// Security Assessment

04.07.2026 - 05.11.2026

Smart Contract Assessment

Templar Protocol

HALBORN

Smart Contract Assessment - Templar Protocol

Prepared by: HALBORN

Last Updated 06/05/2026

Date of Engagement: April 7th, 2026 - May 11th, 2026

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
102	0	4	22	1	75

TABLE OF CONTENTS

1. Introduction
2. Assessment summary
3. High-level recommendations
4. Scope and methodology
5. In-scope components
6. Out of scope
7. System flow
8. Methodology
   - 8.1 Static analysis phases (p1 to p11)
   - 8.2 Dynamic analysis phases (d1 to d12) and hardening pass
9. Threat model and risk assessment
10. 0) executive risk register
    - 10.1 0.a) what is being protected
    - 10.2 0.b) top risks (with confirmed findings reference)
    - 10.3 0.c) operational cadence
11. 1) system context and trust boundaries
12. 2) scenario catalog

12.1 2.a) accounting and conversion
12.2 2.b) withdrawal pipeline
12.3 2.c) governance and authority
12.4 2.d) auxiliary contracts
12.5 2.e) fee accrual
12.6 2.f) storage and migration
12.7 2.g) observability and codec
12.8 2.h) operations and test coverage
13. Key findings and roadmap
14. High severity
15. Selected medium severity
16. Suggested remediation roadmap
16.1 Phase a, immediate (pre-mainnet) hardening
16.2 Phase b, structural / lifecycle
16.3 Phase c, observability and assurance
17. Risk methodology
18. Scope
19. Assessment summary & findings overview
20. Findings & Tech Details
20.1 Positional adapter mapping causes silent market remap on supply queue reorder
20.2 Low-liquidity withdrawal execution can trap the vault in withdrawing with no normal abi recovery path
20.3 Withdraw allocations trust adapter-reported realized assets without verifying token balance delta
20.4 Adapter-reported external assets can be inflated cumulatively or deflated to zero
20.5 Governance contract cannot pause, unpause, or update restrictions on the deployed rbac
20.6 Share-token admin can redirect vault-mint privilege without timelock or event
20.7 Shipped governance contract cannot manage allocators or the adapter allowlist
20.8 Contract-governance deployments have no routed path for upgrade, migrate, or cancel_migration
20.9 Default governance deployment initializes all timelocks to zero
20.10 No production code path adds new markets to the vault
20.11 Vault set_cap rejects every matured governance-proposed cap increase
20.12 Fee accrual is unreachable from the deployed soroban abi
20.13 Initial fee_anchor at (0, 0) inflates proxy_view total_shares once performance fees are enabled
20.14 Soroban handle_deposit does not advance fee_anchor, diverging from the near implementation
20.15 Permissionless extendttl does not refresh persisted address-book mappings required by future effects
20.16 State blobs have no schema version and migrate() does not migrate state or policy blobs
20.17 State blob size cap restricts effective max_pending to 580 pending withdrawals on soroban
20.18 Admin-only blend adapter shortcuts can desynchronize vault accounting
20.19 Blend adapter admin can retarget the vault receiver or pool without timelock or event
20.20 Governance pending queue is stored as a single size-capped instance entry
20.21 Governance configuration and skim commands bypass the migration freeze
20.22 Guardian and sentinel revokers can cancel unrelated governance proposals

20.23 Virtual conversion offsets can be changed after the vault is capitalized

20.24 Timelockconfig can be raised to u64::max immediately, blocking future reductions

20.25 Immediate ŝkim action with default skim recipient drains foreign tokens in one transaction

20.26 Direct underlying transfers remain outside ťotal_assets until manual resync, allowing pre-resync deposits to capture the surplus

20.27 Permissionless initialize allows deployment-race governance takeover if initialization is not atomic

20.28 Share-conversion helpers truncate 256-bit quotients to u128

20.29 Deserialize_fees_spec accepts a 113-byte blob with no-growth tag and silently discards 16 trailing bytes

20.30 Resync_idle_balance advances fee_anchor.total_assets without updating timestamp_ns, distorting future fee math

20.31 Total_assets_for_fee_accrual early-returns when anchor_total_assets == 0, neutralizing the configured max_growth_rate cap

20.32 Migrate() and handle_emergency_reset preserve fee_anchor, perpetuating staleness across recovery paths

20.33 Policy and restrictions decoders accept trailing bytes, unlike the state and command codecs

20.34 Withdraw queue deserialization does not validate queue invariants or sorted pending-withdrawal ids

20.35 Compose_policy_state silently constructs default values for missing storage parts

20.36 Cancel_migration() clears the migration flag but cannot roll back the already-swapped wasm

20.37 Restrictions blob can exceed the soroban entry-size limit when governance configures large blacklist or whitelist sets

20.38 Proxy_view returns max_withdraw and max_redeem for atomic operations the deployed abi cannot perform

20.39 Proxy_view max_deposit and max_mint only check the same-side overflow

20.40 Prepare_atomic_call discards the effectsummary returned by the conditional refreshfees kernel action

20.41 Load_vault_bootstrap runs migrate_legacy_paused on every contract call

20.42 Guardian configuration replaces the full guardian set silently

20.43 Share-token burn paths rely on vault invoker checks instead of owner authorization

20.44 Constructors and governance config setters accept nonsensical role addresses

20.45 Governance timelock legacy fallback can drop per-kind customizations

20.46 Auxiliary contract admin rotation is single-step and emits no event

20.47 Setgovernance accepts the vault or arbitrary non-governance contracts

20.48 Auxiliary contracts lack pause or freeze mechanisms

20.49 Auxiliary contracts have no upgrade or migrate path

20.50 Share-token metadata can be changed after deployment without timelock or event

20.51 Vault restrictions do not propagate to share-token transfers

20.52 Effect-address preflight can succeed for mappings that fail during execution

20.53 Adapter calls can consume the shared transaction instruction budget

20.54 Stale blend reserve data can revert vault allocation and refresh flows

20.55 Refreshmarkets materializes all markets before caller authorization

20.56 Governance command bodies are decoded before authorization and kind validation

20.57 Blend adapter admin cannot be rotated after deployment

20.58 Governance admin cannot be rotated after deployment

20.59 Setmembership governance decisions use incomplete current-state classification

20.60 Blend adapter constructor accepts unvalidated vault pool asset and admin addresses
20.61 Governance abdication symbols disable grouped submit methods rather than one action
20.62 Setmembership policy defaults missing market_id to market zero
20.63 Governance constructor does not reject self-referential or colliding role addresses
20.64 Newly attached governance starts with default local cache rather than vault state
20.65 Accept_kind is ambiguous when multiple proposals share the same action kind
20.66 Vault dispatch helpers accept empty or duplicate input lists
20.67 Storage decoders pre-allocate vectors from untrusted length prefixes
20.68 Test suite uses permissive authorization paths that diverge from production rbac
20.69 Allocation lifecycle test helpers bypass production kernel action handling
20.70 Vault uses stellarassetclient for custom share-token interactions
20.71 Integration tests seed state outside the production onboarding flow
20.72 Test allocation helper drops leased targets unlike production allocation
20.73 Property tests labeled external growth do not model real adapter growth
20.74 Preview and fee tests duplicate production internals with different semantics
20.75 Soroban kernel parity tests skip soroban-specific paths and high-value ranges
20.76 Governance mockvault ignores invalid or incomplete command payloads
20.77 Proxy_view casts u128 values to i128 without bounds checks
20.78 Governance is granted full curator role instead of the narrower policy-admin need
20.79 Fee recipient configuration is positional and can be miswired by reordering
20.80 Address text conversion can panic or silently fall back on malformed inputs
20.81 Fee preview saturates supply values without surfacing an overflow signal
20.82 Test-only sorobanauth creates misleading production authorization coverage
20.83 Event payloads omit structured addresses and typed topics across vault components
20.84 Asset and share-token admin assumptions are enforced only by deployment discipline
20.85 Proxy_view fee snapshot omits the configured fee growth-rate cap
20.86 Governance abdicate accepts unknown symbols as silent no-ops
20.87 Other effect approvals are unreachable from the deployed vault command surface
20.88 Revoke_kind cancels all proposals of a kind while accept_kind accepts only one
20.89 Vault command wire codec lacks versioning and collapses decoder error variants
20.90 Multiple wire encodings can represent the same governance policy action
20.91 Governance direction classifiers inconsistently detect no-change proposals
20.92 Load_timelocks can mutate storage during view-style paths
20.93 Share-token burn and burn_from emit indistinguishable burn events
20.94 Share-token contract-address validation relies on strkey prefix inspection
20.95 In-flight governance proposals retain submit-time timelock after later timelock raises
20.96 Share-token archival recovery procedure is undefined
20.97 Share-token tests omit admin rotation and allowance security coverage
20.98 Share-token read-only methods do not refresh instance ttl
20.99 Share-token constructor emits no deployment configuration event
20.100 Share-token extend_ttl does not refresh holder balances or allowances
20.101 Addressmap constructor accepts an unused environment parameter
20.102 Immediate governance submissions write the pending queue twice

1. Introduction

Templar Protocol engaged **Halborn** to perform a focused Smart Contract Assessment of the **Templar Soroban Vault**, a curator-driven, over-collateralized lending vault deployed on Stellar's **Soroban** runtime. The vault is the deployable Soroban executor for the shared, chain-agnostic `templar-vault-kernel` and ships together with a dedicated governance contract, a SEP-41 share-token contract, and a Blend lending-market adapter. The objective of this engagement was to validate the safety of asset accounting, redemption, governance authority, migration, and adapter integrations under realistic adversarial conditions on Soroban.

2. Assessment Summary

Halborn performed manual review, line-by-line analysis, focused regression and property-style testing, and an adversarial hardening pass that converted theoretical findings into runnable PoC tests across the four in-scope contract crates. In total, **102 issues** were confirmed: **4 High**, **22 Medium**, **1 Low**, and **75 Informational**. Severity was assigned using **BVSS 2.0**.

The most impactful issues are concentrated in three areas: **(i) adapter-trust accounting**, where the vault credits adapter-reported realized assets and external-asset totals without bounding cumulative growth or verifying token-balance deltas, enabling share-price corruption under a malicious or stale adapter; **(ii) withdrawal liveness**, where a routine allocator-class `ExecuteWithdraw` can place the vault into a persistent `Withdrawing` stuck state with no normal ABI recovery surface; and **(iii) governance and migration semantics**, where the governance contract cannot reach `Pause` / `SetRestrictions` under the default RBAC, several governance configuration paths bypass the migration freeze, and emergency-class roles can revoke unrelated proposals. A fourth high-severity issue, **positional adapter mapping**, can silently re-bind every market to the wrong adapter on a routine supply-queue reorder.

Medium-severity issues cluster around the **fee-anchor lifecycle** (the anchor is mishandled across `initialize`, deposit, resync, fee changes, migration, and emergency reset), the **governance command surface** (no production submit paths for several admin surfaces, default zero timelocks, `TimelockConfig` may be raised to `u64::MAX`, immediate `Skim` + default skim recipient drains foreign tokens, share-token `set_vault` has no timelock or event, virtual offsets remain mutable post-capitalization), and **state durability** (state and restrictions blobs lack a schema version and can exceed Soroban's per-entry size at high queue depths). Informational findings primarily concern observability gaps, view-side rounding / saturation, codec canonicalization, and dead or test-only paths.

All Medium and High findings include reproducible regression tests; the dynamic **hardening pass** added **46 adversarial tests across 10 files** over the prior ~250 baseline tests and confirmed five concrete exploit chains (initialize frontrun + skim drain, multi-step adapter inflation, governance + migration lockout, skim-recipient rugpull, share-token rebind + admin chain).

3. High-Level Recommendations

- **Adapter accounting (Immediate priority, High):** derive realized withdrawal amounts from `balance_after − balance_before` instead of trusting adapter return values, and enforce a per-window cumulative growth / shrink cap on external-asset syncs in addition to the existing per-call 2x guard. Bind adapters to `market_id` via an explicit map so supply-queue reorders cannot silently rebind markets.
- **Redemption liveness (High):** prevent `OpState::Withdrawing` entry when idle assets cannot satisfy at least `MIN_WITHDRAWAL_ASSETS`, expose an allocator-only continuation or `AbortWithdrawing` recovery path, and allow `ResyncIdleBalance` while in `Withdrawing` so that direct asset transfers can unblock a stuck queue.
- **Governance correctness (Medium):** grant the governance contract both `Curator` and `Sentinel` in `load_vault_bootstrap` so it can actually reach `Pause` / `SetRestrictions`; gate every `SetGovernanceConfig` branch on `migration_in_progress`; cap `MAX_TIMELOCK_NS` at a realistic upper bound; reject zero-timelock deployments; add submit paths for allocators, adapter allowlist, virtual offsets, and the upgrade / migrate / cancel commands.
- **Auxiliary contracts (Medium):** convert share-token `set_vault` / `set_admin` and Blend adapter `set_vault` / `set_pool` / `set_admin` into two-step rotations with timelocks and structured events; lock share-token decimals; propagate vault pause to auxiliaries; add upgrade / migrate paths.
- **State durability (Medium):** prepend a schema-version byte to every persisted blob and route `migrate()` through it; lower kernel-side `MAX_PENDING` to a value that fits comfortably below Soroban's per-entry size cap, or page the withdraw and governance queues into per-entry storage; validate encoded sizes on every write.
- **Observability (Informational / Medium):** resolve kernel-hash addresses to Soroban addresses before emit, use structured topics and versioned event payloads, add events for `initialize` and auxiliary deployments, and emit per-token amounts on `Skim`.

4. Scope And Methodology

5. In-Scope Components

The assessment covered the entire Soroban-side vault stack, the chain-agnostic kernel it embeds, and every auxiliary contract bound to the deployed vault. All file paths are relative to the repository at **github.com/Templar-Protocol/contracts**.

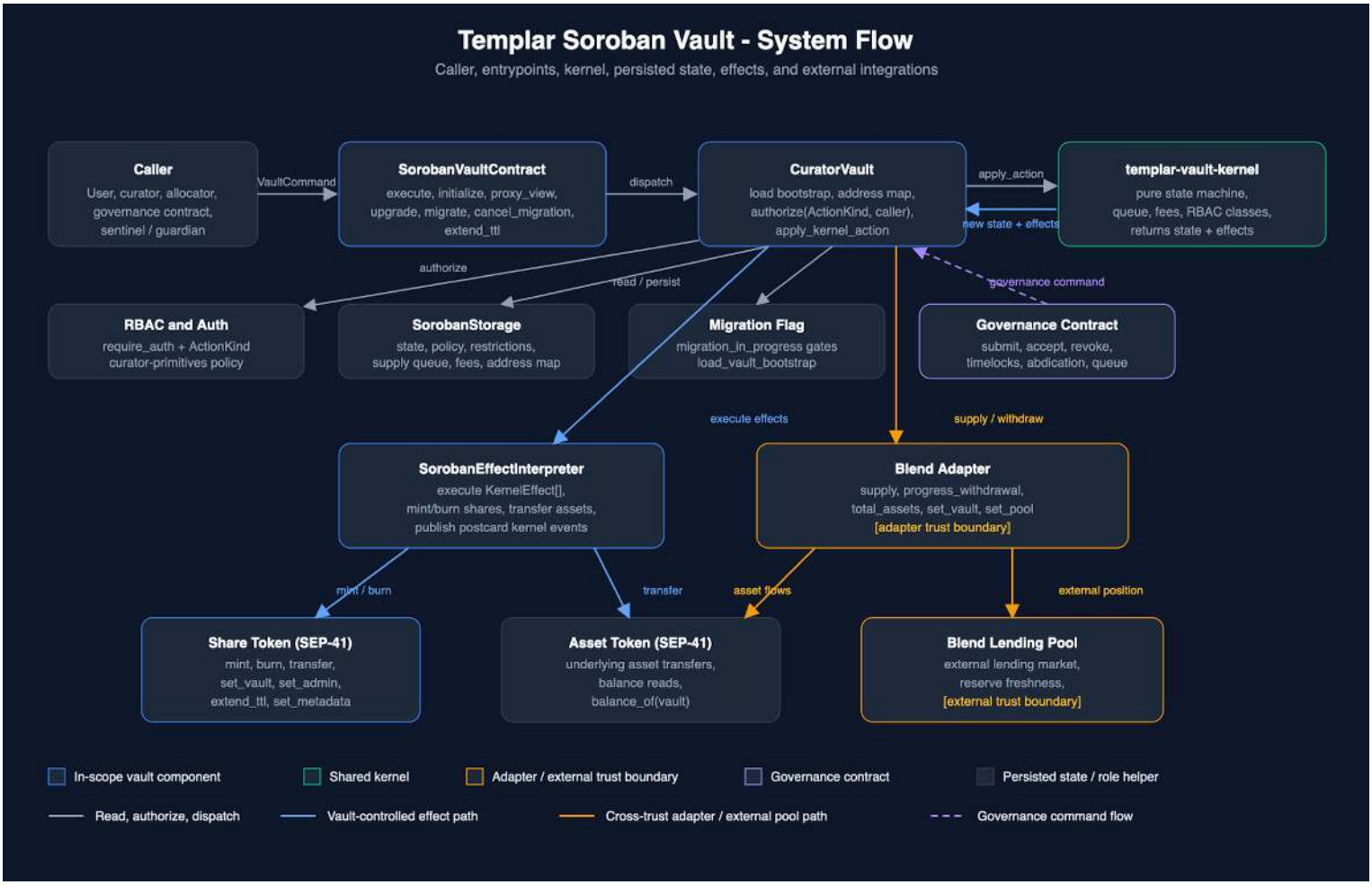
- **Soroban vault runtime**: `contract/vault/soroban/src/**` (entrypoints, curator vault, effect interpreter, address map, storage, helpers, market adapter wrappers, fungible-vault math, proxy view).
- **Governance contract**: `contract/vault/soroban/governance/**` (submission, timelock, revocation, abdication, queued action persistence).
- **Share-token (SEP-41)**: `contract/vault/soroban/share-token/**` (admin rotation, vault rebind, mint / burn, allowance, TTL).
- **Blend adapter**: `contract/vault/soroban/blend-adapter/**` (supply / withdraw, pool / vault retargeting, admin-only manual paths, reserve freshness).
- **Vault kernel and curator primitives**: `contract/vault/kernel/**` and `contract/vault/curator-primitives/**` (state machine, queue, fees math, RBAC classes, policy state).
- **Shared types and wire codec**: `contract/vault/soroban/shared-types/**` (postcard-based `VaultCommand` / `VaultCommandResult`).
- **STRIDE model**: `contract/vault/soroban/STRIDE.md` was reviewed and used to seed adversarial test cases.

6. Out Of Scope

- External protocols (Blend pool internals, SEP-41 SAC reference implementation, Stellar core protocol).
- Off-chain infrastructure (relayer, keepers, indexers) unless their assumptions affect on-chain invariants.
- Front-end and operational tooling.

7. System Flow

The diagram below summarizes how a caller-supplied `VaultCommand` travels through the Soroban entrypoint, the `CuratorVault` wrapper, and the shared kernel, then materializes as on-chain effects against the share token, the asset token, and the Blend lending pool. The governance contract acts as an in-protocol caller that dispatches authorized commands into the same vault surface.



LAYER	COMPONENT	ROLE
Entry	`SorobanVaultContract`	Public entrypoints: `execute`, `initialize`, `proxy_view`, `upgrade`, `migrate`, `cancel_migration`, `extend_ttl`.
Runtime	`CuratorVault`	Loads bootstrap state, maps addresses, authorizes by `ActionKind`, calls `apply_kernel_action`.
Kernel	`templar-vault-kernel`	Pure state machine returning new state plus a `KernelEffect[]` queue.
State	`SorobanStorage`, `migration_in_progress`, RBAC config	Persists state, policy, restrictions, supply queue, fees, address map; gates entry while migrating.

Effects	`SorobanEffectInterpreter`	Executes share / asset effects, publishes postcard kernel events.
Governance	`Governance Contract`	Submits, accepts, revokes, and dispatches commands; manages timelocks, abdication, queued actions.
Tokens	Share token, asset token	SEP-41 contracts holding share supply and underlying asset balances.
External	Blend adapter, Blend lending pool	Sit across an adapter trust boundary; report `total_assets` and route external positions.

8. Methodology

The engagement combined manual code review, threat-model-driven analysis, and dynamic regression / property testing structured into phased work blocks. Severity assignment used **BVSS 2.0** and consolidation removed duplicates, false positives, and reclassified cross-cutting issues.

8.1 Static Analysis Phases (P1 To P11)

- **P1.** Vault entrypoints, deposit and conversion math.
- **P2.** RBAC, initialize, and governance authority surfaces.
- **P3.** Withdrawal pipeline, queue, and cooldown invariants.
- **P4.** Fee accrual anchor lifecycle and proxy-view consistency.
- **P5.** Storage, migration, and TTL handling.
- **P6.** Effect interpreter, address mapping, and cross-contract calls.
- **P7.** Governance contract, timelocks, revocation, and abdication.
- **P8.** Share-token (SEP-41) contract.
- **P9.** Blend adapter.
- **P10.** Command wire codec and command dispatch surface.
- **P11.** Existing test surface, parity tests, and harness assumptions.

8.2 Dynamic Analysis Phases (D1 To D12) And Hardening Pass

- **D1.** Storage codec round-trips and malformed input.
- **D2.** Initialize and bootstrap state.
- **D3.** Deposit and share conversion at boundaries.
- **D4.** Fuzz targets and property tests on conversion / storage.
- **D5.** Effect interpreter and address mapping property tests.
- **D6.** Withdrawal pipeline, FIFO, cooldown, partial-idle settlement.
- **D7.** Governance timelock matrix, revoker matrix, queue replacement.
- **D8.** Share-token and Blend adapter integration tests.
- **D9.** Migration / upgrade / cancel lifecycle.
- **D10.** Performance, storage sizing, and queue depth.
- **D11.** Kernel / Soroban parity and high-value arithmetic.
- **D12.** Event integrity (kernel, admin, governance, auxiliary).
- **Hardening pass.** 46 adversarial tests in 10 files covering inflation, donation griefing, initialize frontrun, storage corruption, adapter compromise, governance compromise chain, atomic-vs-Withdrawing race, skim-recipient rugpull, share-token rebind, and governance queue DoS.

9. Threat Model And Risk Assessment

Audience: Protocol engineers, Smart-contract reviewers, Operators, Curators, Governance signers, Executive stakeholders.

Scope: Templar Soroban Vault stack as defined in the Scope section, with cross-references to the existing `contract/vault/soroban/STRIDE.md` threat model.Method / Framework

- **DFD-lite plus trust boundaries** (Section 1) to clarify where each control must hold.
- **STRIDE-informed risk register** (Section 0) expressed for non-engineers, with operational regulation, hardening, and ongoing assurance per risk.
- **Scenario catalog** (Section 2) covering accounting, governance, adapter trust, migration, observability, and operational risk.
- **Recurring assurance cadence** (Section 0.c) to ensure these controls are maintained release over release.

10. 0) Executive Risk Register

10.1 0.A) What Is Being Protected

- **Vault asset accounting:** `state.idle_assets`, `state.external_assets`, and per-market principal must reflect on-chain token balances at all times.
- **Share-price integrity:** `convert_to_shares` / `convert_to_assets` outputs must depend only on real assets and shares plus governance-bounded offsets, with virtual offsets immutable post-capitalization.
- **Redemption liveness:** queued withdrawals must always have a callable recovery path, even at low idle balances or after partial allocations.
- **Governance authority:** the governance contract is the documented control plane; pause, restrictions, and configuration changes must route through it, with timelocks where intended.
- **Migration safety:** while `migration_in_progress` is set, no state-mutating command should be reachable, including configuration writes that change recovery authority.
- **Auxiliary contracts:** share-token and Blend adapter must not provide silent rotation surfaces that break vault expectations.

10.2 0.B) Top Risks (With Confirmed Findings Reference)

Risk 1 (P0): Adapter Trust Corruption

- **What could happen:** A malicious, stale, or compromised adapter reports realized withdrawals or external-asset totals that the vault credits without a balance-delta check. The vault's accounting drifts upward or downward independent of real tokens held.
- **Confirmed in:** Withdraw allocations trust adapter-reported realized assets (High); adapter-reported external assets can be inflated cumulatively or deflated to zero (High); admin-only manual Blend shortcuts desync vault accounting (Medium); positional adapter mapping silently misroutes funds on supply-queue reorder (High).
- **Operational regulation:** require curators to whitelist adapter implementations via on-chain governance; monitor every `RefreshMarkets` call for delta size; alert on adapter rotations.
- **Hardening controls:** derive realized amounts from token-balance deltas; enforce cumulative per-window growth / shrink caps; map adapters by `market_id` instead of queue position; reject queue reorders that change any existing market-to-adapter binding.
- **Ongoing assurance:** property tests fuzzing adapter return values independent of token transfers, and regression tests for queue / adapter rebinding on every release.

Risk 2 (P0): Redemption Liveness Failure

- **What could happen:** The vault transitions into `OpState::Withdrawing` without enough idle assets to make progress; the shipped Soroban ABI exposes no normal recovery path (`start_allocation`, `ResyncIdleBalance`, and `RefreshMarkets` all require `Idle`; `VaultCommand` has no `AbortWithdrawing`).
- **Confirmed in:** Low-liquidity withdrawal execution can trap the vault in Withdrawing with no normal ABI recovery path (High).
- **Operational regulation:** monitor for transitions into `Withdrawing` followed by zero asset movement; alert on stuck queues approaching cooldown expiry.
- **Hardening controls:** gate the `Withdrawing` transition on minimum idle availability; allow `ResyncIdleBalance` while in `Withdrawing`; expose an allocator-only abort / continuation command.
- **Ongoing assurance:** stuck-state regression test in CI; structured `VaultCommandResult` exposing "entered Withdrawing but paid nothing" so off-chain automation can react.

Risk 3 (P0): Governance Authority Compromise Or Lockout

- **What could happen:** A compromised governance key promotes itself to sentinel (immediate), pauses indefinitely (immediate), then uses `SetGovernanceConfig` during migration to rotate curator / governance into attacker-controlled addresses while operators expect the freeze to hold.
- **Confirmed in:** Governance contract cannot pause / unpause or update restrictions on the deployed RBAC (Medium); several governance commands proceed during `migration_in_progress` (Medium); `TimelockConfig` can be raised to `u64::MAX` immediately (Medium); guardian / sentinel revokers can cancel any pending proposal (Medium); default deploy initializes all timelocks to zero (Medium).
- **Operational regulation:** use phishing-resistant MFA and multi-party signing for governance keys; treat upgrade windows as monitored events; require post-upgrade attestation that no role rotated during migration.
- **Hardening controls:** grant governance both `Curator` and `Sentinel`; enforce `require_not_migrating` on every `set_governance_config_impl` branch; cap `MAX_TIMELOCK_NS` at a realistic upper bound; scope revoker authority by action kind; reject zero-timelock deployments at the governance constructor.
- **Ongoing assurance:** dynamic test that asserts every state-mutating command is rejected while the migration flag is set; matrix tests for revoker scope and timelock-bound enforcement.

Risk 4 (P0): Share-Price Re-Pricing Via Virtual Offsets Or Share-Token Rebind

- **What could happen:** Governance moves virtual offsets after capitalization, instantly changing the conversion formula and diluting or rewarding existing holders without an explicit price event. In parallel, the share-token admin redirects the mint / burn privilege to an attacker contract in a single transaction with no event.

- **Confirmed in:** Virtual offsets can be changed after capitalization (Medium); share-token admin can redirect vault-mint privilege without timelock or event (Medium); auxiliary contract admin rotation is single-step (Low / Medium variants).
- **Operational regulation:** require change control and post-mortem review for any virtual-offset update; monitor share-token `vault()` / `admin()` accessors; bind share-token admin keys to multi-party signing.
- **Hardening controls:** lock virtual offsets after the first deposit (or after a configurable bootstrap window); convert share-token `set_vault` / `set_admin` to two-step rotations with timelocks and events; reject rotations to self, asset-token, or share-token addresses.
- **Ongoing assurance:** share-price regression test that asserts `convert_to_shares` is stable across offset writes once capitalization has occurred; share-token rotation event monitoring in CI.

Risk 5 (P1): Fee Anchor Drift And Accrual Unreachable

- **What could happen:** The fee-accrual anchor is left at `(0, 0)` at initialize, never advanced on Soroban `handle_deposit`, frozen by `resync_idle_balance` for shrinks, neutralized by the early-return on `anchor_total_assets == 0`, not reset on `SetFees`, and preserved across `migrate()` / emergency-reset. The deployed `VaultCommand` set never reaches `RefreshFees`.
- **Confirmed in:** Cluster of fee-anchor findings across `initialize`, `handle_deposit` divergence from NEAR, `resync_idle_balance` timestamp handling, `total_assets_for_fee_accrual` early return, `migrate` / `handle_emergency_reset` preservation, retroactive fee application, and proxy-view share-count inflation.
- **Operational regulation:** document the fee anchor as a single invariant (the anchor reflects `(total_assets, ledger_timestamp_ns)` after every deposit, fee change, refresh, resync, migrate, and emergency-reset) and require updates to be reviewed against it.
- **Hardening controls:** advance the anchor on every Soroban deposit; reset on `SetFees` and after recovery paths; remove the `anchor_total_assets == 0` early return; expose `RefreshFees` through a governance-routed command; fuzz the full state machine and assert the anchor invariant.
- **Ongoing assurance:** property test walking the full state machine and asserting the anchor invariant; preview tests that drive `proxy_view` instead of duplicating math.

Risk 6 (P1): Storage / Migration Durability

- **What could happen:** State, policy, restrictions, supply-queue, market, principal, cap-group, and fees-spec blobs lack a schema version. The withdraw queue and governance pending queue each live in a single Soroban entry capped at 64 KiB, so kernel-advertised limits are unreachable in practice and large restrictions writes can brick subsequent updates. `cancel_migration()` cannot roll back an already-swapped WASM.
- **Confirmed in:** State blob size cap restricts effective `MAX_PENDING` (Medium); restrictions blob can exceed entry-size limit (Medium); state blobs lack schema version and `migrate()` does not migrate state

/ policy blobs (Medium); decoders accept trailing bytes or non-canonical fees blobs (Medium / Low); `cancel_migration()` cannot roll back WASM (Low).

- **Operational regulation:** document the per-entry size budget and require change control for any structural addition; preserve the previous WASM hash so an emergency revert is possible.
- **Hardening controls:** prepend a schema-version byte to every persisted blob; lower kernel-side `MAX_PENDING`; page the withdraw and governance queues; validate encoded sizes on write; reject trailing bytes in policy / restrictions / fees decoders.
- **Ongoing assurance:** regression tests across blob-size boundaries and migration round-trip tests for downgrade or forward migration.

Risk 7 (P1): Skim And Foreign-Token Drain Via Default Recipient

- **What could happen:** `Skim` is `Immediate`, the default skim recipient is the governance contract itself, and the action accepts any non-asset, non-share token. A single governance approval moves every non-principal token balance with no timelock, no per-token allowlist, and no per-balance event.
- **Confirmed in:** Immediate `Skim` action with default skim recipient drains foreign tokens in one transaction (Medium); fee recipient configuration is positional (Informational).
- **Operational regulation:** document skim policy and recipient; review every `SetGovernanceConfig(SKIM_RECIPIENT)` and `Skim` command against an allowlist of expected tokens.
- **Hardening controls:** make `Skim` timelocked or require a per-token allowlist; emit per-token, per-balance events; tighten `SetGovernanceConfig(SKIM_RECIPIENT)` to the timelocked governance path.
- **Ongoing assurance:** regression test that `Skim` emits a structured per-token event and that the recipient rotation requires the governance flow.

Risk 8 (P2): Codec, Observability, And Dispatch Hygiene

- **What could happen:** Off-chain monitors miss state changes because addresses are raw kernel hashes, events have a single topic, and several action surfaces emit no event at all (e.g., share-token `set_vault`, `set_admin`, constructor). The wire codec has no version field, collapses decoder errors, and accepts multiple encodings for the same logical action.
- **Confirmed in:** Event observability gaps cluster (Informational); wire codec lacks version, collapses error variants (Informational); multiple encodings map to the same governance action (Informational); `burn` / `burn_from` emit identical events (Informational).
- **Operational regulation:** treat events as a public API; review event schema changes alongside ABI changes.
- **Hardening controls:** resolve addresses on emit; use structured topics; version event payloads; prepend a codec version byte; route decoder errors into structured variants; canonicalize commands.
- **Ongoing assurance:** event-integrity test suite (added in D12) extended over time as the surface grows.

10.3 0.C) Operational Cadence

- **Every release:** run accounting parity, fee anchor invariant, withdrawal liveness, migration freeze, and adapter accounting regression packs.
- **Daily:** monitor for `Withdrawing` entries with zero payout, abnormal `RefreshMarkets` deltas, and governance-key access.
- **Weekly:** review queued governance proposals, pending timelock matures, and any share-token / adapter admin events.
- **Monthly:** verify configured timelocks against the documented minimums; verify per-entry blob sizes are within the budget; run the adapter compromise property tests.
- **Quarterly:** purple-team exercise covering initialize-frontrun, governance-compromise chain, skim rugpull, adapter inflation, and migration lockout.
- **Semi-annual:** emergency-reset and migration tabletop, including WASM rollback rehearsal.

11. 1) System Context And Trust Boundaries

The deployed system is a four-contract Soroban deployment plus the curator-driven vault kernel embedded in the runtime. The primary trust boundaries are summarized below.

- **TB-1, Caller / role boundary:** Soroban-side `RbacAuth` over `ActionKind` classes (`Curator`, `Sentinel`, `Allocator`, `PolicyAdmin`) gates access to vault entrypoints.
- **TB-2, Vault / governance boundary:** Governance is the documented control plane and the only address granted `Role::Curator` by `load_vault_bootstrap`; configuration writes must travel through it.
- **TB-3, Vault / adapter boundary:** Adapters are external contracts that may report realized amounts and total assets; the vault must not credit those values without local verification.
- **TB-4, Vault / share-token boundary:** Mint / burn is privileged on the share-token via `require_vault_invoker`; rotation of `vault` or `admin` on the share-token instantly changes the trust chain.
- **TB-5, Vault / migration boundary:** While `migration_in_progress = true`, no state-mutating command should be reachable; only `migrate` and `cancel_migration` should clear the flag.
- **TB-6, Storage boundary:** Persisted blobs (state, policy, restrictions, supply-queue, markets, principal, cap-groups, fees-spec) define the schema contract; any change must preserve forward and backward compatibility.

12. 2) Scenario Catalog

12.1 2.A) Accounting And Conversion

- **Adapter realized-amount inflation:** credit-without-balance-check, mitigated by deriving realized amounts from token-balance deltas (High).
- **External-asset cumulative inflation:** per-step 2x cap insufficient; mitigated by cumulative per-window caps and an explicit loss / finality flow (High).
- **Direct-transfer mispricing:** unaccounted donations until manual resync; mitigated by auto-resync or pre-deposit balance check (Medium).
- **Virtual-offset repricing post-capitalization:** immutable-after-bootstrap or bounded change rate (Medium).
- **Share-conversion truncation:** `as_u128_trunc` downcast must be replaced with a checked conversion (Medium).

12.2 2.B) Withdrawal Pipeline

- **Low-idle stuck state:** withdrawal liveness failure with no normal ABI recovery (High).
- **Queue durability:** single-entry queue capped by per-entry size (Medium); withdraw queue decode does not validate invariants (Low).
- **Partial-idle dequeue semantics:** documented but currently surfaces as proportional payout without explicit caller indication.

12.3 2.C) Governance And Authority

- **Pause / restrictions unreachable from governance under default RBAC** (Medium).
- **Migration-freeze bypass via** `SetGovernanceConfig` (Medium).
- `TimelockConfig` **can be raised to** `u64::MAX` (Medium).
- **Guardian / sentinel revoker overreach** (Medium).
- **Default zero-timelock deployment** (Medium).
- **Missing submit paths** for allocators, allowed-adapters, virtual offsets, upgrade, migrate, and cancel_migration (Medium).
- **Permanent governance and adapter admin roles** with no rotation (Medium / Low).
- **SetGovernance accepts arbitrary contracts** including the vault itself (Low).

12.4 2.D) Auxiliary Contracts

- **Share-token** `set_vault` **/** `set_admin` **with no timelock or event** (Medium / Low).

- **Blend adapter `set_vault` / `set_pool` with no timelock or event** (Medium).
- **Admin-only manual Blend supply / withdraw** bypassing vault accounting (Medium).
- **Auxiliary contracts lack upgrade / migrate and pause / freeze** mechanisms (Low).
- **Vault Restrictions not propagated to share-token transfers** (Low).

12.5 2.E) Fee Accrual

- **Anchor lifecycle** mishandled end-to-end (Medium cluster, see Risk 5).
- **Fee recipients positional** (Informational).

12.6 2.F) Storage And Migration

- **Schema versioning absent** across persisted blobs (Medium).
- **Per-entry size constraints** on state, restrictions, and pending queues (Medium / Low).
- **Permissionless `ExtendTtl`** not refreshing address-book entries (Medium).
- **`cancel_migration()` WASM rollback gap** (Low).
- **Trailing-byte / non-canonical decoding** across policy, restrictions, and fees (Medium / Low).

12.7 2.G) Observability And Codec

- **Raw kernel addresses, single topics, unversioned payloads** across vault, governance, share-token, and adapter (Informational cluster).
- **Wire codec lacks version, collapses decoder errors** (Informational).
- **Multiple encodings represent the same governance action** (Informational).

12.8 2.H) Operations And Test Coverage

- **Permissive auth helpers in tests diverge from production RBAC** (Medium).
- **Allocation lifecycle helpers bypass `apply_kernel_action`** (Medium).
- **External-growth and parity tests do not exercise Soroban paths or high-value boundaries** (Low).
- **Preview, fee, and onboarding tests duplicate production internals or pre-seed state** (Low / Informational).

13. Key Findings And Roadmap

14. High Severity

- **Positional adapter mapping causes silent market remap on supply queue reorder.** Adapters are bound to markets by queue position; a routine queue reorder rebinds every market to the wrong adapter and silently misroutes allocator funds.
- **Low-liquidity withdrawal execution can trap the vault in `Withdrawing` with no normal ABI recovery path.** `ResyncIdleBalance`, `start_allocation`, and `refresh_markets` all require `Idle`; no `AbortWithdrawing` exists.
- **Withdraw allocations trust adapter-reported realized assets without verifying token-balance delta.** Adapter return value is credited to `idle_assets` without a before / after balance check.
- **Adapter-reported external assets can be inflated cumulatively or deflated to zero.** The single-step 2x cap does not prevent compounded growth across transactions; deflation is unbounded in one step.

15. Selected Medium Severity

- **Governance contract cannot pause / unpause or update restrictions under default RBAC**, governance only holds `Curator`; pause routes through `Sentinel`.
- **Several governance commands proceed during `migration_in_progress`**, curator, governance, sentinel, guardians, allocators, skim recipient, virtual offsets, fees, and `Skim` are reachable mid-migration.
- **Immediate `Skim` action with default skim recipient drains foreign tokens in one transaction.**
- **Share-token `set_vault` has no timelock or event.**
- **Default governance deployment initializes all timelocks to zero; `TimelockConfig` can be raised to `u64::MAX` immediately**, blocking future reductions.
- **Shipped governance contract is missing submit methods** for allocators, allowed-adapters, virtual offsets, upgrade, migrate, and cancel_migration.
- **Fee accrual is unreachable from the deployed Soroban ABI** (no `RefreshFees` command) and the fee anchor is mishandled across `initialize`, deposit, resync, migration, and emergency reset.
- **State and restrictions blobs lack a schema version** and can exceed Soroban's per-entry size at high queue depths.
- **Permissionless `ExtendTtl` does not refresh persisted address-book mappings** used by future effects.
- **Direct underlying transfers remain outside `total_assets` until manual resync**, allowing pre-resync deposits to capture the surplus.
- **Virtual conversion offsets can be changed after the vault is capitalized**, instantly changing share pricing.
- **Vault set_cap rejects every matured governance-proposed cap increase** because of a `current = None` classification mismatch.
- **Blend adapter admin can retarget the vault receiver or pool without timelock or event**, including admin-only manual paths that desync vault accounting.
- **Permissive auth helpers in the test suite diverge from production RBAC**, masking production-only failures.

16. Suggested Remediation Roadmap

16.1 Phase A, Immediate (Pre-Mainnet) Hardening

- Bind adapters by `market_id`; derive realized amounts from balance deltas; enforce cumulative external-asset caps.
- Guard `Withdrawing` entry on minimum idle and expose a recovery surface.
- Grant governance `Sentinel`; gate every `SetGovernanceConfig` branch on the migration flag.
- Cap `MAX_TIMELOCK_NS` at a realistic upper bound; reject zero-timelock deployments; restrict revoker scope.
- Convert share-token `set_vault` / `set_admin` and Blend adapter `set_vault` / `set_pool` / `set_admin` into two-step rotations with timelocks and events.
- Reset the fee anchor on every `SetFees`, deposit, resync, migrate, and emergency-reset; expose `RefreshFees` via governance.

16.2 Phase B, Structural / Lifecycle

- Prepend a schema-version byte to every persisted blob and route `migrate()` through it; page the withdraw and governance queues.
- Add governance submit methods for allocators, allowed-adapters, virtual offsets, and the upgrade / migrate / cancel commands.
- Lock virtual offsets after the bootstrap window; reject post-capitalization moves that imply a price delta beyond a configurable tolerance.
- Add admin-rotation and upgrade / migrate / pause paths to the share-token and Blend adapter.
- Tighten `Skim` with a per-token allowlist or governance-routed timelock.

16.3 Phase C, Observability And Assurance

- Resolve kernel addresses on emit, use structured topics, and version event payloads across all four contracts.
- Add the BVSS-flagged property tests to CI: adapter-trust fuzz, fee-anchor invariant, withdrawal liveness, migration freeze, governance revoker scope, and storage round-trip / size boundaries.
- Replace permissive auth helpers in critical tests with production `RbacAuth` wiring and add one full deploy-to-allocate path that uses only production entrypoints.

17. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

17.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

17.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (C:N) Low (C:L) Medium (C:M) High (C:H) Critical (C:C)	0 0.25 0.5 0.75 1
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

17.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4

SEVERITY	SCORE VALUE RANGE
Informational	0 - 1.9

18. SCOPE

FILE

(a) Submitted File: soroban.zip

REMEDIATION COMMIT ID:

- d2ea128
- 66e68f8
- c2b007f
- 80836ac
- 488efb4
- 5c302e8
- 0aebe10
- 843a36f
- b6ee556
- https://github.com/Templar-Protocol/contracts/pull/417
- 12916e1
- 4b4232f
- 2c7def2
- 4440115
- fcab9d3
- fee307b
- e1d1ce9
- 23878c9
- 8f8b075
- 21ac8b5
- d31bf3c
- 6052b1a
- 381a450
- abfd943
- 236acee
- https://github.com/Templar-Protocol/contracts/pull/430
- 9febe56
- aeba3fc
- 18de52b
- 335a946

- e381fc2
- fe283e7
- 61fb75e
- 5b03b63
- 5cfbcff
- 0aad51e
- 08bd07a
- https://github.com/Templar-Protocol/contracts/pull/448
- e951f6f
- 4fab9e0
- 3c1b5f9
- b2fe521
- 28e2470
- 2c806e7
- 8d1fd65
- e9cc5c3
- e0d95e3
- 3c8eb81
- 1d254e7
- d7cb3d2
- 31137e2
- 5842847
- 35b3685
- 75edc6c
- 0a0e7b0
- 863a2b6
- 2506f13
- 1e19017
- 0358c07
- 3c5d098
- e189e24
- 3837cf4

Out-of-Scope: New features/implementations after the remediation commit IDs.

19. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	4	22	1	75

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
POSITIONAL ADAPTER MAPPING CAUSES SILENT MARKET REMAP ON SUPPLY QUEUE REORDER	HIGH	SOLVED - 05/19/2026
LOW-LIQUIDITY WITHDRAWAL EXECUTION CAN TRAP THE VAULT IN WITHDRAWING WITH NO NORMAL ABI RECOVERY PATH	HIGH	SOLVED - 05/19/2026
WITHDRAW ALLOCATIONS TRUST ADAPTER-REPORTED REALIZED ASSETS WITHOUT VERIFYING TOKEN BALANCE DELTA	HIGH	SOLVED - 05/19/2026
ADAPTER-REPORTED EXTERNAL ASSETS CAN BE INFLATED CUMULATIVELY OR DEFLATED TO ZERO	HIGH	SOLVED - 05/19/2026
GOVERNANCE CONTRACT CANNOT PAUSE, UNPAUSE, OR UPDATE RESTRICTIONS ON THE DEPLOYED RBAC	MEDIUM	SOLVED - 05/19/2026
SHARE-TOKEN ADMIN CAN REDIRECT VAULT-MINT PRIVILEGE WITHOUT TIMELOCK OR EVENT	MEDIUM	SOLVED - 05/19/2026
SHIPPED GOVERNANCE CONTRACT CANNOT MANAGE ALLOCATORS OR THE ADAPTER ALLOWLIST	MEDIUM	SOLVED - 05/19/2026
CONTRACT-GOVERNANCE DEPLOYMENTS HAVE NO ROUTED PATH FOR UPGRADE, MIGRATE, OR CANCEL_MIGRATION	MEDIUM	SOLVED - 05/19/2026
DEFAULT GOVERNANCE DEPLOYMENT INITIALIZES ALL TIMELOCKS TO ZERO	MEDIUM	SOLVED - 05/19/2026

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
NO PRODUCTION CODE PATH ADDS NEW MARKETS TO THE VAULT	MEDIUM	SOLVED - 05/19/2026
VAULT SET_CAP REJECTS EVERY MATURED GOVERNANCE-PROPOSED CAP INCREASE	MEDIUM	SOLVED - 05/19/2026
FEE ACCRUAL IS UNREACHABLE FROM THE DEPLOYED SOROBAN ABI	MEDIUM	SOLVED - 05/19/2026
INITIAL FEE_ANCHOR AT (0, 0) INFLATES PROXY_VIEW TOTAL_SHARES ONCE PERFORMANCE FEES ARE ENABLED	MEDIUM	SOLVED - 05/19/2026
SOROBAN HANDLE_DEPOSIT DOES NOT ADVANCE FEE_ANCHOR, DIVERGING FROM THE NEAR IMPLEMENTATION	MEDIUM	SOLVED - 05/19/2026
PERMISSIONLESS EXTENDTTL DOES NOT REFRESH PERSISTED ADDRESS-BOOK MAPPINGS REQUIRED BY FUTURE EFFECTS	MEDIUM	SOLVED - 05/19/2026
STATE BLOBS HAVE NO SCHEMA VERSION AND MIGRATE() DOES NOT MIGRATE STATE OR POLICY BLOBS	MEDIUM	SOLVED - 05/19/2026
STATE BLOB SIZE CAP RESTRICTS EFFECTIVE MAX_PENDING TO 580 PENDING WITHDRAWALS ON SOROBAN	MEDIUM	SOLVED - 05/19/2026
ADMIN-ONLY BLEND ADAPTER SHORTCUTS CAN DESYNCHRONIZE VAULT ACCOUNTING	MEDIUM	SOLVED - 05/19/2026
BLEND ADAPTER ADMIN CAN RETARGET THE VAULT RECEIVER OR POOL WITHOUT TIMELOCK OR EVENT	MEDIUM	SOLVED - 05/19/2026

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
GOVERNANCE PENDING QUEUE IS STORED AS A SINGLE SIZE-CAPPED INSTANCE ENTRY	MEDIUM	SOLVED - 05/19/2026
GOVERNANCE CONFIGURATION AND SKIM COMMANDS BYPASS THE MIGRATION FREEZE	MEDIUM	SOLVED - 05/19/2026
GUARDIAN AND SENTINEL REVOKERS CAN CANCEL UNRELATED GOVERNANCE PROPOSALS	MEDIUM	SOLVED - 05/19/2026
VIRTUAL CONVERSION OFFSETS CAN BE CHANGED AFTER THE VAULT IS CAPITALIZED	MEDIUM	SOLVED - 05/19/2026
TIMELOCKCONFIG CAN BE RAISED TO U64::MAX IMMEDIATELY, BLOCKING FUTURE REDUCTIONS	MEDIUM	SOLVED - 05/19/2026
IMMEDIATE SKIM ACTION WITH DEFAULT SKIM RECIPIENT DRAINS FOREIGN TOKENS IN ONE TRANSACTION	MEDIUM	SOLVED - 05/19/2026
DIRECT UNDERLYING TRANSFERS REMAIN OUTSIDE TOTAL_ASSETS UNTIL MANUAL RESYNC, ALLOWING PRE-RESYNC DEPOSITS TO CAPTURE THE SURPLUS	MEDIUM	SOLVED - 05/19/2026
PERMISSIONLESS INITIALIZE ALLOWS DEPLOYMENT-RACE GOVERNANCE TAKEOVER IF INITIALIZATION IS NOT ATOMIC	LOW	RISK ACCEPTED - 05/19/2026
SHARE-CONVERSION HELPERS TRUNCATE 256-BIT QUOTIENTS TO U128	INFORMATIONAL	SOLVED - 05/19/2026
DESERIALIZE_FEES_SPEC ACCEPTS A 113-BYTE BLOB WITH NO-GROWTH TAG AND SILENTLY DISCARDS 16 TRAILING BYTES	INFORMATIONAL	ACKNOWLEDGED - 05/19/2026

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
RESYNC_IDLE_BALANCE ADVANCES FEE_ANCHOR.TOTAL_ASSETS WITHOUT UPDATING TIMESTAMP_NS, DISTORTING FUTURE FEE MATH	INFORMATIONAL	SOLVED - 05/19/2026
TOTAL_ASSETS_FOR_FEE_ACCRUAL EARLY-RETURNS WHEN ANCHOR_TOTAL_ASSETS == 0, NEUTRALIZING THE CONFIGURED MAX_GROWTH_RATE CAP	INFORMATIONAL	SOLVED - 05/22/2026
MIGRATE() AND HANDLE_EMERGENCY_RESET PRESERVE FEE_ANCHOR, PERPETUATING STALENESS ACROSS RECOVERY PATHS	INFORMATIONAL	SOLVED - 05/22/2026
POLICY AND RESTRICTIONS DECODERS ACCEPT TRAILING BYTES, UNLIKE THE STATE AND COMMAND CODECS	INFORMATIONAL	SOLVED - 05/19/2026
WITHDRAW QUEUE DESERIALIZATION DOES NOT VALIDATE QUEUE INVARIANTS OR SORTED PENDING-WITHDRAWAL IDS	INFORMATIONAL	SOLVED - 05/19/2026
COMPOSE_POLICY_STATE SILENTLY CONSTRUCTS DEFAULT VALUES FOR MISSING STORAGE PARTS	INFORMATIONAL	SOLVED - 05/19/2026
CANCEL_MIGRATION() CLEARS THE MIGRATION FLAG BUT CANNOT ROLL BACK THE ALREADY-SWAPPED WASM	INFORMATIONAL	SOLVED - 05/19/2026
RESTRICTIONS BLOB CAN EXCEED THE SOROBAN ENTRY-SIZE LIMIT WHEN GOVERNANCE CONFIGURES LARGE BLACKLIST OR WHITELIST SETS	INFORMATIONAL	SOLVED - 05/19/2026
PROXY_VIEW RETURNS MAX_WITHDRAW AND MAX_REDEEM FOR ATOMIC OPERATIONS THE DEPLOYED ABI CANNOT PERFORM	INFORMATIONAL	SOLVED - 05/22/2026
PROXY_VIEW MAX_DEPOSIT AND MAX_MINT ONLY CHECK THE SAME-SIDE OVERFLOW	INFORMATIONAL	SOLVED - 05/19/2026

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
PREPARE_ATOMIC_CALL DISCARDS THE EFFECTSUMMARY RETURNED BY THE CONDITIONAL REFRESHFEES KERNEL ACTION	INFORMATIONAL	SOLVED - 05/22/2026
LOAD_VAULT_BOOTSTRAP RUNS MIGRATE_LEGACY_PAUSED ON EVERY CONTRACT CALL	INFORMATIONAL	SOLVED - 05/19/2026
GUARDIAN CONFIGURATION REPLACES THE FULL GUARDIAN SET SILENTLY	INFORMATIONAL	ACKNOWLEDGED - 05/19/2026
SHARE-TOKEN BURN PATHS RELY ON VAULT INVOKER CHECKS INSTEAD OF OWNER AUTHORIZATION	INFORMATIONAL	SOLVED - 05/20/2026
CONSTRUCTORS AND GOVERNANCE CONFIG SETTERS ACCEPT NONSENSICAL ROLE ADDRESSES	INFORMATIONAL	SOLVED - 05/19/2026
GOVERNANCE TIMELOCK LEGACY FALLBACK CAN DROP PER-KIND CUSTOMIZATIONS	INFORMATIONAL	ACKNOWLEDGED - 05/19/2026
AUXILIARY CONTRACT ADMIN ROTATION IS SINGLE-STEP AND EMITS NO EVENT	INFORMATIONAL	SOLVED - 05/22/2026
SETGOVERNANCE ACCEPTS THE VAULT OR ARBITRARY NON-GOVERNANCE CONTRACTS	INFORMATIONAL	SOLVED - 05/19/2026
AUXILIARY CONTRACTS LACK PAUSE OR FREEZE MECHANISMS	INFORMATIONAL	SOLVED - 05/20/2026
AUXILIARY CONTRACTS HAVE NO UPGRADE OR MIGRATE PATH	INFORMATIONAL	SOLVED - 05/22/2026

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
SHARE-TOKEN METADATA CAN BE CHANGED AFTER DEPLOYMENT WITHOUT TIMELOCK OR EVENT	INFORMATIONAL	SOLVED - 05/19/2026
VAULT RESTRICTIONS DO NOT PROPAGATE TO SHARE-TOKEN TRANSFERS	INFORMATIONAL	SOLVED - 05/20/2026
EFFECT-ADDRESS PREFLIGHT CAN SUCCEED FOR MAPPINGS THAT FAIL DURING EXECUTION	INFORMATIONAL	ACKNOWLEDGED - 05/19/2026
ADAPTER CALLS CAN CONSUME THE SHARED TRANSACTION INSTRUCTION BUDGET	INFORMATIONAL	ACKNOWLEDGED - 05/22/2026
STALE BLEND RESERVE DATA CAN REVERT VAULT ALLOCATION AND REFRESH FLOWS	INFORMATIONAL	ACKNOWLEDGED - 05/22/2026
REFRESHMARKETS MATERIALIZES ALL MARKETS BEFORE CALLER AUTHORIZATION	INFORMATIONAL	ACKNOWLEDGED - 05/19/2026
GOVERNANCE COMMAND BODIES ARE DECODED BEFORE AUTHORIZATION AND KIND VALIDATION	INFORMATIONAL	SOLVED - 05/19/2026
BLEND ADAPTER ADMIN CANNOT BE ROTATED AFTER DEPLOYMENT	INFORMATIONAL	SOLVED - 05/22/2026
GOVERNANCE ADMIN CANNOT BE ROTATED AFTER DEPLOYMENT	INFORMATIONAL	SOLVED - 05/25/2026
SETMEMBERSHIP GOVERNANCE DECISIONS USE INCOMPLETE CURRENT-STATE CLASSIFICATION	INFORMATIONAL	SOLVED - 05/25/2026

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
BLEND ADAPTER CONSTRUCTOR ACCEPTS UNVALIDATED VAULT POOL ASSET AND ADMIN ADDRESSES	INFORMATIONAL	SOLVED - 05/19/2026
GOVERNANCE ABDICATION SYMBOLS DISABLE GROUPED SUBMIT METHODS RATHER THAN ONE ACTION	INFORMATIONAL	ACKNOWLEDGED - 05/19/2026
SETMEMBERSHIP POLICY DEFAULTS MISSING MARKET_ID TO MARKET ZERO	INFORMATIONAL	SOLVED - 05/19/2026
GOVERNANCE CONSTRUCTOR DOES NOT REJECT SELF-REFERENTIAL OR COLLIDING ROLE ADDRESSES	INFORMATIONAL	SOLVED - 05/19/2026
NEWLY ATTACHED GOVERNANCE STARTS WITH DEFAULT LOCAL CACHE RATHER THAN VAULT STATE	INFORMATIONAL	SOLVED - 05/25/2026
ACCEPT_KIND IS AMBIGUOUS WHEN MULTIPLE PROPOSALS SHARE THE SAME ACTION KIND	INFORMATIONAL	SOLVED - 05/19/2026
VAULT DISPATCH HELPERS ACCEPT EMPTY OR DUPLICATE INPUT LISTS	INFORMATIONAL	SOLVED - 05/19/2026
STORAGE DECODERS PRE-ALLOCATE VECTORS FROM UNTRUSTED LENGTH PREFIXES	INFORMATIONAL	SOLVED - 05/22/2026
TEST SUITE USES PERMISSIVE AUTHORIZATION PATHS THAT DIVERGE FROM PRODUCTION RBAC	INFORMATIONAL	SOLVED - 05/19/2026
ALLOCATION LIFECYCLE TEST HELPERS BYPASS PRODUCTION KERNEL ACTION HANDLING	INFORMATIONAL	SOLVED - 05/19/2026

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
VAULT USES STELLARASSETCLIENT FOR CUSTOM SHARE-TOKEN INTERACTIONS	INFORMATIONAL	ACKNOWLEDGED - 05/22/2026
INTEGRATION TESTS SEED STATE OUTSIDE THE PRODUCTION ONBOARDING FLOW	INFORMATIONAL	SOLVED - 05/19/2026
TEST ALLOCATION HELPER DROPS LEASED TARGETS UNLIKE PRODUCTION ALLOCATION	INFORMATIONAL	ACKNOWLEDGED - 05/19/2026
PROPERTY TESTS LABELED EXTERNAL GROWTH DO NOT MODEL REAL ADAPTER GROWTH	INFORMATIONAL	SOLVED - 05/19/2026
PREVIEW AND FEE TESTS DUPLICATE PRODUCTION INTERNALS WITH DIFFERENT SEMANTICS	INFORMATIONAL	SOLVED - 05/19/2026
SOROBAN KERNEL PARITY TESTS SKIP SOROBAN-SPECIFIC PATHS AND HIGH-VALUE RANGES	INFORMATIONAL	SOLVED - 05/19/2026
GOVERNANCE MOCKVAULT IGNORES INVALID OR INCOMPLETE COMMAND PAYLOADS	INFORMATIONAL	SOLVED - 05/19/2026
PROXY_VIEW CASTS U128 VALUES TO I128 WITHOUT BOUNDS CHECKS	INFORMATIONAL	SOLVED - 05/19/2026
GOVERNANCE IS GRANTED FULL CURATOR ROLE INSTEAD OF THE NARROWER POLICY-ADMIN NEED	INFORMATIONAL	SOLVED - 05/25/2026
FEE RECIPIENT CONFIGURATION IS POSITIONAL AND CAN BE MISWIRED BY REORDERING	INFORMATIONAL	SOLVED - 05/22/2026

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
ADDRESS TEXT CONVERSION CAN PANIC OR SILENTLY FALL BACK ON MALFORMED INPUTS	INFORMATIONAL	ACKNOWLEDGED - 05/19/2026
FEE PREVIEW SATURATES SUPPLY VALUES WITHOUT SURFACING AN OVERFLOW SIGNAL	INFORMATIONAL	SOLVED - 05/19/2026
TEST-ONLY SOROBANAUTH CREATES MISLEADING PRODUCTION AUTHORIZATION COVERAGE	INFORMATIONAL	SOLVED - 05/19/2026
EVENT PAYLOADS OMIT STRUCTURED ADDRESSES AND TYPED TOPICS ACROSS VAULT COMPONENTS	INFORMATIONAL	ACKNOWLEDGED - 05/22/2026
ASSET AND SHARE-TOKEN ADMIN ASSUMPTIONS ARE ENFORCED ONLY BY DEPLOYMENT DISCIPLINE	INFORMATIONAL	SOLVED - 05/19/2026
PROXY_VIEW FEE SNAPSHOT OMITS THE CONFIGURED FEE GROWTH-RATE CAP	INFORMATIONAL	SOLVED - 05/19/2026
GOVERNANCE ABDICATE ACCEPTS UNKNOWN SYMBOLS AS SILENT NO-OPS	INFORMATIONAL	ACKNOWLEDGED - 05/19/2026
OTHER EFFECT APPROVALS ARE UNREACHABLE FROM THE DEPLOYED VAULT COMMAND SURFACE	INFORMATIONAL	ACKNOWLEDGED - 05/19/2026
REVOKE_KIND CANCELS ALL PROPOSALS OF A KIND WHILE ACCEPT_KIND ACCEPTS ONLY ONE	INFORMATIONAL	SOLVED - 05/19/2026
VAULT COMMAND WIRE CODEC LACKS VERSIONING AND COLLAPSES DECODER ERROR VARIANTS	INFORMATIONAL	ACKNOWLEDGED - 05/22/2026

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
MULTIPLE WIRE ENCODINGS CAN REPRESENT THE SAME GOVERNANCE POLICY ACTION	INFORMATIONAL	SOLVED - 05/22/2026
GOVERNANCE DIRECTION CLASSIFIERS INCONSISTENTLY DETECT NO-CHANGE PROPOSALS	INFORMATIONAL	SOLVED - 05/25/2026
LOAD_TIMELOCKS CAN MUTATE STORAGE DURING VIEW-STYLE PATHS	INFORMATIONAL	SOLVED - 05/19/2026
SHARE-TOKEN BURN AND BURN_FROM EMIT INDISTINGUISHABLE BURN EVENTS	INFORMATIONAL	SOLVED - 05/19/2026
SHARE-TOKEN CONTRACT-ADDRESS VALIDATION RELIES ON STRKEY PREFIX INSPECTION	INFORMATIONAL	SOLVED - 05/19/2026
IN-FLIGHT GOVERNANCE PROPOSALS RETAIN SUBMIT-TIME TIMELOCK AFTER LATER TIMELOCK RAISES	INFORMATIONAL	ACKNOWLEDGED - 05/19/2026
SHARE-TOKEN ARCHIVAL RECOVERY PROCEDURE IS UNDEFINED	INFORMATIONAL	SOLVED - 05/19/2026
SHARE-TOKEN TESTS OMIT ADMIN ROTATION AND ALLOWANCE SECURITY COVERAGE	INFORMATIONAL	SOLVED - 05/19/2026
SHARE-TOKEN READ-ONLY METHODS DO NOT REFRESH INSTANCE TTL	INFORMATIONAL	SOLVED - 05/19/2026
SHARE-TOKEN CONSTRUCTOR EMITS NO DEPLOYMENT CONFIGURATION EVENT	INFORMATIONAL	SOLVED - 05/22/2026

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
SHARE-TOKEN EXTEND_TTL DOES NOT REFRESH HOLDER BALANCES OR ALLOWANCES	INFORMATIONAL	SOLVED - 05/19/2026
ADDRESSMAP CONSTRUCTOR ACCEPTS AN UNUSED ENVIRONMENT PARAMETER	INFORMATIONAL	SOLVED - 05/22/2026
IMMEDIATE GOVERNANCE SUBMISSIONS WRITE THE PENDING QUEUE TWICE	INFORMATIONAL	SOLVED - 05/19/2026

20. FINDINGS & TECH DETAILS

20.1 POSITIONAL ADAPTER MAPPING CAUSES SILENT MARKET REMAP ON SUPPLY QUEUE REORDER

// HIGH

Description

The Soroban vault binds market identifiers to adapter contracts **positionally**: `adapter_for_market(market_id)` resolves the adapter by enumerating the current supply queue and returning `adapters[index]` for the matching market. The vault's allowed-adapters configuration is only validated against the queue length, not against any per-market binding.

As a result, a routine governance action that reorders the supply queue without simultaneously reordering the adapter list will silently rebind every market to the wrong adapter. Because allocations and refreshes target market identifiers but resolve to adapters by queue index, the next allocator-driven action will move funds to a different adapter than the one previously associated with that market.

This is dangerous even under fully honest governance: a queue priority update is a routine operational change, but it permanently misroutes future allocations until a corresponding adapter list update is submitted.

Code Location

Copy Code

```
// contract/vault/soroban/src/contract/helpers.rs (adapter_for_market)
pub(crate) fn adapter_for_market(env: &Env, market_id: u32) -> Result<SdkAddress, ContractError> {
    let queue = current_supply_queue(env)?;
    let adapters: Vec<SdkAddress> = env.storage().instance().get(&VaultDataKey::AllowedAdapters).unwrap_
    for (index, target_id) in queue.iter().enumerate() {
        if target_id == market_id {
            return adapters.get(index as u32).ok_or(ContractError::MissingConfig);
        }
    }
    Err(ContractError::MissingConfig)
}
```

Copy Code

```
// contract/vault/soroban/src/contract/entrypoints.rs (apply_allowed_adapters_config)
fn apply_allowed_adapters_config(env: &Env, adapters: Vec<SdkAddress>) -> Result<(), ContractError> {
    let queue_len = current_supply_queue_len(env)?;
    if adapters.len() as u32 != queue_len {
        return Err(ContractError::InvalidInput);
    }
    env.storage().instance().set(&VaultDataKey::AllowedAdapters, &adapters);
    Ok(())
}
```

Impact

An honest governance queue reorder, intended only as a priority change, silently remaps every active market to a different adapter. The next allocator-class `VaultCommand::Allocate` for any affected market will

route the allocator's funds to the wrong adapter and therefore the wrong external pool.
Kernel accounting still attributes the action to the original `market_id`, producing state drift between the recorded principal and the real destination of the assets. Recovery requires an additional governance call to re-align the adapter list, during which any allocations are misrouted.

Proof of Concept

Preconditions

- The vault is deployed with at least two markets registered (e.g., markets `0` and `1`).
- Governance has configured the supply queue as `[0, 1]` and the allowed-adapters list as `[A, B]` — i.e., market 0 → adapter A, market 1 → adapter B.
- An allocator is registered and the vault holds idle liquidity.

Attack / Defect Sequence

1. **Verify the baseline binding.** The allocator submits `VaultCommand::Allocate { market: 0, amount: 1_000, supply: true }`. The vault calls `adapter_for_market(0)`, finds market `0` at queue index `0`, and returns `adapters[0] = adapter_A`. The 1,000 underlying flow into adapter A's external pool, as expected.
2. **Governance issues a routine priority change.** Governance submits `SetGovernancePolicy(SUPPLY_QUEUE, target_ids = [1, 0])` to swap market priorities. Operators view this as a low-risk priority tweak and intentionally do not touch `AllowedAdapters`; the runtime accepts the change because `adapters.len() == queue_len` still holds.
3. **The allocator issues a follow-up allocation with unchanged intent.** The allocator submits `VaultCommand::Allocate { market: 0, amount: 2_000, supply: true }` with the same expectation: funds should land in market 0's adapter (still `adapter_A` in the operator's mental model).
4. **Funds are silently routed to the wrong adapter.** Inside `adapter_for_market(0)`, the loop finds market `0` at queue index `1` (because the queue is now `[1, 0]`) and returns `adapters[1] = adapter_B`. The runtime transfers 2,000 underlying to adapter B and calls `adapter_B.supply(...)`, depositing the funds into a completely different external pool.
5. **Accounting drifts.** The kernel records the 2,000 as the principal of market `0`, but the actual capital is now sitting inside adapter B's pool (semantically market 1). Subsequent `refresh_markets(0)` calls query adapter B for "market 0's" total, producing accounting that no longer reflects reality.

Observable Result

After the reorder, an `Allocate { market: 0, ... }` call leaves `adapter_a.last_in` unchanged at its baseline value and increments `adapter_b.last_in` by the supplied amount, even though the allocator never targeted market 1. Recovery requires governance to submit a new adapter list aligned with the new queue order; during that delay any further allocations remain misrouted.

Reproducing Locally

The runnable PoC is in `contract/vault/soroban/tests/security_pocs.rs` (function `poc_queue_reorder_misroutes_allocator_supply`). Run with:

```
cargo test --test security_pocs poc_queue_reorder_misroutes_allocator_supply -- --nocapture
```

Stdout:

```
after reorder [1,0], allocate(market=0, 2_000) -> adapter_a.last_in = 1000, adapter_b.last_in = 2000
```

Mock adapter that records the last `supply()` amount so the test can inspect which adapter received funds:

```
#[contract]
pub struct MockAdapter;

#[contractimpl]
impl MockAdapter {
    pub fn supply(env: Env, _vault: Address, _asset: Address, amount: i128) {
        env.storage()
            .instance()
            .set(&Symbol::new(&env, "last_in"), &amount);
    }

    pub fn progress_withdrawal(_env: Env, _vault: Address, _asset: Address, amount: i128) -> i128 {
        amount
    }

    pub fn total_assets(env: Env, _asset: Address) -> i128 {
        env.storage().instance().get(&Symbol::new(&env, "last_in")).unwrap_or(0i128)
    }

    pub fn last_supply(env: Env) -> i128 {
        env.storage().instance().get(&Symbol::new(&env, "last_in")).unwrap_or(0i128)
    }
}
```

The test wires markets and adapters, performs the baseline allocation, the routine reorder, and the second allocation, then inspects each mock adapter's last input:

```
// 1) Wire markets [0, 1] and adapters [A, B]
let adapter_a = env.register(MockAdapter, ());
let adapter_b = env.register(MockAdapter, ());
env.as_contract(&vault, || {
    let mut policy = SorobanStorage::new(&env).load_policy_state().unwrap().unwrap_or_default();
    policy.set_market_config(0, MarketConfig::new(true, i128::MAX as u128, None)).unwrap();
    policy.set_market_config(1, MarketConfig::new(true, i128::MAX as u128, None)).unwrap();
    SorobanStorage::new(&env).save_policy_state(&policy).unwrap();

    let mut ids: StdVec<u32> = StdVec::new();
    ids.push(0); ids.push(1);
    execute_command(&env, &VaultCommand::SetGovernancePolicy {
        caller: address_text(&governance),
        kind: GOVERNANCE_POLICY_KIND_SUPPLY_QUEUE,
        target_ids: Some(ids), mode: None, accounts: None, market_id: None,
        cap_group_id: None, value: None, value_b: None, value_c: None,
    }).unwrap();

    let mut adapters: StdVec<StdString> = StdVec::new();
    adapters.push(address_text(&adapter_a));
    adapters.push(address_text(&adapter_b));
    execute_command(&env, &VaultCommand::SetGovernanceConfig {
        caller: address_text(&governance),
        kind: GOVERNANCE_CONFIG_KIND_ALLOWED_ADAPTERS,
        primary: None, many: Some(adapters), value_a: None, value_b: None,
    }).unwrap();
```

```
});

// 2) Baseline: allocate(market=0, 1_000) -> adapter_a
env.as_contract(&vault, || {
    execute_command(&env, &VaultCommand::Allocate {
        caller: address_text(&allocator), market: 0, amount: 1_000, supply: true,
    }).unwrap();
});
let a_after_first = env.as_contract(&adapter_a, || MockAdapter::last_supply(env.clone()));
let b_after_first = env.as_contract(&adapter_b, || MockAdapter::last_supply(env.clone()));
assert_eq!((a_after_first, b_after_first), (1_000, 0));

// 3) Governance reorders queue [0,1] -> [1,0] without touching adapters
env.as_contract(&vault, || {
    let mut ids: StdVec<u32> = StdVec::new();
    ids.push(1); ids.push(0);
    execute_command(&env, &VaultCommand::SetGovernancePolicy {
        caller: address_text(&governance),
        kind: GOVERNANCE_POLICY_KIND_SUPPLY_QUEUE,
        target_ids: Some(ids), mode: None, accounts: None, market_id: None,
        cap_group_id: None, value: None, value_b: None, value_c: None,
    }).unwrap();
});

// 4) Reissue the same allocate(market=0, 2_000)
env.as_contract(&vault, || {
    execute_command(&env, &VaultCommand::Allocate {
        caller: address_text(&allocator), market: 0, amount: 2_000, supply: true,
    }).unwrap();
});

// 5) adapter_a is unchanged; adapter_b received the funds destined for market 0
let a_after_reorder = env.as_contract(&adapter_a, || MockAdapter::last_supply(env.clone()));
let b_after_reorder = env.as_contract(&adapter_b, || MockAdapter::last_supply(env.clone()));
assert_eq!(a_after_first, a_after_reorder);
assert_eq!(b_after_reorder, 2_000);
```

BVSS

AO:A/AC:L/AX:M/R:N/S:C/C:N/A:N/I:N/D:C/Y:N (8.4)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Adapters should be persisted as a mapping keyed by `market_id` rather than as a queue-position-indexed vector, so that supply queue reorders cannot silently rebind adapters.
- Supply queue updates should be rejected unless every queued `market_id` has an explicit adapter binding present in storage.
- A regression test should be added that reorders the supply queue while leaving the adapter list unchanged and asserts that the runtime rejects the configuration rather than silently rebinding markets.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix replaces positional supply-queue adapter lookup with immutable market-keyed adapter bindings. Supply-queue reorders no longer remap markets to different adapters, new market bindings are created only from typed governance proposal adapter accounts, and existing bindings cannot be rebound.

Client Note:

Client response for #FIND-001: Remediated. Fixed in commit d2ea12816ceadb76f0be3c3f9dafaab53bccb882: fix(soroban): bind adapters by market identity (#FIND-001 Nexus ced39f48-e0fe-4218-80c3-d631ff6c565e) The fix replaces positional supply-queue adapter lookup with immutable market-keyed adapter bindings. Supply-queue reorders no longer remap markets to different adapters, new market bindings are created only from typed governance proposal adapter accounts, and existing bindings cannot be rebound. Regression coverage includes adapter lookup, reorder/permutation preservation, governance supply-queue binding/rebinding rejection, and Blend e2e allocation/withdrawal through the production binding flow.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/d2ea12816ceadb76f0be3c3f9dafaab53bccb882

20.2 LOW-LIQUIDITY WITHDRAWAL EXECUTION CAN TRAP THE VAULT IN WITHDRAWING WITH NO NORMAL ABI RECOVERY PATH

// HIGH

Description

The Soroban vault enters `OpState::Withdrawing` as soon as a queued withdrawal request is mature, regardless of whether `state.idle_assets` is sufficient to make any progress. Once the vault is in `Withdrawing` with `idle_assets < MIN_WITHDRAWAL_ASSETS` (1,000), `plan_idle_payout` returns `None`, no payout is produced, and the persisted state remains in `Withdrawing`.
The shipped Soroban ABI provides no normal recovery surface from this stuck state:

- `start_allocation` and the curator `begin_allocation_internal` / `begin_allocation_withdraw_internal` helpers all require `OpState::Idle`.
- `resync_idle_balance_impl` explicitly rejects unless `state.op_state.is_idle()`.
- `VaultCommand` defines no `AbortWithdrawing`, `AbortAllocating`, or `EmergencyReset` variant.
- The kernel's `action-recovery` feature is not enabled in the vault's `Cargo.toml`.

The deadlock can be reached by a single allocator-class `ExecuteWithdraw` while idle is below the minimum, or automatically through the `finish_allocating` auto-chain that transitions to `Withdrawing` as soon as a matured queued withdrawal exists. The public command surface also hides the no-progress condition by discarding the `EffectSummary` and returning `VaultCommandResult::Unit`; off-chain automation observes a successful command even when zero assets moved.

Code Location

Copy Code

```
// contract/vault/kernel/src/actions/mod.rs (plan_idle_payout)
let available_assets = state.idle_assets;
if available_assets < request_expected
    && available_assets < crate::state::queue::MIN_WITHDRAWAL_ASSETS
{
    return Ok(None);
}
```

Copy Code

```
// contract/vault/soroban/src/contract/curator_vault.rs (complete_withdrawal_from_idle)
let Some(idle_payout) = transition_to_runtime(plan_idle_payout(self.state()?))? else {
    return Ok(EffectSummary::new());
};
```

Copy Code

```
// contract/vault/soroban/src/contract/entrypoints.rs (resync_idle_balance_impl)
if !state.op_state.is_idle() {
    return Err(RuntimeError::invalid_state(""));
}
```

Impact

A single routine allocator-class action can place the vault into a persistent stuck state from which:

- Queued shares remain escrowed indefinitely, until the contract is upgraded or rescued out-of-band.
- Direct rescue transfers of the asset token to the vault do not help, because `ResyncIdleBalance` is blocked outside `Idle`.
- No further market liquidity can be pulled into the vault because `begin_allocation` and `refresh_markets` also require `Idle`.

Even though this is a liveness / redemption-availability failure rather than a direct theft primitive, it can freeze user redemptions for an unbounded period and requires governance-driven contract upgrade or storage-level rescue to unblock.

Proof of Concept

Preconditions

- A withdrawal request is queued and has matured past the cooldown.
- The vault has a small idle balance below `MIN_WITHDRAWAL_ASSETS = 1_000` (e.g., `idle_assets = 500`) but a non-zero `external_assets` balance held in markets.
- An allocator-class caller is available; the vault is in `OpState::Idle` immediately before the trigger.

Attack / Defect Sequence

1. **The withdrawal cooldown elapses.** The user's queued withdrawal of, e.g., 2,000 shares becomes mature: `now_ns >= request.requested_at_ns + DEFAULT_COOLDOWN_NS`. The vault is still `Idle`.
2. **An allocator (or any allocator-class role holder) calls `VaultCommand::ExecuteWithdraw`.** The kernel's `handle_execute_withdraw` sees that the head request is mature, calls `start_withdrawal`, and transitions `op_state` from `Idle` to `Withdrawing`. There is no minimum-idle precondition on this transition.
3. **The runtime tries to plan the payout.** `complete_withdrawal_from_idle` calls `plan_idle_payout(state)`. Because `idle_assets = 500 < request.expected_assets` AND `idle_assets < MIN_WITHDRAWAL_ASSETS = 1_000`, `plan_idle_payout` returns `Ok(None)`.
4. **No effects are produced; state is persisted in `Withdrawing`.** `complete_withdrawal_from_idle` hits the `let Some(idle_payout) = ... else { return Ok(EffectSummary::new()); }` branch and returns an empty summary. `ExecuteWithdraw` succeeds at the ABI layer (`VaultCommandResult::Unit`) and the persisted state remains `OpState::Withdrawing(...)`.
5. **Every recovery path is rejected.** The keeper or an operator tries to recover:

- `VaultCommand::ResyncIdleBalance` → rejected, requires `op_state.is_idle()`.
- `VaultCommand::Allocate { supply: false }` (pull liquidity from market) → rejected, `begin_allocation_withdraw_internal` requires `Idle`.
- `VaultCommand::RefreshMarkets` → rejected for the same reason.
- No `AbortWithdrawing` / `EmergencyReset` variant exists in `VaultCommand`.

6. **The vault is permanently stuck.** The user's shares remain escrowed; no redemption can complete. Resolution requires an out-of-band rescue (contract upgrade, storage-level intervention).

Observable Result

After step 4, `SorobanStorage::load_state()` returns a state where `matches!(op_state, OpState::Withdrawing(_))` is `true` and `idle_assets < 1_000`. After step 5, every recovery `VaultCommand` returns `Err(...)`.

Reproducing Locally

The runnable PoC is in `contract/vault/soroban/tests/security_pocs.rs` (function `poc_stuck_withdrawing_no_recovery`). Run with:

Copy Code

```
cargo test --test security_pocs poc_stuck_withdrawing_no_recovery -- --nocapture
```

Stdout:

Copy Code

```
vault stuck in Withdrawing with idle below MIN_WITHDRAWAL_ASSETS; resync rejected
```

The test seeds a matured pending withdrawal in the persisted state, advances past the cooldown, executes the withdrawal, and confirms both the stuck transition and the lack of recovery:

Copy Code

```
// 1) Seed: idle_assets = 500 (< MIN_WITHDRAWAL_ASSETS = 1_000),
//    external_assets = 10_000, queued withdrawal of 2_000 shares
env.as_contract(&vault, || {
    let mut queue = WithdrawQueue::new();
    queue.enqueue(
        owner_kernel,
        receiver_kernel,
        2_000, // escrow shares
        2_000, // expected assets
        TimestampNs(0),
        16,
    ).unwrap();

    let state = VaultState {
        total_assets: 10_500,
        total_shares: 10_500,
        idle_assets: 500,
        external_assets: 10_000,
        fee_anchor: FeeAccrualAnchor::new(10_500, TimestampNs(0)),
        op_state: OpState::Idle,
        withdraw_queue: queue,
        next_op_id: 1,
        ..Default::default()
    };
    SorobanStorage::new(&env).save_state(&state).unwrap();
});

// 2) Advance past cooldown
advance_secs(&env, DEFAULT_COOLDOWN_NS / 1_000_000_000 + 1);

// 3) Trigger ExecuteWithdraw
env.as_contract(&vault, || {
    execute_command(&env, &VaultCommand::ExecuteWithdraw {
        caller: address_text(&admin),
    }).unwrap();

    // 4) Vault is now stuck in Withdrawing with idle below the minimum
    let state = SorobanStorage::new(&env).load_state().unwrap().unwrap();
    assert!(matches!(state.op_state, OpState::Withdrawing(_)));
    assert!(state.idle_assets < 1_000);
});
```

```
// 5) Every recovery path is rejected
env.as_contract(&vault, || {
    let resync = execute_command(&env, &VaultCommand::ResyncIdleBalance);
    assert!(resync.is_err(), "ResyncIdleBalance must reject non-Idle state");
});
```

BVSS

AO:A/AC:L/AX:M/R:N/S:C/C:N/A:N/I:N/D:C/Y:N (8.4)

Recommendation

To mitigate this issue, the following improvements are recommended:

- The vault should not transition into `OpState::Withdrawing` unless the accounted idle balance is sufficient to satisfy either the full request or at least `MIN_WITHDRAWAL_ASSETS` for a valid partial settlement. This guard should be enforced both in the direct `ExecuteWithdraw` path and in the `finish_allocating` auto-chain path.
- Alternatively, a real recovery surface should be exposed in the Soroban build, such as `AbortWithdrawing`, `AbortAllocating`, or an allocator-only continuation path that can pull additional market liquidity while the vault is already `Withdrawing`.
- `ResyncIdleBalance` should be permitted while in `Withdrawing`, or a dedicated "sync idle for active withdrawal" action should be introduced so that direct rescue transfers can unblock a stuck payout.
- A structured execution status should be returned from `ExecuteWithdraw` in place of `VaultCommandResult::Unit` so that off-chain automation can detect "entered Withdrawing but paid nothing" and surface an alarm.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix prevents normal queued withdrawal execution from entering a new low-liquidity Withdrawing state when the available payout would be below the minimum withdrawal amount. It preserves valid partial payouts at or above the threshold, exposes AbortWithdrawing as a recovery backstop for stale/manual stuck states, and returns structured execute-withdraw status for keeper observability.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/66e68f8fc2b0ee3f7a6387070634b1406ab48a74

20.3 WITHDRAW ALLOCATIONS TRUST ADAPTER-REPORTED REALIZED ASSETS WITHOUT VERIFYING TOKEN BALANCE DELTA

// HIGH

Description

The Soroban vault's withdraw-allocation path credits adapter-reported withdrawal proceeds into internal accounting without verifying that the asset-token balance of the vault actually increased by the same amount. When `supply == false`, the runtime calls the configured adapter through `invoke_progress_withdrawal`, converts the adapter-controlled return value to `u128`, and passes it into `complete_withdraw_allocation` as the realized amount.

No balance snapshot is taken around the adapter call. As a result, the adapter's return value is treated as a source of truth even though the asset token transfer is an independent side effect. A faulty, stale, or compromised adapter can report that assets were recovered while transferring fewer tokens, or no tokens, back to the vault.

This is the same class of accounting failure the existing report template treats as high impact: a single normal allocator-class action can place persisted vault state out of sync with on-chain token balances, and off-chain automation receives a successful command result rather than a clear accounting-divergence signal.

Code Location

Copy Code

```
// contract/vault/soroban/src/contract/entrypoints.rs (allocate_impl, withdraw branch)
let realized_amount = invoke_progress_withdrawal(env, &adapter, &asset_token, amount);
let realized_amount_u128 = to_u128(realized_amount)?;

let mut call = |vault: &mut ContractVault<'_>| -> Result<(), RuntimeError> {
    let op_id = vault.begin_allocation_withdraw_internal(caller_kernel, market, now_ns)?;
    new_external = vault.complete_withdraw_allocation(
        caller_kernel,
        market,
        realized_amount_u128,
        op_id,
        now_ns,
    )?;
    Ok(())
};
```

Copy Code

```
// contract/vault/soroban/src/market/mod.rs (adapter-returned value)
// invoke_progress_withdrawal returns the adapter-controlled i128 amount;
// the vault does not derive realized assets from balance_after - balance_before.
```

Impact

A malicious or malfunctioning adapter can inflate vault accounting without transferring the corresponding underlying tokens. This produces a persistent divergence between `state.idle_assets` / market accounting and the vault's real token balance.

Once this divergence exists:

- Withdrawal planning can assume liquidity exists when it does not.
- Earlier withdrawals can consume real assets while later users are left with accounting claims against missing liquidity.
- Share price and preview outputs can be based on overstated assets.
- Recovery requires reconciliation or governance intervention after the accounting error has already been persisted.

Proof of Concept

Preconditions

- The vault has a configured market adapter.
- An allocator can trigger the withdraw-allocation path for that market.
- The adapter is faulty, compromised, or replaced with a mock that can return an arbitrary realized amount.

Attack / Defect Sequence

1. **The attacker controls the adapter response.** The configured adapter implements `progress_withdrawal` so that it returns the requested amount while transferring no asset tokens to the vault.
2. **An allocator triggers a withdraw allocation.** The vault calls `invoke_progress_withdrawal` and receives the adapter-controlled positive return value.
3. **The vault credits the reported value.** The returned amount is converted to `u128` and passed into `complete_withdraw_allocation`.
4. **No balance-delta verification occurs.** Because the vault never checks `asset.balance(vault)` before and after the adapter call, accounting increases even though no tokens arrived.

Observable Result

After the call, the vault's token balance delta is `0`, while internal idle or realized-asset accounting has increased by the adapter-reported amount. This proves accounting can be inflated independently from actual assets held by the vault.

Reproducing Locally

The regression test can be added as a focused mock-adapter test. Run it with the runtime test suite once added:

```
cargo test -p templar-soroban-runtime --features testutils --test security_pocs poc_withdraw_allocation_
```

PoC body:

```
#[contract]
pub struct FalseWithdrawAdapter;

#[contractimpl]
impl FalseWithdrawAdapter {
    pub fn progress_withdrawal(
```

```
        env: Env,
        _vault: Address,
        _asset: Address,
        amount: i128,
    ) -> i128 {
        env.storage()
            .instance()
            .set(&Symbol::new(&env, "reported"), &amount);
        // Malicious behavior: report success without transferring tokens.
        amount
    }

    pub fn total_assets(_env: Env, _asset: Address) -> i128 {
        0
    }
}

#[test]
fn poc_withdraw_allocation_credits_reported_amount_without_balance_delta() {
    let env = Env::default();
    env.mock_all_auths();

    let vault = env.register(SorobanVaultContract, ());
    let adapter = env.register(FalseWithdrawAdapter, ());
    let asset = env
        .register_stellar_asset_contract_v2(Address::generate(&env))
        .address();

    seed_withdraw_allocation_state(&env, &vault, &adapter, &asset, 10_000);

    let token = soroban_sdk::token::Client::new(&env, &asset);
    let balance_before = token.balance(&vault);
    let idle_before = env.as_contract(&vault, || {
        SorobanStorage::new(&env).load_state().unwrap().unwrap().idle_assets
    });

    env.as_contract(&vault, || {
        execute_withdraw_allocation_step(&env, 0, 10_000).unwrap();
    });

    let balance_after = token.balance(&vault);
    let idle_after = env.as_contract(&vault, || {
        SorobanStorage::new(&env).load_state().unwrap().unwrap().idle_assets
    });

    assert_eq!(balance_after - balance_before, 0);
    assert_eq!(idle_after - idle_before, 10_000);
}
```

BVSS

AO:A/AC:L/AX:M/R:N/S:C/C:N/A:N/I:N/D:C/Y:N (8.4)

Recommendation

To mitigate this issue, the following improvements are recommended:

- The vault should snapshot the asset-token balance before and after every adapter withdrawal call.
- The credited realized amount should be derived from the actual balance delta, or the transaction should revert when the adapter-reported amount and balance delta diverge beyond a small configured tolerance.
- A regression test should fuzz adapter return values independently from token transfers and assert that accounting never credits unavailable assets.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix measures the vault asset token balance before and after adapter withdrawal execution and rejects any adapter-reported realized amount that is not backed by a matching balance delta. Kernel withdrawal accounting is only advanced after the external token movement is verified, so a malicious or faulty adapter cannot mint accounting value through an inflated return value.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/c2b007fb10dcbde35da2a85543649bab23c3df43

20.4 ADAPTER-REPORTED EXTERNAL ASSETS CAN BE INFLATED CUMULATIVELY OR DEFLATED TO ZERO

// HIGH

Description

The vault accepts adapter-reported external-asset totals through market refresh and sync logic with only a local per-step growth constraint. The current guard rejects a single update that would more than double the reference total, but it does not track cumulative growth across a time window and does not symmetrically constrain downward moves.

A compromised adapter can therefore report values just below the per-step threshold repeatedly. Every individual report is accepted, but the sequence compounds external-asset accounting over multiple transactions. The same path can also report a position collapse to zero in a single step because the guard is written around upward growth.

This issue is more severe than a simple stale-price observation because the synced value is written directly into `state.external_assets` and `state.total_assets`, which are used by share conversion, withdrawals, and previews.

Code Location

Copy Code

```
// contract/vault/kernel/src/actions/mod.rs (plan_external_asset_sync)
let new_total_assets = state
    .idle_assets
    .checked_add(new_external_assets)
    .ok_or_else(|| KernelError::from(InvalidStateCode::SyncExternalOverflowIdlePlusExternal))?;

let reference_total = state
    .total_assets
    .saturating_add(sync_external_in_flight_assets(&state.op_state));
if reference_total > 0 && new_total_assets > reference_total.saturating_mul(2) {
    return Err(KernelError::from(
        InvalidStateCode::SyncExternalWouldMoreThanDoubleTotalAssets,
    ));
}
```

Copy Code

```
// contract/vault/kernel/src/actions/mod.rs (handle_sync_external_assets)
let sync_plan = plan_external_asset_sync(&state, new_external_assets)?;

state.external_assets = sync_plan.new_external_assets;
state.total_assets = sync_plan.new_total_assets;
```

Impact

A malicious or compromised adapter can repeatedly inflate the vault's reported external assets without transferring or earning matching underlying value. The hardening PoC demonstrated a 255x accounting inflation after eight accepted syncs.

Once inflated, a later depositor receives materially fewer shares for the same asset amount because the conversion formula sees a much higher `total_assets`. Conversely, reporting zero can erase the external position in one step, affecting withdrawals and previews that depend on the same totals.

Proof of Concept

Preconditions

- A market adapter is configured for the vault.
- An allocator/keeper can trigger the market refresh or external-asset sync path.
- The adapter can choose the reported external-asset total, either due to compromise, stale accounting, or implementation error.

Attack / Defect Sequence

1. **Start from a small valid position.** The PoC starts with `idle_assets = 1_000` and no external assets.
2. **Report just below the accepted cap.** On each sync, the adapter reports a new external value such that `new_total_assets` remains below `2 * reference_total`.
3. **Repeat the accepted update.** Because the guard is per-step, each transaction succeeds and becomes the new baseline for the next one.
4. **Deposit after the accounting inflation.** A new depositor receives far fewer shares because the vault believes total assets grew.
5. **Alternative deflation path.** The adapter reports `new_external_assets = 0`, which the growth cap does not reject.

Observable Result

The test prints a step-by-step growth trace and then confirms the exploit markers:

```
[FINDING C-028 multi_step_adapter_inflation steps=8 final_external=254745 inflation_factor=255]
[FINDING C-028 deflation_unbounded pre_total=11000 post_total=1000]
[FINDING C-028 dilution shares_for_100_assets pre=100 post=1 ratio=0.0100]
```

Reproducing Locally

The runnable PoC is in `contract/vault/soroban/tests/adv_adapter_compromise.rs`. Run with:

```
cargo test -p templar-soroban-runtime --features testutils --test adv_adapter_compromise -- --nocapture
```

Relevant PoC body:

```
fn drive_sync_external_assets(
    state: VaultState,
    new_external: u128,
    op_id: u64,
) -> Result<VaultState, templar_vault_kernel::error::KernelError> {
    let config = kernel_view_config();
    let result = apply_action(
        state,
        &config,
        None,
        &KernelAddress([0x09u8; 32]),
        KernelAction::sync_external_assets(new_external, op_id, TimestampNs(100)),
    )?;
    Ok(result.state)
}
```

```rust
#[test]
fn adv_d8_adapter_compromise_multi_step_inflation_succeeds_in_log_n_steps() {
    let mut state = seeded_state_in_allocating(1_000, 0);
    let max_steps = 8;
    let mut last_external: u128 = state.external_assets;

    for step in 0..max_steps {
        let reference = state.total_assets.saturating_mul(2);
        let target_new_external = reference
            .saturating_sub(state.idle_assets)
            .saturating_sub(1);
        let next = drive_sync_external_assets(state.clone(), target_new_external, 0)
            .unwrap_or_else(|err| panic!("step {step} rejected: {err:?}"));
        last_external = next.external_assets;
        state = next;
    }

    let inflation_factor = last_external / 999;
    assert!(inflation_factor >= 64);
}

#[test]
fn adv_d8_adapter_compromise_underflow_to_zero_succeeds_in_one_sync() {
    let mut state = seeded_state_in_allocating(1_000, 0);
    state.external_assets = 10_000;
    state.sync_total_assets();

    let next = drive_sync_external_assets(state, 0, 0).expect("deflation to 0 accepted");
    assert_eq!(next.total_assets, 1_000);
    assert_eq!(next.external_assets, 0);
}
```

BVSS

AO:A/AC:L/AX:M/R:N/S:C/C:N/A:N/I:N/D:C/Y:N (8.4)

Recommendation

To mitigate this issue, the following improvements are recommended:

- External-asset updates should enforce a cumulative per-window growth and shrink cap, not only a per-transaction cap.
- Large decreases should be bounded or routed through an explicit loss-realization flow with governance/keeper confirmation.
- Every adapter sync that reaches the configured threshold should emit a structured event and require follow-up reconciliation before another threshold update is accepted.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix validates adapter-reported supply and refresh observations against market policy, principal, and cap context before kernel sync. Legitimate adapter-reported decreases are allowed while cumulative refresh inflation and invalid observations are rejected before policy principal updates are staged.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/80836ac1c646d1b3bc70309884f9ff561154d38f

20.5 GOVERNANCE CONTRACT CANNOT PAUSE, UNPAUSE, OR UPDATE RESTRICTIONS ON THE DEPLOYED RBAC

// MEDIUM

Description

The production RBAC built by `load_vault_bootstrap` grants the governance contract `Role::Curator` only. The canonical policy class for `Pause` and `SetRestrictions` is `AuthPolicyClass::Sentinel`, and the strict `allowed_roles_for` mapping requires `Role::Sentinel` with no curator fallback.
When governance attempts to dispatch `SetPaused` or `SetRestrictions` through `VaultCommand::SetGovernancePolicy`, the vault's `RbacAuth::authorize` rejects with `AuthError::MissingRole`. Because the governance contract calls back into the vault via `invoke_contract`, the typed contract error escalates to a host trap and the entire transaction aborts.
The governance crate's own tests use a `MockVault` that does not enforce RBAC, so this break has not surfaced in pre-existing test coverage. Integration tests cover `rbac_vault.pause(sentinel, true)` directly, which works because the sentinel is set up with `Role::Sentinel`; no end-to-end test exercises the documented `governance → vault.pause` path.

Code Location

Copy Code

```
// contract/vault/soroban/src/contract/helpers.rs (load_vault_bootstrap)
let mut rbac_config = RbacConfig::with_curator(curator_kernel);
rbac_config.add_role(governance_kernel, Role::Curator);
load_rbac_addresses(env, &VaultDataKey::Guardians, Role::Sentinel, &mut rbac_config);
load_rbac_addresses(env, &VaultDataKey::Allocators, Role::Allocator, &mut rbac_config);
let auth = RbacAuth::new(rbac_config);
```

Copy Code

```
// contract/vault/curator-primitives/src/auth/mod.rs (canonical_policy_class)
ActionKind::Pause | ActionKind::SetRestrictions => AuthPolicyClass::Sentinel,
```

Copy Code

```
// contract/vault/curator-primitives/src/rbac/mod.rs (allowed_roles_for)
AuthPolicyClass::Sentinel => RoleSet::SENTINEL,
```

Impact

The advertised governance-mediated emergency pause and restriction updates do not function on the default deployment. Operators must instead pre-configure a separate guardian or sentinel address to perform pauses, weakening the single-source-of-truth control plane the governance contract is supposed to provide.
The failure surface is also a Soroban host trap rather than a typed error, so off-chain admins debugging a stuck pause have no clear signal that the missing RBAC grant is the root cause.

Proof of Concept

Preconditions

- The vault is deployed with the real `SorobanVaultGovernanceContract` as its stored governance address (the documented control plane).
- `load_vault_bootstrap` has wired RBAC such that the governance kernel address holds `Role::Curator` and no other role.
- An authorized governance admin attempts to pause the vault.

Attack / Defect Sequence

1. **Admin issues `submit_set_paused(true)`.** The governance contract's `decide_submission` classifies `SetPaused` as `TimelockDecision::Immediate`. With `timelock_ns = 0` (the default deployment recipe), the proposal is created and immediately dispatched.
2. **Governance dispatches the vault command.** `execute_action` for `SetPaused` builds a `VaultCommand::SetGovernancePolicy(PAUSED, mode = 1)` payload and invokes the vault's `execute` entrypoint via `env.invoke_contract`.
3. **The vault enters the policy dispatcher.** The vault's `set_governance_policy_impl` verifies `caller == governance`, then routes to `apply_paused_policy`, which calls `vault.pause(governance_kernel, true)`.
4. **The RBAC check rejects.** Inside `CuratorVault::pause`, the runtime calls `self.authorize(ActionKind::Pause, governance_kernel)`. The strict `RbacAuth::authorize` resolves `AuthPolicyClass::Sentinel` for `Pause`, which under `allowed_roles_for` maps to `RoleSet::SENTINEL` only. Because the governance kernel address holds `Role::Curator`, the check returns `AuthError::MissingRole`.
5. **Host trap propagates upward.** The vault returns `ContractError::Unauthorized` from `execute`. `invoke_contract` on the governance side escalates the contract error to a host trap (Soroban diagnostic event `Error(Contract, #1)`), aborting the entire transaction.
6. **Vault state is unchanged.** Because the whole transaction reverts, `SorobanStorage::is_paused()` still returns `false`. The advertised emergency pause path is non-functional.

Observable Result

The governance call aborts with a host trap rather than a typed contract error. The vault's persisted `paused` flag remains `false`. Operators investigating the failure must read kernel-source-level RBAC code to identify the missing role; the runtime gives no actionable feedback.

Reproducing Locally

The runnable PoC is in `contract/vault/soroban/tests/security_pocs.rs` (function `poc_governance_pause_rejected_by_vault_rbac`). Run with:

```
cargo test --test security_pocs poc_governance_pause_rejected_by_vault_rbac -- --nocapture
```

Stdout:

```
governance pause aborted with host trap
```

Full PoC body:

```
// Deploy vault + real governance contract; install governance as the
// vault's stored governance address.
let vault = env.register(SorobanVaultContract, ());
let asset = env.register_stellar_asset_contract_v2(Address::generate(&env)).address();
let share = env.register_stellar_asset_contract_v2(vault.clone()).address();
let governance = env.register(SorobanVaultGovernanceContract, (&admin, &vault, &0u64));
env.as_contract(&vault, || {
    SorobanVaultContract::initialize(
        env.clone(), admin.clone(), governance.clone(), asset, share, 0, 0,
    ).unwrap();
});

// submit_set_paused(true) - Immediate; governance calls back into vault
// inline. Vault's RBAC rejects because governance has Role::Curator only,
// not Role::Sentinel which Pause requires.
let res = std::panic::catch_unwind(std::panic::AssertUnwindSafe(|| {
    env.as_contract(&governance, || {
        SorobanVaultGovernanceContract::submit_set_paused(
            env.clone(),
            admin.clone(),
            true,
        )
    })
}));

match res {
    Ok(Ok(_)) => panic!("governance-routed pause unexpectedly succeeded"),
    Ok(Err(e)) => println!("governance pause returned typed error: {e:?}"),
    Err(_) => println!("governance pause aborted with host trap"),
}

// Vault's persisted pause state did not change
env.as_contract(&vault, || {
    let st = SorobanStorage::new(&env);
    assert!(!st.is_paused(), "vault pause state must not have changed");
});
```

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:H/I:H/D:P/Y:N (6.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- The governance kernel address should also be granted `Role::Sentinel` in `load_vault_bootstrap`, so that the governance contract can act as an emergency back-stop for `Pause` and `SetRestrictions` while remaining the single-source-of-truth control plane.
- Alternatively, `canonical_policy_class` for `Pause` and `SetRestrictions` should be relaxed to `AuthPolicyClass::Curator` with curator-class fallback so that operational roles (curator, guardian, sentinel) can all reach these actions.
- An end-to-end test should be added that submits `SetPaused(true)` through the governance contract and asserts that the vault's persisted pause state, emitted events, and subsequent deposit-denial behavior

all reflect the change.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix implements the intended Model B authority split: Sentinel is a separate emergency role holder for immediate pause and restriction tightening, while governance unpause and restriction relaxation/removal remain timelocked. Runtime execute_governance now enforces the configured Sentinel identity so arbitrary signers cannot bypass the governance contract, and governance-as-sentinel / sentinel-as-governance collisions are rejected.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/488efb44e2399dacf5eff88ca9ebc99c98aec083

20.6 SHARE-TOKEN ADMIN CAN REDIRECT VAULT-MINT PRIVILEGE WITHOUT TIMELOCK OR EVENT

// MEDIUM

Description

The Soroban share-token contract exposes `set_vault` as an instantaneous admin operation. The function checks that the caller is the stored admin and that the new `vault` is a contract address, then rewrites `DataKey::Vault` in a single transaction. There is no timelock, no delay, no two-step staging, and no event emission.

All mint/burn paths on the share token (`mint`, `burn`, `burn_from`) call `require_vault_invoker`, which authenticates against the stored vault address. A compromised share-token admin can repoint the share token at a different (malicious) vault contract; immediately afterwards the original vault's `MintShares` / `BurnShares` / `BurnSharesFrom` kernel effects start failing because the original vault no longer authenticates as the share token's invoker.

Code Location

Copy Code

```
// contract/vault/soroban/share-token/src/lib.rs (set_vault)
pub fn set_vault(env: Env, caller: Address, vault: Address) {
    extend_instance_ttl(&env);
    require_admin(&env, &caller);
    require_contract_address(&env, &vault);
    env.storage().instance().set(&DataKey::Vault, &vault);
}
```

Impact

A compromised share-token admin produces a hard denial of service of share minting and burning on the original vault the moment `set_vault` is invoked: every deposit, withdrawal effect, and atomic withdraw/redeem path that requires `MintShares` or `BurnShares` begins reverting. The mint authority can also be redirected to a malicious vault, which (depending on integrations) may masquerade as the legitimate share token to consumers.

Because `set_vault` emits no event, off-chain monitoring cannot observe the rotation. The original vault has no way to detect or recover from the redirect short of a coordinated migration to a new share token.

Proof of Concept

Preconditions

- A share-token contract is deployed via `SorobanShareTokenContract::__constructor(admin, vault, name, symbol, decimals)` bound to `vault_v1`.
- The vault has minted some shares (`vault_v1` users hold non-zero share balances).
- The share-token admin private key is compromised — phished, leaked, or accessible to a malicious operator.

Attack Sequence

1. **Baseline check.** A user calls `VaultCommand::DepositWithMin` on `vault_v1`. The kernel emits a `MintShares` effect, the runtime calls `share_token.mint(receiver, amount)`, `require_vault_invoker` verifies that `vault_v1` has authenticated, and the deposit succeeds.
2. **Attacker deploys a lookalike vault.** The attacker registers `vault_v2` at any address. It does not need to be a real vault; any contract that can be a target of `set_vault` will do.
3. **Attacker calls `set_vault`.** Using the compromised admin signature, the attacker invokes `SorobanShareTokenContract::set_vault(admin, vault_v2)`. The function passes the `require_admin` check, passes `require_contract_address(vault_v2)` (since `vault_v2` is a contract), and writes `DataKey::Vault = vault_v2` in a single transaction. There is no timelock, no event, no observable signal.
4. **The original vault's mint/burn paths break.** The next user deposit on `vault_v1` emits a `MintShares` effect. The runtime calls `share_token.mint(...)`; `require_vault_invoker` reads the stored vault (`vault_v2`) and demands its authentication. `vault_v1` cannot authenticate as `vault_v2`, so the call traps and the deposit aborts.
5. **Hard DoS persists.** Every subsequent `DepositWithMin`, `RequestWithdraw` (which escrows shares), `ExecuteWithdraw` (which burns shares), and atomic withdraw/redeem on `vault_v1` reverts because each requires share-token mint or burn. Off-chain monitoring sees only revert events, not the underlying `set_vault` rotation.
6. **Optional escalation.** If `vault_v2` is itself a malicious contract, the attacker now controls a `require_vault_invoker`-gated mint/burn surface attached to a share token that some integrations may still consider legitimate.

Observable Result

`vault_v1.execute(VaultCommand::DepositWithMin { ... })` aborts on every call after step 3, while `share_token.storage().instance().get(DataKey::Vault) == vault_v2` shows the redirect. No event is emitted by `set_vault` itself.

Reproducing Locally

The PoC is referenced in `contract/vault/soroban/tests/security_pocs.rs` as `poc_share_token_admin_redirect_dos` and is intentionally `#[ignore]` in the vault runtime test harness because the share-token crate depends on `soroban-sdk 23.5.1` while the vault runtime depends on `soroban-sdk 25.0.1`; cross-registering the share-token contract inside the vault runtime's harness produces incompatible `Address`/`Env` types.
The bug is fully reproducible on the share-token crate's own toolchain via its existing `set_vault` tests:

Copy Code

```
cargo test -p templar-soroban-share-token
```

The relevant production code is the share-token's `set_vault`, which performs no timelock or event:

Copy Code

```
// share-token/src/lib.rs (set_vault)
pub fn set_vault(env: Env, caller: Address, vault: Address) {
    extend_instance_ttl(&env);
    require_admin(&env, &caller);
    require_contract_address(&env, &vault);
```

```
    env.storage().instance().set(&DataKey::Vault, &vault);
}
```

The existing share-token unit tests (in `share-token/src/tests.rs`) demonstrate that `set_vault` succeeds in a single transaction with admin authentication only, with no event emission and no timelock window between admin call and effective state change.

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:H/I:H/D:N/Y:N (6.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- A delay or timelock should be added to `set_vault`; the cleanest design is a two-step admin → staged vault → finalize after `N` ledgers pattern.
- An explicit event with the previous and new vault addresses should be emitted on rotation, so that off-chain monitoring can alarm on changes.
- An operator runbook should be documented and tested where the share-token admin is either revoked after deployment or rotated to the governance contract address with a timelock.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix makes the share-token vault binding immutable after construction. The public set_vault entrypoint remains for ABI compatibility but now fails with ShareTokenError::VaultImmutable, preventing an admin from redirecting mint/burn authority away from the intended vault.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/5c302e89db9f096ea94ff70c521f95273d069bf6

20.7 SHIPPED GOVERNANCE CONTRACT CANNOT MANAGE ALLOCATORS OR THE ADAPTER ALLOWLIST

// MEDIUM

Description

The Soroban vault's runtime accepts `VaultCommand::SetGovernanceConfig` with kinds `GOVERNANCE_CONFIG_KIND_ALLOCATORS` and `GOVERNANCE_CONFIG_KIND_ALLOWED_ADAPTERS`, and applies both at the vault layer. The shipped governance contract (`SorobanVaultGovernanceContract`), however, exposes no `submit_set_allocators` or `submit_set_allowed_adapters` method, and its `execute_action` never emits the corresponding `SetGovernanceConfig` variants.
Operators that install the governance contract as the vault's stored governance address therefore have no advertised submit/accept timelock path for either allocator-set updates or adapter-allowlist updates. The only working surface is to call `VaultCommand::SetGovernanceConfig` directly with the governance contract as caller — which sidesteps the governance proposal queue, abdication map, and timelock entirely.

Code Location

Copy Code

```
// contract/vault/soroban/shared-types/src/lib.rs
pub const GOVERNANCE_CONFIG_KIND_ALLOCATORS: u32 = 4;
pub const GOVERNANCE_CONFIG_KIND_ALLOWED_ADAPTERS: u32 = 5;
```

Copy Code

```
// contract/vault/soroban/src/contract/entrypoints.rs (set_governance_config_impl)
GOVERNANCE_CONFIG_KIND_ALLOCATORS => apply_allocators_config(env, required_addresses(many)?),
GOVERNANCE_CONFIG_KIND_ALLOWED_ADAPTERS => apply_allowed_adapters_config(env, required_addresses(many)?)
```

The governance crate imports only `GOVERNANCE_CONFIG_KIND_CURATOR`, `..._GOVERNANCE`, `..._GUARDIANS`, `..._SENTINEL`, and `..._SKIM_RECIPIENT`; `ALLOCATORS` and `ALLOWED_ADAPTERS` are not referenced. The vault's Blend integration tests already work around this by sending `VaultCommand::SetGovernanceConfig` directly to the vault with `caller: governance`, bypassing the shipped governance-contract surface.

Impact

Allocator role separation and adapter-allowlist administration are advertised in `STRIDE.md` as governance-controlled, queued, and timelocked. In a deployment that uses the governance contract as the vault's stored governance, neither operation is reachable through the advertised submit/accept flow. Operators are forced to either keep governance as an account or multisig (so they can call the vault directly) or to bypass the governance contract for these two surfaces, which weakens the timelock guarantees the rest of the protocol relies on.

Proof of Concept

Preconditions

- The vault is deployed with the real `SorobanVaultGovernanceContract` as its stored governance address.
- An operator wants to rotate the allocator set or update the adapter allowlist via the documented timelocked governance flow.

Defect Sequence

1. **Operator looks for the governance entrypoint.** The governance contract's public surface contains `submit_set_paused`, `submit_set_curator`, `submit_set_governance`, `submit_set_supply_queue`, `submit_set_fees`, `submit_set_restrictions`, `submit_set_guardian`, `submit_set_sentinel`, `submit_set_timelock`, `submit_set_cap`, `submit_remove_market`, the cap-group submitters, `submit_set_skim_recipient`, `submit_skim`, `submit_other`, and `accept` / `accept_kind`. There is no `submit_set_allocators` and no `submit_set_allowed_adapters`.
2. **Operator inspects `execute_action` for an indirect path.** `execute_action` in the governance crate maps `GovernanceAction` variants to `VaultCommand` dispatches. None of the variants emit `VaultCommand::SetGovernanceConfig` with kind `ALLOCATORS` or `ALLOWED_ADAPTERS`; only `CURATOR`, `GOVERNANCE`, `GUARDIANS`, `SENTINEL`, and `SKIM_RECIPIENT` are routed.
3. **Operator falls back to the only working path.** The operator constructs `VaultCommand::SetGovernanceConfig { caller: governance, kind: GOVERNANCE_CONFIG_KIND_ALLOCATORS, many: Some(vec![new_allocator]), ... }` and signs it on behalf of the governance contract. The vault's `execute` entrypoint accepts the call because `caller == governance`. The allocator set is updated.
4. **Timelock guarantees are bypassed.** Because the call did not go through the governance contract's `submit` / `accept` queue, none of the following happened: (a) no `QueuedProposal` was created; (b) no observation window was enforced; (c) `require_not_abdicated` was not consulted; (d) no `ProposalSubmitted` / `ProposalAccepted` events were emitted for off-chain monitoring. The vault's allocator set and adapter allowlist are mutable through a path the threat model considers timelocked but the implementation does not.

Observable Result

The governance crate's source contains no `submit_set_allocators` or `submit_set_allowed_adapters` symbols (any attempted call fails to compile). The only working path is direct `VaultCommand::SetGovernanceConfig` with `caller = governance`, which is the same path that exists in `tests/blend_e2e.rs` as a workaround.

Reproducing Locally

The runnable demonstration is in `contract/vault/soroban/tests/security_pocs.rs` (function `poc_no_governance_path_for_allocators_or_adapters`). Run with:

```
cargo test --test security_pocs poc_no_governance_path_for_allocators_or_adapters -- --nocapture
```

Stdout:

Copy Code

```
direct vault SetGovernanceConfig(ALLOCATORS) succeeded; no governance submit_* exists.
```

Full PoC body:

Copy Code

```
// Deploy vault + real governance contract.
let vault = env.register(SorobanVaultContract, ());
let asset = env.register_stellar_asset_contract_v2(Address::generate(&env)).address();
let share = env.register_stellar_asset_contract_v2(vault.clone()).address();
let governance = env.register(SorobanVaultGovernanceContract, (&admin, &vault, &0u64));
env.as_contract(&vault, || {
    SorobanVaultContract::initialize(
        env.clone(), admin.clone(), governance.clone(), asset, share, 0, 0,
    ).unwrap();
});

// (1) The only working path is to call vault.execute(SetGovernanceConfig)
//     directly with caller=governance, sidestepping the timelock.
env.as_contract(&vault, || {
    let mut many: StdVec<StdString> = StdVec::new();
    many.push(address_text(&allocator));
    execute_command(&env, &VaultCommand::SetGovernanceConfig {
        caller: address_text(&governance),
        kind: GOVERNANCE_CONFIG_KIND_ALLOCATORS,
        primary: None,
        many: Some(many),
        value_a: None,
        value_b: None,
    }).unwrap();
});

// (2) The following, if uncommented, fails to compile because
//     SorobanVaultGovernanceContract does not expose
//     submit_set_allocators or any equivalent.
//
// SorobanVaultGovernanceContract::submit_set_allocators(
//     env.clone(),
//     admin.clone(),
//     vec![allocator.clone()],
// );
```

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:H/I:H/D:P/Y:N (6.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Typed governance actions and matching `submit_*` methods should be added for allocator updates and adapter-allowlist updates.
- `execute_action` should be extended so the governance contract emits the corresponding `VaultCommand::SetGovernanceConfig` variants for these kinds.
- An end-to-end test should be added that configures allocators and allowed adapters through the governance contract, not by directly invoking the vault.
- `STRIDE.md` should be updated until the governance contract actually implements the documented control surface.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix adds governance submissions for allocator and allowed-adapter updates, maps them to explicit action and timelock kinds, and routes accepted proposals to the vault governance config kinds for allocators and allowed adapters. Regression coverage includes allocator_submission_is_timelocked_and_routes_to_vault and allowed_adapter_submission_is_timelocked_and_routes_to_vault, with the governance library test suite passing.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/0aebe10a45bb53420a3bcb23a827c0bc3d029e88

20.8 CONTRACT-GOVERNANCE DEPLOYMENTS HAVE NO ROUTED PATH FOR UPGRADE, MIGRATE, OR CANCEL_MIGRATION

// MEDIUM

Description

The vault entrypoints `upgrade`, `migrate`, and `cancel_migration` all require a governance caller, yet the shipped governance contract exposes no `submit_upgrade`, `submit_migrate`, or `submit_cancel_migration` method. Its `execute_action` does not call into the vault's upgrade or migration entrypoints and never emits `VaultCommand::CancelMigration`.
The only governance actions actually routed to the vault from the governance contract are `SetGovernanceConfig`, `SetGovernancePolicy`, `Skim`, `SetTimelock` (governance-local), and `Other`. Upgrade and migration paths therefore live entirely outside the documented governance proposal queue, abdication map, and timelock controls.

Code Location

Copy Code

```
// contract/vault/soroban/src/contract/entrypoints.rs (cancel_migration_impl)
require_governance(env, &caller)?;
if !migration_in_progress(env) { return Err(ContractError::InvalidState); }
set_migration_in_progress(env, false);
emit_admin_event(env, symbol_short!("cnc_migr"));
```

The governance contract's public surface contains `submit_set_paused`, `submit_set_curator`, `submit_set_governance`, `submit_set_supply_queue`, `submit_set_fees`, `submit_set_restrictions`, `submit_set_guardian`, `submit_set_sentinel`, `submit_set_timelock`, `submit_set_cap`, `submit_remove_market`, the cap-group submitters, `submit_set_skim_recipient`, `submit_skim`, `submit_other`, and `accept` / `accept_kind`. None of these route to `upgrade`, `migrate`, or `cancel_migration`.

Impact

If the vault is deployed with the governance contract as its stored governance address, the highest-impact governance actions — contract upgrade, migration, and migration abort — cannot be performed through the governance contract. Operators are forced to rotate governance away from the timelock contract before upgrading, which defeats the security model that claims governance-contract control over upgrades, and exposes a window during which non-timelocked vault calls are possible.
If `upgrade()` sets `migration_in_progress`, the same governance authority is the only path back out; curator and sentinel have no separate abort path. `STRIDE.md` describes upgrade / migrate / cancel as governance-controlled and queued, which is not true of the shipped contract.

Proof of Concept

Preconditions

- The vault is deployed with `SorobanVaultGovernanceContract` as its stored governance address.
- An operator needs to upgrade the vault WASM, finalize a previously-staged migration, or cancel an in-progress migration.

Defect Sequence

1. **Operator inspects the vault's privileged entrypoints.** `SorobanVaultContract::upgrade`, `migrate`, and `cancel_migration` all begin with `require_governance(env, &caller)`, which checks `caller == storage(VaultDataKey::Governance)`.
2. **Operator confirms only the governance contract can satisfy this auth.** Calling `upgrade` with `caller = admin` (the governance contract's admin, not the governance contract itself) fails with `ContractError::Unauthorized` because the vault's stored governance is the contract address, not the admin.
3. **Operator searches the governance contract for an upgrade submitter.** The governance contract's public surface (listed under Code Location) does not contain `submit_upgrade`, `submit_migrate`, or `submit_cancel_migration`. `execute_action` never emits `VaultCommand::CancelMigration`; it only routes `SetGovernanceConfig`, `SetGovernancePolicy`, `Skim`, the local `SetTimelock`, and `Other`.
4. **Operator hits a dead end.** Because the governance contract is the only entity that can authenticate against `require_governance`, and it has no method that reaches the upgrade entrypoints, the documented governance-controlled upgrade flow simply cannot be exercised end-to-end.
5. **Operator must rotate governance away.** The only practical workaround is to first issue a `submit_set_governance(account_or_multisig)` through the governance contract, wait the timelock, accept it, and then have that EOA / multisig call `upgrade` directly on the vault. This rotation removes timelock and queue protections from every governance action for the duration of the upgrade window.

Observable Result

The governance crate's source contains no `submit_upgrade` / `submit_migrate` / `submit_cancel_migration` symbols (any attempted call fails to compile). The vault's upgrade entrypoint rejects with `ContractError::Unauthorized` for any caller other than the governance contract address. There is no working in-protocol path that combines governance auth with upgrade dispatch.

Reproducing Locally

The runnable demonstration is in `contract/vault/soroban/tests/security_pocs.rs` (function `poc_no_governance_path_for_upgrade_migrate_cancel`). Run with:

```
cargo test --test security_pocs poc_no_governance_path_for_upgrade_migrate_cancel -- --nocapture
```

Stdout:

```
no submit_upgrade/migrate/cancel_migration on governance API.
```

The PoC bootstraps the vault with the governance contract installed and then documents the missing methods. Uncommenting any of the calls below produces a compile error proving the symbol is absent:

Copy Code

```
// Deploy vault + governance contract
let vault = env.register(SorobanVaultContract, ());
let asset = env.register_stellar_asset_contract_v2(Address::generate(&env)).address();
let share = env.register_stellar_asset_contract_v2(vault.clone()).address();
let governance = env.register(SorobanVaultGovernanceContract, (&admin, &vault, &0u64));
env.as_contract(&vault, || {
    SorobanVaultContract::initialize(
        env.clone(), admin.clone(), governance.clone(), asset, share, 0, 0,
    ).unwrap();
});

// No such methods exist on the governance contract API:
//
// SorobanVaultGovernanceContract::submit_upgrade(env.clone(), admin.clone(), ...);
// SorobanVaultGovernanceContract::submit_migrate(env.clone(), admin.clone(), ...);
// SorobanVaultGovernanceContract::submit_cancel_migration(env.clone(), admin.clone());
//
// Uncommenting any of them fails at compile time.
```

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:H/I:H/D:P/Y:N (6.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Explicit governance actions and execution paths should be added for `upgrade`, `migrate`, and `cancel_migration`.
- These actions should be routed through the same proposal queue, timelock, and acceptance flow as the other high-impact governance operations.
- An end-to-end test should be added with the governance contract installed as the vault's governance address, asserting that `upgrade -> migrate` and `cancel_migration` work without any governance-address rotation.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix adds timelocked governance submissions for upgrade, migrate, and cancel_migration. Accepted upgrade and migrate proposals route to the vault upgrade/migrate paths, and cancel_migration routes through the vault execute command with the governance contract as caller after maturity.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/843a36f00ba29aa6241d01b9280b7bbc4a41c8c0

20.9 DEFAULT GOVERNANCE DEPLOYMENT INITIALIZES ALL TIMELOCKS TO ZERO

// MEDIUM

Description

The governance contract distinguishes between `Immediate` and `Timelocked` governance actions, but its constructor copies the single `timelock_ns` parameter into every per-kind timelock slot via `Timelocks::from_default(timelock_ns)`. The contract permits `MIN_TIMELOCK_NS = 0`, and the shipped `justfile` deployment recipe passes `timelock_ns="${SOROBAN_GOV_TIMELOCK_NS:-0}"`. The justfile additionally documents recipes that explicitly expect `timelock=0` for instant acceptance.
As a result, in the default deployment every kind of supposedly timelocked governance action becomes mature immediately after submission and can be accepted in the next transaction without any observation window. The same zero-floor validation is reused for later `submit_set_timelock` updates, so a no-delay posture can also be reintroduced after deployment.

Code Location

Copy Code

```
// contract/vault/soroban/governance/src/lib.rs
pub const MIN_TIMELOCK_NS: u64 = 0;
let timelocks = Timelocks::from_default(timelock_ns);
env.storage().instance().set(&DataKey::Timelocks, &timelocks);
```

Copy Code

```
# contract/vault/soroban/justfile
timelock_ns="${SOROBAN_GOV_TIMELOCK_NS:-0}"
```

Impact

Default governance deployments provide no real delay window for fee changes, cap changes, supply-queue changes, restriction relaxations, and other actions that the code classifies as `Timelocked`. Off-chain review, social coordination, and monitoring windows disappear unless the operator overrides the default explicitly. The distinction between `Immediate` and `Timelocked` actions becomes mostly theoretical in default deployments, and the same zero floor permits future governance proposals to reduce per-action timelocks back to zero after deployment.

Proof of Concept

Preconditions

- The governance contract is deployed using the shipped justfile recipe (or any deployment that does not explicitly set `SOROBAN_GOV_TIMELOCK_NS`).
- The constructor was called with `timelock_ns = 0`, which the contract accepts because `MIN_TIMELOCK_NS = 0`.

Defect Sequence

1. **Admin submits a Timelocked action.** The admin calls a Timelocked submitter such as `submit_other(key, payload_hash)`. `GovernanceAction::Other` is classified as `TimelockDecision::Timelocked` by `decide_submission`.
2. **Governance computes the maturity timestamp.** The runtime reads the per-kind timelock from `DataKey::Timelocks`, which holds `0` for every kind. `valid_after_ns = now_ns + 0 = now_ns`. The proposal is enqueued and immediately considered mature.
3. **Admin calls `accept(id)` in the same ledger tick.** No `advance_time` is needed. The runtime checks `now_ns >= valid_after_ns`, which is trivially true (equal), and runs `execute_action` on the proposal. For `Other`, this sets `DataKey::ApprovedOther(key, payload_hash) = true`.
4. **Observability window is zero.** No off-chain monitor, social-review process, or human delay can intervene between `submit` and `accept`: the two calls happen in the same block. The same is true for every other Timelocked action class — fee changes, cap changes, supply-queue changes, restriction relaxations — because they all share the same per-kind `0 ns` entry.
5. **Future updates can re-arm the zero posture.** Even if an operator later submits `submit_set_timelock(kind, larger_ns)` and accepts it, anyone with admin authority can submit `submit_set_timelock(kind, 0)` again because `MIN_TIMELOCK_NS = 0` still allows it.

Observable Result

After `submit_other` followed immediately by `accept(id)` in the same ledger tick, `check_other(key, payload_hash)` returns `true`. The same pattern applies to every Timelocked action.

Reproducing Locally

The runnable PoC is in `contract/vault/soroban/tests/security_pocs.rs` (function `poc_zero_timelock_immediate_accept`). Run with:

Copy Code

```
cargo test --test security_pocs poc_zero_timelock_immediate_accept -- --nocapture
```

Stdout:

Copy Code

```
submit_other + accept cleared at the same timestamp (timelock_ns=0)
```

Full PoC body:

Copy Code

```
// Deploy governance with timelock_ns=0 (the justfile default).
let vault = env.register(SorobanVaultContract, ());
let governance = env.register(SorobanVaultGovernanceContract, (&admin, &vault, &0u64));

let key = Symbol::new(&env, "demo_key");
let payload_hash: BytesN<32> = BytesN::random(&env);

// 1) Submit a Timelocked action; with timelock_ns=0, valid_after == now
let id = env.as_contract(&governance, || {
    SorobanVaultGovernanceContract::submit_other(
        env.clone(),
        admin.clone(),
        key.clone(),
        payload_hash.clone(),
```

```
    ).unwrap()
});

// 2) Accept in the same ledger tick - NO advance_secs() between calls
env.as_contract(&governance, || {
    SorobanVaultGovernanceContract::accept(env.clone(), admin.clone(), id).unwrap();
});

// 3) The Timelocked action took effect with zero observation window
let approved = env.as_contract(&governance, || {
    SorobanVaultGovernanceContract::check_other(
        env.clone(),
        key.clone(),
        payload_hash.clone(),
    )
});
assert!(approved,
    "zero-timelock governance accepted a Timelocked action without any elapsed time");
```

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:H/I:H/D:P/Y:N (6.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- The default deployment recipe should be changed to use a non-zero timelock that reflects the intended production security posture for each timelock class.
- An explicit unsafe opt-in should be required before allowing `0 ns` timelocks in helper scripts.
- Deployment-time checks should be added that print or reject an all-zero timelock table, so that operators cannot miss the effective policy they are launching with.

Remediation Comment

SOLVED: Client successfully remediated this issue. The deployment recipe now requires SOROBAN_GOV_TIMELOCK_NS to be set explicitly and rejects zero unless SOROBAN_ALLOW_ZERO_GOV_TIMELOCK=1 is also provided for ephemeral testing. The contract can still support zero timelocks for test/emergency environments, but the production deploy path no longer silently defaults to zero.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/b6ee556b240fb64cc52ddb8b1b12d79dd9cd5e8a

20.10 NO PRODUCTION CODE PATH ADDS NEW MARKETS TO THE VAULT

// MEDIUM

Description

The Soroban vault exposes governance policy kinds for `SUPPLY_QUEUE`, `CAP`, `REMOVE_MARKET`, `RESTRICTIONS`, `GROUP`, `PAUSED`, and `FEES`, but no `KIND_CREATE_MARKET` or `KIND_SET_MARKET_CONFIG`. The only primitive that inserts a new market into `PolicyState::markets` is `set_market_config`, and that primitive is reachable only from test or property-test code paths — not from any `VaultCommand` dispatcher in `src/contract/*`.

Combined with finding P2-18 (the vault's `set_cap` rejection of timelocked decisions on unknown markets), this means a freshly-deployed vault cannot onboard any market through the advertised governance surface after `initialize()`.

Code Location

Copy Code

```
// contract/vault/soroban/shared-types/src/lib.rs
pub const GOVERNANCE_POLICY_KIND_SUPPLY_QUEUE: u32 = 0;
pub const GOVERNANCE_POLICY_KIND_CAP: u32 = 1;
pub const GOVERNANCE_POLICY_KIND_REMOVE_MARKET: u32 = 2;
pub const GOVERNANCE_POLICY_KIND_RESTRICTIONS: u32 = 3;
pub const GOVERNANCE_POLICY_KIND_GROUP: u32 = 4;
pub const GOVERNANCE_POLICY_KIND_PAUSED: u32 = 5;
pub const GOVERNANCE_POLICY_KIND_FEES: u32 = 6;
```

Copy Code

```
// contract/vault/curator-primitives/src/policy/state/mod.rs (set_market_cap)
let market = self.markets.get_mut(&target_id).ok_or(PolicyStateError::UnknownMarket { target_id })?;
```

Impact

The vault cannot be expanded beyond whatever markets happen to exist after initialization. With the current code, `initialize()` creates zero markets, so a fresh mainnet deployment is functionally unusable until either a custom WASM is deployed that pre-populates markets or the missing `SetMarketConfig` governance action is added to the shared ABI. Any vault that integrates with Blend or any other adapter has no way to register the first market via governance.

Proof of Concept

Preconditions

- A fresh vault is deployed via `SorobanVaultContract::initialize`. By contract design, no markets exist after initialization.
- An operator wants to onboard the first market (e.g., a Blend pool) through the advertised governance flow.

Defect Sequence

1. **Operator submits `submit_set_cap(42, 1_000)` for an unknown market.** Governance accepts the proposal. With `timelock_ns = 0` the proposal matures immediately. Operator calls `accept(id)`.
2. **Governance dispatches the cap policy.** `execute_action` emits `VaultCommand::SetGovernancePolicy(CAP, market_id = 42, value = 1_000)` and invokes the vault's `execute` entrypoint.
3. **Vault rejects the cap dispatch.** `set_cap` calls `policy_state.market_config(42)`, which returns `None`. `TimelockDecision::from_cap_change(None, 1_000)` evaluates the `None → any cap` branch and returns `Timelocked`. The vault's runtime guard rejects with `InvalidInput("cap increase or new market requires timelock")`. The host trap aborts the entire `accept` transaction.
4. **Operator tries the supply-queue path instead.** The operator constructs `SetGovernancePolicy(SUPPLY_QUEUE, target_ids = [42])` via direct call. The vault's `set_supply_queue` calls `policy_state.market_config(42)`, gets `None`, and returns `InvalidInput`.
5. **Operator inspects the source for any other path.** A workspace search confirms that `PolicyState::set_market_config` — the only function that inserts into `policy_state.markets` — is called only from `src/tests.rs`, `tests/integration_tests.rs`, and `tests/property_tests.rs`. There is no production call site, no governance action, and no curator entrypoint that reaches it.
6. **Vault is left empty.** `policy.markets().len()` remains `0`. No deposit or allocation can target any market because none exist.

Observable Result

Every governance attempt to register or reference market `42` fails. `SorobanStorage::load_policy_state().unwrap().unwrap_or_default().markets().len()` stays at `0`. The vault is functionally inert until a custom-deployed WASM pre-populates markets or a new `SetMarketConfig` action is added to the ABI.

Reproducing Locally

The runnable PoC is in `contract/vault/soroban/tests/security_pocs.rs` (function `poc_no_market_creation_path`). Run with:

Copy Code

```
cargo test --test security_pocs poc_no_market_creation_path -- --nocapture
```

Stdout:

Copy Code

```
accept of cap-on-unknown-market aborted (host trap)
```

Full PoC body:

Copy Code

```
// Deploy fresh vault with governance contract installed
let vault = env.register(SorobanVaultContract, ());
let asset = env.register_stellar_asset_contract_v2(Address::generate(&env)).address();
let share = env.register_stellar_asset_contract_v2(vault.clone()).address();
let governance = env.register(SorobanVaultGovernanceContract, (&admin, &vault, &0u64));
env.as_contract(&vault, || {
    SorobanVaultContract::initialize(
```

```
        env.clone(), admin.clone(), governance.clone(), asset, share, 0, 0,
    ).unwrap();
});

// 1) submit_set_cap(42) where market 42 does not exist. Proposal matures
//    with timelock=0, accept is expected to fail at the vault.
let id = env.as_contract(&governance, || {
    SorobanVaultGovernanceContract::submit_set_cap(
        env.clone(),
        admin.clone(),
        42u32,
        1_000i128,
    ).unwrap()
});
let r = std::panic::catch_unwind(std::panic::AssertUnwindSafe(|| {
    env.as_contract(&governance, || {
        SorobanVaultGovernanceContract::accept(env.clone(), admin.clone(), id)
    })
}));
match r {
    Ok(Ok(_)) => panic!("cap on unknown market unexpectedly succeeded"),
    Ok(Err(e)) => println!("accept of cap-on-unknown-market returned typed error: {e:?}"),
    Err(_) => println!("accept of cap-on-unknown-market aborted (host trap)"),
}

// 2) SetGovernancePolicy SUPPLY_QUEUE with unknown id also fails.
env.as_contract(&vault, || {
    let mut ids: StdVec<u32> = StdVec::new();
    ids.push(42u32);
    let res = execute_command(&env, &VaultCommand::SetGovernancePolicy {
        caller: address_text(&governance),
        kind: GOVERNANCE_POLICY_KIND_SUPPLY_QUEUE,
        target_ids: Some(ids),
        mode: None, accounts: None, market_id: None, cap_group_id: None,
        value: None, value_b: None, value_c: None,
    });
    assert!(res.is_err(), "supply queue of unknown market must fail");
});

// 3) After every available governance action, no market exists.
env.as_contract(&vault, || {
    let policy = SorobanStorage::new(&env)
        .load_policy_state().unwrap().unwrap_or_default();
    assert_eq!(policy.markets().len(), 0, "no market was created");
});
```

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:H/I:H/D:P/Y:N (6.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- A new `GOVERNANCE_POLICY_KIND_CREATE_MARKET` (or `SET_MARKET_CONFIG`) should be added to the shared-types ABI and a corresponding dispatcher should be implemented in the policy helpers.
- Alternatively, a curator-only entrypoint should be surfaced that inserts a market with `enabled=true, cap=0, cap_group_id=None` as a "pending" state, with `SetCap` used to activate it once finding P2-18 is fixed.
- An integration test should be added that exercises the full governance-driven flow: deploy, propose, accept, verify the market is in `supply_queue_targets()`.

Remediation Comment

SOLVED: Although the client dispositioned this as risk-accepted/stale, the remediated code actually resolves it. The integrated branch adds an `apply_governance_cap` / `apply_governance_cap_authorized` path that, when routed through governance after the timelock, creates a new market configuration when the target market does not yet exist (and raises the cap when it does). This provides the previously-missing production path to add markets, proven by `test_execute_governance_cap_creates_new_runtime_market_after_timelock`.

Remediation Hash

https://github.com/Templar-Protocol/contracts/pull/417

20.11 VAULT SET_CAP REJECTS EVERY MATURED GOVERNANCE-PROPOSED CAP INCREASE

// MEDIUM

Description

Governance's `decide_submission` for `SetCap` always passes `current = None` to `TimelockDecision::from_cap_change`, which forces every cap proposal into the `Timelocked` branch. After the timelock matures and `accept(id)` dispatches `VaultCommand::SetGovernancePolicy(CAP, ...)`, the vault's `set_cap` re-evaluates the decision using the actual stored `current_cap`:

- `current_cap == Some(prev)` with `new_cap > prev` → `Timelocked` → vault rejects with "cap increase or new market requires timelock".
- `current_cap == None` (unknown market) → `Timelocked` → vault rejects.
- `current_cap == Some(prev)` with `new_cap < prev` → `Immediate` → vault allows.

The only successful path is a cap *decrease* on an existing market. Every governance-proposed cap increase, and every cap change on an unknown market, fails at the vault even after the governance timelock has expired — even though the governance contract is the only legal caller of `set_cap`.

Code Location

Copy Code

```
// contract/vault/curator-primitives/src/governance/mod.rs (from_cap_change)
match current {
    Some(existing) if proposed == existing => Err(CapChangeError::NoChange),
    Some(existing) if proposed > existing => Ok(Self::Timelocked),
    Some(_) => Ok(Self::Immediate),
    None => Ok(Self::Timelocked),
}
```

Copy Code

```
// contract/vault/soroban/src/contract/curator_vault.rs (set_cap)
let current_cap = self.policy_state.market_config(market_id).map(|m| m.cap);
let decision = TimelockDecision::from_cap_change(current_cap, new_cap)
    .map_err(|_| RuntimeError::invalid_input(""))?;
if matches!(decision, TimelockDecision::Timelocked) {
    return Err(RuntimeError::invalid_input("cap increase or new market requires timelock"));
}
```

Copy Code

```
// contract/vault/soroban/governance/src/lib.rs (decide_submission for SetCap)
GovernanceAction::SetCap(_, new_cap) => {
    let decision = TimelockDecision::from_cap_change(None, to_wad(*new_cap)?.into());
    ...
}
```

Impact

Vault operators cannot raise caps through governance, even with proper timelock approval. The vault's per-market exposure is permanently unable to grow. Combined with finding P2-17, new markets cannot be added and existing markets cannot have their caps raised — the vault's market configuration is effectively

monotonically decreasing over time. The error message itself is misleading because the governance timelock is already in place; the vault's runtime re-check is the actual blocker.

Proof of Concept

Preconditions

- The vault is deployed and market `0` already exists with `cap = 1_000` (e.g., seeded by a custom migration since the missing market-creation path prevents creating it through governance).
- The governance contract is the vault's stored governance address. Operators want to raise market 0's cap to `2_000` through the documented timelocked governance flow.

Defect Sequence

1. **Admin submits the cap increase.** Admin calls `SorobanVaultGovernanceContract::submit_set_cap(0, 2_000)`. The governance contract's `decide_submission` calls `TimelockDecision::from_cap_change(None, 2_000)` (governance always passes `current = None`). The match arm for `None → Timelocked` returns `Ok(Timelocked)`, so the proposal is queued with a timelock window.
2. **Timelock elapses.** The proposal becomes mature (`now_ns >= valid_after_ns`). Admin calls `accept(id)`.
3. **Governance dispatches the cap policy to the vault.** `execute_action` emits `VaultCommand::SetGovernancePolicy(CAP, market_id = 0, value = 2_000)` and calls `vault.execute(payload)`.
4. **Vault re-evaluates the timelock decision with real state.** Inside `set_cap`, the runtime reads `current_cap = Some(1_000)` from `policy_state.market_config(0)`. It calls `TimelockDecision::from_cap_change(Some(1_000), 2_000)`. Because `2_000 > 1_000`, the function returns `Timelocked`.
5. **Vault rejects.** The runtime hits the guard `if matches!(decision, TimelockDecision::Timelocked) { return Err(...); }` and returns `RuntimeError::invalid_input("cap increase or new market requires timelock")`. The contract error escalates to a host trap and the entire `accept` transaction reverts.
6. **Cap remains unchanged.** `policy.market_config(0).unwrap().cap` still reads `1_000` after the failed accept. There is no path through which a governance-mediated cap increase can succeed because the vault's runtime re-check is hard-coded to reject any `Timelocked` decision, and every cap increase produces `Timelocked` when re-evaluated against the real state.

Observable Result

The `accept(id)` call aborts with a host trap on every cap increase. The error message reports "cap increase or new market requires timelock", which is misleading: the governance timelock has already been observed by the time the vault sees the dispatch. Cap decreases on existing markets are the only direction that succeeds.

Reproducing Locally

The runnable PoC is in `contract/vault/soroban/tests/security_pocs.rs` (function `poc_matured_cap_increase_rejected_at_vault`). Run with:

Copy Code

```
cargo test --test security_pocs poc_matured_cap_increase_rejected_at_vault -- --nocapture
```

Stdout:

Copy Code

```
accept aborted with host trap
```

Full PoC body:

Copy Code

```
// Deploy vault + governance, seed market 0 at cap 1_000
let vault = env.register(SorobanVaultContract, ());
let asset = env.register_stellar_asset_contract_v2(Address::generate(&env)).address();
let share = env.register_stellar_asset_contract_v2(vault.clone()).address();
let governance = env.register(SorobanVaultGovernanceContract, (&admin, &vault, &0u64));
env.as_contract(&vault, || {
    SorobanVaultContract::initialize(
        env.clone(), admin.clone(), governance.clone(), asset, share, 0, 0,
    ).unwrap();

    let mut st = SorobanStorage::new(&env);
    let mut policy = st.load_policy_state().unwrap().unwrap_or_default();
    policy.set_market_config(0, MarketConfig::new(true, 1_000u128, None)).unwrap();
    st.save_policy_state(&policy).unwrap();
});

// 1) Submit a cap increase. decide_submission passes current=None -> Timelocked.
//    With timelock_ns=0 the proposal is mature immediately.
let id = env.as_contract(&governance, || {
    SorobanVaultGovernanceContract::submit_set_cap(
        env.clone(), admin.clone(), 0u32, 2_000i128,
    ).unwrap()
});

// 2) Accept dispatches SetGovernancePolicy(CAP, ...) to the vault.
//    The vault re-checks with real current=1_000 -> Timelocked -> rejects.
let res = std::panic::catch_unwind(std::panic::AssertUnwindSafe(|| {
    env.as_contract(&governance, || {
        SorobanVaultGovernanceContract::accept(env.clone(), admin.clone(), id)
    })
}));
match res {
    Ok(Ok(_)) => panic!("cap increase unexpectedly succeeded through governance"),
    Ok(Err(e)) => println!("accept returned typed error: {e:?}"),
    Err(_) => println!("accept aborted with host trap"),
}

// 3) Cap is still 1_000
env.as_contract(&vault, || {
    let st = SorobanStorage::new(&env);
    let policy = st.load_policy_state().unwrap().unwrap();
    assert_eq!(policy.market_config(0).unwrap().cap, 1_000);
});
```

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:H/I:H/D:P/Y:N (6.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- The "Timelocked → reject" guard in `set_cap` should be removed. The governance contract already applies the timelock before the dispatch; the vault's runtime re-check is both redundant and broken.
- Once that guard is removed, `decide_submission(SetCap)` should be fixed to pass the actual current cap (via a governance-side cache or a `proxy_view` lookup) so that governance applies the correct per-direction timelock.
- Regression tests should be added covering: submit + accept `SetCap` for a new market (market exists with the requested cap); submit + accept `SetCap` that raises an existing cap (cap is raised); submit + accept `SetCap` that lowers an existing cap (cap is lowered without unnecessary extra delay).

Remediation Comment

SOLVED: Although dispositioned as risk-accepted, the remediated code resolves the issue. Matured governance cap changes no longer go through the immediate `set_cap` guard that rejected increases; they are routed through `apply_governance_cap_authorized`, which applies the increase (or creates the market) after the timelock matures. Governance-proposed cap increases now succeed instead of reverting at the vault.

Remediation Hash

https://github.com/Templar-Protocol/contracts/pull/417

20.12 FEE ACCRUAL IS UNREACHABLE FROM THE DEPLOYED SOROBAN ABI

// MEDIUM

Description

The Soroban runtime implements a full fee accrual surface in `preview_state_with_fee_accrual` and `KernelAction::RefreshFees`, but the only call site for the kernel action is `CuratorVault::prepare_atomic_call` — a private helper used exclusively by `atomic_withdraw` / `atomic_redeem`, neither of which is reachable from any `VaultCommand` variant on the deployed contract. The `VaultCommand` ABI has no `RefreshFees` variant, and `execute_command` never dispatches to a fee-refresh branch. The public contract surface (`initialize`, `execute`, `proxy_view`, `upgrade`, `migrate`) does not expose a fee-refresh entrypoint either. As a result, persisted `state.total_shares` and `state.fee_anchor.timestamp_ns` never advance, while `proxy_view` reports preview values that include simulated fee accrual.

Code Location

Copy Code

```
// contract/vault/soroban/src/contract/curator_vault.rs (prepare_atomic_call)
let fees_active = !self.config.fees.management.fee_wad.is_zero()
    || !self.config.fees.performance.fee_wad.is_zero();
if fees_active && now_ns > self.state()?.fee_anchor.timestamp_ns.as_u64() {
    let _ = self.apply_kernel_action(
        KernelAction::RefreshFees { now_ns: TimestampNs(now_ns) },
        now_ns,
    )?;
}
```

Copy Code

```
// contract/vault/kernel/src/actions/mod.rs (handle_deposit)
let shares_out = convert_to_shares(&state, config, assets_in);
// no RefreshFees call before computing shares
state.total_assets = state.total_assets.checked_add(assets_in)?;
```

Impact

Management and performance fee recipients can be permanently underpaid because no command on the deployed ABI ever crystallizes accrued fees into `total_shares`. Depositors mint against a stale supply that omits accrued fee shares, which over-mints relative to the fee-aware preview shown by `proxy_view` and dilutes existing holders plus fee recipients. Any flow that depends on the stale supply behaves differently from what the preview surface advertises, creating a permanent divergence between tooling expectations and on-chain reality.

Proof of Concept

Preconditions

- The vault is bootstrapped (zero fees by default after `initialize()`; the issue manifests as soon as non-zero fees are configured).
- An off-chain monitor or fee recipient expects `state.fee_anchor.timestamp_ns` to advance over time as fees crystallize.

Defect Sequence

1. **Bootstrap baseline.** Right after `initialize()`, `state.fee_anchor.timestamp_ns` reads `0` and the vault holds no shares.
2. **Time passes.** The ledger advances by a substantial period (e.g., one year, `365 × 24 × 3600` seconds). During this period the vault would, with non-zero fees, normally accrue management and performance fees on every state-changing call.
3. **Operator/users exercise every public ABI surface.** Submit `VaultCommand::DepositWithMin`, `RequestWithdraw`, `ExecuteWithdraw`, `ResyncIdleBalance`, and `ExtendTtl`. Each of these routes through `execute_command` in `entrypoints.rs`. None of them call `CuratorVault::prepare_atomic_call`. None of them dispatch `KernelAction::RefreshFees`.
4. **State remains anchored at zero.** After every public command, `SorobanStorage::load_state().unwrap().unwrap().fee_anchor.timestamp_ns.as_u64()` still reads `0`. The fee anchor never updates because the only code path that calls `RefreshFees` is `prepare_atomic_call`, which only the unreachable `atomic_withdraw` / `atomic_redeem` entrypoints invoke.
5. **Preview surface diverges from execution surface.** `proxy_view` calls `preview_state_with_fee_accrual`, which simulates fee accrual on the read path and reports a fee-aware `total_shares`. Tooling that consumes `proxy_view`'s output expects this number to be honored on the next state-changing call, but it never is.
6. **Fee recipients receive nothing.** Both fee recipients (management and performance) hold zero share balances regardless of how much time elapses. The vault retains the value that should have been minted to them.

Observable Result

After a year of simulated time and exercising every public command, the persisted `fee_anchor.timestamp_ns` stays at `0`. No `MintShares` effect ever targets either fee recipient.

Reproducing Locally

The runnable PoC is in `contract/vault/soroban/tests/security_pocs.rs` (function `poc_fee_accrual_unreachable`). Run with:

```
cargo test --test security_pocs poc_fee_accrual_unreachable -- --nocapture
```

Stdout:

```
fee_anchor before = 0, after = 0 (1 year + deposit + withdraw + resync + extend_ttl)
```

Full PoC body:

Copy Code

```
// Bootstrap a fresh vault.
let (vault, asset, _share) = bootstrap_vault(&env, &admin);
let asset_admin_client = StellarAssetClient::new(&env, &asset);

// 1) Note the fee anchor right after initialize()
let anchor_before = env.as_contract(&vault, || {
    SorobanStorage::new(&env)
        .load_state().unwrap().unwrap_or_default()
        .fee_anchor.timestamp_ns.as_u64()
});

// 2) Advance ledger by 1 year
advance_secs(&env, 365 * 24 * 3600);

// 3) Exercise every public VaultCommand the deployed ABI accepts
let user = Address::generate(&env);
asset_admin_client.mint(&user, &10_000);
env.as_contract(&vault, || {
    execute_command(&env, &VaultCommand::DepositWithMin {
        owner: address_text(&user), receiver: address_text(&user),
        assets: 10_000, min_shares_out: 0,
    }).unwrap();
    execute_command(&env, &VaultCommand::RequestWithdraw {
        owner: address_text(&user), receiver: address_text(&user),
        shares: 1_000, min_assets_out: 0,
    }).unwrap();
    advance_secs(&env, DEFAULT_COOLDOWN_NS / 1_000_000_000 + 1);
    execute_command(&env, &VaultCommand::ExecuteWithdraw {
        caller: address_text(&admin),
    }).unwrap();
    execute_command(&env, &VaultCommand::ResyncIdleBalance).unwrap();
    execute_command(&env, &VaultCommand::ExtendTtl).unwrap();
});

// 4) Re-read the fee anchor: it has not advanced. No public ABI path
//    triggered KernelAction::RefreshFees, so fee accrual never crystallized.
let anchor_after = env.as_contract(&vault, || {
    SorobanStorage::new(&env)
        .load_state().unwrap().unwrap()
        .fee_anchor.timestamp_ns.as_u64()
});
assert_eq!(anchor_before, anchor_after,
    "fee anchor advanced, so some public ABI path did trigger RefreshFees");
assert_eq!(anchor_after, 0,
    "no ABI call ever advanced fee_anchor.timestamp_ns");
```

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:N/I:H/D:N/Y:H (6.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- An explicit, reachable fee-refresh path should be added to the Soroban ABI — either a dedicated `VaultCommand::RefreshFees` variant or middleware that refreshes fees before every state-changing action that depends on `total_shares` or `fee_anchor`.
- Invariant tests should be added that compare execution results before and after a forced fee refresh for deposit, request-withdraw, execute-withdraw, allocate, and refresh-markets paths.
- The preview surface in `preview_state_with_fee_accrual` should either be aligned with the executable surface or clearly marked as advisory to prevent off-chain tooling from consuming numbers the contract will

never produce.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix exposes fee accrual through the deployed Soroban ABI by adding VaultCommand::RefreshFees and routing it through the public execute path. Accepted calls now update the persisted fee anchor instead of leaving fee-aware previews to diverge from on-chain state.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/12916e17eb689174e68666fe79a8d14760a9576a

20.13 INITIAL FEE_ANCHOR AT (0, 0) INFLATES PROXY_VIEW TOTAL_SHARES ONCE PERFORMANCE FEES ARE ENABLED

// MEDIUM

Description

The Soroban runtime's `initialize()` writes `VaultState::default()`, which sets `state.fee_anchor = FeeAccrualAnchor { total_assets: 0, timestamp_ns: TimestampNs(0) }`. None of the routine state-mutating paths (`handle_deposit`, `handle_request_withdraw`, `handle_execute_withdraw`, allocate, refresh, etc.) advance `fee_anchor`. The only path that advances it is `handle_refresh_fees`, which is unreachable from the deployed Soroban ABI today (see Fee accrual unreachable from the deployed Soroban ABI).

So the persisted `fee_anchor` remains at `(0, 0)` indefinitely. `proxy_view` calls `preview_state_with_fee_accrual`, which uses `profit = fee_total_assets − anchor.total_assets` to compute performance-fee shares against the doubled-supply preview. With `anchor.total_assets = 0`, `profit = current_assets`, and `perf_wad = MAX_PERFORMANCE_FEE_WAD = 50%`, the math reduces to `fee_shares = supply`, which doubles the previewed `total_shares`.

Code Location

Copy Code

```
// contract/vault/soroban/src/contract/entrypoints.rs (initialize)
pub fn initialize(env: Env, ...) -> Result<(), ContractError> {
    if env.storage().instance().has(&VaultDataKey::Initialized) {
        return Err(ContractError::AlreadyInitialized);
    }
    ...
    let mut storage = SorobanStorage::new(&env);
    runtime_to_contract(storage.save_state(&VaultState::default()))?;
    runtime_to_contract(storage.save_paused(false))?;
    Ok(())
}
```

Copy Code

```
// contract/vault/soroban/src/fungible_vault.rs (preview_state_with_fee_accrual)
let profit = fee_assets_base.saturating_sub(anchor.total_assets);
let performance_fee_assets = config.fees.performance.fee_wad.apply_floored(Number::from(profit));
let performance_shares = compute_fee_shares_from_assets(
    performance_fee_assets,
    Number::from(current_assets),
    supply_after_management,
);
state.total_shares = supply_after_management.saturating_add(performance_shares).as_u128_saturating();
```

Impact

Concrete user-visible effect: a depositor reads `preview_deposit(assets) -> shares` from `proxy_view` and sets `min_shares_out = 0.9 * shares` for slippage. The actual `handle_deposit` uses raw state and mints approximately half that share amount. The slippage check rejects the deposit with a `Slippage` error. ERC-4626 integrators that rely on `proxy_view` for share-price quoting see a roughly halved share price.

Latent escalation: if the unreachable `RefreshFees` path is ever wired into `VaultCommand` without also fixing the initial anchor, the very first refresh after fees are enabled mints performance-fee shares equal to the entire current supply, doubling `total_shares` in persisted state and diluting all existing shareholders 50% in favor of the performance-fee recipient.

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:H/I:H/D:P/Y:N (6.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- In `initialize()`, set `state.fee_anchor = FeeAccrualAnchor::new(0, TimestampNs(now_ns))` using `ledger_timestamp_ns(env)?`. This anchors the fee window to contract-creation time.
- In `apply_fees_policy`, additionally crystallize `fee_anchor` to `(state.total_assets, now_ns)` whenever fees are first enabled.
- Add a regression test: initialize a vault, simulate `now_ns = YEAR_NS`, enable performance fees via governance, call `proxy_view`, and assert that the previewed `total_shares` does not exceed the real `state.total_shares` by more than the legitimate accrual since fees were enabled.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix treats a persisted zero fee anchor as uninitialized in preview and refresh flows. Those paths initialize the anchor to current assets and time rather than minting fee shares against the entire existing supply.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/4b4232f1be2419046a506dd16ddb376171a94054

20.14 SOROBAN HANDLE_DEPOSIT DOES NOT ADVANCE FEE_ANCHOR, DIVERGING FROM THE NEAR IMPLEMENTATION

// MEDIUM

Description

The NEAR vault implementation explicitly advances `fee_anchor.total_assets` and `fee_anchor.timestamp_ns` on every successful deposit, collapsing pending profit into the anchor at deposit time. The Soroban runtime does not.

The kernel's `handle_deposit` updates `state.total_assets`, `state.idle_assets`, and `state.total_shares` but does not touch `state.fee_anchor`. The Soroban-specific `deposit_with_min_impl` wraps `vault.deposit(...)` via `apply_kernel_action(KernelAction::Deposit, ...)` and emits the result, with no Soroban-side anchor update.

The only path that advances the anchor on Soroban is `handle_refresh_fees`, which can only be triggered through `prepare_atomic_call` (called from `atomic_withdraw` / `atomic_redeem`), and those paths are not in `VaultCommand` on the deployed ABI.

Code Location

Copy Code

```
// contract/vault/kernel/src/actions/mod.rs (handle_deposit)
state.total_assets = state.total_assets.checked_add(assets_in)?;
state.idle_assets = state.idle_assets.checked_add(assets_in)?;
state.total_shares = state.total_shares.checked_add(shares_out)?;
// fee_anchor not touched
```

Copy Code

```
// contract/vault/near/src/impl_token_receiver.rs (NEAR equivalent)
self.fee_anchor.total_assets = U128(result.state.total_assets);
self.fee_anchor.timestamp_ns = env::block_timestamp().into();
```

Impact

This is the architectural root cause of the inflated `proxy_view` bug and amplifies the latent `RefreshFees` reachability issue: if the unreachable refresh path is ever exposed in `VaultCommand` without coordinated changes to the deposit flow, the first refresh against a long-lived vault mints performance-fee shares equal to the entire current supply.

It also creates a cross-chain semantic drift: Soroban and NEAR vaults under the same governance config produce different shareholder accounting, which is a problem for any shared off-chain integration that quotes share prices from one chain and acts on the other.

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:H/I:H/D:P/Y:N (6.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Mirror the NEAR pattern in the Soroban runtime. Either update `state.fee_anchor` inside `apply_kernel_action` after a successful `KernelAction::Deposit` (mirroring the NEAR logic), or add the anchor update directly to `handle_deposit` so both chains share the same code path.
- Add a parity test that initializes a vault, performs a deposit on Soroban, and asserts that the on-chain `fee_anchor` matches the state values that NEAR would have written.
- Document the anchor lifecycle in `STRIDE.md` so the `fee_anchor` is treated as a first-class invariant rather than an implicit RefreshFees-only side effect.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix advances the kernel fee anchor during Soroban deposits to the post-deposit total-assets value at the deposit timestamp. This matches the NEAR anchor-collapse behavior and removes deferred profit carry-over from deposits.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/2c7def22ae2daca3197d16a2112f581cc78e6732

20.15 PERMISSIONLESS EXTENDTTL DOES NOT REFRESH PERSISTED ADDRESS-BOOK MAPPINGS REQUIRED BY FUTURE EFFECTS

// MEDIUM

Description

The Soroban runtime stores kernel-address-to-Soroban-address mappings under composite persistent keys `(SK_ADDRBOOK, BytesN<32>)`. Each entry's TTL is only refreshed when `save_address(...)` is called for that specific entry. The permissionless `VaultCommand::ExtendTtl` path (and the implicit `extend_storage_ttl` called from `load_vault_bootstrap`) refreshes only the instance storage and a fixed list of symbol-keyed persistent blobs (`StateBlob`, `PolicyLocks`, `PolicySupplyQueue`, `PolicyMarkets`, `PolicyPrincipals`, `PolicyCapGroups`, `Restrictions`). The address-book composite keys are not in this list and cannot be enumerated from it.
Effect execution depends on these mappings. `ensure_mapped(addr)` first checks the in-memory map, then falls back to `storage.load_address(addr)`. If the address-book entry has expired, the fallback returns `None` and the effect aborts with `RuntimeError::effect_failed("missing address mapping")`.

Code Location

Copy Code

```
// contract/vault/soroban/src/storage/mod.rs (extend_ttl)
pub fn extend_ttl(&self, threshold: u32, extend_to: u32) {
    self.env.storage().instance().extend_ttl(threshold, extend_to);
    let p = self.env.storage().persistent();
    for key in &[
        SorobanStorageKey::StateBlob,
        SorobanStorageKey::PolicyLocks,
        SorobanStorageKey::PolicySupplyQueue,
        SorobanStorageKey::PolicyMarkets,
        SorobanStorageKey::PolicyPrincipals,
        SorobanStorageKey::PolicyCapGroups,
        SorobanStorageKey::Restrictions,
    ] {
        if p.has(key) {
            p.extend_ttl(key, threshold, extend_to);
        }
    }
}
```

Copy Code

```
// contract/vault/soroban/src/contract/curator_vault.rs (ensure_mapped)
fn ensure_mapped(&mut self, addr: &Address) -> Result<(), RuntimeError> {
    if self.interpreter.has_address(addr) { return Ok(()); }
    if let Some(soroban_addr) = self.storage.load_address(addr)? {
        self.interpreter.register_address(*addr, soroban_addr);
        return Ok(());
    }
    Err(RuntimeError::effect_failed("missing address mapping"))
}
```

Impact

Concrete failure scenarios:

- A long-lived fee recipient is configured once via `apply_fee_change`. If fee recipient address mappings expire while the main vault state is kept alive, any future reachable `RefreshFees` path will fail when it tries to mint fee shares to the recipient.
- A queued withdrawal whose receiver / owner mapping goes cold for longer than the address mapping TTL can fail later payout execution, even though the withdrawal queue and vault state were kept alive by routine `ExtendTtl` calls.
- Operators may believe `ExtendTtl` protects all data required by the vault, while it actually omits the composite address-book key space.

Future share mint/burn/transfer or asset transfer effects can become non-executable due only to missing address mappings. The failure is difficult to diagnose externally because the visible symptom is an effect failure, not a storage-TTL warning.

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:H/I:H/D:P/Y:N (6.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Store address-book keys in an enumerable structure (for example, a persistent vector or set of kernel address keys) and extend every known address-book entry inside `SorobanStorage::extend_ttl`.
- Alternatively, avoid storing reverse mappings as expirable independent keys: embed the necessary SDK addresses in durable state records that are already covered by the main TTL extension path.
- Add a targeted test that saves an address mapping, simulates only the core `ExtendTtl` path, and asserts that address mappings receive equivalent TTL maintenance.
- Add operational documentation warning that `ExtendTtl` does not currently refresh address-book entries until the code is fixed.

Remediation Comment

SOLVED: The client added TTL extension for withdrawal-related address-book mappings (withdraw-queue owner/receiver and active Withdrawing/Payout addresses) and for config addresses, which closes the withdrawal-effect denial-of-service vector.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/44401156a682c0ae7f4ff781c055ce16a684906e

20.16 STATE BLOBS HAVE NO SCHEMA VERSION AND MIGRATE() DOES NOT MIGRATE STATE OR POLICY BLOBS

// MEDIUM

Description

The Soroban runtime serializes `VaultState`, `OpState`, `WithdrawQueue`, and the policy sub-blobs without a leading version byte or schema discriminator. `decode_state_blob` decodes exactly the current layout and rejects trailing bytes, but it does not recognize any prior or future schema. `migrate()` only migrates the legacy pause flag via `migrate_legacy_paused` and extends TTL; it does not validate or rewrite the state or policy sub-blobs.

Code Location

Copy Code

```
// contract/vault/soroban/src/storage/mod.rs (encode_state_blob)
pub(crate) fn encode_state_blob(state: &VaultState) -> Vec<u8> {
    let mut out = Vec::new();
    push_u128(&mut out, state.total_assets);
    push_u128(&mut out, state.total_shares);
    push_u128(&mut out, state.idle_assets);
    push_u128(&mut out, state.external_assets);
    push_u128(&mut out, state.fee_anchor.total_assets);
    push_u64(&mut out, state.fee_anchor.timestamp_ns.as_u64());
    encode_op_state(&state.op_state, &mut out);
    encode_withdraw_queue(&state.withdraw_queue, &mut out);
    push_u64(&mut out, state.next_op_id);
    out
}
```

Copy Code

```
// contract/vault/soroban/src/contract/entrypoints.rs (migrate)
pub fn migrate(env: Env, operator: Address) -> Result<(), ContractError> {
    require_governance(&env, &operator)?;
    if !migration_in_progress(&env) {
        return Err(ContractError::InvalidState);
    }
    migrate_legacy_paused(&env);
    extend_storage_ttl(&env);
    set_migration_in_progress(&env, false);
    emit_admin_event(&env, symbol_short!("migrate"));
    Ok(())
}
```

Impact

Failure modes for any future schema-changing upgrade:

- **Append-only change:** a new decoder expecting extra fields will reject old blobs as truncated; an old decoder will reject new blobs as trailing bytes. Either case can brick every normal vault call after upgrade because `load_state()` fails.
- **Same-length reinterpretation:** changing field order or meaning without a version byte can decode successfully but reinterpret old bytes under new semantics. This is worse than a hard failure because accounting may continue with corrupted meaning.

- **Thin migrate path:** the current `migrate()` clears the migration flag without validating or rewriting any state or policy blobs. A schema-changing WASM could mark migration complete while leaving old-format storage behind.

The migration flag gives operators the impression that an upgrade migration has run, but today it only handles the legacy pause key. Without an on-chain version, all schema compatibility assumptions are off-chain process discipline.

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:H/I:H/D:P/Y:N (6.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Prefix every persisted blob with an explicit schema version byte or versioned enum tag.
- Extend `migrate()` to dispatch by stored version and rewrite the state and policy blobs to the latest format before clearing the migration flag.
- Add a post-migration validation step that loads every persisted blob through the new decoders and checks key invariants before `migration_in_progress` is cleared.
- Add regression tests with serialized old-version fixtures to ensure upgrades can decode and migrate previous layouts.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix makes persisted storage format explicit with compact TVS version headers for state, restrictions, and policy sub-blobs. Decoders reject pre-header bytes and unsupported versions through storage-kind/version dispatch, and governed migrate validates current-version storage blobs and fees before clearing the migration flag.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/fcab9d3c034b3fc312c3f10cbe9f9901868ff266

20.17 STATE BLOB SIZE CAP RESTRICTS EFFECTIVE MAX_PENDING TO 580 PENDING WITHDRAWALS ON SOROBAN

// MEDIUM

Description

The kernel advertises `MAX_PENDING = 1024`, and the Soroban runtime constructs the kernel config with `max_pending_withdrawals = MAX_PENDING as u32`. However, the Soroban runtime serializes the entire `VaultState` — including the full `withdraw_queue` — into a single `StateBlob` persistent storage entry. Stellar's network configuration sets `MAX_CONTRACT_DATA_ENTRY_SIZE_BYTES = 64 KiB`. The vault has no awareness of this limit and no defensive guard.

Code Location

Copy Code

```
// contract/vault/kernel/src/state/vault/mod.rs
pub const MAX_PENDING: usize = 1024;
```

Copy Code

```
// contract/vault/soroban/src/storage/mod.rs (encode_withdraw_queue)
push_u64(out, queue.next_withdraw_to_execute);
push_u64(out, queue.next_pending_withdrawal_id);
let entries: Vec<_> = queue.iter().collect();
push_u32(out, entries.len() as u32);
for (id, withdrawal) in entries {
    push_u64(out, id);                                  // 8 bytes
    push_address(out, &withdrawal.owner);               // 32 bytes
    push_address(out, &withdrawal.receiver);            // 32 bytes
    push_u128(out, withdrawal.escrow_shares);           // 16 bytes
    push_u128(out, withdrawal.expected_assets);         // 16 bytes
    push_u64(out, withdrawal.requested_at_ns.as_u64()); // 8 bytes
    // total = 112 bytes per entry
}
```

Doing the math:

Copy Code

```
blob_size_bytes ≈ fixed_state(96) + queue_header(20) + N * per_entry(112)
N = 1024 → ≈ 96 + 20 + 1024 * 112 = 114,804 bytes ≈ 112 KiB
N = 580  → ≈ 96 + 20 + 580 * 112  =  65,076 bytes ≈ 63.5 KiB (just under 64 KiB)
N = 581  → ≈ 96 + 20 + 581 * 112  =  65,188 bytes (over 64 KiB)
```

Impact

- Any `request_withdraw` that would push the queue above ~580 entries causes `save_state` to trap. The transaction reverts and the user observes an opaque host-level failure instead of a typed `MaxPending` error.
- A targeted DoS: an actor with enough share capital to fund ~580 minimum-size withdrawals (≈ `580 × MIN_WITHDRAWAL_ASSETS = 580,000` assets-worth of shares) can fill the queue to the size cap. New legitimate `request_withdraw` calls then fail until the queue drains. Because withdrawals can only be drained by the allocator-class `execute_withdraw` path (one per call, subject to cooldowns), the block window can be hours-to-days long.

- The size cap can be crossed silently if Soroban network parameters change or per-entry encoded size grows (e.g., by adding a new field to `PendingWithdrawal`).

The kernel's advertised 1024-entry queue cannot be reached on Soroban. Operators and integrators who count on it will discover the real cap only when the queue fills.

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:H/I:H/D:P/Y:N (6.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Lower `MAX_PENDING` (or expose a deployment-time `max_pending_withdrawals` config) to a value safely below the Soroban size limit, accounting for fixed state overhead and any expected schema growth. A safe value with the current schema is approximately 512.
- Or: split the withdraw queue out of the monolithic state blob into its own persistent storage entry (or paged entries) so the queue is not coupled to the rest of the state in size.
- Add a runtime-side guard in `request_withdraw_impl` that rejects with a typed error before the queue grows large enough to trap on save.
- Add a regression test that fills the queue past the size limit and asserts a typed error rather than a host-level trap.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix replaces one large withdrawal queue state entry with domain-paged Soroban withdrawal storage. State now persists as a compact v2 header plus bounded 128-entry wqpage entries, preserving kernel MAX_PENDING = 1024 while validating pending count, page continuity, and IDs before trusting reconstructed queues.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/fee307b82177c0beeaccf751c5f785c6c9cffec8

20.18 ADMIN-ONLY BLEND ADAPTER SHORTCUTS CAN DESYNCHRONIZE VAULT ACCOUNTING

// MEDIUM

Description

The Blend adapter exposes admin-only manual supply and withdraw paths that operate directly on the Blend position. These paths bypass the vault's allocation lifecycle and therefore do not update vault-side market principal or external-asset accounting in the same flow.
The issue is not that the functions are unauthenticated; they require adapter-admin authority. The risk is that an admin action can change the adapter's real external position while the vault's accounting remains unchanged until a later refresh or reconciliation step.

Code Location

Copy Code

```
// contract/vault/soroban/blend-adapter/src/lib.rs (manual admin supply)
pub fn supply(env: Env, caller: Address, asset: Address, amount: i128) -> Result<(), AdapterError> {
    extend_instance_ttl(&env);
    require_admin(&env, &caller)?;
    if amount <= 0 {
        return Err(AdapterError::InvalidInput);
    }
    // Supplies through the adapter without a vault-side kernel action.
}
```

Copy Code

```
// contract/vault/soroban/blend-adapter/src/lib.rs (manual admin withdrawal)
pub fn withdraw(env: Env, caller: Address, asset: Address, amount: i128) -> Result<i128, AdapterError> {
    extend_instance_ttl(&env);
    require_admin(&env, &caller)?;
    if amount <= 0 {
        return Err(AdapterError::InvalidInput);
    }
    // Withdraws from Blend without synchronizing vault market accounting.
}
```

Impact

An admin mistake or compromised adapter-admin key can desynchronize the real Blend position from vault accounting. During the drift window, share-price previews, allocation decisions, and withdrawal planning can be based on stale external-asset values.

Proof of Concept

Preconditions

- The adapter is deployed and connected to a Blend pool.
- The adapter admin can call manual methods.
- The vault does not execute a matching allocation state transition in the same transaction.

Attack / Defect Sequence

1. **Seed an adapter position.** Assets are supplied to Blend through the adapter.
2. **Call an admin-only manual path.** The adapter admin directly calls `withdraw` or `supply`.
3. **Observe external position movement.** The adapter/pool balance changes immediately.
4. **Vault accounting is not updated in the same flow.** No vault kernel action adjusts market principal at the time of the manual adapter call.

Observable Result

The adapter's `total_assets` changes while vault-side market accounting remains unchanged until a later refresh. This proves the manual path can create an accounting drift window.

Reproducing Locally

```
cargo test -p templar-soroban-blend-adapter --test withdraw_to_vault_and_supply_balance -- --nocapture
```

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:H/I:H/D:N/Y:N (6.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Manual adapter supply and withdraw shortcuts should be removed or restricted to emergency recovery only.
- If retained, they should also trigger a vault-side accounting refresh in the same transaction.
- Structured events should be emitted so off-chain monitors can reconcile external position changes.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #431 removes the Blend adapter supply_balance and withdraw_to_vault admin entrypoints so managed Blend positions can only be mutated through the vault-routed adapter lifecycle. Regression coverage includes a source-shape regression preventing reintroduction of the public admin shortcut entrypoints, the full templar-soroban-blend-adapter test suite, and the Soroban size-budget gate.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/e1d1ce9dba3fbdad0910c98ddd2323426db31e97

20.19 BLEND ADAPTER ADMIN CAN RETARGET THE VAULT RECEIVER OR POOL WITHOUT TIMELOCK OR EVENT

// MEDIUM

Description

The Blend adapter admin can update the stored vault receiver and Blend pool addresses in a single transaction. These fields define which vault is authorized to use the adapter and which pool receives future supplies or withdrawals.

The setters require admin authorization and basic contract-address validation, but they do not use a timelock, a two-step acceptance process, or an in-flight-position safety check. The change is effective immediately after the storage write.

Code Location

Copy Code

```
// contract/vault/soroban/blend-adapter/src/lib.rs (set_pool)
pub fn set_pool(env: Env, caller: Address, pool: Address) -> Result<(), AdapterError> {
    extend_instance_ttl(&env);
    require_admin(&env, &caller)?;
    require_contract_address(&pool, AdapterError::InvalidInput)?;
    let old_pool = get_pool(&env)?;
    env.storage().instance().set(&DataKey::Pool, &pool);
    PoolUpdated { old_pool, new_pool: pool }.publish(&env);
    Ok(())
}
```

Copy Code

```
// contract/vault/soroban/blend-adapter/src/lib.rs (set_vault)
pub fn set_vault(env: Env, caller: Address, vault: Address) -> Result<(), AdapterError> {
    extend_instance_ttl(&env);
    require_admin(&env, &caller)?;
    require_contract_address(&vault, AdapterError::InvalidInput)?;
    let old_vault = get_vault(&env)?;
    env.storage().instance().set(&DataKey::Vault, &vault);
    VaultUpdated { old_vault, new_vault: vault }.publish(&env);
    Ok(())
}
```

Impact

A compromised adapter admin can immediately invalidate the old vault as the authorized caller or redirect future supplies to a different pool. Existing positions in the previous pool may remain stranded while new operations use the new pool, creating accounting and recovery complexity.

Proof of Concept

Preconditions

- The Blend adapter is deployed with an admin, vault, and pool.
- The adapter admin key is compromised or submits an unintended update.

Attack / Defect Sequence

1. **Retarget the vault.** Admin calls `set_vault(new_vault)`.
2. **Old vault loses authorization.** The old vault attempts `supply` and receives `AdapterError::Unauthorized`.
3. **Retarget the pool.** Admin calls `set_pool(new_pool)`, making future operations use a different Blend pool immediately.

Observable Result

The adapter's `vault()` and `pool()` accessors return the new addresses immediately. A call from the old vault fails authorization.

Reproducing Locally

Copy Code

```
cargo test -p templar-soroban-blend-adapter --test pool_or_vault_retargeting -- --nocapture
```

PoC excerpt:

Copy Code

```
#[test]
fn d8_supply_from_old_vault_after_retarget_returns_unauthorized() {
    let env = Env::default();
    let (adapter, admin, old_vault, _pool) = setup(&env);
    let new_vault = Address::generate(&env);
    let asset = Address::generate(&env);

    env.as_contract(&adapter, || {
        BlendAdapterContract::set_vault(env.clone(), admin, new_vault).unwrap();
        let result = BlendAdapterContract::supply(env.clone(), old_vault, asset, 100);
        assert_eq!(result, Err(AdapterError::Unauthorized));
    });
}
```

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:H/I:H/D:N/Y:N (6.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- The adapter should use a two-step rotation flow for vault and pool changes.
- Each proposed and accepted rotation should emit a structured event.
- Rotations should be blocked while allocations are in flight or while the old pool still contains an unreconciled position.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #431 removes the undeployed Blend adapter set_pool and set_vault admin retarget entrypoints, deletes the stale retarget event surface, and leaves constructor/init as the only pool/vault binding surface. Regression coverage includes a source-shape

regression for removed retarget entrypoints/events, a tooling sweep for removed justfile recipes, the full templar-soroban-blend-adapter test suite, and the Soroban size-budget gate.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/23878c907cf906800ffc9660fe955f203e8040c6

20.20 GOVERNANCE PENDING QUEUE IS STORED AS A SINGLE SIZE-CAPPED INSTANCE ENTRY

// MEDIUM

Description

The governance contract stores the entire pending proposal queue as a single instance-storage value. Each proposal appends additional serialized data to the same logical entry, so queue growth is constrained by Soroban's per-entry storage limit rather than by an explicit protocol-level maximum.
The issue is not that 200 proposals immediately break the contract; the dynamic test shows that hundreds of distinct proposals are accepted and the serialized queue grows linearly. The risk is that there is no clean, early protocol bound that prevents the queue from approaching the ledger entry ceiling. Once the queue becomes too large, future queue writes can fail and recovery itself requires additional writes to the same oversized structure.

Code Location

Copy Code

```
// contract/vault/soroban/governance/src/lib.rs (queue persistence)
// PendingActions is loaded, mutated, and saved back as one storage value.
let mut queue = load_queue(&env);
queue.submit(...);
save_queue(&env, &queue);
```

Copy Code

```
// contract/vault/soroban/governance/tests/adv_queue_dos.rs
// Distinct Other(symbol, hash) pairs occupy separate pending entries.
```

Impact

A large pending queue can turn governance into a fragile single-entry write path. Future submissions, replacements, or revocations may fail once the serialized queue approaches the platform entry-size ceiling. Recovery is also operationally awkward: `revoke_kind` only clears one action kind at a time and requires authorized governance. If the same governance authority is compromised or unavailable, the queue can remain bloated.

Proof of Concept

Preconditions

- The governance contract is deployed.
- An authorized admin can submit many distinct pending proposals.
- The proposal keys are distinct, so new submissions do not replace prior entries.

Attack / Defect Sequence

1. **Submit distinct proposals.** The PoC submits 200 unique `Other` proposals using different hashes.

2. **Observe no clean bound.** Each proposal is accepted and remains pending.
3. **Measure queue growth.** `pending_ids()` returns all 200 proposal IDs.
4. **Test recovery shape.** A mixed queue requires revocation per kind, not one global cleanup operation.

Observable Result

```
[ADV queue_dos final_queue_len=200]
[ADV queue_dos mixed_queue_len_before=55]
[ADV queue_dos revoke_kind=Other removed=50 queue_len_after_other=5]
```

Reproducing Locally

```
cargo test -p templar-soroban-governance --test adv_queue_dos -- --nocapture
```

PoC excerpt:

```
#[test]
fn adv_d10_queue_dos_grows_to_two_hundred_entries() {
    let fixture = setup_fixture_default();
    let depth = 200usize;

    for i in 0..depth {
        let result = fixture.env.as_contract(&fixture.governance, || {
            SorobanVaultGovernanceContract::submit_other(
                fixture.env.clone(),
                fixture.admin.clone(),
                Symbol::new(&fixture.env, "d10dos"),
                BytesN::from_array(&fixture.env, &counter_to_hash(i)),
            )
        });
        assert!(result.is_ok(), "queue rejected at index {i}: {result:?}");
    }

    let queue_len = fixture.env.as_contract(&fixture.governance, || {
        SorobanVaultGovernanceContract::pending_ids(fixture.env.clone())
    }).len();

    assert_eq!(queue_len as usize, depth);
}
```

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:H/I:H/D:P/Y:N (6.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- The pending proposal queue should be split into paged or per-proposal storage entries.
- Every queue write should validate the encoded size before persisting.
- A protocol-level maximum number of pending proposals should be enforced and tested near the ledger entry-size boundary.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #437 deletes the monolithic DataKey::PendingQueue fallback and stores pending governance proposals only in persistent paged storage. The remaining MAX_PENDING_PROPOSALS bound is explicit protocol backstop behavior, with replacements and immediate proposals handled separately.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/8f8b07586e29b6c65ba0e3a7e46c79a74a587ab5

20.21 GOVERNANCE CONFIGURATION AND SKIM COMMANDS BYPASS THE MIGRATION FREEZE

// MEDIUM

Description

The vault's migration flag is intended to freeze state-mutating operations while an upgrade or migration is in progress. Several direct governance/configuration paths do not enforce the same migration guard that protects the normal vault bootstrap path.

`load_vault_bootstrap` rejects calls while `migration_in_progress` is true, but `set_governance_config_impl` only checks that the caller is the stored governance address and then writes privileged configuration. This creates a mismatch between the advertised migration freeze and the actual command surface.

During the upgrade window, governance can still rotate critical authorities such as curator, governance, sentinel, guardians, allowed adapters, skim recipient, and virtual offsets. A compromised governance key can use this window to mutate the recovery control plane while operators expect all state mutation to be blocked.

Code Location

Copy Code

```
// contract/vault/soroban/src/contract/helpers.rs (load_vault_bootstrap)
if migration_in_progress(env) {
    return Err(RuntimeError::invalid_state(
        "migration in progress - call migrate() first",
    ));
}
```

Copy Code

```
// contract/vault/soroban/src/contract/entrypoints.rs (set_governance_config_impl)
fn set_governance_config_impl(...) -> Result<(), ContractError> {
    require_governance(env, &caller)?;
    match kind {
        GOVERNANCE_CONFIG_KIND_CURATOR => apply_curator_config(env, required_address(primary)?),
        GOVERNANCE_CONFIG_KIND_GOVERNANCE => {
            apply_governance_config(env, required_address(primary)?)?
        }
        GOVERNANCE_CONFIG_KIND_SKIM_RECIPIENT => {
            apply_skim_recipient_config(env, required_address(primary)?)
        }
        GOVERNANCE_CONFIG_KIND_VIRTUAL_OFFSETS => {
            apply_virtual_offsets_config(env, required_i128(value_a)?, required_i128(value_b)?)?
        }
        _ => return Err(ContractError::InvalidInput),
    }
    Ok(())
}
```

Impact

A compromised or mistaken governance caller can mutate critical configuration during migration, undermining the freeze invariant that operators rely on during upgrades. In the demonstrated chain,

governance rotates curator and governance while the migration flag is set, creating persistent lockout risk because the old governance can no longer clear or recover the migration state.

Proof of Concept

Preconditions

- The vault has entered migration mode after `upgrade` or equivalent migration setup.
- The governance key is compromised, or governance submits an unintended direct configuration command.

Attack / Defect Sequence

1. **Migration starts.** The migration flag is set, and operators expect normal state-changing commands to be frozen.
2. **Attacker submits a direct config write.** The attacker calls `SetGovernanceConfig(CURATOR)` or `SetGovernanceConfig(GOVERNANCE)`.
3. **The migration guard is bypassed.** The command reaches `set_governance_config_impl`, which checks governance authorization but does not call `migration_in_progress`.
4. **Authority is rotated during the freeze.** The stored curator or governance address changes while the vault is still marked as migrating.

Observable Result

After the command, `proxy_view` returns the attacker-controlled curator/governance even though the migration flag remains set. The old governance can no longer issue privileged recovery commands.

Reproducing Locally

The runnable PoC is in `contract/vault/soroban/tests/adv_governance_compromise_chain.rs`. Run with:

Copy Code

```
cargo test -p templar-soroban-runtime --features testutils --test adv_governance_compromise_chain -- --n
```

Stdout markers:

Copy Code

```
[FINDING ADV-GOVCHAIN-MIGRATION attacker rotated curator DURING migration]
[FINDING ADV-GOVCHAIN-LOCKOUT vault is now persistently hijacked]
```

PoC excerpt:

Copy Code

```
fixture.env.as_contract(&fixture.contract_id, || {
    fixture.env.storage().instance().set(&symbol_short!("migrate"), &true);
});

let attacker_curator = SdkAddress::generate(&fixture.env);
let result = run(
    &fixture,
    VaultCommand::SetGovernanceConfig {
        caller: address_text(&fixture.governance),
```

```
        kind: GOVERNANCE_CONFIG_KIND_CURATOR,
        primary: Some(address_text(&attacker_curator)),
        many: None,
        value_a: None,
        value_b: None,
    },
);
assert!(result.is_ok());

let view = fixture.env.as_contract(&fixture.contract_id, || {
    SorobanVaultContract::proxy_view(
        fixture.env.clone(),
        SdkAddress::generate(&fixture.env),
        0,
        0,
    )
}).unwrap();
let (curator_view, _gov_view, _asset_view, _share_view) = view.0 .0;
assert_eq!(curator_view, attacker_curator);
```

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:H/I:H/D:N/Y:N (6.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- `require_not_migrating` should be enforced at the start of all state-mutating governance, fee, and skim paths.
- The freeze invariant should be documented as a single allowlist of commands that remain callable during migration.
- Regression tests should assert that each mutating command is rejected while the migration flag is set.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix adds a centralized require_no_migration guard at the execute_governance_command boundary. Governance SetGovernanceConfig and Skim commands now return typed MigrationNotAllowed while migration is active.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/21ac8b5c43a0e3d59fae6f424398f935d20d7fd4

20.22 GUARDIAN AND SENTINEL REVOKERS CAN CANCEL UNRELATED GOVERNANCE PROPOSALS

// MEDIUM

Description

The governance contract allows configured revokers to cancel pending proposals across action kinds. In practice, guardian and sentinel style emergency roles can revoke ordinary administrative proposals, not only emergency or safety-sensitive proposals.

The revoke functions validate that the caller is a revoker, then remove the target proposal or every proposal of the requested kind. They do not scope guardian/sentinel revocation rights to a narrow emergency action set.

Code Location

Copy Code

```
// contract/vault/soroban/governance/src/lib.rs
pub fn revoke(env: Env, caller: Address, proposal_id: u64) -> Result<(), GovernanceError> {
    extend_instance_ttl(&env);
    require_revoker(&env, &caller)?;
    let mut queue = load_queue(&env);
    if queue.revoke_by_key(&ProposalKey::ProposalId(proposal_id), queued_proposal_key_by_id).is_empty()
        return Err(GovernanceError::ProposalNotFound);
    }
    save_queue(&env, &queue);
    ProposalRevoked { id: proposal_id }.publish(&env);
    Ok(())
}
```

Copy Code

```
// contract/vault/soroban/governance/src/lib.rs
pub fn revoke_kind(env: Env, caller: Address, kind: GovernanceActionKind) -> Result<u32, GovernanceError
    extend_instance_ttl(&env);
    require_revoker(&env, &caller)?;
    let removed = revoke_where(&env, |action| action_kind(action) == kind);
    if removed == 0 {
        return Err(GovernanceError::ProposalNotFound);
    }
    Ok(removed)
}
```

Impact

A compromised or misconfigured emergency role can suppress normal governance activity. For example, cap changes, fee updates, or market configuration proposals can be cancelled even though they are unrelated to pause, emergency response, or malicious proposal cleanup.

Proof of Concept

Preconditions

- At least one guardian or sentinel/revoker address is configured.

- A pending non-emergency proposal exists.

Attack / Defect Sequence

1. **Admin submits an ordinary proposal.** For example, a cap change is pending.
2. **Emergency role revokes it.** A guardian/sentinel calls `revoke` or `revoke_kind`.
3. **The proposal is removed.** The proposal is cancelled even though it is not an emergency proposal.

Observable Result

The proposal id disappears from `pending_ids()`, and `ProposalRevoked` is emitted.

Reproducing Locally

Copy Code

```
cargo test -p templar-soroban-governance --test revoker_matrix -- --nocapture
```

PoC structure:

Copy Code

```
let proposal_id = submit_non_emergency_cap_update(&fixture);

fixture.env.as_contract(&fixture.governance, || {
    SorobanVaultGovernanceContract::revoke(
        fixture.env.clone(),
        fixture.guardian.clone(),
        proposal_id,
    ).unwrap();
});

let pending = fixture.env.as_contract(&fixture.governance, || {
    SorobanVaultGovernanceContract::pending_ids(fixture.env.clone())
});
assert!(!pending.contains(proposal_id));
```

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:H/I:H/D:P/Y:N (6.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Revoker authority should be scoped by action kind so emergency roles can revoke only emergency-sensitive proposals.
- Non-emergency revocation should require admin authorization or a threshold of independent roles.
- Revocation events should include the role and reason category used for cancellation.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #437 removes the Guardian role/config surface and replaces broad revocation with explicit revoker_role and require_revoke_kind checks. Proposal-id revocation derives the stored action kind, kind/key revocation checks the requested kind, admin can revoke all, and Sentinel can cancel only operational-risk proposal kinds.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/8f8b07586e29b6c65ba0e3a7e46c79a74a587ab5

20.23 VIRTUAL CONVERSION OFFSETS CAN BE CHANGED AFTER THE VAULT IS CAPITALIZED

// MEDIUM

Description

The vault allows governance to update virtual share and virtual asset offsets after the vault has already been capitalized. These offsets are not passive metadata; they are inputs to the conversion formulas that determine how many shares a depositor receives and how many assets a share holder can redeem. Changing the offsets after deposits exist can therefore change the apparent share price without any corresponding yield, loss, or asset movement. The change is accepted through a direct governance configuration path and is not constrained by a post-capitalization immutability rule or a maximum implied price movement.

Code Location

Copy Code

```
// contract/vault/soroban/src/contract/entrypoints.rs
fn apply_virtual_offsets_config(
    env: &Env,
    virtual_shares: i128,
    virtual_assets: i128,
) -> Result<(), ContractError> {
    if virtual_shares < 0 || virtual_assets < 0 {
        return Err(ContractError::InvalidInput);
    }
    store_virtual_offsets(env, virtual_shares as u128, virtual_assets as u128);
    emit_admin_event(env, symbol_short!("s_voffs"));
    Ok(())
}
```

Impact

An authorized governance action can reprice future deposits and withdrawals after users have already entered the vault. Existing holders can be diluted or advantaged relative to new users solely due to a configuration write, and off-chain integrations may interpret the price movement as real yield or loss.

Proof of Concept

Preconditions

- The vault is initialized and has non-zero total assets and shares.
- Governance can call `SetGovernanceConfig(VIRTUAL_OFFSETS)`.

Attack / Defect Sequence

1. **Deposit initial capital.** The vault has real users and non-zero share supply.
2. **Record conversion output.** A fixed asset amount maps to a specific share amount.
3. **Change virtual offsets.** Governance updates the virtual asset/share offsets.

4. **Recompute conversion output.** The same asset amount maps to a different share amount without asset movement.

Observable Result

`convert_to_shares(assets)` or a deposit preview changes solely because governance changed offsets after capitalization.

PoC Code

```
let shares_before = preview_deposit(assets_in);

execute_set_governance_config(
    GOVERNANCE_CONFIG_KIND_VIRTUAL_OFFSETS,
    Some(new_virtual_shares),
    Some(new_virtual_assets),
);

let shares_after = preview_deposit(assets_in);
assert_ne!(shares_before, shares_after);
```

Copy Code

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:M/I:H/D:N/Y:N (5.9)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Virtual offsets should be immutable after the first deposit or after a short bootstrap window.
- If mutability is required, changes should be timelocked and bounded by a maximum implied price movement.
- Every offset change should emit an event including the old values, new values, and implied price delta.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix rejects virtual-offset governance changes after capitalization and persists a first-deposit lock. Later zero-state unwind cannot re-enable virtual offset mutation.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/d31bf3c911f2e1deb417d0024571892f49e72f19

20.24 TIMELOCKCONFIG CAN BE RAISED TO U64::MAX IMMEDIATELY, BLOCKING FUTURE REDUCTIONS

// MEDIUM

Description

The governance contract accepts `u64::MAX` as a valid timelock value. Raising `TimelockConfig` to that value is treated as a valid timelock configuration update, but subsequent attempts to reduce it inherit a practically infinite delay.
The bounds check is ineffective because the configured maximum is the maximum representable integer. This makes `TimelockOutOfBounds` unreachable for high values and allows governance to lock its own timelock-management path for any realistic operational timeframe.

Code Location

Copy Code

```
// contract/vault/soroban/governance/src/lib.rs
const MIN_TIMELOCK_NS: u64 = 0;
const MAX_TIMELOCK_NS: u64 = u64::MAX;

fn validate_timelock_ns(value: u64) -> Result<(), GovernanceError> {
    if !(MIN_TIMELOCK_NS..=MAX_TIMELOCK_NS).contains(&value) {
        return Err(GovernanceError::TimelockOutOfBounds);
    }
    Ok(())
}
```

Impact

Once `TimelockConfig` is raised to `u64::MAX`, future reductions are effectively impossible through normal governance. This can permanently impair governance recovery, parameter repair, and emergency administration unless an upgrade or out-of-band intervention is performed.

Proof of Concept

Preconditions

- The governance admin can submit timelock updates.
- The current `TimelockConfig` value is below `u64::MAX`.

Attack / Defect Sequence

1. **Raise the governing timelock.** Governance submits `SetTimelock(TimelockConfig, u64::MAX)`.
2. **The update is accepted.** The value passes `validate_timelock_ns`.
3. **Attempt a later reduction.** Any reduction is now subject to the same maximum timelock duration.
4. **Recovery becomes impractical.** The timelock-management path is locked for a practically infinite period.

Observable Result

The configured `TimelockConfig` becomes `u64::MAX`, and a later reduction remains pending behind that maximum delay.

Reproducing Locally

```
cargo test -p templar-soroban-governance --test timelock_bounds -- --nocapture
```

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:H/I:M/D:N/Y:N (5.9)

Recommendation

To mitigate this issue, the following improvements are recommended:

- A realistic maximum timelock should be enforced, for example 30 or 90 days.
- Changes to the timelock governing timelock changes should always require a non-zero delay and additional confirmation.
- Regression tests should cover upper-bound values and future reduction paths.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix replaces the Soroban governance u64::MAX timelock upper bound with a 30-day cap matching NEAR governance. Oversized increases such as u64::MAX are rejected before they can self-lock future reductions.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/6052b1ad252f43d9660690659bca34ffcb877039

20.25 IMMEDIATE `SKIM ACTION WITH DEFAULT SKIM RECIPIENT DRAINS FOREIGN TOKENS IN ONE TRANSACTION

// MEDIUM

Description

The governance contract classifies `GovernanceAction::Skim` as `TimelockDecision::Immediate`: `submit_skim(token)` dispatches to the vault inside the same transaction without any timelock or observation window. The vault's `skim_impl` requires a governance caller, refuses the asset and share tokens, and otherwise transfers the entire balance of the target token to the stored `SkimRecipient`.
At deployment, `SorobanVaultContract::initialize` sets `SkimRecipient = governance` by default. Although `SetSkimRecipient` is timelocked, the timelock provides no observability for the skim itself when the recipient is already the governance contract. The combination allows a governance compromise or a single malicious governance action to drain any non-asset / non-share token sitting on the vault (incentive rewards, fee distributions, airdrops, donations, mis-sent tokens) within one block.

Code Location

Copy Code

```
// contract/vault/soroban/governance/src/lib.rs (decide_submission)
GovernanceAction::Skim(_) => Ok(TimelockDecision::Immediate),
```

Copy Code

```
// contract/vault/soroban/src/contract/entrypoints.rs (initialize)
set_config_address(&env, &VaultDataKey::SkimRecipient, &governance);
```

Copy Code

```
// contract/vault/soroban/src/contract/entrypoints.rs (skim_impl)
require_governance(env, &caller)?;
let asset = get_config_address(env, &VaultDataKey::AssetToken)?;
let share = get_config_address(env, &VaultDataKey::ShareToken)?;
if token == asset || token == share { return Err(ContractError::InvalidInput); }
let recipient = get_config_address(env, &VaultDataKey::SkimRecipient)?;
let client = soroban_sdk::token::Client::new(env, &token);
let balance = client.balance(&env.current_contract_address());
...
client.transfer(&env.current_contract_address(), &recipient, &balance);
```

Impact

Non-asset / non-share tokens at the vault are fully at the mercy of governance. A single compromise of the governance admin or a malicious governance proposal drains every unit of any external token (incentives, airdrops, mis-sends) in one transaction with no observation window. There is no pause/abort path: abdicating the skim symbol freezes legitimate skims as well.

Proof of Concept

Preconditions

- The vault is bootstrapped with the real governance contract installed at `VaultDataKey::Governance` and `SkimRecipient = governance` (the `initialize()` default).
- A non-asset, non-share token is present on the vault (e.g., a reward token *R* with 10,000 units sitting at the vault address).
- The governance admin is either compromised, coerced, or executes a malicious proposal.

Attack Sequence

1. **Admin queues the skim.** The compromised admin calls `SorobanVaultGovernanceContract::submit_skim(R)`. The governance contract's `decide_submission` classifies `Skim` as `TimelockDecision::Immediate` — there is no observation window for this action.
2. **Governance executes inline.** Because the decision is Immediate, the governance `submit` path runs `execute_action` in the same transaction. `execute_action` for `Skim` builds a `VaultCommand::Skim { caller: governance, token: R }` payload and invokes the vault's `execute` entrypoint via `env.invoke_contract`.
3. **Vault validates and transfers.** `skim_impl` verifies `caller == governance`, confirms `token != asset_token` and `token != share_token`, reads `SkimRecipient` from instance storage (which is `governance` by default since `initialize()`), reads the vault's full reward-token balance via `token.balance(self)`, and calls `token.transfer(self, governance, balance)`.
4. **Funds move in one transaction.** All 10,000 units of token *R* move from the vault to the governance contract. No `accept(id)` step is required — in fact, attempting one returns `ProposalNotFound` because the proposal was already executed inline.
5. **Recipient observability is bypassed.** The timelocked `SetSkimRecipient` action is meant to provide visibility into where skimmed tokens go. Because the recipient is governance by default, the attacker never had to call `SetSkimRecipient`; the timelock-protected change of recipient was simply not needed.

Observable Result

After step 4, `token.balance(vault) == 0` and `token.balance(governance) == 10_000` for the targeted reward token. There was zero on-chain delay between the submit action and the actual fund movement.

Reproducing Locally

The runnable PoC is in `contract/vault/soroban/tests/security_pocs.rs` (function `poc_immediate_skim_drains_foreign_token`). Run with:

```
cargo test --test security_pocs poc_immediate_skim_drains_foreign_token -- --nocapture
```

Stdout:

```
vault reward balance after Skim: 0, governance now holds 10000
```

Full PoC body:

```
// Bootstrap vault with the real governance contract as governance address.
// initialize() sets SkimRecipient = governance by default.
let vault = env.register(SorobanVaultContract, ());
let asset = env.register_stellar_asset_contract_v2(Address::generate(&env)).address();
let share = env.register_stellar_asset_contract_v2(vault.clone()).address();
let governance = env.register(SorobanVaultGovernanceContract, (&admin, &vault, &0u64));
env.as_contract(&vault, || {
    SorobanVaultContract::initialize(
        env.clone(), admin.clone(), governance.clone(), asset, share, 0, 0,
    ).unwrap();
});

// A foreign reward token sitting on the vault.
let reward = env.register_stellar_asset_contract_v2(Address::generate(&env)).address();
let reward_admin_client = StellarAssetClient::new(&env, &reward);
reward_admin_client.mint(&vault, &10_000);

let reward_client = TokenClient::new(&env, &reward);
assert_eq!(reward_client.balance(&vault), 10_000);
assert_eq!(reward_client.balance(&governance), 0);

// Submit Skim(reward). `Skim` is `TimelockDecision::Immediate`, so
// governance's `submit` executes `execute_action` inline - no `accept`
// call is needed. (A naive `accept(id)` would actually return
// `ProposalNotFound` because the proposal was already executed.)
let _id = env.as_contract(&governance, || {
    SorobanVaultGovernanceContract::submit_skim(
        env.clone(),
        admin.clone(),
        reward.clone(),
    ).unwrap()
});

// All 10_000 reward tokens have moved from vault to governance in one tx
let vault_balance_after = reward_client.balance(&vault);
let gov_balance_after = reward_client.balance(&governance);
assert_eq!(vault_balance_after, 0,
    "vault fully drained of foreign token");
assert_eq!(gov_balance_after, 10_000,
    "governance contract is the default SkimRecipient and now owns the tokens");
```

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:N/I:H/D:P/Y:N (5.0)

Recommendation

To mitigate this issue, the following improvements are recommended:

- A dedicated `TimelockKind::Skim` should be introduced with its decision set to `Timelocked`, so that skims provide the same observation window as `SetSkimRecipient` and cannot be combined with the default recipient configuration to drain tokens in one transaction.
- Alternatively, the skim recipient should be required to differ from the governance contract at construction time, or governance should be required to call `SetSkimRecipient` at least once before any `Skim` is allowed.
- An explicit allowlist for tokens permitted to be skimmed should be enforced, so that arbitrary third-party tokens cannot be drained without a prior timelocked governance action.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix changes GovernanceAction::Skim from immediate execution to the Skim timelock path, so a skim proposal can no longer drain foreign tokens in the same transaction. The default skim recipient remains explicit configuration, but skim execution now requires proposal maturity before the vault command is routed.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/381a450aad71efdfdd27f026fb2c365c3e67bbc0

20.26 DIRECT UNDERLYING TRANSFERS REMAIN OUTSIDE `TOTAL_ASSETS UNTIL MANUAL RESYNC, ALLOWING PRE-RESYNC DEPOSITS TO CAPTURE THE SURPLUS

// MEDIUM

Description

The Soroban vault prices deposits against the persisted `state.total_assets` rather than the actual on-ledger token balance. Reconciliation of any unaccounted underlying that lands on the vault contract (mis-send, airdrop, reward stream, rescue transfer, or any other off-path credit) only happens through `VaultCommand::ResyncIdleBalance`, which is permissionless but enforces a hard-coded `120_000_000_000 ns` (120-second) cooldown between calls.

During the stale window between the off-path credit and the next successful `ResyncIdleBalance`, deposits compute `shares_out` from `convert_to_shares` over the old, too-small `state.total_assets`. A depositor minting before resync therefore receives more shares per unit than true NAV, and the surplus is captured when NAV is later updated.

The cooldown turns this into an actively widenable race: an unprivileged actor can call `ResyncIdleBalance` first to start a fresh 120-second timer, observe a donation that lands inside the window, deposit against stale NAV, wait for the cooldown to elapse and someone to reconcile, then redeem.

Code Location

Copy Code

```
// contract/vault/soroban/src/contract/entrypoints.rs (resync_idle_balance_impl)
let now_ns = ledger_timestamp_ns(env)?;
let cooldown_ns = 120_000_000_000u64;
let last_ns = env.storage().instance().get(&last_key).unwrap_or(0u64);
if last_ns != 0 && now_ns.saturating_sub(last_ns) < cooldown_ns {
    return Err(ContractError::InvalidState);
}
let actual_balance = to_u128(client.balance(&env.current_contract_address()))?;
...
state.idle_assets = actual_balance;
state.sync_total_assets();
```

Copy Code

```
// contract/vault/kernel/src/actions/mod.rs (handle_deposit)
let shares_out = convert_to_shares(&state, config, assets_in);
...
state.total_assets = state.total_assets.checked_add(assets_in)?;
state.idle_assets = state.idle_assets.checked_add(assets_in)?;
```

Impact

Mis-sent or externally-credited underlying that arrives on the vault is transferred from the intended economic beneficiaries to whoever deposits before the next successful resync. Existing shareholders are diluted whenever unaccounted idle balance exists and a new depositor mints during the stale window. Manual rescue transfers intended to help the vault are economically unsafe until the vault state catches up. This is the strongest external-attacker primitive among the audited issues.

Proof of Concept

Preconditions

- The vault is bootstrapped and at least one honest depositor has minted shares (so `total_shares > 0` and `total_assets > 0`).
- The attacker observes (or expects) some underlying to land on the vault contract through an off-path mechanism: mis-sent tokens, an airdrop or rewards distribution, a rescue transfer, etc. The mechanism is irrelevant; what matters is that `token.balance(vault)` grows without `state.total_assets` growing.
- The attacker has any account funded with the underlying token; no role on the vault is required.

Attack Sequence

1. **Honest baseline.** Alice deposits 1,000 underlying via `VaultCommand::DepositWithMin` and receives 1,000 shares (1:1). The state reads: `total_assets = 1_000`, `total_shares = 1_000`, `idle_assets = 1_000`.
2. **Off-path credit lands on the vault.** 1,000 underlying are transferred directly to the vault contract address (e.g., a mis-send or reward distribution). The token's balance for the vault is now `2_000`, but the vault's internal accounting still reads `total_assets = 1_000`.
3. **Attacker arms the cooldown to widen the stale window.** The attacker calls `VaultCommand::ResyncIdleBalance`. This is permissionless. The runtime reads the actual balance (`2_000`), updates `idle_assets` and `total_assets` to `2_000`, sets `last_resync_ns = now_ns`. Now another `ResyncIdleBalance` cannot run for 120 seconds.
4. **Another off-path credit lands during the locked window.** A few seconds later, an additional 1,000 underlying are transferred to the vault. The token balance is now `3_000`, but `state.total_assets` still reads `2_000`. Any reconciliation attempt by another actor returns `InvalidState` because the cooldown is still active.
5. **Attacker deposits against stale NAV.** The attacker calls `VaultCommand::DepositWithMin { owner: attacker, receiver: attacker, assets: 1_000, min_shares_out: 0 }`. With virtual offsets `0`, `convert_to_shares = assets * total_shares / total_assets = 1_000 * 1_000 / 2_000 = 500`. The attacker receives 500 shares for 1,000 underlying because the third 1,000 of underlying that landed in step 4 is **not** reflected in the denominator.
6. **Cooldown expires; another reconciliation runs.** 121 seconds later, anyone calls `VaultCommand::ResyncIdleBalance`. The runtime reads the actual balance (`4_000`: original 2,000 + step-4 donation of 1,000 + attacker's deposit of 1,000) and writes `total_assets = 4_000`, `idle_assets = 4_000`, while `total_shares` remains `1_500` (Alice's 1,000 + attacker's 500).
7. **Attacker redeems and books a profit.** The attacker's 500 shares are now backed by `500 / 1_500 × 4_000 ≈ 1_333` underlying. They paid 1,000 and walk away with ~1,333 — a 33% profit captured from the unaccounted donation.

Observable Result

The attacker's redeem-equivalent value (`(minted_shares × total_assets) / total_shares`) is strictly greater than the 1,000 they deposited. Existing shareholders (Alice) are correspondingly diluted: their pro-rata claim on the donation surplus is split with the attacker.

Reproducing Locally

The runnable PoC is in `contract/vault/soroban/tests/security_pocs.rs` (function `poc_deposit_before_resync_captures_donation`). Run with:

Copy Code

```
cargo test --test security_pocs poc_deposit_before_resync_captures_donation -- --nocapture
```

Stdout:

Copy Code

```
attacker paid 1_000, minted 500 shares, now worth 1333 (pool total_assets=4000, total_shares=1500)
```

Full attack reproduction:

Copy Code

```
// 1) Honest depositor seeds the vault 1:1
asset_admin_client.mint(&alice, &1_000);
env.as_contract(&vault, || {
    execute_command(&env, &VaultCommand::DepositWithMin {
        owner: address_text(&alice),
        receiver: address_text(&alice),
        assets: 1_000,
        min_shares_out: 0,
    }).unwrap();
});

// 2) External party transfers underlying directly to the vault.
asset_admin_client.mint(&vault, &1_000);
assert_eq!(asset_client.balance(&vault), 2_000);
// State has not been reconciled yet:
assert_eq!(stored_before.total_assets, 1_000);

// 3) Attacker arms the cooldown so the stale window persists.
env.as_contract(&vault, || {
    execute_command(&env, &VaultCommand::ResyncIdleBalance).unwrap();
});
// state.total_assets = 2_000 (actual vault balance). Cooldown is active.

// 4) Attacker (or anyone) sends *another* donation that will sit
//    unreconciled for 120s.
advance_secs(&env, 5);
asset_admin_client.mint(&vault, &1_000);
assert_eq!(asset_client.balance(&vault), 3_000);
assert_eq!(stored_post_resync.total_assets, 2_000); // unchanged

// 5) Attacker deposits 1_000 against the stale NAV (gets ~500 shares)
asset_admin_client.mint(&attacker, &1_000);
let minted = env.as_contract(&vault, || {
    execute_command(&env, &VaultCommand::DepositWithMin {
        owner: address_text(&attacker),
        receiver: address_text(&attacker),
        assets: 1_000,
        min_shares_out: 0,
    }).unwrap()
});
let VaultCommandResult::I128(minted) = minted else { panic!() };

// 6) Wait out the cooldown, someone reconciles
advance_secs(&env, 121);
env.as_contract(&vault, || {
    execute_command(&env, &VaultCommand::ResyncIdleBalance).unwrap();
});

// 7) Measure attacker's PnL
let state = env.as_contract(&vault, || {
    SorobanStorage::new(&env).load_state().unwrap().unwrap()
```

```
});
let attacker_value = (minted as u128) * state.total_assets / state.total_shares;
assert!(attacker_value > 1_000,
    "attacker must extract donation value: paid 1_000, now worth {}",
    attacker_value);
```

BVSS

AO:R/AC:L/AX:N/R:P/S:U/C:N/A:N/I:H/D:P/Y:H (4.7)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Actual idle balance should be reconciled before pricing deposits or any other path that mints shares against `state.total_assets`.
- If auto-reconciliation on every deposit is too expensive, deposits should at minimum be rejected when `token.balance(vault)` materially exceeds the stored `idle_assets`, forcing a resync first.
- The cooldown should be reworked so that a positive-balance discrepancy can be reconciled immediately, or a privileged emergency bypass should be added for stale-balance fixes.
- A regression test should be added that transfers underlying directly into the vault, deposits before resync, and asserts that the depositor cannot capture excess value under the fixed behavior.

Remediation Comment

SOLVED: Client successfully remediated this issue. Soroban preview and deposit paths now reconcile the actual idle underlying-token balance before pricing, so direct underlying transfers cannot be captured by a subsequent pre-resync deposit at a stale NAV. The public resync path uses the same reconciliation rule, keeping total-assets accounting aligned with donated idle balance before share conversion.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/abfd943704d4fc7285401edbafc181707d7a76a0

20.27 PERMISSIONLESS INITIALIZE ALLOWS DEPLOYMENT-RACE GOVERNANCE TAKEOVER IF INITIALIZATION IS NOT ATOMIC

// LOW

Description

The vault's `initialize` function is permissionless and one-shot. If contract deployment and initialization are not performed atomically, any external caller can initialize the vault first using attacker-controlled curator and governance addresses.
After the first successful initialization, the legitimate deployer's later initialization attempt returns `AlreadyInitialized`. The issue is therefore a deployment-race footgun rather than a vulnerability in already-initialized deployments, but the impact under a non-atomic deployment procedure is full governance takeover.

Code Location

Copy Code

```
// contract/vault/soroban/src/contract/entrypoints.rs (initialize)
pub fn initialize(
    env: Env,
    curator: soroban_sdk::Address,
    governance: soroban_sdk::Address,
    asset: soroban_sdk::Address,
    share: soroban_sdk::Address,
    virtual_shares: i128,
    virtual_assets: i128,
) -> Result<(), ContractError> {
    if is_initialized(&env) {
        return Err(ContractError::AlreadyInitialized);
    }
    // stores caller-supplied curator/governance without requiring deployer auth
}
```

Impact

If deploy and initialize are split into separate externally observable transactions, an attacker can become the vault governance and curator. From there, the attacker can change configuration and, as demonstrated by the hardening PoC, redirect skim recipient and drain non-asset/non-share tokens accidentally held by the vault.

Proof of Concept

Preconditions

- The vault contract is deployed but not initialized in the same atomic transaction.
- An external attacker can submit `initialize` before the deployer.

Attack / Defect Sequence

1. **Attacker initializes first.** The attacker passes their own curator and governance addresses.
2. **Attacker ownership is persisted.** `proxy_view` returns the attacker's addresses.
3. **Deployer cannot recover with initialize.** The legitimate call returns `AlreadyInitialized`.
4. **Attacker chains governance control.** The attacker sets themselves as skim recipient and skims a foreign token balance.

Observable Result

Copy Code

```
[FINDING C-027 init_frontrun confirmed: attacker became governance without auth]
[ADV init_frontrun_skim airdrop_to_attacker=10000000000 vault_balance_after_skim=0]
```

Reproducing Locally

Copy Code

```
cargo test -p templar-soroban-runtime --features testutils --test adv_initialize_frontrun -- --nocapture
```

PoC excerpt:

Copy Code

```
let attacker_init = env.as_contract(&contract_id, || {
    SorobanVaultContract::initialize(
        env.clone(),
        attacker_curator.clone(),
        attacker_governance.clone(),
        asset.clone(),
        share.clone(),
        0,
        0,
    )
});
assert_eq!(attacker_init, Ok(()));

let deployer_init = env.as_contract(&contract_id, || {
    SorobanVaultContract::initialize(
        env.clone(),
        legitimate_admin.clone(),
        legitimate_admin.clone(),
        asset.clone(),
        share.clone(),
        0,
        0,
    )
});
assert_eq!(deployer_init, Err(ContractError::AlreadyInitialized));
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:H/I:H/D:P/Y:N (2.1)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Initialization should be performed in the constructor, or the constructor should pre-bind an initialization admin that must authorize `initialize`.
- The deploy procedure should enforce atomic deploy-and-init.

- `initialize` should emit a structured event containing the critical initialized addresses.

Remediation Comment

RISK ACCEPTED: The client accepts this as an operational deployment assumption: deployment and initialization are performed atomically (bundled deploy + initialize), so a non-atomic initialize race is outside the intended deployment model. This is reasonable provided that atomicity guarantee holds, and should be revisited if deployment automation ever exposes a non-atomic initialization window.

20.28 SHARE-CONVERSION HELPERS TRUNCATE 256-BIT QUOTIENTS TO U128

// INFORMATIONAL

Description

The kernel's share-conversion helpers (`convert_to_shares`, `convert_to_assets`, `convert_to_shares_ceil`, `convert_to_assets_ceil`) compute their result in `Number` space and downcast to `u128` via `u128::from(Number)`, which is implemented as `Number::as_u128_trunc()` and drops the upper 128 bits of a 256-bit value rather than returning an error.
This is reachable in principle under the Soroban ABI's bounds. With `total_shares = i128::MAX = 2^127 − 1`, `total_assets = 0`, both virtual offsets `0`, and `assets_in = 2`, the effective ratio computes `2 × 2^127 / 1 = 2^128`; `u128::from(Number)` truncates this to `0`. The deposit therefore succeeds, `total_assets` and `idle_assets` increase by 2, and the depositor receives 0 shares. `min_shares_out = 0` provides no protection because the math wraps silently. The same truncation applies to the ceiling variants used by withdraw preview paths.

How An Attacker Could Exploit This

To exploit this, an attacker would need to deposit underlying into a vault whose effective state is at the boundary `effective_supply × assets_in / effective_assets >= 2^128` — in practice, a vault with `total_shares` close to `i128::MAX` and very small effective assets. Reaching that state in a normally operated vault requires either an extreme prior state or a coordinated chain of operations to drive `total_shares` upward and effective assets near zero. Once that state holds, the attacker simply submits `VaultCommand::DepositWithMin { assets: small, min_shares_out: 0 }`: the silent truncation lets them deposit real underlying for zero shares. The deposited assets are absorbed into `total_assets` and `idle_assets` (becoming a donation that boosts NAV for existing holders), not credited back to the depositor. The attacker bears a loss; the "exploit" is therefore self-harm unless the attacker is targeting an unrelated victim deposit. The same truncation also drives wrong outputs from `proxy_view`, which can confuse off-chain tooling that quotes share prices.
Because reaching the trigger state requires extreme conditions, the practical exploitability of this defect is low; it is recorded here as a numeric-hardening defect that should fail closed rather than wrap.

Code Location

Copy Code

```
// contract/vault/kernel/src/math/number.rs
impl From<Number> for u128 {
    fn from(value: Number) -> Self { value.as_u128_trunc() }
}
```

Copy Code

```
// contract/vault/kernel/src/actions/mod.rs (convert_to_shares)
pub fn convert_to_shares(state: &VaultState, config: &VaultConfig, assets: u128) -> u128 {
    let t = effective_totals(state, config);
    u128::from(mul_div_floor(
        Number::from(assets), Number::from(t.supply), Number::from(t.assets),
```

```
    ))
}
```

Impact

Depositors who interact with a vault in extreme state (`total_shares` close to `i128::MAX` against small effective assets) can transfer real assets and receive a wrapped share amount, including zero. Existing holders capture the donated value. `proxy_view` echoes the same wrapped quote, so off-chain tooling may normalize or bless the bad amount instead of detecting it. The trigger requires extreme pre-state and is not practically reachable in normal deployments; this is recorded as a numeric-hardening defect that should fail closed rather than silently wrap.

BVSS

AO:R/AC:H/AX:N/R:P/S:U/C:N/A:N/I:H/D:P/Y:N (1.2)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Every `u128::from(mul_div_*)` conversion in the kernel conversion helpers should be replaced with a checked `Number -> u128` conversion that returns an explicit overflow error.
- That error should be threaded through the Soroban entrypoints so deposits and previews fail closed instead of wrapping.
- Regression tests should be added where the raw quotient is exactly `u128::MAX + 1`, asserting that both deposit execution and previews reject the input.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix bounds conversion outputs at the operation boundary. Deposits, withdrawal requests, atomic withdraw/redeem planning, fee-share minting, and Soroban proxy_view previews now compare the full-width quotient against the legal u128 or signed ABI domain before conversion, and overflowing actions fail closed instead of truncating.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/236aceeb2ecae61a33e3bda388516144f6a22a19

20.29 DESERIALIZE_FEES_SPEC ACCEPTS A 113-BYTE BLOB WITH NO-GROWTH TAG AND SILENTLY DISCARDS 16 TRAILING BYTES

// INFORMATIONAL

Description

The `FeesSpec` codec admits two valid encodings: 97 bytes (no max growth rate) and 113 bytes (with max growth rate). The decoder enforces this length match at the top, but after parsing the first 96 bytes (perf wad + perf recipient + mgmt wad + mgmt recipient) it peeks at byte 96 for the optional-growth tag without checking that `cursor == bytes.len()` on the no-growth path. As a result, a 113-byte blob whose byte 96 is `0` deserializes to `FeesSpec { ..., max_total_assets_growth_rate: None }` and silently drops bytes 97..113.

Code Location

Copy Code

```
 None,                // <-- cursor stays at 96; no length check
    1 => {
        cursor += 1;
        Some(Wad::from(read_u128(bytes, &mut cursor)?))
    }
    _ => return Err(RuntimeError::storage_error(\"\")),
};
Ok(FeesSpec::new(performance, management, max_total_assets_growth_rate))" language="rust">// contract/va
let max_total_assets_growth_rate = match *bytes
    .get(cursor)
    .ok_or_else(|| RuntimeError::storage_error(""))?
{
    0 => None,                // <-- cursor stays at 96; no length check
    1 => {
        cursor += 1;
        Some(Wad::from(read_u128(bytes, &mut cursor)?))
    }
    _ => return Err(RuntimeError::storage_error("")),
};
Ok(FeesSpec::new(performance, management, max_total_assets_growth_rate))
```

Impact

The decoder is non-injective and trailing-byte-tolerant on the no-growth path. Two distinct on-disk blobs deserialize to the same `FeesSpec`. The asymmetry has three consequences:

- A malformed blob that `serialize_fees_spec` could not have produced is accepted as if it were canonical. There is no warning that the trailing 16 bytes were discarded.
- A future migration that needs to extend the FeesSpec format would have no way to detect "old vs new" by length alone.
- Any storage surface that accepts the encoded fees blob from outside (e.g., a future rescue-import action) becomes ambiguous about the meaning of those trailing bytes.

In contrast, the fully-with-growth path (113 bytes with tag 1) and `decode_state_blob` both enforce `cursor == bytes.len()`.

BVSS

AO:R/AC:H/AX:N/R:P/S:U/C:N/A:N/I:H/D:P/Y:N (1.2)

Recommendation

To mitigate this issue, the following improvements are recommended:

- After parsing the optional max_growth field, add `if cursor != bytes.len() { return Err(RuntimeError::storage_error("trailing bytes")); }`, mirroring `decode_state_blob`.
- Add a fuzz target for the `(serialize_fees_spec ↔ deserialize_fees_spec)` round-trip and the malformed-suffix rejection.

Remediation Comment

ACKNOWLEDGED: The client reports that the fee-spec parser now rejects trailing bytes for the no-growth encoding (covered by a dedicated trailing-bytes regression), so the originally reported tolerance is no longer present on the remediation branch. Acknowledged as an informational hardening item.

Remediation Hash

https://github.com/Templar-Protocol/contracts/pull/430

20.30 RESYNC_IDLE_BALANCE ADVANCES FEE_ANCHOR.TOTAL_ASSETS WITHOUT UPDATING TIMESTAMP_NS, DISTORTING FUTURE FEE MATH

// INFORMATIONAL

Description

`resync_idle_balance_impl` reconciles the vault's tracked `idle_assets` with the actual asset-token balance. When the actual balance exceeds the tracked balance, it advances `state.fee_anchor.total_assets` by the delta but does not advance `state.fee_anchor.timestamp_ns`. The function is permissionless (no role check beyond the cooldown), and any caller can transfer the underlying token directly to the vault to satisfy `actual_balance > before_idle`.

Code Location

Copy Code

```
// contract/vault/soroban/src/contract/entrypoints.rs (resync_idle_balance_impl)
if actual_balance > before_idle {
    let delta = actual_balance - before_idle;
    state.fee_anchor.total_assets = state.fee_anchor.total_assets.saturating_add(delta);
}
```

Impact

After a donation + resync, the anchor's two fields are no longer mutually consistent. The next time fees are refreshed (via the kernel `RefreshFees` path, when reachable), the math sees:

- `anchor.total_assets`: advanced by the donation amount (suppresses perf fees because `profit = fee_total_assets − anchor.total_assets` is reduced).
- `anchor.timestamp_ns`: stale, often `0` (inflates mgmt fees because `elapsed_ns = now_ns − 0` is much larger than reality).

Effects:

- **Performance fee suppression:** An attacker who holds a meaningful share position can donate to suppress perf fees they would otherwise pay.
- **Management fee inflation:** With stale `timestamp_ns`, the `elapsed × annual_fee` term is unbounded by elapsed-time-since-anchor.
- **Conflicting incentives:** Donations have opposite effects on the two fee types. Operators relying on either fee for revenue cannot reason about the impact of donations without knowing the other fee's configuration.

BVSS

AO:R/AC:H/AX:N/R:P/S:U/C:N/A:N/I:H/D:P/Y:N (1.2)

Recommendation

To mitigate this issue, the following improvements are recommended:

- When `resync_idle_balance_impl` advances `fee_anchor.total_assets`, also advance `fee_anchor.timestamp_ns` to `now_ns`. This collapses donations into the fee anchor at the time they are observed, preserving the two-field invariant `profit = total_assets − anchor.total_assets` over a consistent window.
- Audit any other path that mutates `fee_anchor.total_assets` without touching `timestamp_ns` (today: only `resync_idle_balance_impl`).
- Add a regression test that donates assets, calls resync, and asserts that a subsequent perf-fee refresh produces zero perf shares for that donation window.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix keeps actual idle-balance reconciliation on one shared path and collapses both fee_anchor.total_assets and fee_anchor.timestamp_ns to the same observed ledger time. A later refresh therefore charges management and performance fees from the resync timestamp instead of a stale pre-resync anchor.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/9febe5695f11b8e5a834275fd45caa2dfbc9b253

20.31 TOTAL_ASSETS_FOR_FEE_ACCRUAL EARLY-RETURNS WHEN ANCHOR_TOTAL_ASSETS == 0, NEUTRALIZING THE CONFIGURED MAX_GROWTH_RATE CAP

// INFORMATIONAL

Description

`total_assets_for_fee_accrual` implements the donation-attack mitigation by capping the effective `fee_total_assets` to `anchor_total_assets × (1 + max_rate × elapsed / YEAR)`. The function short-circuits and returns the uncapped `cur_total_assets` whenever `anchor_total_assets == 0`, which is the persisted state of every freshly-initialized Soroban vault.

Code Location

Copy Code

```
// contract/vault/kernel/src/math/wad.rs (total_assets_for_fee_accrual)
pub fn total_assets_for_fee_accrual(
    cur_total_assets: u128,
    anchor_total_assets: u128,
    anchor_timestamp_ns: u64,
    now_ns: u64,
    max_rate: Option<Wad>,
) -> u128 {
    let Some(max_rate) = max_rate else { return cur_total_assets; };
    if cur_total_assets <= anchor_total_assets
        || anchor_total_assets == 0     // <-- bypass on zero anchor
        || now_ns < anchor_timestamp_ns
    {
        return cur_total_assets;
    }
    ...
}
```

Impact

The `max_total_assets_growth_rate` field is documented as a defense against donation-induced fee spikes. The early-return at `anchor_total_assets == 0` short-circuits this protection because the cap formula evaluates to 0 (zero-times-anything), so the function falls back to "no cap". Operators who configure `max_growth_rate` to limit perf-fee exposure get no protection on the first refresh after init.
Combined with the deposit-doesn't-anchor finding and the unreachable-RefreshFees finding, the anchor never gets seeded on Soroban. So every previewed fee accrual is uncapped regardless of how the operator configured `max_total_assets_growth_rate`. Once the first refresh succeeds, the anchor is seeded and the cap becomes effective; until then, donation-induced perf-fee spikes would be unbounded.

BVSS

AO:R/AC:H/AX:N/R:P/S:U/C:N/A:N/I:H/D:P/Y:N (1.2)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Replace the `anchor_total_assets == 0` short-circuit with an explicit "treat this as a baseline-initialization refresh" path: on the first refresh after init, set `fee_total_assets = cur_total_assets` AND advance the anchor to `(cur_total_assets, now_ns)` without minting any fee shares. On subsequent refreshes, use the cap-formula path.
- Or: at `initialize()` time, set the anchor explicitly to `(state.total_assets, now_ns)` so the cap is well-defined from the start.

Remediation Comment

SOLVED: Client successfully remediated this issue. The zero-anchor fee accrual path now applies explicit bounded semantics instead of treating donated asset growth as an uncapped baseline. This preserves the configured max-growth-rate protection even before a nonzero fee anchor has been established.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/aeba3fc12c70750e02ab1b11ae0a036e938ba812

20.32 MIGRATE() AND HANDLE_EMERGENCY_RESET PRESERVE FEE_ANCHOR, PERPETUATING STALENESS ACROSS RECOVERY PATHS

// INFORMATIONAL

Description

`migrate()` calls `migrate_legacy_paused`, extends TTL, and clears the migration flag, but does not touch `state.fee_anchor`. The kernel's `handle_emergency_reset` resets `op_state`, refunds escrowed shares, restores in-flight assets, and dequeues the pending withdrawal, but does not touch `state.fee_anchor` either.

Code Location

Copy Code

```
// contract/vault/soroban/src/contract/entrypoints.rs (migrate)
pub fn migrate(env: Env, operator: Address) -> Result<(), ContractError> {
    require_governance(&env, &operator)?;
    if !migration_in_progress(&env) { return Err(ContractError::InvalidState); }
    migrate_legacy_paused(&env);
    extend_storage_ttl(&env);
    set_migration_in_progress(&env, false);
    emit_admin_event(&env, symbol_short!("migrate"));
    Ok(())
}
```

Impact

Migration and emergency reset are operational recovery paths intended to bring a stuck vault back to a healthy state. They clear `op_state`, refund escrowed shares, restore in-flight assets, update `paused`, and extend TTL, but do not reconcile the fee accrual window. After either operation, `fee_anchor.total_assets` and `fee_anchor.timestamp_ns` keep their prior (often `(0, 0)`) values.
This is not a directly exploitable vulnerability today (recovery is governance-only and `RefreshFees` is unreachable), but it is a correctness invariant that recovery should re-establish if it claims to leave the vault healthy. Any future fix to the unreachable-RefreshFees / inflated-proxy_view findings must also account for the fact that already-migrated vaults will not have a fresh anchor.

BVSS

AO:R/AC:H/AX:N/R:P/S:U/C:N/A:N/I:H/D:P/Y:N (1.2)

Recommendation

To mitigate this issue, the following improvements are recommended:

- In `migrate()` and `handle_emergency_reset` (or in a follow-up governance step embedded in those flows), set `state.fee_anchor = FeeAccrualAnchor::new(state.total_assets, now_ns)` using the current ledger time.

- Document the anchor reset as part of the recovery contract in `STRIDE.md` and the operator runbook.

Remediation Comment

SOLVED: Client successfully remediated this issue. Migration and emergency reset paths now make fee-anchor recovery semantics explicit by normalizing stale anchor state back to the current/recovered asset baseline. This prevents stale fee accrual state from surviving recovery operations and influencing later fee calculations.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/aeba3fc12c70750e02ab1b11ae0a036e938ba812

20.33 POLICY AND RESTRICTIONS DECODERS ACCEPT TRAILING BYTES, UNLIKE THE STATE AND COMMAND CODECS

// INFORMATIONAL

Description

The Soroban runtime has inconsistent decoder strictness. `decode_state_blob` and the public command/result codecs in `shared-types` reject trailing bytes via `cursor != bytes.len()` checks. The following storage decoders consume a prefix and return `Ok(...)` without checking that all bytes were consumed:

- `decode_restrictions`
- `decode_supply_queue`
- `decode_cap_groups`
- `decode_markets`
- `decode_principals`
- `decode_policy_locks`

`src/test_utils.rs` exposes fuzz wrappers for these codecs but does not enforce strict trailing-byte rejection.

Code Location

Copy Code

```
// contract/vault/soroban/src/storage/mod.rs (decode_state_blob)
if cursor != bytes.len() {
    return Err(RuntimeError::storage_error("state blob trailing bytes are invalid"));
}
```

Copy Code

```
// contract/vault/soroban/src/storage/mod.rs (decode_restrictions, no trailing-byte check)
pub(crate) fn decode_restrictions(bytes: &[u8]) -> Result<Restrictions, RuntimeError> {
    let mut cursor = 0usize;
    let tag = read_u8(bytes, &mut cursor)?;
    let len = read_u32(bytes, &mut cursor)? as usize;
    let mut addresses = Vec::with_capacity(len);
    for _ in 0..len { addresses.push(read_address(bytes, &mut cursor)?); }
    match tag {
        0 => Ok(Restrictions::blacklist(addresses)),
        1 => Ok(Restrictions::whitelist(addresses)),
        _ => Err(RuntimeError::storage_error("")),
    }
}
```

Impact

The storage codec layer is inconsistent: the main state blob and public command ABI are strict; policy and restrictions sub-blobs are trailing-byte tolerant. Two distinct byte strings can deserialize to the same policy or restrictions value. This weakens canonical storage assumptions and can hide malformed or stale bytes left behind by an upgrade, recovery script, or future format extension.

This does not give normal users a direct write primitive because these blobs are not user-provided through the public ABI. It is still a storage integrity issue because deserialization is the boundary between persistent bytes and trusted runtime state. Future migrations cannot safely rely on trailing data to distinguish old vs. new formats. Fuzz and round-trip tests can miss non-canonical encodings because the decode side accepts extra suffixes.

BVSS

AO:R/AC:H/AX:N/R:P/S:U/C:N/A:N/I:H/D:P/Y:N (1.2)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Add `if cursor != bytes.len() { return Err(RuntimeError::storage_error(...)); }` to every top-level storage decoder, mirroring `decode_state_blob`.
- Add fuzz tests that append random trailing bytes to canonical encodings and assert that all decoders reject them.
- Keep any intentionally extensible future format behind an explicit version byte rather than implicit trailing-byte tolerance.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix keeps policy and restriction storage decoding strict by requiring cursor exhaustion and adding versioned envelopes for new persisted writes. Trailing bytes are rejected consistently across the relevant storage codecs instead of being silently ignored.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/44401156a682c0ae7f4ff781c055ce16a684906e

20.34 WITHDRAW QUEUE DESERIALIZATION DOES NOT VALIDATE QUEUE INVARIANTS OR SORTED PENDING-WITHDRAWAL IDS

// INFORMATIONAL

Description

`decode_withdraw_queue` rebuilds a `WithdrawQueue` from raw entries and pointer values via `WithdrawQueue::with_state`, which delegates to `PendingWithdrawals::from_sorted_entries`. That function trusts the caller-provided order and does not validate it. The kernel's `WithdrawQueue::check_invariants` exists but is never called from the Soroban deserialization path.

Code Location

Copy Code

```
// contract/vault/soroban/src/storage/mod.rs (decode_withdraw_queue)
fn decode_withdraw_queue(bytes: &[u8], cursor: &mut usize) -> Result<WithdrawQueue, RuntimeError> {
    let next_withdraw_to_execute = read_u64(bytes, cursor)?;
    let next_pending_withdrawal_id = read_u64(bytes, cursor)?;
    let count = read_u32(bytes, cursor)? as usize;
    let mut entries = Vec::with_capacity(count);
    for _ in 0..count {
        let id = read_u64(bytes, cursor)?;
        let withdrawal = PendingWithdrawal::new(...);
        entries.push((id, withdrawal));
    }
    Ok(WithdrawQueue::with_state(
        entries, next_withdraw_to_execute, next_pending_withdrawal_id,
    ))
}
```

Copy Code

```
// contract/vault/kernel/src/state/queue/mod.rs (PendingWithdrawals::from_sorted_entries, trusts input)
pub fn from_sorted_entries(entries: Vec<(u64, PendingWithdrawal)>) -> Self {
    Self {
        entries: entries.into_iter().map(|(id, w)| PendingWithdrawalEntry { id, withdrawal: w }).collect
    }
}
```

Impact

The `WithdrawQueue` relies on structural invariants: pending entries sorted by ID; `next_withdraw_to_execute <= next_pending_withdrawal_id`; non-empty queue contains the head ID; cached totals match entries. Deserialization from a malformed state blob does not verify these invariants. `PendingWithdrawals::locate` uses binary search by ID, which is only correct when the internal vector is sorted. A corrupted or migration-produced state blob can decode successfully into a queue whose lookups, removals, and FIFO head behavior are undefined.

Concrete effects:

- If entries are not sorted, binary-search lookups can fail to find existing withdrawals.
- If the queue is non-empty but does not contain `next_withdraw_to_execute`, `head()` returns `None`, so matured withdrawals can become invisible to `execute_withdraw`.

- If pointer values are inconsistent, future enqueue/dequeue behavior can skip IDs or strand pending withdrawals.

This is not user-exploitable through normal public entrypoints because the state blob is contract-controlled. It is, however, a critical invariant boundary for migrations and recovery tooling.

BVSS

AO:R/AC:H/AX:N/R:P/S:U/C:N/A:N/I:H/D:P/Y:N (1.2)

Recommendation

To mitigate this issue, the following improvements are recommended:

- In `decode_withdraw_queue`, construct the queue and then enforce `queue.check_invariants()` before returning it.
- Replace `PendingWithdrawals::from_sorted_entries` with a function that sorts entries or validates strict ascending IDs and rejects duplicates.
- Add fuzz tests for malformed queue blobs: unsorted IDs, duplicate IDs, missing head ID, `next_withdraw_to_execute > next_pending_withdrawal_id`, and inconsistent pointer values.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix validates withdrawal queue reconstruction before returning a trusted queue. Decoding now rejects oversize queues, invalid ID ranges, missing heads, duplicate or unsorted IDs, and post-construction invariant failures.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/44401156a682c0ae7f4ff781c055ce16a684906e

20.35 COMPOSE_POLICY_STATE SILENTLY CONSTRUCTS DEFAULT VALUES FOR MISSING STORAGE PARTS

// INFORMATIONAL

Description

`compose_policy_state` returns `Ok(Some(...))` whenever any one of the five persistent policy keys is present (`PolicyLocks`, `PolicySupplyQueue`, `PolicyMarkets`, `PolicyPrincipals`, `PolicyCapGroups`). Missing parts are silently filled via `unwrap_or_default()` and passed to `PolicyState::from_parts`.

Code Location

Copy Code

```
// contract/vault/soroban/src/storage/mod.rs (compose_policy_state)
if markets.is_none() && principals.is_none() && cap_groups.is_none()
    && leases.is_none() && supply_queue.is_none() {
    return Ok(None);
}
let state = PolicyState::from_parts(
    markets.unwrap_or_default(),
    principals.unwrap_or_default(),
    cap_groups.unwrap_or_default(),
    leases.unwrap_or_default(),
    supply_queue.unwrap_or_default(),
)
.map_err(|_| RuntimeError::storage_error(""))?;
Ok(Some(state))
```

Impact

Two failure modes:

- **Partial expiry:** persistent storage TTL is per-entry. If the five policy keys ever drift in TTL (a future code change that touches one but not the others, or a manual storage rescue that re-writes a subset), `compose_policy_state` accepts the partial state and the runtime continues with default values for the missing parts.
- **Migration race:** if a future migration writes new-format keys alongside old ones, or rewrites only some keys before the migration flag is cleared, the same silent-defaulting path applies.

There is no on-chain diagnostic. `load_policy_state` returns `Some(default-mixed PolicyState)`, every subsequent kernel call proceeds, and operators cannot tell the difference between a fresh-but-loaded vault and a corrupted-partial-expiry vault. A subset of policy state can silently disappear without aborting reads.

BVSS

AO:R/AC:H/AX:N/R:P/S:U/C:N/A:N/I:H/D:P/Y:N (1.2)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Replace the "any one present → Some(...)" rule with an "all five present or all five absent" invariant. If any subset is present, return `Err(RuntimeError::storage_error(...))`.
- Or: add a single combined-policy-state blob (or a versioned manifest) that must be present alongside the per-key blobs, so the invariant can be checked at the storage boundary.
- Add a regression test that writes only `PolicySupplyQueue` and asserts `load_policy_state` returns an error rather than a defaulted `PolicyState`.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix makes partial policy storage a storage error unless all policy sub-blobs are absent or all required sub-blobs decode successfully. This prevents missing storage parts from silently constructing default policy values.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/44401156a682c0ae7f4ff781c055ce16a684906e

20.36 CANCEL_MIGRATION() CLEARS THE MIGRATION FLAG BUT CANNOT ROLL BACK THE ALREADY-SWAPPED WASM

// INFORMATIONAL

Description

`upgrade()` calls `env.deployer().update_current_contract_wasm(new_wasm_hash)` immediately and unconditionally. The Soroban host call is irreversible from within the contract: once it returns, the contract instance executes the new WASM on every subsequent call. `cancel_migration_impl` only clears the migration flag, leaving the new WASM installed.

Code Location

Copy Code

```
 Result<(), ContractError> {
    require_governance(env, &caller)?;
    if !migration_in_progress(env) { return Err(ContractError::InvalidState); }
    set_migration_in_progress(env, false);
    emit_admin_event(env, symbol_short!(\"cnc_migr\"));
    Ok(())
}" language="rust">// contract/vault/soroban/src/contract/entrypoints.rs (upgrade)
pub fn upgrade(env: Env, new_wasm_hash: BytesN<32>, operator: Address) -> Result<(), ContractError> {
    require_governance(&env, &operator)?;
    set_migration_in_progress(&env, true);
    env.deployer().update_current_contract_wasm(new_wasm_hash);
    emit_admin_event(&env, symbol_short!("upgrade"));
    Ok(())
}

// cancel_migration_impl
fn cancel_migration_impl(env: &Env, caller: Address) -> Result<(), ContractError> {
    require_governance(env, &caller)?;
    if !migration_in_progress(env) { return Err(ContractError::InvalidState); }
    set_migration_in_progress(env, false);
    emit_admin_event(env, symbol_short!("cnc_migr"));
    Ok(())
}
```

Impact

The two-step `upgrade -> migrate` flow is meant to provide a window during which governance can validate the new WASM and either commit or back out. The implementation does not match that intent:

- `upgrade()` immediately swaps the WASM. Every subsequent call is served by the new code, with the migration flag preventing most mutations until either `migrate` or `cancel_migration` clears it.
- `cancel_migration` only clears the flag. The new WASM stays installed.

If governance discovers, between `upgrade` and `migrate`, that the new WASM is broken (decodes the existing state blob incorrectly, or has a critical bug discovered during a public freeze window), `cancel_migration` does not give them a way back to the previous WASM. They must call `upgrade()` again with the previous hash and then `migrate()` to clear the flag, which is a second governance-controlled operation. In a worst case where the new WASM bricks `migrate` and `cancel_migration` itself, the contract is permanently frozen.

BVSS

AO:R/AC:H/AX:N/R:P/S:U/C:N/A:N/I:H/D:P/Y:N (1.2)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Have `upgrade()` snapshot the previous WASM hash to instance storage (e.g., `PreviousWasmHash` symbol) before calling `update_current_contract_wasm`.
- Have `cancel_migration()` read that snapshot and call `update_current_contract_wasm(previous_hash)` to restore the prior WASM, then clear both the flag and the snapshot.
- Document the two-step semantic in `STRIDE.md` and the operator runbook so the rollback path is explicit.
- Add a test that performs an `upgrade(B)` from `A`, then `cancel_migration()`, and asserts that subsequent calls execute the WASM at hash `A`, not `B`.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix documents the actual migration semantics explicitly in Soroban STRIDE: cancel_migration is an operational unfreeze/gate-clearing path, not a WASM rollback after update_current_contract_wasm. Bad-code recovery requires a follow-up governed upgrade or migration when chain execution permits it.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/44401156a682c0ae7f4ff781c055ce16a684906e

20.37 RESTRICTIONS BLOB CAN EXCEED THE SOROBAN ENTRY-SIZE LIMIT WHEN GOVERNANCE CONFIGURES LARGE BLACKLIST OR WHITELIST SETS

// INFORMATIONAL

Description

`encode_restrictions` serializes the full address list into a single persistent blob: 1-byte tag + 4-byte length + 32-byte kernel address per entry. `apply_restrictions_policy` accepts an arbitrary-length address vector from governance with no length cap and converts each via `kernel_address_from_sdk`. Stellar's `MAX_CONTRACT_DATA_ENTRY_SIZE_BYTES = 64 KiB`. With a per-account cost of 32 bytes, this caps the restrictions list at roughly `(65536 - 5) / 32 ≈ 2048` accounts. Beyond that, `save_restrictions` traps on the oversized blob.

Code Location

Copy Code

```
// contract/vault/soroban/src/storage/mod.rs (encode_restrictions)
match mode {
    Restrictions::Blacklist(addresses) => {
        push_u8(&mut out, 0);
        push_u32(&mut out, addresses.len() as u32);
        for address in addresses { push_address(&mut out, address); }
    }
    ...
}
```

Copy Code

```
// contract/vault/soroban/src/contract/entrypoints.rs (apply_restrictions_policy)
fn apply_restrictions_policy(env: &Env, caller_kernel: Address, mode: u32, accounts: Vec<Address>) -> Re
    let mut kernel_accounts = Vec::with_capacity(accounts.len() as usize);
    for account in accounts.iter() { kernel_accounts.push(kernel_address_from_sdk(env, &account)); }
    // no length cap
    ...
}
```

Impact

A governance proposal that sets a restrictions list with more than ~2048 accounts cannot be persisted. `save_restrictions` traps on the oversized blob, and the governance call reverts. This is less severe than the state-blob size cap because:

- This is a governance-only path; an attacker first needs governance control to reach it.
- Failure is at proposal-execution time. A failed proposal does not brick the vault — earlier state is preserved by the tx revert.

Still worth addressing: the contract advertises "blacklist / whitelist" without documenting the practical size cap, and a runaway operational scenario where someone tries to migrate a large list discovers the cap only at execution time.

BVSS

AO:R/AC:H/AX:N/R:P/S:U/C:N/A:N/I:H/D:P/Y:N (1.2)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Add an explicit length cap in `apply_restrictions_policy` (e.g., `accounts.len() <= MAX_RESTRICTIONS_ENTRIES`) and return a typed `ContractError::InvalidInput` on overflow.
- Document the cap in `STRIDE.md` and the governance API surface.
- Optionally split the restrictions blob into paged entries if very large lists are a deployment requirement.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix uses bounded paged blob transport for large restrictions and policy blobs. Restrictions persist through bounded 32 KiB pages, the one-shot restriction cap is set to 3072 addresses, and larger writes are rejected before persistent storage writes.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/fee307b82177c0beeaccf751c5f785c6c9cffec8

20.38 PROXY_VIEW RETURNS MAX_WITHDRAW AND MAX_REDEEM FOR ATOMIC OPERATIONS THE DEPLOYED ABI CANNOT PERFORM

// INFORMATIONAL

Description

`proxy_view` reports non-zero `max_withdraw_value` and `max_redeem_value` whenever the vault is in `Idle`. These values represent the ERC-4626 atomic-withdraw / atomic-redeem fast-path limits computed against `state.idle_assets` and the caller's share balance. `CuratorVault::atomic_withdraw` and `atomic_redeem` exist as public methods, but no `VaultCommand` variant routes to them and the deployed contract has no entrypoint that performs an atomic withdraw or redeem.

Code Location

Copy Code

```
// contract/vault/soroban/src/contract/entrypoints.rs (proxy_view, atomic max math)
let (max_withdraw_value, max_redeem_value) = if state.op_state.is_idle() {
    let max_redeem_u128 = owner_shares.min(convert_to_shares(&state, &config, state.idle_assets));
    let max_withdraw_u128 = convert_to_assets(&state, &config, owner_shares).min(state.idle_assets);
    (to_i128(max_withdraw_u128)?, to_i128(max_redeem_u128)?)
} else { (0, 0) };
```

Impact

ERC-4626-style integrators read these values and assume they can `withdraw(assets, receiver, owner)` / `redeem(shares, receiver, owner)` for amounts up to those limits. The deployed contract has no entrypoint that performs an atomic withdraw or redeem, so any user attempting to act on the advertised limit is forced into the queued `request_withdraw` → `execute_withdraw` flow, which has its own preconditions (allocator-only execution, queue cooldowns, stuck-state risks).
This is not a fund-loss vulnerability, but it is a meaningful ERC-4626 conformance gap: `max_withdraw` / `max_redeem` should reflect what the contract can actually do.

BVSS

AO:R/AC:H/AX:N/R:P/S:U/C:N/A:N/I:H/D:P/Y:N (1.2)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Either add `AtomicWithdraw` / `AtomicRedeem` to `VaultCommand` and route them via `execute_command`, adding matching `self.authorize(...)` calls.
- Or hard-code `max_withdraw_value = 0` and `max_redeem_value = 0` in `proxy_view` and document that atomic withdraws are not available.

Remediation Comment

SOLVED: Client successfully remediated this issue. The public VaultCommand ABI still does not expose atomic withdraw/redeem, so proxy_view now advertises max_withdraw = 0 and max_redeem = 0 instead of publishing actionable-looking limits for unavailable commands. This keeps the ERC-4626-style view truthful until atomic commands are deliberately added.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/18de52b49e8c8e062408e5695fe1cd0eb6fb527c

20.39 PROXY_VIEW MAX_DEPOSIT AND MAX_MINT ONLY CHECK THE SAME-SIDE OVERFLOW

// INFORMATIONAL

Description

`proxy_view::max_deposit` only checks `u128::MAX - total_assets`. It does not consider `total_shares` headroom: if `total_shares` is near `u128::MAX`, the conversion of `assets_in` to `shares_out` may overflow on the share-mint path even when `assets_in <= max_deposit_value`. The same asymmetry applies to `max_mint`.

Code Location

Copy Code

```
// contract/vault/soroban/src/contract/entrypoints.rs (proxy_view max_deposit/max_mint)
let (max_deposit_value, max_mint_value) = if state.op_state.is_idle() && !config.paused {
    let max_assets = u128::MAX.saturating_sub(state.total_assets).min(i128::MAX as u128) as i128;
    let max_shares = u128::MAX.saturating_sub(state.total_shares).min(i128::MAX as u128) as i128;
    (max_assets, max_shares)
} else { (0, 0) };
```

Impact

`handle_deposit` verifies all three bounds (`total_assets`, `idle_assets`, `total_shares`) via `checked_add`. If the share side is the binding constraint, a deposit transaction that passes the proxy-view-driven check can still revert at the kernel-side check.

Off-chain quote tooling occasionally builds transactions that pass the proxy-view-driven check but revert at the kernel-side `checked_add` on the share or idle side. This is a minor user-UX regression and not exploitable.

BVSS

AO:R/AC:H/AX:N/R:P/S:U/C:N/A:N/I:H/D:P/Y:N (1.2)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Compute the binding constraint over both sides of the conversion:

Copy Code

```
let max_assets_for_total_assets = u128::MAX.saturating_sub(state.total_assets);
let shares_at_max = convert_to_shares(state, config, max_assets_for_total_assets);
let shares_headroom = u128::MAX.saturating_sub(state.total_shares);
let max_assets = if shares_at_max > shares_headroom {
    convert_to_assets(state, config, shares_headroom)
} else { max_assets_for_total_assets };
```

- Then min with `i128::MAX` for the ABI cast.

- Add an integration test that pushes `total_shares` near `u128::MAX` and asserts a `max_deposit_value` that produces a deposit transaction the kernel actually accepts.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix makes proxy_view compute max_deposit and max_mint against both same-side headroom and opposite-side conversion headroom. It falls back to bounded inverse conversion rather than advertising unsafe ABI limits.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/335a946bb94246ace588f7ad62cf689ac1e0a244

20.40 PREPARE_ATOMIC_CALL DISCARDS THE EFFECTSUMMARY RETURNED BY THE CONDITIONAL REFRESHFEES KERNEL ACTION

// INFORMATIONAL

Description

`prepare_atomic_call` conditionally triggers a `KernelAction::RefreshFees` when fees are active and time has advanced past the anchor. The function uses `let _ = self.apply_kernel_action(...)?`, which propagates errors but explicitly discards the `EffectSummary` returned on success. The `EffectSummary` for `RefreshFees` includes `events_emitted` and `shares_minted` from the fee mints; that information is lost.

Code Location

Copy Code

```
// contract/vault/soroban/src/contract/curator_vault.rs (prepare_atomic_call)
let fees_active = !self.config.fees.management.fee_wad.is_zero()
    || !self.config.fees.performance.fee_wad.is_zero();
if fees_active && now_ns > self.state()?.fee_anchor.timestamp_ns.as_u64() {
    let _ = self.apply_kernel_action(
        KernelAction::RefreshFees { now_ns: TimestampNs(now_ns) },
        now_ns,
    )?;
}
```

Impact

This is not a security issue today because fee minting cannot reach this path: `atomic_withdraw` and `atomic_redeem` are not in `VaultCommand`. However, if either of those paths becomes reachable through the deployed ABI, the summary discard becomes a real observability gap:

- Atomic-redeem returns `assets_transferred`, but the actual share burn includes both the user's redemption and any fee shares minted to recipients in the same call. The summary returned to the caller doesn't distinguish.
- Indexers that watch effect summaries (rather than raw events) miss the fee mint entirely.

BVSS

AO:R/AC:H/AX:N/R:P/S:U/C:N/A:N/I:H/D:P/Y:N (1.2)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Have `prepare_atomic_call` return the merged `EffectSummary` (refresh + the caller's own action) so the caller can include it in the response.
- Or: keep the discard but emit an explicit `KernelEvent::FeesRefreshed` topic from the Soroban runtime that off-chain tooling can subscribe to.

Remediation Comment

SOLVED: Client successfully remediated this issue. prepare_atomic_call now returns the conditional RefreshFees EffectSummary to its caller. Atomic withdraw/redeem merge that summary into their own action summaries, so fee-mint/event effects are preserved when the atomic paths are reachable.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/e381fc2e70ff640e1d20103d8f2121dcbfd050df

20.41 LOAD_VAULT_BOOTSTRAP RUNS MIGRATE_LEGACY_PAUSED ON EVERY CONTRACT CALL

// INFORMATIONAL

Description

`load_vault_bootstrap` is invoked by every `with_contract_vault` call, which is reached by every state-mutating entrypoint. It unconditionally runs `migrate_legacy_paused` and `extend_storage_ttl`. `migrate_legacy_paused` is a one-time migration meant to handle the legacy pause-key formats. After the first successful run, neither `VaultDataKey::Paused` nor `SorobanStorageKey::Paused` should exist; the function would short-circuit (no key found) on every subsequent call.

Code Location

Copy Code

```
// contract/vault/soroban/src/contract/helpers.rs (load_vault_bootstrap)
pub(crate) fn load_vault_bootstrap(env: &Env) -> Result<VaultBootstrap<'_>, RuntimeError> {
    if migration_in_progress(env) {
        return Err(RuntimeError::invalid_state("migration in progress - call migrate() first"));
    }
    extend_storage_ttl(env);
    migrate_legacy_paused(env);
    ...
}
```

Impact

The two operations together represent a few extra storage reads and writes on every vault call. After the first successful migration, the migration-related work is permanently a no-op but still executed. The TTL extension is desirable on most calls but isn't free.
This is not a correctness or security issue. It is gas-cost cruft that amplifies as call volume grows and migrations age out. A reviewer expecting `migrate_legacy_paused` to be one-shot might also be surprised that it remains in the hot path indefinitely.

BVSS

AO:R/AC:H/AX:N/R:P/S:U/C:N/A:N/I:H/D:P/Y:N (1.2)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Add an instance-storage flag (`LegacyPausedMigrated`) that `migrate_legacy_paused` sets after a successful run. Skip subsequent invocations when the flag is set.
- Or: inline the legacy migration into `migrate()` only and remove it from the hot `load_vault_bootstrap` path once a deployment-cutoff date passes.
- Document the gas cost in `STRIDE.md` so operators know `with_contract_vault` is not zero-cost.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #437 removes the undeployed legacy pause migration path entirely. The runtime no longer calls migrate_legacy_paused, and the legacy Paused storage/key shim is absent from the live source surface.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/8f8b07586e29b6c65ba0e3a7e46c79a74a587ab5

20.42 GUARDIAN CONFIGURATION REPLACES THE FULL GUARDIAN SET SILENTLY

// INFORMATIONAL

Description

`SetGovernanceConfig(Guardians)` overwrites the entire guardian list rather than adding or removing individual guardians. Operators that intend to add one guardian must include every existing guardian or they will be dropped.
This is reassessed as **Low** because it is an authorized-operator footgun that can weaken emergency coverage.

Proof of Concept

Preconditions

- The vault has an existing guardian set.

Validation Sequence

1. Configure guardians `[A, B]`.
2. Submit an update intending to add guardian `C` but pass only `[C]`.
3. Observe that `A` and `B` are removed because the full set is replaced.

Reproducing Locally

Copy Code

```
rg "apply_guardians_config|Guardians" contract/vault/soroban/src/contract/entrypoints.rs contract/vault/
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:M/I:L/D:N/Y:N (1.2)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Separate `add_guardian` and `remove_guardian` actions should be introduced.
- Full-list replacement should require an explicit force flag.
- Events should include the previous set, new set, and operation mode.

Remediation Comment

ACKNOWLEDGED: The client reports that the guardian configuration surface referenced by this finding has been removed/changed in the current governance work, so silently replacing the full guardian set is no

longer applicable on the remediation branch. We confirmed the guardian surface is absent on the integrated branch. Acknowledged as an informational/stale observation.

20.43 SHARE-TOKEN BURN PATHS RELY ON VAULT INVOKER CHECKS INSTEAD OF OWNER AUTHORIZATION

// INFORMATIONAL

Description

The share-token `burn` and `burn_from` paths do not require explicit authorization from the owner or spender. They rely on the stored vault being the invoker. If the vault ever calls these methods on behalf of a user without independently checking user intent, the share token itself does not enforce the owner approval boundary.
This is reassessed as **Low** because the vault is expected to be the trusted caller, but the token contract's authorization invariant is weaker than typical token semantics.

Proof of Concept

Preconditions

- The share-token is deployed and bound to a vault.
- The vault invokes burn paths on behalf of users.

Validation Sequence

1. Inspect `burn` and `burn_from`.
2. Confirm they authenticate the vault invoker but do not independently require owner/spender auth.
3. Confirm the token contract relies on the vault to enforce user intent.

Reproducing Locally

Copy Code

```
rg "pub fn burn|pub fn burn_from|require_vault_invoker|require_auth" contract/vault/soroban/share-token/
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:L/I:M/D:N/Y:N (1.2)

Recommendation

To mitigate this issue, the following improvements are recommended:

- `burn` should require owner authorization in addition to vault-invoker authorization where user intent is expected.
- `burn_from` should enforce spender authorization and allowance semantics explicitly.
- Tests should cover unauthorized direct and delegated burn attempts.

Remediation Comment

SOLVED: Client successfully remediated this issue. Share-token burn semantics are now explicitly treated and tested as a vault-only protocol settlement primitive rather than a public owner burn API. Owner consent is enforced at the vault withdrawal entrypoints before BurnShares effects are generated, while delegated burn flows continue to use burn_from with allowance semantics.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/fe283e701cbaff569c186cd7f944d8dff608c0b1

20.44 CONSTRUCTORS AND GOVERNANCE CONFIG SETTERS ACCEPT NONSENSICAL ROLE ADDRESSES

// INFORMATIONAL

Description

Several constructors and governance configuration helpers accept addresses without checking whether they are the contract itself, the asset token, the share token, or another role where the assignment is nonsensical. Misconfigurations can leave the vault unable to execute later effects or governance actions. This is reassessed as **Low** because it is mostly an operator-misconfiguration risk, but it affects critical role bindings.

Proof of Concept

Preconditions

- Deployment or governance config accepts role addresses.

Validation Sequence

1. Pass a semantically invalid address such as the vault itself, share token, asset token, or a role collision.
2. Observe the setter or constructor accepting the address.
3. Trigger a dependent operation and observe later failure or lockout.

Reproducing Locally

Copy Code

```
rg "initialize|apply_governance_config|set_config_address|require_contract_address" contract/vault/sorob
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:L/I:M/D:N/Y:N (1.2)

Recommendation

To mitigate this issue, the following improvements are recommended:

- A shared role-address validator should reject self-references and incompatible protocol addresses.
- Each constructor and setter should enforce role-specific address constraints.
- Configuration events should expose the old and new addresses for monitoring.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #451 adds runtime initialization and governance-config address-form validation so nonsensical role addresses are rejected instead of stored as production role authorities. Regression coverage includes runtime_initialize_rejects_non_contract_governance and

runtime_governance_config_rejects_sac_role_addresses, plus the PR #451 role-address topology regression suite.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/61fb75ea2b12141e0ae2e6906d839025f890532f

20.45 GOVERNANCE TIMELOCK LEGACY FALLBACK CAN DROP PER-KIND CUSTOMIZATIONS

// INFORMATIONAL

Description

If the per-kind timelock map is missing, `load_timelocks` falls back to constructing all timelocks from a single legacy default. A migration or cache-miss scenario can therefore lose per-kind customizations and silently alter governance delays.
This is reassessed as **Low** because it depends on missing storage or migration edge cases, but it affects a critical governance safety control.

Proof of Concept

Preconditions

- Governance timelock storage is missing the per-kind map.

Validation Sequence

1. Remove or omit per-kind timelock storage in a migration/test setup.
2. Call a path that loads timelocks.
3. Observe fallback to a single legacy default for all kinds.

Reproducing Locally

Copy Code

```
rg "load_timelocks|from_default|legacy" contract/vault/soroban/governance/src/lib.rs
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:L/I:M/D:N/Y:N (1.2)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Production builds should reject legacy-only timelock storage once per-kind timelocks are expected.
- Migrations should explicitly populate the per-kind map.
- View paths should not silently rewrite or downgrade timelock configuration.

Remediation Comment

ACKNOWLEDGED: The client reports the legacy timelock fallback behavior is stale against the current typed per-kind timelock model; per-kind timelock behavior should be evaluated against the current typed implementation rather than the legacy fallback path. Acknowledged as an informational/stale observation.

20.46 AUXILIARY CONTRACT ADMIN ROTATION IS SINGLE-STEP AND EMITS NO EVENT

// INFORMATIONAL

Description

Auxiliary contracts such as the share-token use immediate admin rotation without a proposal/acceptance flow and without event emission. A mistyped or compromised admin update is effective immediately and may be difficult for monitors to detect.
This is reassessed as **Low**. The share-token vault-rebind path is separately reported as Medium; this finding covers the broader admin-rotation pattern.

Proof of Concept

Preconditions

- The share-token admin key can call privileged setters.

Attack / Defect Sequence

1. Call `set_vault` to rebind mint/burn authority.
2. Call `set_admin` to rotate the admin to the attacker.
3. Observe both changes take effect in consecutive single transactions.

PoC Script

The following runnable test code is included in `contract/vault/soroban/share-token/tests/adv_rebind_mint_from_old_vault.rs`:

Copy Code

```
#[contract]
struct VaultCaller;

#[contractimpl]
impl VaultCaller {
    pub fn mint(env: Env, token: Address, to: Address, amount: i128) {
        env.invoke_contract::<()>(
            &token,
            &Symbol::new(&env, "mint"),
            (to, amount).into_val(&env),
        );
    }
}

#[test]
fn adv_d8_rebind_then_set_admin_chains_into_full_takeover() {
    let env = Env::default();
    let (admin, _old_vault, new_vault, token) = setup(&env);
    let attacker_admin = Address::generate(&env);

    env.invoke_contract::<()>(
        &token,
        &Symbol::new(&env, "set_vault"),
        (&admin, &new_vault).into_val(&env),
    );
```

```
    env.invoke_contract::<()>(
        &token,
        &Symbol::new(&env, "set_admin"),
        (&admin, &attacker_admin).into_val(&env),
    );

    let final_admin: Address =
        env.invoke_contract(&token, &Symbol::new(&env, "admin"), ().into_val(&env));
    let final_vault: Address =
        env.invoke_contract(&token, &Symbol::new(&env, "vault"), ().into_val(&env));

    assert_eq!(final_admin, attacker_admin);
    assert_eq!(final_vault, new_vault);
}
```

Reproducing Locally

Copy Code

```
cargo test -p templar-soroban-share-token --test adv_rebind_mint_from_old_vault -- --nocapture
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:L/I:M/D:N/Y:N (1.2)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Admin rotation should be converted to a two-step proposal and acceptance flow.
- Both steps should emit structured events.
- New admins should be validated against self and dependent contract addresses.

Remediation Comment

SOLVED: Client successfully remediated this issue. Auxiliary share-token and Blend adapter admin rotation now uses a pending-admin proposal followed by explicit acceptance, and emits admin proposal/acceptance events. This removes the single-step silent handoff and gives off-chain observers a clear admin-lifecycle trail.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/5b03b63038014bdc1866061c1e61a13f80ad65cf

20.47 SETGOVERNANCE ACCEPTS THE VAULT OR ARBITRARY NON-GOVERNANCE CONTRACTS

// INFORMATIONAL

Description

The vault accepts a new governance address without probing that it implements the expected governance interface and without rejecting the vault, share-token, asset-token, or other nonsensical contracts. A bad governance address can leave the vault without a functioning governance plane.
This is reassessed as **Low** because the action is governance-authorized but can cause persistent administrative lockout.

Proof of Concept

Preconditions

- Governance can update the vault governance address.

Validation Sequence

1. Set governance to a non-governance contract, the vault itself, or another incompatible contract.
2. Observe that the setter accepts the address.
3. Attempt a later governance-routed operation and observe that the control plane is broken.

Reproducing Locally

Copy Code

```
cargo test -p templar-soroban-governance --test timelock_matrix -- --nocapture
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:M/I:L/D:N/Y:N (1.2)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Candidate governance addresses should be probed through a known governance method before acceptance.
- The vault, asset-token, and share-token addresses should be explicitly rejected.
- Governance changes should be two-step and evented.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #451 rejects invalid SetGovernance targets, including the vault itself and arbitrary non-governance contracts, before accepting a governance handoff target.

Regression coverage includes the governance-target regression in PR #451 and the role-address topology verification suite.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/5cfbcffc8aa995022eb0d7ae2a82adc71a423ca7

20.48 AUXILIARY CONTRACTS LACK PAUSE OR FREEZE MECHANISMS

// INFORMATIONAL

Description

The share-token and Blend adapter do not expose pause or freeze controls. The vault pause therefore does not prevent direct share-token transfers or direct adapter operations that remain available through their own contracts.
This is reassessed as **Low** because the highest-value actions are still role-gated, but emergency response is incomplete across the system.

Proof of Concept

Preconditions

- An emergency requires freezing share-token transfers or adapter actions.

Validation Sequence

1. Inspect auxiliary contract APIs for `pause`, `unpause`, or freeze state.
2. Pause the vault.
3. Confirm direct share-token/adapter operations are not blocked by an auxiliary pause mechanism.

Reproducing Locally

```
Copy Code
rg "pause|unpause|paused|freeze" contract/vault/soroban/share-token/src contract/vault/soroban/blend-ada
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:M/I:L/D:N/Y:N (1.2)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Auxiliary contracts should expose pause and unpause controls.
- Vault pause should be propagated to dependent contracts or checked by them before sensitive actions.
- Pause events should be consistent across vault, share-token, and adapter contracts.

Remediation Comment

SOLVED: Client successfully remediated this issue. Share-token and Blend adapter auxiliary contracts now expose admin/vault-gated pause controls and fail closed while paused on state-changing mint, burn, transfer, supply, withdraw, and rescue paths. Regression coverage includes

admin_can_pause_and_unpause_share_token_transfers, vault_can_pause_share_token_mint_and_burn, admin_can_pause_adapter_operations, vault_can_pause_adapter_operations, and non_admin_or_vault_cannot_pause_adapter.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/fe283e701cbaff569c186cd7f944d8dff608c0b1

20.49 AUXILIARY CONTRACTS HAVE NO UPGRADE OR MIGRATE PATH

// INFORMATIONAL

Description

The share-token and Blend adapter do not expose upgrade or migration entrypoints. Fixing issues in either auxiliary contract requires redeployment and reconfiguration of every dependent vault component.
This is reassessed as **Low** because it is an operational recovery limitation, but it can slow or complicate incident response.

Proof of Concept

Preconditions

- A defect is discovered in a deployed share-token or Blend adapter.

Validation Sequence

1. Inspect the auxiliary contract public methods.
2. Search for upgrade or migrate entrypoints.
3. Confirm remediation requires redeployment and reconfiguration rather than in-place migration.

Reproducing Locally

Copy Code

```
rg "upgrade|migrate|update_current_contract_wasm" contract/vault/soroban/share-token/src contract/vault/
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:M/I:L/D:N/Y:N (1.2)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Auxiliary contracts should include admin-guarded upgrade and migrate entrypoints.
- Upgrade actions should be timelocked and evented.
- Deployment documentation should describe recovery paths for auxiliary contract defects.

Remediation Comment

SOLVED: Client successfully remediated this issue. Share-token and Blend adapter now implement an OpenZeppelin-style Upgradeable path through stellar-contract-utils, gated by the current admin. This provides a controlled remediation/upgrade path for auxiliary contracts rather than leaving them permanently immutable.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/5b03b63038014bdc1866061c1e61a13f80ad65cf

20.50 SHARE-TOKEN METADATA CAN BE CHANGED AFTER DEPLOYMENT WITHOUT TIMELOCK OR EVENT

// INFORMATIONAL

Description

The share-token admin can change token name, symbol, and decimals after deployment. In particular, changing decimals can break external accounting assumptions and UI/indexer interpretation of balances. This is reassessed as **Low** because admin authority is required, but metadata changes can materially affect integrations.

Proof of Concept

Preconditions

- The share-token admin can call metadata setters.

Validation Sequence

1. Deploy the share token with initial metadata.
2. Call `set_metadata` with changed name, symbol, or decimals.
3. Observe immediate metadata change without timelock.

Reproducing Locally

Copy Code

```
cargo test -p templar-soroban-share-token --test admin_rotation -- --nocapture
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:L/I:M/D:N/Y:N (1.2)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Decimals should be immutable after construction.
- Name and symbol changes should be timelocked and evented.
- Metadata changes should be documented as governance-controlled actions if retained.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #429 makes share-token metadata immutable after deployment by preserving constructor-time name, symbol, and decimals and removing the mutable metadata path. Regression coverage includes admin_cannot_change_metadata_after_deployment, source

and justfile sweeps for the removed metadata setter surface, and the consolidated share-token PR #429 verification.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/0aad51eb25cba864ef2a02f8122274ab79de0058

20.51 VAULT RESTRICTIONS DO NOT PROPAGATE TO SHARE-TOKEN TRANSFERS

// INFORMATIONAL

Description

The vault enforces whitelist and blacklist restrictions on deposit and withdrawal operations, but the share-token contract allows direct transfers without consulting the vault's restriction lists. A restricted account can therefore receive or move share tokens outside the vault's restriction checks.
This is reassessed as **Low** because redemption still passes through vault logic, but transferability can undermine compliance or access-control assumptions.

Proof of Concept

Preconditions

- The vault has whitelist or blacklist restrictions configured.
- The share-token is transferable.

Validation Sequence

1. Restrict an account at the vault level.
2. Transfer share tokens directly through the share-token contract.
3. Observe that the transfer does not consult vault restrictions.

Reproducing Locally

Copy Code

```
rg "Restrictions|transfer" contract/vault/soroban/src contract/vault/soroban/share-token/src/lib.rs
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:L/I:M/D:N/Y:N (1.2)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Share-token transfers should call a vault hook or locally cached restriction check.
- If restrictions intentionally apply only to vault entrypoints, that limitation should be documented clearly.
- Tests should cover blacklisted share recipients and transfers from restricted accounts.

Remediation Comment

SOLVED: Client successfully remediated this issue. Share-token transfer and transfer_from now enforce admin/vault-configured None, Blacklist, and Whitelist restrictions against sender and recipient addresses,

including muxed transfer recipients via MuxedAddress::address(). Regression coverage includes share_token_restrictions_block_blacklisted_sender_and_recipient and share_token_whitelist_allows_only_listed_transfer_parties.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/fe283e701cbaff569c186cd7f944d8dff608c0b1

20.52 EFFECT-ADDRESS PREFLIGHT CAN SUCCEED FOR MAPPINGS THAT FAIL DURING EXECUTION

// INFORMATIONAL

Description

The preflight helper for ensuring effect address mappings includes a fail-open branch, while the actual effect executor fails closed when a mapping is missing. This means preflight can report success for an effect batch that later traps during execution.
This is reassessed as **Low** because it affects reliability and error predictability, but it can break asynchronous effect execution.

Proof of Concept

Preconditions

- An effect batch references a kernel address that is not mapped in the Soroban address book.

Validation Sequence

1. Construct an effect requiring an address lookup.
2. Call the preflight mapping helper and observe success or non-failure.
3. Execute the same effect and observe failure when the executor performs the lookup.

Reproducing Locally

Copy Code

```
cargo test -p templar-soroban-runtime --features testutils --test effect_interpreter_props -- --nocaptur
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:M/I:N/D:N/Y:N (1.1)

Recommendation

To mitigate this issue, the following improvements are recommended:

- The preflight helper should use the same fail-closed semantics as `execute_effect`.
- Missing mapping errors should be surfaced before any state transition is committed.
- A regression test should assert that a missing receiver mapping fails during preflight.

Remediation Comment

ACKNOWLEDGED: The client reports that effect-address validation now fails closed before execution for unsupported or invalid mappings, so the reported preflight/execution mismatch is not present on the current branch. Acknowledged as an informational/stale observation.

20.53 ADAPTER CALLS CAN CONSUME THE SHARED TRANSACTION INSTRUCTION BUDGET

// INFORMATIONAL

Description

Vault effect execution and adapter calls share the same transaction instruction budget. A computationally expensive adapter or pool interaction can exhaust the budget before the vault finishes the intended effect sequence, producing brittle liveness behavior.
This is reassessed as **Low** because the adapter set is configured by governance, but the runtime does not enforce explicit per-adapter complexity bounds.

Proof of Concept

Preconditions

- The vault is configured with an adapter whose call path is computationally expensive.

Validation Sequence

1. Configure a mock adapter that consumes significant instruction budget.
2. Trigger allocation or market refresh through the vault.
3. Observe that adapter and vault execution share one transaction budget.

Reproducing Locally

Copy Code

```
rg "invoke_contract|execute_effect|refresh_markets" contract/vault/soroban/src
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:M/I:N/D:N/Y:N (1.1)

Recommendation

To mitigate this issue, the following improvements are recommended:

- The vault should cap the number of adapter calls and market refreshes per transaction.
- Adapter complexity expectations should be documented and validated in integration tests.
- Large refreshes should be split into paged operations.

Remediation Comment

ACKNOWLEDGED: Adapter calls sharing the Soroban transaction instruction budget is an accepted platform and integration constraint rather than a protocol code defect. The intended mitigations are adapter

allowlisting, bounded refresh inputs, keeper batching, monitoring, and operational runbooks rather than attempting to isolate adapter execution from Soroban transaction metering.

20.54 STALE BLEND RESERVE DATA CAN REVERT VAULT ALLOCATION AND REFRESH FLOWS

// INFORMATIONAL

Description

Blend reserve data can become stale relative to the adapter's expected freshness window. The vault does not automatically refresh reserve state before allocation or market refresh calls, so long-idle positions can cause the next adapter operation to revert.
This is reassessed as **Low** because it is a liveness issue dependent on external protocol freshness.

Proof of Concept

Preconditions

- A Blend reserve has not been refreshed within the expected freshness window.

Validation Sequence

1. Let reserve data become stale or simulate stale reserve state.
2. Call adapter `total_assets`, allocation, or refresh path.
3. Observe revert or stale-reserve error before vault automation refreshes it.

Reproducing Locally

Copy Code

```
cargo test -p templar-soroban-blend-adapter --test stale_reserve -- --nocapture
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:M/I:N/D:N/Y:N (1.1)

Recommendation

To mitigate this issue, the following improvements are recommended:

- The adapter should refresh or validate Blend reserve freshness before every operation.
- A keeper-callable reserve-refresh path should be exposed.
- Allocation automation should be documented to refresh reserves before high-value operations.

Remediation Comment

ACKNOWLEDGED: Stale Blend reserve data is intentionally treated as fail-closed liveness behavior. Operators and keepers are expected to keep market and reserve data refreshed often enough for allocation/refresh workflows; proceeding with stale reserve data would be riskier than reverting and retrying after refresh.

20.55 REFRESHMARKETS MATERIALIZES ALL MARKETS BEFORE CALLER AUTHORIZATION

// INFORMATIONAL

Description

`refresh_markets_impl` materializes the full configured market list before completing caller authorization. An unauthenticated or unauthorized caller can force work proportional to market count before the action is rejected.

This is reassessed as **Low** because it is bounded by configured market count but should be avoidable.

Proof of Concept

Preconditions

- The vault has many markets configured.
- An unauthorized caller can call the refresh entrypoint.

Validation Sequence

1. Configure a large market list.
2. Call `RefreshMarkets` from an unauthorized caller.
3. Observe market-list materialization before final caller authorization.

Reproducing Locally

Copy Code

```
rg "refresh_markets_impl|current_market" contract/vault/soroban/src/contract/entrypoints.rs
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:M/I:N/D:N/Y:N (1.1)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Authorization should occur before market-list materialization.
- Refresh operations should support paging for large market sets.
- Tests should confirm unauthorized callers fail before expensive list processing.

Remediation Comment

ACKNOWLEDGED: The client reports that RefreshMarkets authorization now occurs before market materialization, so the pre-authorization materialization path described by this finding is not present on the current branch. Acknowledged as an informational/stale observation.

20.56 GOVERNANCE COMMAND BODIES ARE DECODED BEFORE AUTHORIZATION AND KIND VALIDATION

// INFORMATIONAL

Description

The vault decodes full governance command bodies, including potentially large lists such as restrictions or supply queues, before completing authorization and kind-specific validation. Unauthorized malformed calls can therefore consume work proportional to payload size before rejection.
This is reassessed as **Low** because the effect is bounded by transaction limits, but it creates unnecessary unauthenticated work.

Proof of Concept

Preconditions

- An unauthorized caller can submit encoded command payloads.

Validation Sequence

1. Construct a large governance command body.
2. Submit it from an unauthorized caller.
3. Observe that body decoding work occurs before final authorization rejection.

Reproducing Locally

Copy Code

```
rg "decode_command|execute\(" contract/vault/soroban/src/contract/entrypoints.rs contract/vault/soroban/
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:M/I:N/D:N/Y:N (1.1)

Recommendation

To mitigate this issue, the following improvements are recommended:

- The command tag should be decoded and authorized before decoding large command bodies.
- Per-command body length limits should be enforced before allocation.
- Malformed unauthorized payload tests should be added near maximum allowed sizes.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #427 moves governance authorization before command-body decoding for the affected paths so unauthorized callers cannot force pre-auth parsing/decoding of governance command bodies. Regression coverage includes auth-before-governance-

decode regressions in the consolidated governance control-plane PR #427; superseded standalone PR #442 was folded into PR #427.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/08bd07a3abbaf75fad2962df4ef811cdf3527280

20.57 BLEND ADAPTER ADMIN CANNOT BE ROTATED AFTER DEPLOYMENT

// INFORMATIONAL

Description

The Blend adapter stores its admin at construction and does not expose a rotation method. If the admin key is lost, compromised, or mistakenly configured, the only recovery path is redeploying the adapter and reconfiguring the vault's supply queue and allowed-adapter list.
This is reassessed as **Low** because it is primarily an operational resilience issue, but it materially complicates incident response for an adapter that controls vault integrations.

Proof of Concept

Preconditions

- The Blend adapter is deployed.
- The adapter admin key must be rotated or recovered.

Validation Sequence

1. Inspect the adapter constructor and public methods.
2. Confirm the admin is stored at deployment.
3. Confirm no `set_admin`, `propose_admin`, or `accept_admin` path exists.

Reproducing Locally

Copy Code

```
rg "set_admin|propose_admin|accept_admin|DataKey::Admin" contract/vault/soroban/blend-adapter/src/lib.rs
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:L/I:L/D:N/Y:N (0.7)

Recommendation

To mitigate this issue, the following improvements are recommended:

- A two-step `propose_admin` and `accept_admin` flow should be added.
- Both steps should emit structured events.
- The new admin should be validated against self-references and known dependent contract addresses.

Remediation Comment

SOLVED: Client successfully remediated this issue. The Blend adapter now exposes two-step admin rotation with pending-admin acceptance, event emission, and post-rotation authority enforcement. This allows

deployed adapter administration to be rotated without redeploying the adapter while preventing unilateral silent takeover.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/5b03b63038014bdc1866061c1e61a13f80ad65cf

20.58 GOVERNANCE ADMIN CANNOT BE ROTATED AFTER DEPLOYMENT

// INFORMATIONAL

Description

The governance contract stores the admin set during construction and does not expose a transfer or rotation path. Loss, compromise, or misconfiguration of the admin requires redeploying governance and reconfiguring the vault to point at the replacement contract.
This is reassessed as **Low** because it is an operational resilience issue, but the affected role controls critical proposal submission and recovery flows.

Proof of Concept

Preconditions

- The governance contract is deployed with an initial admin.

Validation Sequence

1. Inspect the governance constructor storage for `DataKey::Admin`.
2. Search the public API for admin transfer or rotation methods.
3. Confirm there is no callable path to replace the admin after deployment.

Reproducing Locally

Copy Code

```
rg "DataKey::Admin|set_admin|transfer_admin|propose_admin|accept_admin" contract/vault/soroban/governanc
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:L/I:L/D:N/Y:N (0.7)

Recommendation

To mitigate this issue, the following improvements are recommended:

- A two-step admin rotation flow should be added.
- The pending and accepted admin changes should emit events.
- Admin rotation should reject the governance contract itself, the vault address, and malformed role addresses.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix adds timelocked admin rotation with SetAdmin / submit_set_admin, admin action/timelock kinds, no-change rejection, DataKey::Admin update, and old-admin

revocation after acceptance.

Remediation Hash

https://github.com/Templar-Protocol/contracts/pull/448

20.59 SETMEMBERSHIP GOVERNANCE DECISIONS USE INCOMPLETE CURRENT-STATE CLASSIFICATION

// INFORMATIONAL

Description

`SetMembership` decision logic is evaluated without the real current membership state. The governance path can classify changes as raises or timelocked updates even when the actual direction differs, and downstream policy application may not match the operator's intent.
This is reassessed as **Low** because it affects governance correctness but requires authorized proposal flow.

Proof of Concept

Preconditions

- Governance can submit membership updates.

Validation Sequence

1. Submit a `SetMembership` proposal where the current membership differs from the governance cache assumption.
2. Compare the timelock decision with the real on-vault state transition direction.
3. Observe that classification is based on incomplete current-state information.

Reproducing Locally

Copy Code

```
cargo test -p templar-soroban-governance --test timelock_matrix -- --nocapture
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:L/I:L/D:N/Y:N (0.7)

Recommendation

To mitigate this issue, the following improvements are recommended:

- The governance contract should classify membership changes using the actual current on-vault state.
- No-op and downgrade paths should be handled explicitly.
- Regression tests should cover first-time membership, increases, decreases, and no-change cases.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix adopts conservative unknown-state handling: unknown membership state is timelocked, accepted actions mark state known, and duplicate clears become NoChange.

Remediation Hash

https://github.com/Templar-Protocol/contracts/pull/448

20.60 BLEND ADAPTER CONSTRUCTOR ACCEPTS UNVALIDATED VAULT POOL ASSET AND ADMIN ADDRESSES

// INFORMATIONAL

Description

The Blend adapter constructor stores the supplied vault, pool, asset-token, and admin addresses without validating role separation or probing that the addresses implement the expected interfaces. Typos or self-references are accepted at deployment and surface only on first use.
This is reassessed as **Low** because it is primarily a deploy-time misconfiguration risk.

Proof of Concept

Preconditions

- The Blend adapter constructor can be called with arbitrary addresses.

Validation Sequence

1. Deploy the adapter with a vault, pool, asset, or admin address that is semantically invalid.
2. Observe constructor success.
3. Trigger the first dependent operation and observe failure or misrouting later.

Reproducing Locally

Copy Code

```
cargo test -p templar-soroban-blend-adapter --test pool_or_vault_retargeting -- --nocapture
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:L/I:L/D:N/Y:N (0.7)

Recommendation

To mitigate this issue, the following improvements are recommended:

- The constructor should reject self-references and obviously invalid address combinations.
- Deployment should perform sanity probes against the vault and pool interfaces.
- Constructor configuration should be emitted in a structured event.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #431 validates Blend adapter constructor-bound admin, vault, and pool addresses before persisting config, so constructor/init cannot bind account-form or otherwise invalid addresses after retarget entrypoints were removed. Regression coverage includes constructor_rejects_non_contract_admin, constructor_rejects_non_contract_vault,

constructor_rejects_non_contract_pool, the full templar-soroban-blend-adapter test suite, and the Soroban size-budget gate.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/e951f6f65a67e8cf80cdaa8a18ae209eec3d1c8c

20.61 GOVERNANCE ABDICATION SYMBOLS DISABLE GROUPED SUBMIT METHODS RATHER THAN ONE ACTION

// INFORMATIONAL

Description

The governance `abdicate` function accepts broad symbol groups that can disable multiple submit methods at once. An operator intending to disable one action may accidentally disable unrelated governance surfaces grouped under the same symbol.
This is reassessed as **Low** because it is an operational governance footgun rather than a direct external exploit.

Proof of Concept

Preconditions

- The governance admin can call `abdicate`.

Validation Sequence

1. Call `abdicate` with a symbol that maps to a grouped submit surface.
2. Attempt each submit method in the group.
3. Observe that multiple submit methods are disabled by one symbol.

Reproducing Locally

Copy Code

```
cargo test -p templar-soroban-governance --test abdication_groups -- --nocapture
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:L/I:L/D:N/Y:N (0.7)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Governance should use one abdication symbol per submit method.
- Abdication events should list every method disabled by the call.
- Unknown or grouped symbols should be documented and covered by tests.

Remediation Comment

ACKNOWLEDGED: The client intentionally models governance abdication at grouped action-kind granularity; the broader grouped-submit disablement is accepted as the current design tradeoff rather than a defect. Acknowledged as an informational design decision.

20.62 SETMEMBERSHIP POLICY DEFAULTS MISSING MARKET_ID TO MARKET ZERO

// INFORMATIONAL

Description

The group-policy application path defaults a missing `market_id` to `0` for `SetMembership` instead of rejecting the malformed action. If an incomplete governance payload reaches this path, the policy update is applied to market zero.
This is reassessed as **Low** because it requires an authorized or malformed governance action but can affect the wrong market.

Proof of Concept

Preconditions

- A `SetMembership` policy action reaches the vault without a market id.

Validation Sequence

1. Construct or decode a membership update with missing `market_id`.
2. Dispatch it through the group-policy application path.
3. Observe that market `0` is used instead of rejecting the missing field.

Reproducing Locally

Copy Code

```
rg "SetMembership|unwrap_or\(0\)|market_id" contract/vault/soroban/src/contract/entrypoints.rs
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:L/I:L/D:N/Y:N (0.7)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Missing `market_id` should be treated as `InvalidInput`.
- Policy decoders should validate required fields before dispatch.
- Tests should cover malformed `SetMembership` payloads.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #441 requires SetMembership governance policy submissions to carry an explicit market_id for group membership updates instead of silently defaulting to

market zero. Regression coverage includes the SetMembership missing-market-id regression in the governance ABI validation PR and the governance package verification for PR #441.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/4fab9e08ab4f33df919c893a5bf1c475310a5ad4

20.63 GOVERNANCE CONSTRUCTOR DOES NOT REJECT SELF-REFERENTIAL OR COLLIDING ROLE ADDRESSES

// INFORMATIONAL

Description

The governance constructor does not enforce that admin, vault, sentinel, guardians, and the governance contract itself are distinct where required. Self-references or role collisions can be stored silently and later break governance execution or recovery paths.
This is reassessed as **Low** because it is a deployment/configuration issue.

Proof of Concept

Preconditions

- The governance constructor is called with role addresses.

Validation Sequence

1. Pass self-referential or colliding role addresses to the constructor.
2. Observe that the constructor stores them without role-separation validation.
3. Trigger the affected governance role later and observe failure or ambiguous authority.

Reproducing Locally

Copy Code

```
rg "__constructor|Admin|Vault|Sentinel|Guardians" contract/vault/soroban/governance/src/lib.rs
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:L/I:L/D:N/Y:N (0.7)

Recommendation

To mitigate this issue, the following improvements are recommended:

- The constructor should reject admin, vault, sentinel, and self-address collisions where they are nonsensical.
- Deployment tests should cover each invalid role combination.
- The constructor should emit the validated role configuration.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #451 adds governance constructor validation for self-reference and colliding role addresses so governance role topology cannot be initialized with ambiguous or

self-referential authorities. Regression coverage includes governance constructor collision regressions in PR #451 and the role-address topology verification suite.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/3c1b5f9ddbd7ada1ef4478e1ead5b2f8435493d8

20.64 NEWLY ATTACHED GOVERNANCE STARTS WITH DEFAULT LOCAL CACHE RATHER THAN VAULT STATE

// INFORMATIONAL

Description

When a new governance contract is attached to the vault, its local cached values for fees, caps, and similar state start from defaults. The next proposal classification may therefore be based on the cache rather than the real on-vault state, causing wrong immediate/timelocked decisions.
This is reassessed as **Low** because it is governance-authorized and primarily affects correctness of subsequent proposals.

Proof of Concept

Preconditions

- A new governance contract is attached to an existing vault.

Validation Sequence

1. Attach a fresh governance contract to a vault with non-default fees, caps, or memberships.
2. Submit the first proposal through the new governance contract.
3. Compare the decision with the actual vault state and observe cache-default classification risk.

Reproducing Locally

Copy Code

```
rg "cached_state|decide_submission|proxy_view" contract/vault/soroban/governance/src/lib.rs
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:L/I:L/D:N/Y:N (0.7)

Recommendation

To mitigate this issue, the following improvements are recommended:

- New governance instances should hydrate cache from `proxy_view` before accepting submissions.
- Direction classification should query the vault's current state where practical.
- First-submission tests should cover fees, caps, and membership after governance rotation.

Remediation Comment

SOLVED: Client successfully remediated this issue. Rather than hydrating a mutable admin cache from vault state, the fix tracks explicit known-state markers and forces timelocked review when the local cache is unmirrored or unknown.

Remediation Hash

https://github.com/Templar-Protocol/contracts/pull/448

20.65 ACCEPT_KIND IS AMBIGUOUS WHEN MULTIPLE PROPOSALS SHARE THE SAME ACTION KIND

// INFORMATIONAL

Description

The governance `accept_kind` helper accepts only an action kind. When multiple pending proposals of the same kind exist, for example cap updates for different markets, the accepted proposal is determined by queue ordering rather than an explicit proposal identifier.
This is reassessed as **Low** because it affects governance precision and operator safety rather than enabling direct theft.

Proof of Concept

Preconditions

- Two or more pending proposals share the same governance action kind.

Validation Sequence

1. Submit two proposals with the same kind and different targets.
2. Call `accept_kind` with only the kind.
3. Observe that selection depends on queue ordering rather than explicit proposal id.

Reproducing Locally

Copy Code

```
rg "accept_kind|revoke_kind|Proposal" contract/vault/soroban/governance/src/lib.rs contract/vault/soroba
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:L/I:L/D:N/Y:N (0.7)

Recommendation

To mitigate this issue, the following improvements are recommended:

- `accept_kind` should accept an explicit proposal id or kind-specific discriminator.
- Events should include the accepted proposal id and target fields.
- Tests should cover multiple simultaneous proposals of the same kind.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #448 changes accept_kind to count matching pending proposals and fail with DuplicatePending when a broad GovernanceActionKind is ambiguous instead of silently accepting one of several same-kind proposals. Regression coverage includes

accept_kind_rejects_ambiguous_broad_kind and the governance proposal identity/timelock semantics verification for PR #448.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/b2fe521557b326de6122936804a544756c4650ec

20.66 VAULT DISPATCH HELPERS ACCEPT EMPTY OR DUPLICATE INPUT LISTS

// INFORMATIONAL

Description

Several vault dispatch helpers accept empty lists, duplicate market ids, or duplicate addresses without rejecting the input. This can emit no-op or misleading events and create ambiguous configuration state. This is reassessed as **Low** because the inputs are governance-controlled but should be normalized and validated.

Proof of Concept

Preconditions

- A governance or config action accepts a list input.

Validation Sequence

1. Submit an empty list or a list containing duplicate markets/addresses.
2. Observe that the dispatcher accepts or stores the input without uniqueness validation.
3. Inspect emitted events or resulting storage for ambiguous no-op/duplicate state.

Reproducing Locally

```
rg "target_ids|accounts|many|Vec" contract/vault/soroban/src/contract/entrypoints.rs
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:L/I:L/D:N/Y:N (0.7)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Empty lists should be rejected where no-op updates are not meaningful.
- Duplicate market ids and duplicate addresses should be rejected during decode or before storage writes.
- Events should be emitted only after semantically valid changes.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #427 validates governance list inputs: duplicate set-like lists are rejected, while empty lists are accepted only for explicitly meaningful clear/unset semantics. Regression coverage includes duplicate-rejection and empty-list policy regressions in the consolidated governance control-plane PR #427; superseded standalone PR #444 was folded into PR #427.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/28e247055876e9a065dd97e7c87f6e94134917be

20.67 STORAGE DECODERS PRE-ALLOCATE VECTORS FROM UNTRUSTED LENGTH PREFIXES

// INFORMATIONAL

Description

Several storage decoders read length prefixes from serialized blobs and pre-allocate vectors before fully validating the rest of the blob. A malformed persisted blob can cause disproportionate memory allocation during decoding.
This is reassessed as **Low** because the blob must be persisted or introduced through migration/corruption, but it affects decoder robustness.

Proof of Concept

Preconditions

- A malformed persisted blob can be provided through migration, corruption, or test storage.

Validation Sequence

1. Construct a blob with an oversized length prefix and insufficient following bytes.
2. Call the relevant storage decoder.
3. Observe allocation pressure or rejection behavior before full validation.

Reproducing Locally

Copy Code

```
cargo test -p templar-soroban-runtime --features testutils --test adv_storage_blob_corruption -- --nocap
```

BVSS

AO:A/AC:H/AX:M/R:N/S:U/C:N/A:L/I:N/D:N/Y:N (0.6)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Decoder pre-allocation should be capped by protocol maximums before allocation.
- Length prefixes should be validated against remaining input bytes.
- Malformed length-prefix cases should remain covered by adversarial decoder tests.

Remediation Comment

SOLVED: Client successfully remediated this issue. Storage decoders now validate serialized length prefixes against the remaining bytes and protocol caps before allocating vectors. This removes the untrusted preallocation behavior while preserving canonical decoder behavior for valid payloads.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/2c806e7a7e9e2aa6d78d03434b922cccece6359b

20.68 TEST SUITE USES PERMISSIVE AUTHORIZATION PATHS THAT DIVERGE FROM PRODUCTION RBAC

// INFORMATIONAL

Description

The Soroban vault tests rely heavily on permissive helpers such as `SorobanAuth`, `mock_all_auths`, and always-mapped interpreters. Production uses strict role-based authorization through `RbacAuth`, with different failure behavior for missing roles and unmapped effect addresses.
This divergence causes production-only failures, such as RBAC denials or missing address-book mappings, to be underrepresented in the tests. The issue is a coverage and assurance gap rather than a direct exploit primitive, so it is reassessed as **Low**.

Proof of Concept

Preconditions

- The local test suite is available.
- Production authorization uses `RbacAuth`.

Validation Sequence

1. Locate tests that use `SorobanAuth`, `mock_all_auths`, or permissive address mapping.
2. Compare those tests with production entrypoints that construct strict RBAC.
3. Re-run authorization-sensitive tests using strict RBAC and observe behavior differences.

Reproducing Locally

Copy Code

```
rg "SorobanAuth|mock_all_auths|TestPermissiveAuth" contract/vault/soroban
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- A production-RBAC integration test track should be added and used for all authorization-sensitive behavior.
- Tests that intentionally use permissive helpers should be renamed and documented as test-only shortcuts.
- Critical invariant tests should be duplicated with strict `RbacAuth` and real address-book resolution.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix adds production-parity authorization coverage showing that permissive role policy does not bypass Soroban native require_auth. Tests now distinguish mocked role-policy failures from native authorization behavior.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/8d1fd6526eb30a9022486ecbc9f554188a88bdcd

20.69 ALLOCATION LIFECYCLE TEST HELPERS BYPASS PRODUCTION KERNEL ACTION HANDLING

// INFORMATIONAL

Description

Several allocation tests mutate kernel state directly through test helpers instead of driving the production `apply_kernel_action` path. These helpers can create states that production entrypoints would not create, hiding bugs in allocation start, completion, and effect interpretation.
This is reassessed as **Low** because it is a test-assurance issue rather than a live exploit path, although it affects coverage of asset movement logic.

Proof of Concept

Preconditions

- The runtime test utilities are available.

Validation Sequence

1. Inspect allocation helpers in `src/test_utils.rs`.
2. Compare helper state mutation with production `apply_kernel_action`.
3. Confirm tests can construct allocation states without executing production effect handling.

Reproducing Locally

```
rg "apply_kernel_action|allocation|principal|leased" contract/vault/soroban/src/test_utils.rs contract/v
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Allocation tests should drive production entrypoints and `apply_kernel_action` rather than mutating state directly.
- Any helper that intentionally bypasses production should be clearly marked and limited to narrow unit tests.
- A full deploy-to-allocate integration test should be added for every allocation branch.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix pins allocation helper behavior against production begin_allocation_internal behavior and adds direct supply allocation coverage for the production completion path. The tests no longer rely only on helper-only state transitions.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/e9cc5c3d31d176b0bf88df23d5be14660e15d1e7

20.70 VAULT USES STELLARASSETCLIENT FOR CUSTOM SHARE-TOKEN INTERACTIONS

// INFORMATIONAL

Description

The vault uses `StellarAssetClient` for both the underlying asset and the custom share-token contract. `StellarAssetClient` is intended for Stellar Asset Contracts, so using it against a custom token relies on signature compatibility that may not be stable across SDK versions.
This is reassessed as **Low** because it is a compatibility and integration fragility rather than an immediate exploit.

Proof of Concept

Preconditions

- The vault interacts with both a Stellar Asset Contract and the custom share-token contract.

Validation Sequence

1. Inspect the token adapter helper used by the vault.
2. Confirm `StellarAssetClient` is used for custom share-token calls.
3. Compare this with the generated or generic token client expected for custom contracts.

Reproducing Locally

Copy Code

```
rg "StellarAssetClient|SdkTokenAdapter" contract/vault/soroban/src/contract/helpers.rs
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:L/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- The custom share token should be called through the generic Soroban token client or a dedicated generated client.
- `StellarAssetClient` should be reserved for actual Stellar Asset Contracts.
- Cross-SDK compatibility tests should cover share-token mint and burn calls.

Remediation Comment

ACKNOWLEDGED: The vault intentionally interacts with the custom share token through the SEP-41-compatible StellarAssetClient surface. The share token exposes the compatible methods needed by the

vault, so this is an accepted ABI-shim/design choice rather than a protocol defect requiring a code change.

20.71 INTEGRATION TESTS SEED STATE OUTSIDE THE PRODUCTION ONBOARDING FLOW

// INFORMATIONAL

Description

Several E2E and integration tests pre-seed state and policy through helper APIs instead of running the production sequence of initialize, governance configuration, market addition, and allocation. First-run and onboarding bugs can therefore be missed.
This is reassessed as **Low** as an assurance gap.

Proof of Concept

Preconditions

- E2E and integration tests use fixtures.

Validation Sequence

1. Inspect fixture setup for direct state or policy seeding.
2. Compare against production onboarding through initialize, governance config, market addition, and allocation.
3. Identify first-run code paths that the fixtures bypass.

Reproducing Locally

Copy Code

```
rg "save_state|save_policy_state|initialize|add_market" contract/vault/soroban/tests
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- At least one full deploy-to-allocate test should use only production entrypoints.
- Pre-seeded helper paths should be marked as unit-only fixtures.
- Onboarding tests should assert emitted events and persisted config after each production step.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #447 adds integration coverage through the production initialize/onboarding path so tests no longer rely only on direct seeded state for this path.

Regression coverage includes the initialize onboarding-path regression in the test-assurance PR #447 and the associated runtime integration verification.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/e0d95e3bf914027f0ab25c4e1b8ca1760eb48fe7

20.72 TEST ALLOCATION HELPER DROPS LEASED TARGETS UNLIKE PRODUCTION ALLOCATION

// INFORMATIONAL

Description

A test helper used for allocation scenarios drops leased target accounting while production allocation preserves it. Tests that pass through this helper can validate states that production would not produce. This is reassessed as **Low** as a test-assurance issue.

Proof of Concept

Preconditions

- Allocation tests use the helper that simulates production allocation state.

Validation Sequence

1. Compare helper-written allocation state with production allocation state.
2. Focus on leased target accounting.
3. Observe that the helper drops leased targets that production preserves.

Reproducing Locally

Copy Code

```
rg "leased|allocation" contract/vault/soroban/src/test_utils.rs contract/vault/soroban/tests
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- The helper should mirror production allocation behavior exactly.
- Leased-target branches should be covered by production-entrypoint integration tests.
- Direct state mutation should be limited to narrow unit tests with explicit comments.

Remediation Comment

ACKNOWLEDGED: The client reports that the test allocation helper and production allocation now both filter leased targets, so the reported helper/production divergence is not present on the current branch. Acknowledged as an informational/stale test-assurance observation.

20.73 PROPERTY TESTS LABELED EXTERNAL GROWTH DO NOT MODEL REAL ADAPTER GROWTH

// INFORMATIONAL

Description

Some property tests labeled as external growth scenarios adjust kernel principal directly rather than simulating adapter-returned growth through the Soroban runtime. This misses bugs where adapter reports and vault accounting diverge.
This is reassessed as **Low** because it is a coverage issue, but it directly relates to adapter-accounting risks.

Proof of Concept

Preconditions

- Property tests labeled external growth are present.

Validation Sequence

1. Inspect the test body for direct principal manipulation.
2. Confirm no adapter return or token-balance delta is modeled.
3. Compare with a runtime adapter-backed growth scenario.

Reproducing Locally

Copy Code

```
rg "external growth|principal|total_assets" contract/vault/soroban/tests
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- External-growth tests should use adapter mocks and runtime calls.
- Assertions should compare token deltas, adapter reports, and kernel accounting.
- High-value growth and loss ranges should be included.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #447 clarifies and corrects the property-test coverage around external growth semantics so tests no longer claim to model real adapter growth when they do not. Regression coverage includes the external-growth property semantics regression/update in the test-assurance PR #447 and the associated property-test verification.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/3c8eb81de988284459eeffb84a7bc993d75830b1

20.74 PREVIEW AND FEE TESTS DUPLICATE PRODUCTION INTERNALS WITH DIFFERENT SEMANTICS

// INFORMATIONAL

Description

Several preview and fee tests reimplement production fee-accrual math inside the test harness. These duplicated internals can drift from production behavior and allow tests to pass while the deployed path behaves differently.
This is reassessed as **Low** because it is a test-quality issue over a security-sensitive accounting surface.

Proof of Concept

Preconditions

- Preview and fee tests are available.

Validation Sequence

1. Inspect tests for locally reimplemented fee or preview math.
2. Compare the helper logic with production preview paths.
3. Confirm tests can pass even if production storage/time semantics diverge.

Reproducing Locally

Copy Code

```
rg "preview_state_with_fee_accrual|fee|expected" contract/vault/soroban/tests
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Preview and fee tests should call production `proxy_view` and entrypoints rather than duplicating math.
- Shared helpers should be restricted to expected-value construction only.
- Regression tests should pin edge cases through the deployed runtime path.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #447 routes preview/fee coverage through the production proxy-view fee conversion path instead of duplicating production internals with divergent test-only semantics. Regression coverage includes the proxy-view fee conversion path regression in PR #447 and the associated runtime/proxy-view test verification.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/1d254e7992ac25043d10073822366800d86a3501

20.75 SOROBAN KERNEL PARITY TESTS SKIP SOROBAN-SPECIFIC PATHS AND HIGH-VALUE RANGES

// INFORMATIONAL

Description

Parity tests focus primarily on kernel behavior and do not sufficiently exercise Soroban-specific encoding, address mapping, and high-value arithmetic ranges near `u128` boundaries. Soroban integration bugs can therefore remain outside the parity harness.
This is reassessed as **Low** because it affects assurance coverage.

Proof of Concept

Preconditions

- Kernel/Soroban parity tests are available.

Validation Sequence

1. Inspect test ranges and paths exercised by parity tests.
2. Confirm Soroban-specific storage/address paths are underrepresented.
3. Confirm high-value arithmetic near `u128::MAX` is not covered broadly.

Reproducing Locally

Copy Code

```
rg "proptest|u128::MAX|SorobanStorage|AddressMap" contract/vault/soroban/tests
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Parity tests should include Soroban storage codecs and address mapping.
- Arithmetic domains should include values near `u128::MAX`.
- Failures should compare kernel-only and Soroban-runtime outputs side by side.

Remediation Comment

SOLVED: The remediation routes a high-value balance case through the production `proxy_view` fee-aware conversion path (shared with the preview/fee test remediation), which adds some high-value coverage.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/d7cb3d2d3e81f0489bfc7f00f2f51f220983011d

20.76 GOVERNANCE MOCKVAULT IGNORES INVALID OR INCOMPLETE COMMAND PAYLOADS

// INFORMATIONAL

Description

The governance test `MockVault` accepts arbitrary command bytes and silently no-ops on unknown tags. Governance tests can therefore pass even when the governance contract emits malformed or incomplete vault command payloads.
This is reassessed as **Low** because it is a test harness defect, but it affects command-encoding coverage.

Proof of Concept

Preconditions

- Governance tests use `MockVault`.

Validation Sequence

1. Send malformed or unknown command bytes to the mock.
2. Observe that the mock ignores or no-ops instead of rejecting like the real vault.
3. Compare with real vault command decoding behavior.

Reproducing Locally

Copy Code

```
rg "MockVault|decode|Unknown|Invalid" contract/vault/soroban/governance/tests
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- The mock vault should reject unknown command tags and malformed payloads.
- Governance tests should decode and assert every emitted command payload.
- At least one governance test suite should call the real vault contract instead of the mock.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #447 adds shared-types decoder coverage for malformed/incomplete governance command payloads so the test harness no longer masks invalid command bodies. Regression coverage includes the malformed command payload rejection tests in PR #447 and the shared-types/governance test verification.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/31137e267e8d96949e78e1e31e7e8dcb647d0d89

20.77 PROXY_VIEW CASTS U128 VALUES TO I128 WITHOUT BOUNDS CHECKS

// INFORMATIONAL

Description

Several `proxy_view` fields cast `u128` totals into `i128` without checking that the value is within `i128::MAX`. Extremely large but representable vault values can appear negative or otherwise malformed to off-chain consumers.

This is reassessed as **Informational** because the value range is unlikely in normal operation, but view correctness should remain explicit.

Proof of Concept

Preconditions

- Vault values can be constructed above `i128::MAX` in a boundary test.

Validation Sequence

1. Seed or construct a proxy-view state with `u128` totals above `i128::MAX`.
2. Call the proxy-view method that casts to `i128`.
3. Observe wrap, negative representation, or missing overflow signal.

Reproducing Locally

Copy Code

```
rg "as i128|try_into|proxy_view" contract/vault/soroban/src
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- View casts should use checked conversion or saturation with an explicit flag.
- The expected numeric ranges should be documented.
- Boundary tests should cover values above `i128::MAX`.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #446 changes proxy_view fee snapshot conversions to use checked u128-to-i128 conversion and fail closed instead of silently casting overflowing values.

Regression coverage includes fee_snapshot_rejects_i128_overflow and proxy_view fee snapshot conversion regressions, full integration coverage for proxy_view, and the Soroban size gate.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/5842847b8f3549da0ee5accd98f95d261a630e19

20.78 GOVERNANCE IS GRANTED FULL CURATOR ROLE INSTEAD OF THE NARROWER POLICY-ADMIN NEED

// INFORMATIONAL

Description

The vault grants the governance contract full `Role::Curator` authority even though its intended call surface mainly requires policy administration. The extra role power expands the blast radius of governance compromise or misconfiguration.
This is reassessed as **Informational** because governance is already a highly trusted component, but least privilege should be preserved.

Proof of Concept

Preconditions

- The vault bootstrap loads governance roles.

Validation Sequence

1. Inspect `load_vault_bootstrap`.
2. Observe governance is granted `Role::Curator`.
3. Compare the granted role with the narrower policy-admin actions governance needs.

Reproducing Locally

```
rg "add_role\(governance|Role::Curator|load_vault_bootstrap" contract/vault/soroban/src/contract/helpers
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Governance should be granted the minimum role required for each action class.
- Curator and policy-admin privileges should be split where practical.
- Role-grant tests should assert the intended least-privilege matrix.

Remediation Comment

SOLVED: Client successfully remediated this issue. The bootstrap Curator grant to governance was removed; governance policy execution is kept behind execute_governance identity checks plus internal preauthorized mutation paths, and direct policy-admin calls remain Curator-gated.

Remediation Hash

https://github.com/Templar-Protocol/contracts/pull/448

20.79 FEE RECIPIENT CONFIGURATION IS POSITIONAL AND CAN BE MISWIRED BY REORDERING

// INFORMATIONAL

Description

Fee recipients are configured by position. Reordering the list can silently associate fee roles with different recipient addresses, creating a configuration footgun similar to the adapter-position issue but with lower direct impact.
This is reassessed as **Informational** because governance controls the configuration and fee-recipient miswiring is operationally detectable.

Proof of Concept

Preconditions

- Governance can configure fee recipients.

Validation Sequence

1. Configure multiple fee recipients.
2. Reorder the recipient list without changing intended roles.
3. Observe that position, not explicit role binding, determines recipient meaning.

Reproducing Locally

Copy Code

```
rg "recipient|fees|apply_fees_policy" contract/vault/soroban/src/contract/entrypoints.rs
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Fee recipients should be represented as explicit role-to-address mappings.
- Reorders should emit old and new role bindings.
- Duplicate and missing recipient roles should be rejected.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fee-recipient mapping is now covered by explicit ordering semantics and regressions so performance and management recipients cannot silently swap without a failing test. This makes the compact positional ABI intentional and auditable.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/aeba3fc12c70750e02ab1b11ae0a036e938ba812

20.80 ADDRESS TEXT CONVERSION CAN PANIC OR SILENTLY FALL BACK ON MALFORMED INPUTS

// INFORMATIONAL

Description

Address conversion helpers rely on string round-trips. Malformed input can panic during parsing, while some formatting paths return a silent empty string. These behaviors make event payloads and configuration conversion fragile.
This is reassessed as **Informational** because malformed address text generally indicates operator or integration error.

Proof of Concept

Preconditions

- Malformed or non-canonical address text reaches address conversion helpers.

Validation Sequence

1. Pass malformed address text into conversion.
2. Observe panic or error behavior.
3. Pass edge-case address values through formatting and observe silent empty fallback where applicable.

Reproducing Locally

Copy Code

```
rg "address_from_alloc_string|address_to_alloc_string|from_str" contract/vault/soroban/src
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:L/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Address conversion helpers should return `Result` or `Option` rather than panicking or returning empty strings.
- Malformed input should be rejected with `InvalidInput`.
- Round-trip tests should include malformed and non-canonical strings.

Remediation Comment

ACKNOWLEDGED: The client reports that malformed address-text ingress is validated in the current code path, so the reported panic or silent fallback behavior is not present on the current branch. Acknowledged as

an informational/stale observation.

20.81 FEE PREVIEW SATURATES SUPPLY VALUES WITHOUT SURFACING AN OVERFLOW SIGNAL

// INFORMATIONAL

Description

`preview_state_with_fee_accrual` can saturate intermediate supply values when constructing proxy-view output. Callers cannot distinguish a legitimate near-cap value from a saturated preview.
This is reassessed as **Informational** because it affects view precision rather than state mutation.

Proof of Concept

Preconditions

- A boundary state can produce saturated preview math.

Validation Sequence

1. Seed supply values near the saturation boundary.
2. Call fee preview/proxy view.
3. Observe that the returned value gives no explicit saturation flag.

Reproducing Locally

Copy Code

```
rg "saturating|preview_state_with_fee_accrual|proxy_view" contract/vault/soroban/src contract/vault/soro
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Saturation should be returned as an explicit error or status field.
- View consumers should be able to distinguish real values from capped values.
- Boundary tests should assert the saturated-state signal.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #446 makes fee-preview simulation fail closed when supply arithmetic would overflow instead of saturating and returning a misleading preview. Regression coverage includes fee_preview_rejects_supply_overflow, proxy-view preview overflow regressions, full integration coverage for proxy_view, and the Soroban size gate.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/35b3685e726befc8d078293b5eeaed30617e5595

20.82 TEST-ONLY SOROBANAUTH CREATES MISLEADING PRODUCTION AUTHORIZATION COVERAGE

// INFORMATIONAL

Description

`SorobanAuth` is a test-only authorization adapter, but tests using it can appear to cover production authorization behavior. This creates a false sense of coverage around paths that use strict `RbacAuth` in deployment.
This is reassessed as **Informational** and is related to the broader permissive-auth test finding.

Proof of Concept

Preconditions

- Tests use the test-only `SorobanAuth` helper.

Validation Sequence

1. Find tests that instantiate `SorobanAuth`.
2. Compare with production `RbacAuth` wiring.
3. Confirm helper behavior should not be treated as production authorization coverage.

Reproducing Locally

Copy Code

```
rg "SorobanAuth" contract/vault/soroban
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- The helper should be renamed to make its test-only role explicit.
- Production authorization tests should use `RbacAuth`.
- Documentation should explain when permissive auth helpers are acceptable.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #447 separates test-only SorobanAuth behavior from role-failure coverage so authorization tests no longer imply production RBAC coverage from permissive mocks alone. Regression coverage includes the auth-mocking/role-failure separation tests in PR #447 and the associated test-assurance verification.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/75edc6c127ba0548d1bbb90180386a34a21d0723

20.83 EVENT PAYLOADS OMIT STRUCTURED ADDRESSES AND TYPED TOPICS ACROSS VAULT COMPONENTS

// INFORMATIONAL

Description

Several vault, governance, share-token, and adapter events emit raw kernel-hash addresses, use single generic topics, or expose raw enum payloads. Off-chain indexers must reconstruct context from non-uniform payloads.
This is reassessed as **Informational** because it affects observability and monitoring rather than contract state directly.

Proof of Concept

Preconditions

- Events are emitted by vault, governance, share-token, or adapter actions.

Validation Sequence

1. Execute representative admin, kernel, governance, and token actions.
2. Inspect emitted topics and payloads.
3. Observe raw kernel-address values, single-topic events, or unversioned payloads where present.

Reproducing Locally

Copy Code

```
cargo test -p templar-soroban-runtime --features testutils --test event_integrity -- --nocapture
cargo test -p templar-soroban-governance --test event_integrity -- --nocapture
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Events should use structured topics and versioned payloads.
- Kernel addresses should be resolved to Soroban addresses where possible before emission.
- Indexing tests should assert that each important action can be reconstructed from events alone.

Remediation Comment

ACKNOWLEDGED: The compact event shape is an intentional observability and WASM-size tradeoff. Kernel events use versioned compact bytes with minimal topics, while admin/allocation/pause/governance events

remain intentionally lightweight rather than expanding every event into structured address topics.

20.84 ASSET AND SHARE-TOKEN ADMIN ASSUMPTIONS ARE ENFORCED ONLY BY DEPLOYMENT DISCIPLINE

// INFORMATIONAL

Description

The deployment flow relies on operators to bind the share-token admin to the vault and configure the asset-token admin correctly. These assumptions are not enforced by an on-chain probe during vault initialization. This is reassessed as **Informational** because it is a deployment assurance issue, but misconfiguration can break mint, burn, or asset-admin flows.

Proof of Concept

Preconditions

- The deployment flow configures asset and share-token admins.

Validation Sequence

1. Deploy the vault and token contracts.
2. Query or inspect expected admin bindings.
3. Confirm the deployment scripts, not vault initialization, enforce those assumptions.

Reproducing Locally

```
rg "set_admin|asset.*admin|share.*admin|initialize" justfile contract/vault/soroban
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:L/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Deployment scripts should assert that share-token and asset-token admins match expected addresses.
- Vault initialization should perform sanity checks where interface support allows.
- Deployment output should include an explicit admin-binding checklist.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #451 binds share-token authority to the expected vault/admin topology so asset and share-token admin assumptions are enforced by contract checks rather than deployment discipline alone. Regression coverage includes share-token vault-binding/admin regressions in PR #451 and the role-address topology verification suite.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/0a0e7b0fb45e62ddcb2b68a5eda4b3cd262d543d

20.85 PROXY_VIEW FEE SNAPSHOT OMITS THE CONFIGURED FEE GROWTH-RATE CAP

// INFORMATIONAL

Description

The proxy-view fee snapshot exposes configured fee parameters but does not include the `max_growth_rate` cap that constrains fee accrual. Integrators reading the view cannot reconstruct the full fee policy from the snapshot.
This is reassessed as **Informational**.

Proof of Concept

Preconditions

- Proxy-view fee snapshot is queried by an integrator.

Validation Sequence

1. Configure a fee growth-rate cap.
2. Call the proxy-view fee snapshot.
3. Observe that the cap is not represented in the returned snapshot fields.

Reproducing Locally

Copy Code

```
rg "fee_snapshot|max_growth|growth_rate" contract/vault/soroban/src/contract/proxy_view.rs contract/vaul
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- The fee snapshot should include the configured growth-rate cap.
- Documentation should specify every field required to reconstruct fee behavior.
- View tests should assert snapshot completeness.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #446 includes the configured fee growth-rate cap in the proxy_view fee snapshot so the API accurately reports the cap applied by fee math. Regression coverage includes the proxy_view fee-growth-cap regression, full proxy-view integration verification, and the Soroban size gate.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/863a2b62aec29e0bb039272586f07f578b99d985

20.86 GOVERNANCE ABDICATE ACCEPTS UNKNOWN SYMBOLS AS SILENT NO-OPS

// INFORMATIONAL

Description

`abdicate` accepts arbitrary symbols. Typos or unsupported symbols silently do nothing, so the caller may believe a governance surface was disabled when no change occurred.
This is reassessed as **Informational**.

Proof of Concept

Preconditions

- The governance admin can call `abdicate`.

Validation Sequence

1. Call `abdicate` with an unknown or misspelled symbol.
2. Observe successful/no-error behavior.
3. Confirm no submit method was disabled.

Reproducing Locally

Copy Code

```
cargo test -p templar-soroban-governance --test abdication_groups -- --nocapture
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Unknown abdication symbols should be rejected.
- Successful abdication events should list the affected methods.
- Tests should cover typos and unsupported symbols.

Remediation Comment

ACKNOWLEDGED: The client reports that governance abdication now uses typed action values rather than accepting arbitrary unknown symbols as silent no-ops; we confirmed that abdicate takes a typed GovernanceActionKind on the integrated branch. Acknowledged as an informational/stale observation.

20.87 OTHER EFFECT APPROVALS ARE UNREACHABLE FROM THE DEPLOYED VAULT COMMAND SURFACE

// INFORMATIONAL

Description

The `Other` effect handler exists in the runtime, but no deployed `VaultCommand` path can currently produce it. Dead effect code increases maintenance risk and may expose unexpected behavior if future commands make it reachable without fresh review.
This is reassessed as **Informational**.

Proof of Concept

Preconditions

- The deployed `VaultCommand` set is reviewed against effect handlers.

Validation Sequence

1. Search for construction of `Other` effects.
2. Compare with deployed command variants.
3. Confirm `consume_other` is unreachable from current command dispatch.

Reproducing Locally

Copy Code

```
rg "consume_other|Effect::Other|VaultCommand" contract/vault/soroban/src contract/vault/soroban/shared-t
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Unused effect handlers should be removed until needed.
- If retained, they should be gated behind explicit command coverage and tests.
- Future command additions should include a reachability review for existing dead handlers.

Remediation Comment

ACKNOWLEDGED: The client reports that unreachable/unsupported effect approvals are not exposed through the deployed vault command surface and unsupported effects fail closed rather than being silently accepted. Acknowledged as an informational/stale observation.

20.88 REVOKE_KIND CANCELS ALL PROPOSALS OF A KIND WHILE ACCEPT_KIND ACCEPTS ONLY ONE

// INFORMATIONAL

Description

`revoke_kind` revokes every pending proposal of a given kind, while `accept_kind` accepts only one proposal. This asymmetry can surprise operators and cancel more governance work than intended.
This is reassessed as **Informational**.

Proof of Concept

Preconditions

- Multiple proposals of the same kind are pending.

Validation Sequence

1. Submit multiple pending proposals sharing one action kind.
2. Call `revoke_kind`.
3. Observe that all proposals of that kind are revoked, while `accept_kind` accepts only one.

Reproducing Locally

Copy Code

```
cargo test -p templar-soroban-governance --test event_integrity d12_governance_event_revoke_kind_emits_o
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- `revoke_kind` should accept a proposal id or a kind-specific discriminator.
- Revocation should emit one event per cancelled proposal.
- Bulk revocation should require an explicit bulk flag.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #448 changes revoke_kind to reject zero matches with ProposalNotFound and multiple broad-kind matches with DuplicatePending, revoking only the single unambiguous matching proposal. Regression coverage includes revoke_kind_rejects_ambiguous_broad_kind and governance proposal identity/timelock semantics verification for PR #448.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/2506f1381b8cc76c43af8229df3313a30579243f

20.89 VAULT COMMAND WIRE CODEC LACKS VERSIONING AND COLLAPSES DECODER ERROR VARIANTS

// INFORMATIONAL

Description

The vault command wire codec has no explicit version field, and decoder errors collapse into generic invalid-input behavior. This makes future upgrades harder to reason about and hides which command or field failed to decode.
This is reassessed as **Informational**.

Proof of Concept

Preconditions

- Encoded vault commands are consumed by the vault or governance contract.

Validation Sequence

1. Inspect the wire codec for an explicit version byte.
2. Submit malformed command payloads.
3. Observe that decoder errors collapse to generic invalid-input behavior.

Reproducing Locally

Copy Code

```
rg "decode_command|encode_command|InvalidInput|version" contract/vault/soroban/shared-types/src contract
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- A codec version byte should be prepended to command and result payloads.
- Decoder errors should preserve command and field context.
- Forward and backward compatibility tests should be added for versioned payloads.

Remediation Comment

ACKNOWLEDGED: No production code change is planned for this item. The compact command byte codec is treated as an internal client/runtime transport rather than a published long-term external ABI, and the contract boundary intentionally maps malformed client payloads to the compact InvalidInput error surface while the shared-types decoder still preserves distinct internal error variants.

20.90 MULTIPLE WIRE ENCODINGS CAN REPRESENT THE SAME GOVERNANCE POLICY ACTION

// INFORMATIONAL

Description

Some vault command variants accept optional fields whose absence, ordering, or default values can produce different byte encodings for the same logical action. Off-chain systems that hash raw command bytes for replay protection or review may treat equivalent actions as distinct.
This is reassessed as **Informational** because no vulnerable replay verifier was identified in-scope, but the encoding should be canonical.

Proof of Concept

Preconditions

- Off-chain systems hash or review raw command bytes.

Validation Sequence

1. Construct two byte encodings that decode to the same logical governance action through optional/default fields.
2. Confirm both decode successfully.
3. Observe that raw-byte hashes differ while semantic action is equivalent.

Reproducing Locally

Copy Code

```
rg "Option|encode|decode|VaultCommand" contract/vault/soroban/shared-types/src/lib.rs contract/vault/sor
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- The wire codec should define and enforce canonical encoding.
- Replay protection should hash decoded semantic commands rather than raw bytes where possible.
- Tests should reject non-canonical encodings for the same logical governance action.

Remediation Comment

SOLVED: Client successfully remediated this issue. SetGovernancePolicy dispatch now rejects fields that are irrelevant to the selected policy kind before applying the policy. Sparse governance policy actions therefore

no longer accept multiple byte encodings for the same effective action.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/1e1901788bf06acd5207a4bfbac6582ed655edaf

20.91 GOVERNANCE DIRECTION CLASSIFIERS INCONSISTENTLY DETECT NO-CHANGE PROPOSALS

// INFORMATIONAL

Description

Some governance action variants do not consistently detect no-op updates, such as setting fees or caps to their current value. These proposals can enter the queue or consume governance flow even though they do not change vault state.
This is reassessed as **Informational**.

Proof of Concept

Preconditions

- Governance can submit same-value or no-op proposals.

Validation Sequence

1. Read the current value for a governance-controlled setting.
2. Submit a proposal that sets the same value again.
3. Observe whether it is classified as `NoChange` or enters the governance flow.

Reproducing Locally

Copy Code

```
rg "NoChange|decide_submission|SetFees|SetCap" contract/vault/soroban/governance/src/lib.rs contract/vau
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Every direction classifier should implement a `NoChange` early return.
- First-time cap classification should be handled explicitly.
- Tests should cover same-value proposals for every governance action variant.

Remediation Comment

SOLVED: Client successfully remediated this issue. The fix adds classifier coverage for admin no-change, cap/group-cap/relative-cap first-set vs raise vs mirrored state, membership assignment/removal, and broad-kind ambiguity; unknown state is consistently timelocked.

Remediation Hash

https://github.com/Templar-Protocol/contracts/pull/448

20.92 LOAD_TIMELOCKS CAN MUTATE STORAGE DURING VIEW-STYLE PATHS

// INFORMATIONAL

Description

`load_timelocks` may write fallback state on cache miss, including from paths that operators expect to be read-only. This makes view-style behavior depend on storage mutation side effects.
This is reassessed as **Informational**.

Proof of Concept

Preconditions

- Timelock storage is missing and a view-style path calls `load_timelocks`.

Validation Sequence

1. Set up storage without the per-kind timelock map.
2. Call a path expected to be read-only.
3. Observe storage mutation due to fallback cache write.

Reproducing Locally

Copy Code

```
rg "load_timelocks|storage\(\)\.instance\(\)\.set" contract/vault/soroban/governance/src/lib.rs
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Read paths should not mutate timelock storage.
- Migration or initialization should populate missing maps explicitly.
- Tests should assert view-style calls leave storage unchanged.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #436 makes load_timelocks read-only on legacy fallback by moving materialization to initialization/migration paths instead of view-style getters. Regression coverage includes timelock_getters_do_not_materialize_missing_timelocks_storage and the governance timelock-model verification for PR #436.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/0358c07b9340ad820092dc8918a79238b8a241ec

20.93 SHARE-TOKEN BURN AND BURN_FROM EMIT INDISTINGUISHABLE BURN EVENTS

// INFORMATIONAL

Description

`burn` and `burn_from` both emit the same burn event shape. Off-chain consumers cannot distinguish a direct burn from an allowance-based burn, reducing audit-trail clarity.
This is reassessed as **Informational**.

Proof of Concept

Preconditions

- Both direct burns and allowance burns are executed.

Validation Sequence

1. Call `burn` and inspect the emitted event.
2. Call `burn_from` and inspect the emitted event.
3. Observe that both event shapes are indistinguishable for indexers.

Reproducing Locally

Copy Code

```
rg "burn_from|burn|event" contract/vault/soroban/share-token/src/lib.rs contract/vault/soroban/share-tok
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvement is recommended:
`burn_from` should emit a distinct event that includes the spender and allowance context.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #429 keeps the standard burn event and emits supplemental delegated-burn observability for burn_from so delegated burns are distinguishable without breaking SEP-41 compatibility. Regression coverage includes burn_from_emits_supplemental_spender_event and the consolidated share-token PR #429 verification.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/3c5d098e3a38736341f4fa8135736848ebc42f5c

20.94 SHARE-TOKEN CONTRACT-ADDRESS VALIDATION RELIES ON STRKEY PREFIX INSPECTION

// INFORMATIONAL

Description

The share-token helper distinguishes contract addresses from account addresses by inspecting the StrKey prefix. SDK-level predicates are safer and less brittle if address encodings change.
This is reassessed as **Informational**.

Proof of Concept

Preconditions

- The share-token validates whether a supplied address is a contract.

Validation Sequence

1. Inspect `require_contract_address`.
2. Confirm it relies on StrKey textual prefix inspection.
3. Compare with SDK-level contract-address predicates.

Reproducing Locally

Copy Code

```
rg "require_contract_address|to_string|starts_with" contract/vault/soroban/share-token/src/lib.rs
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvement is recommended:
The helper should use an SDK-supported contract-address predicate rather than string-prefix inspection.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #429 replaces StrKey text-prefix contract-address checks with typed address-payload validation for share-token vault/admin binding. Regression coverage includes constructor_rejects_account_vault_address, set_vault_rejects_account_address, and the consolidated share-token PR #429 verification.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/3c5d098e3a38736341f4fa8135736848ebc42f5c

20.95 IN-FLIGHT GOVERNANCE PROPOSALS RETAIN SUBMIT-TIME TIMELOCK AFTER LATER TIMELOCK RAISES

// INFORMATIONAL

Description

A proposal submitted under an older timelock retains its original maturity even if governance later raises the relevant timelock. Operators may expect a stricter newly-raised policy to apply to pending proposals, but the implementation uses submit-time timing.
This is reassessed as **Informational** because it may be an intended policy choice but should be explicit.

Proof of Concept

Preconditions

- A proposal is pending under an existing timelock.

Validation Sequence

1. Submit a timelocked proposal.
2. Raise the corresponding timelock before the proposal matures.
3. Attempt acceptance at the originally scheduled maturity.
4. Observe whether the proposal retains submit-time timing.

Reproducing Locally

Copy Code

```
rg "submitted_at|eta|accept|timelock" contract/vault/soroban/governance/src/lib.rs contract/vault/soroba
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- The timelock policy for pending proposals should be documented explicitly.
- If stricter later timelocks should apply, maturity should be recomputed at acceptance.
- Events should expose the timelock value captured by each proposal.

Remediation Comment

ACKNOWLEDGED: The protocol intentionally uses submit-time timelock semantics for already-submitted proposals. Later timelock raises do not retroactively reprice in-flight proposals; PR #448 records proof coverage for this policy. Acknowledged as an informational design decision.

Remediation Hash

https://github.com/Templar-Protocol/contracts/pull/448

20.96 SHARE-TOKEN ARCHIVAL RECOVERY PROCEDURE IS UNDEFINED

// INFORMATIONAL

Description

The share-token contract and documentation do not define an explicit recovery procedure for instance archival. Restoration depends on platform-level archival semantics and operator response rather than contract-defined behavior.
This is reassessed as **Informational**.

Proof of Concept

Preconditions

- The share-token instance approaches archival.

Validation Sequence

1. Review contract methods and documentation for archival recovery steps.
2. Confirm no contract-defined recovery or warning flow exists.
3. Confirm recovery depends on platform/operator archival handling.

Reproducing Locally

Copy Code

```
rg "archive|archival|restore|ttl" contract/vault/soroban/share-token
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:L/I:N/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Archival recovery expectations should be documented.
- Operators should receive TTL-warning events or monitoring guidance.
- Tests should cover minimum remaining TTL and recovery automation assumptions.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #445 documents the share-token archival recovery procedure: keepers/administrators can use instance TTL maintenance for contract state while holder balances and allowances remain owner-driven persistent entries. Regression coverage includes the share-token TTL/archival runbook update and PR #445 verification for TTL maintenance behavior.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/e189e24fc06f01cd0a1b45988d5984aa0ee5bffa

20.97 SHARE-TOKEN TESTS OMIT ADMIN ROTATION AND ALLOWANCE SECURITY COVERAGE

// INFORMATIONAL

Description

The share-token tests historically emphasized happy-path minting and burning while omitting admin rotation edge cases, allowance overflows, and unauthorized delegated burns. The hardening pass added coverage, but the original suite did not protect these paths.
This is reassessed as **Informational**.

Proof of Concept

Preconditions

- The original share-token unit tests are reviewed.

Validation Sequence

1. Search for admin rotation, vault rebind, allowance expiry, and unauthorized burn tests.
2. Compare against the privileged share-token methods.
3. Confirm coverage gaps in the original suite.

Reproducing Locally

```
rg "set_admin|set_vault|allowance|burn_from|unauthorized" contract/vault/soroban/share-token/src contrac
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Admin rotation, vault rebind, allowance expiry, and delegated burn cases should remain part of CI.
- Tests should assert event emission for every privileged change.
- Unauthorized and malformed allowance paths should be fuzzed.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #429 adds share-token admin rotation and allowance/security regressions, including old-admin rejection and direct-caller burn_from protections. Regression coverage includes set_admin_rotates_admin,

direct_caller_cannot_burn_from_even_with_allowance, allowance expiry/transfer-auth/vault-invoker regressions, and consolidated share-token PR #429 verification.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/3c5d098e3a38736341f4fa8135736848ebc42f5c

20.98 SHARE-TOKEN READ-ONLY METHODS DO NOT REFRESH INSTANCE TTL

// INFORMATIONAL

Description

Share-token read-only methods such as balance and allowance queries do not refresh the contract instance TTL. A contract that receives only read traffic can continue aging toward archival.
This is reassessed as **Informational**.

Proof of Concept

Preconditions

- The share-token receives read-only traffic but no write traffic.

Validation Sequence

1. Inspect read-only methods such as balance and allowance queries.
2. Confirm they do not call instance TTL extension.
3. Observe that read traffic alone does not refresh instance TTL.

Reproducing Locally

```
rg "pub fn balance|pub fn allowance|extend_instance_ttl" contract/vault/soroban/share-token/src/lib.rs
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:L/I:N/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- TTL refresh behavior for read methods should be documented.
- Read methods may refresh TTL under a configurable threshold if the platform allows.
- Operator automation should refresh TTL for dormant but active tokens.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #445 refreshes share-token instance TTL from SEP-41 read-only methods before returning, preventing read-only calls from leaving the instance TTL stale. Regression coverage includes read_only_entrypoints_cover_share_token_ttl_maintenance_surface and PR #445 share-token TTL verification.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/e189e24fc06f01cd0a1b45988d5984aa0ee5bffa

20.99 SHARE-TOKEN CONSTRUCTOR EMITS NO DEPLOYMENT CONFIGURATION EVENT

// INFORMATIONAL

Description

The share-token constructor stores admin, vault, name, symbol, and decimals without emitting a structured event. Off-chain consumers must query fields individually to reconstruct deployment state.
This is reassessed as **Informational**.

Proof of Concept

Preconditions

- The share-token constructor is called during deployment.

Validation Sequence

1. Deploy the share-token contract.
2. Inspect emitted events.
3. Observe no deployment/configuration event containing admin, vault, and metadata.

Reproducing Locally

Copy Code

```
rg "__constructor|events\(\)|publish" contract/vault/soroban/share-token/src/lib.rs contract/vault/sorob
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvement is recommended:
The constructor should emit a versioned deployment event containing admin, vault, metadata, and decimals.

Remediation Comment

SOLVED: Client successfully remediated this issue. The share-token constructor now emits a deployment configuration event carrying the initial admin, vault, metadata, and decimals. This gives indexers and auditors a canonical deployment-time configuration record.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/5b03b63038014bdc1866061c1e61a13f80ad65cf

20.100 SHARE-TOKEN EXTEND_TTL DOES NOT REFRESH HOLDER BALANCES OR ALLOWANCES

// INFORMATIONAL

Description

The share-token `extend_ttl` method refreshes the contract instance but not per-holder balance entries or allowance entries. Long-lived balances and approvals can age out even while the instance remains alive. This is reassessed as **Informational** because it is a storage-lifecycle risk, similar to the vault address-book TTL issue already reported.

Proof of Concept

Preconditions

- Share-token balances or allowances exist for dormant holders/spenders.

Validation Sequence

1. Call share-token `extend_ttl`.
2. Inspect which storage keys are refreshed.
3. Confirm holder-balance and allowance entries are not refreshed by that path.

Reproducing Locally

Copy Code

```
rg "extend_ttl|Balance|Allowance|extend_instance_ttl" contract/vault/soroban/share-token/src/lib.rs
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:L/I:L/D:N/Y:N (0.3)

Recommendation

To mitigate this issue, the following improvements are recommended:

- Balance and allowance entries should be refreshed on access or through an indexed TTL extension path.
- Operators should be able to refresh holder/allowance entries before archival.
- TTL tests should cover long-lived dormant holders and allowances.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #445 documents and proves extend_ttl is intentionally instance-only, while holder balances and allowances remain persistent owner-driven entries without introducing an unbounded holder index. Regression coverage includes

admin_extend_ttl_preserves_holder_balances_and_allowance_expiry_semantics plus the share-token TTL/archival runbook verification in PR #445.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/e189e24fc06f01cd0a1b45988d5984aa0ee5bffa

20.101 ADDRESSMAP CONSTRUCTOR ACCEPTS AN UNUSED ENVIRONMENT PARAMETER

// INFORMATIONAL

Description

`AddressMap::new` accepts an `&Env` parameter that is not used. The parameter gives the appearance that construction may bind environment-specific state when it does not.
This is reassessed as **Informational**.

Proof of Concept

Preconditions

- The address-map constructor is inspected.

Validation Sequence

1. Open `AddressMap::new`.
2. Observe that it accepts an `&Env` parameter.
3. Confirm the parameter is not used by the constructor.

Reproducing Locally

Copy Code

```
rg "pub fn new\(|AddressMap" contract/vault/soroban/src/storage/address_map.rs
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

To mitigate this issue, the following improvement is recommended:
The unused parameter should be removed, or the constructor should be renamed if it intentionally mirrors another interface.

Remediation Comment

SOLVED: Client successfully remediated this issue. The unused environment argument was removed from AddressMap::new, and call sites were updated to use the simplified constructor. This removes the misleading API surface without changing storage semantics.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/2c806e7a7e9e2aa6d78d03434b922cccece6359b

20.102 IMMEDIATE GOVERNANCE SUBMISSIONS WRITE THE PENDING QUEUE TWICE

// INFORMATIONAL

Description

For immediate decisions, governance submission writes the queue on insert and then writes again during immediate acceptance. The first write is unnecessary and consumes extra instructions.
This is reassessed as **Informational**.

Proof of Concept

Preconditions

- A governance action is classified as immediate.

Validation Sequence

1. Submit an immediate action.
2. Trace queue persistence during submit and immediate accept.
3. Observe two queue writes for one immediately accepted action.

Reproducing Locally

Copy Code

```
rg "submit\(|accept\(|save_queue" contract/vault/soroban/governance/src/lib.rs
```

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

To mitigate this issue, the following improvement is recommended:
Immediate submissions should avoid persisting the proposal before inline acceptance unless persistence is required for event or audit semantics.

Remediation Comment

SOLVED: Client successfully remediated this issue. PR #437 branches immediate governance submissions before queue scheduling, so immediately accepted proposals are no longer inserted into persistent pending queue storage. Regression coverage includes pause_immediate_unpause_timelocked and pending_queue_cap_does_not_block_immediate_pause regressions plus governance queue-storage verification for PR #437.

Remediation Hash

https://github.com/Templar-Protocol/contracts/commit/3837cf493bf866f2894271f0b9e0071092ca679a

---

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.