// Security Assessment

04.29.2026 - 05.08.2026

Smart Contract Assessment

TruYields

HALBORN

Smart Contract Assessment - TruYields

Prepared by: HALBORN

Last Updated 05/14/2026

Date of Engagement: April 29th, 2026 - May 8th, 2026

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
6	0	0	1	1	4

TABLE OF CONTENTS

1. Introduction

`Trufin` engaged `Halborn` to conduct a security assessment of their **TRUBILL Solana Program** beginning on April 29th, 2026, and ending on May 8th, 2026. The security assessment was scoped to the Solana Programs provided in solana-vaults-audit GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

TruBill is a Solana vault that accepts USDC deposits and issues TruBILL share tokens in return. It routes the majority of deposits through an external protocol (Delta Manager) to earn yield on ULTRA, while keeping a configurable USDC reserve for immediate liquidity. Share prices are updated once per epoch based on Delta Manager's published NAV.

Users can exit either instantly drawing from the reserve with a small fee or through a slower path where shares are burned now and USDC is claimed after the operator completes a redemption cycle through Delta Manager. An owner governs risk parameters and an operator handles all external asset movements.

2. Assessment Summary

`Halborn` was provided 7 days for the engagement and assigned one full-time security engineer to review the security of the Solana Programs in scope. The engineer is a blockchain and smart contract security expert with advanced smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the Solana Programs.
- Ensure that smart contract functionality operates as intended.

In summary, `Halborn` identified some improvements to reduce the likelihood and impact of risks, which have been addressed by the `Trufin team`. The main recommendations were:

- `Track the redeem-request id in a per-user state PDA rather than the global vault_config`
- `Track the sum of unclaimed RedeemRequest.usdc_owed in a dedicated field on UsdcAccounting and require the post-reconcile owed_to_users to remain at or above that sum`

3. Test Approach And Methodology

Halborn performed a combination of a manual review of the source code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the program assessment. While manual testing is recommended to uncover flaws in business logic, processes, and implementation; automated testing techniques help enhance coverage of programs and can quickly identify items that do not follow security best practices.

The following phases and associated tools were used throughout the term of the assessment:

- Research into the architecture, purpose, and use of the platform.
- Manual program source code review to identify business logic issues.
- Mapping out possible attack vectors
- Thorough assessment of safety and usage of critical Rust variables and functions in scope that could lead to arithmetic vulnerabilities.
- Scanning dependencies for known vulnerabilities (`cargo audit`).
- Local runtime testing (`just test`)

4. 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.

4.1 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
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
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:

$$E = \prod m_e$$

4.2 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

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
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:

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

4.3 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
Informational	0 - 1.9

5. SCOPE

REPOSITORY

(a) Repository: solana-vaults-audit

(b) Assessed Commit ID: 6d4a13c

(c) Items in scope:

- programs/trubill-vault/Cargo.toml
- programs/trubill-vault/src/generated/external_addresses.rs
- programs/trubill-vault/src/generated/mod.rs
- programs/trubill-vault/src/instructions/claim_withdrawal.rs
- programs/trubill-vault/src/instructions/request_redeem.rs
- programs/trubill-vault/src/instructions/reconcile_ultra.rs
- programs/trubill-vault/src/instructions/receive_deposit_refund.rs
- programs/trubill-vault/src/instructions/update_total_assets.rs
- programs/trubill-vault/src/instructions/reconcile_usdc.rs
- programs/trubill-vault/src/instructions/setters.rs
- programs/trubill-vault/src/instructions/initialize.rs
- programs/trubill-vault/src/instructions/deposit.rs
- programs/trubill-vault/src/instructions/receive_ultra.rs
- programs/trubill-vault/src/instructions/mod.rs
- programs/trubill-vault/src/instructions/instant_redeem.rs
- programs/trubill-vault/src/instructions/request_ultra_redemption.rs
- programs/trubill-vault/src/instructions/request_ultra_mint.rs
- programs/trubill-vault/src/instructions/receive_redeem_refund.rs
- programs/trubill-vault/src/instructions/receive_usdc.rs
- programs/trubill-vault/src/constants.rs
- programs/trubill-vault/src/error.rs
- programs/trubill-vault/src/lib.rs
- programs/trubill-vault/src/state
- programs/trubill-vault/src/state/types.rs
- programs/trubill-vault/src/state/events.rs
- programs/trubill-vault/src/state/mod.rs
- programs/trubill-vault/src/helpers.rs
- programs/trubill-vault/src/math.rs

Out-of-Scope: External Dependencies i.e DeltaManager, KYC program, Staker whitelist program

FILE

(a) Submitted File: solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51.zip

(b) Items in scope:

- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/docs/ARCHITECTURE.md
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/generated/external_addresses.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/generated/mod.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/instructions/claim_withdrawal.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/instructions/deposit.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/instructions/initialize.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/instructions/instant_redeem.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/instructions/mod.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/instructions/receive_deposit_refund.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/instructions/receive_redeem_refund.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/instructions/receive_ultra.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/instructions/receive_usdc.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/instructions/reconcile_ultra.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/instructions/reconcile_usdc.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/instructions/request_redeem.rs

- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/instructions/request_ultra_mint.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/instructions/request_ultra_redemption.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/instructions/setters.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/instructions/update_total_assets.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/state/events.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/state/mod.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/state/types.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/constants.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/error.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/helpers.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/lib.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/src/math.rs
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/programs/trubill-vault/Cargo.toml
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/Cargo.toml
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/README.md
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/Anchor.toml
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/trubill/rust-toolchain.toml
- /solana-vaults-audit-6d4a13c741cfb6f198032ba8781fb1ae857eec51/README.md

REMEDIATION COMMIT ID:

- 3f74fb3

- 50a7a32
- fc09903
- 2b90dee
- b040489

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	1	1

INFORMATIONAL
4

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
GLOBAL REDEEM REQUEST COUNTER CAUSES RACE CONDITION AND DOS	MEDIUM	SOLVED - 05/13/2026
RECONCILE USDC OVERSTATEMENT BLOCKS SLOW-PATH REDEEM CLAIMS	LOW	SOLVED - 05/13/2026
INSTANT REDEEM FEE CAN BE BYPASSED VIA SMALL REDEEM AMOUNTS	INFORMATIONAL	SOLVED - 05/13/2026
RISK OF INITIALIZATION FRONT-RUNNING	INFORMATIONAL	ACKNOWLEDGED - 05/13/2026
REDEEMER USDC CAN BE PERMANENTLY LOCKED BY WHITELIST REVOCATION AFTER REQUEST_REDEEM	INFORMATIONAL	SOLVED - 05/13/2026

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
MISSING INPUT VALIDATION ACROSS MULTIPLE INSTRUCTIONS	INFORMATIONAL	SOLVED - 05/13/2026

7. FINDINGS & TECH DETAILS

7.1 GLOBAL REDEEM REQUEST COUNTER CAUSES RACE CONDITION AND DOS

// MEDIUM

Description

The `process_request_redeem` instruction lets a whitelisted user burn TruBill shares and create a `RedeemRequest` PDA recording the USDC owed to them. The instruction takes a `redeem_request_id` parameter, requires it to equal the global `vault_config.next_redeem_request_id` counter, and increments the counter on success. The `RedeemRequest` PDA is derived from `[REDEEM_REQUEST, user.key(), redeem_request_id]`, so per-user uniqueness is guaranteed by the user key.

However, when N users prepare `request_redeem` transactions concurrently, all of them read `next_redeem_request_id = K` off-chain and submit `redeem_request_id = K`. This creates a race condition for the redeemers and only the first transaction succeeds and `next_redeem_request_id` becomes `K+1`, and every other transaction (and any subsequent transaction still using `K`) fails with `InvalidRedeemRequestId`. As a result, this causes a Denial of Service for honest redeemers with repeated failed transactions. It can happen naturally when a lot of redeemers try to redeem for that epoch.

Additionally, griefers can also cause DOS by queuing small amounts to prevent users from calling the redeem path.

programs/trubill-vault/src/instructions/request_redeem.rs

Copy Code

```
31      require!(trubill_amount > 0, VaultError::InvalidRedeemAmount);
32      require!(redeem_request_id == config.next_redeem_request_id, VaultError::InvalidRedeemRequ
33      require_latest_snapshot_epoch(epoch, config.last_snapshot_epoch)?;
```

programs/trubill-vault/src/instructions/request_redeem.rs

Copy Code

```
211     #[account(
212         init,
213         payer = user,
214         seeds = [seeds::REDEEM_REQUEST, user.key().as_ref(), &redeem_request_id.to_le_bytes()]
215         bump,
216         space = ANCHOR_DISCRIMINATOR + RedeemRequest::INIT_SPACE,
217     )]
218     pub redeem_request: Account<'info, RedeemRequest>,
```

Gaps In A User's Redeem-Request IDs

Because the counter is global, a given user's `RedeemRequest` PDAs are scattered across non-contiguous ids. A user whose very first redeem lands at id `520` has no records at ids 0 through 519, and subsequent

requests from the same user typically jump by hundreds depending on overall vault interaction. As a result, on-chain enumeration of a user's redeem history becomes inconsistent and scattered, making it harder to track and maintain.

Proof of Concept

Scenario

Two whitelisted users, alice and bob, each hold TruBill shares and prepare `request_redeem` transactions in the same slot. Both read the on-chain `vault_config.next_redeem_request_id` counter off-chain, observe the value `0`, and sign their transactions with `redeem_request_id = 0`. Alice's transaction lands first, the counter increments to `1`, and bob's transaction now carries a stale id and is rejected by the equality check in `process_request_redeem` with `InvalidRedeemRequestId`.

Test Code

tests/litesvm/integration/multi-user.test.ts

Copy Code

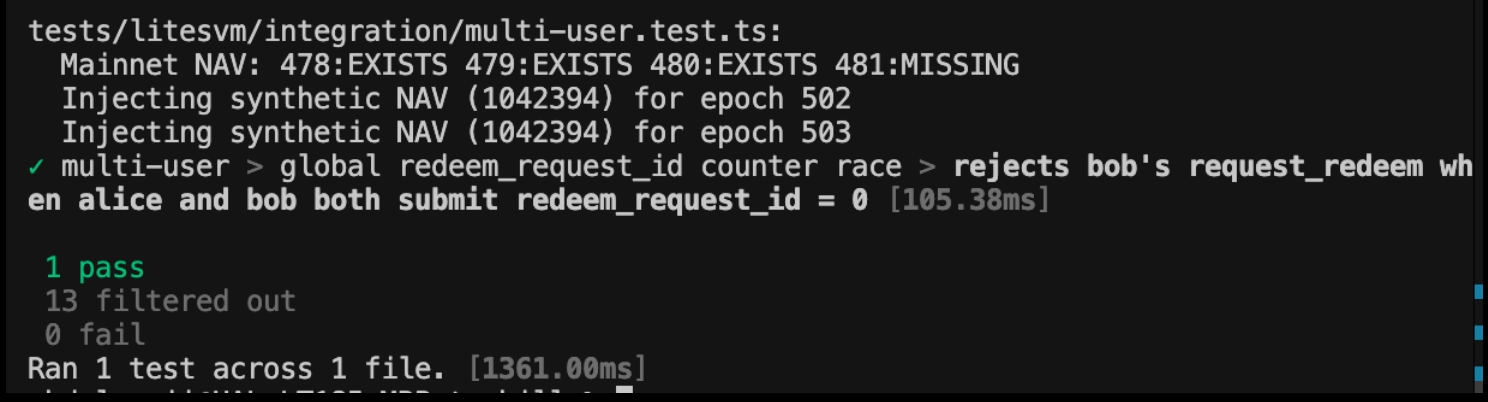
```
    it("rejects bob's request_redeem when alice and bob both submit redeem_request_id = 0", as
      const epoch = await advanceEpochAndSnapshot(ctx, ctx.snapshotEpoch);

      // Both alice and bob read next_redeem_request_id = 0 off-chain and sign with that id.
      // Alice's transaction lands first and wins.
      await Ixs.requestRedeem(ctx, {
        epoch,
        redeemRequestId: ZERO_BN,
        trubillAmount: new BN(ctx.getBalances(ctx.alice.keys.publicKey).trubillBalance),
        userKeys: ctx.alice.keys,
      });

      // Bob's transaction lands second; the on-chain counter is now 1, so id=0 fails.
      await expectError(
        Ixs.requestRedeem(ctx, {
          epoch,
          redeemRequestId: ZERO_BN,
          trubillAmount: new BN(ctx.getBalances(bob.keys.publicKey).trubillBalance),
          userKeys: bob.keys,
        }),
        "InvalidRedeemRequestId",
      );
    });
```

Result

Alice's transaction succeeds and her `RedeemRequest` PDA is created at id `0`. Bob's transaction reverts with `InvalidRedeemRequestId` because the global counter has already advanced. The same pattern repeats for any number of concurrent redeemers, so a single griefer submitting cheap dust redemptions in every slot is sufficient to deny service to all honest users on the slow-path redeem queue, and concurrent honest users incur repeated failed transactions until they coordinate off-chain on the next id.



BVSS

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

Recommendation

To address this finding, it is recommended to track the redeem-request id in a per-user state PDA rather than the global `vault_config`, so concurrent redeemers no longer collide on a shared counter. Alternatively, drop the equality check entirely and rely on Anchor's `init` constraint to reject duplicate per-user PDAs.

Remediation Comment

SOLVED: The **Trufin** team resolved the finding by tracking the redeem request id in a per-user state PDA rather than the global `vault_config`.

Remediation Hash

3f74fb3aa62c8754c2a33b45094519f7bb902f2c

7.2 RECONCILE USDC OVERSTATEMENT BLOCKS SLOW-PATH REDEEM CLAIMS

// LOW

Description

The `process_reconcile_usdc` instruction lets the vault owner sweep two kinds of surplus from the vault USDC ATA into `usdc.reserve`: unaccounted USDC arriving from donations or rounding (computed as `vault_balance − expected_ata_balance`), and a NAV-difference surplus that accumulates inside `owed_to_users` because users are credited at NAV at `request_redeem` time but Delta Manager settles at a later, higher NAV. The owner specifies the NAV-difference component as the `pending_withdrawal_surplus` argument, and the instruction guards it only with `usdc.owed_to_users >= pending_withdrawal_surplus`.

However, whenever any open `RedeemRequest` PDAs hold unclaimed `usdc_owed`, the `>=` check does not distinguish between true NAV surplus and money still backing those obligations. As a result, an overstated `pending_withdrawal_surplus` reduces `owed_to_users` below the sum of unclaimed user obligations, and subsequent `claim_withdrawal` calls revert with `InsufficientFunds`. The affected users' TruBILL was already burned during `request_redeem`, so their `RedeemRequest` PDAs are permanently stuck and the diverted USDC remains in `usdc.reserve` with no easy way to recover them.

programs/trubill-vault/src/instructions/reconcile_usdc.rs

Copy Code

```
38      require!(usdc.sent_for_minting == 0 && ultra.sent_for_redemption == 0, VaultError::Reconci
39      require!(usdc.owed_to_users >= pending_withdrawal_surplus, VaultError::InsufficientFunds);
40
41      let actual_balance = ctx.accounts.vault_usdc_ata.amount;
42      let accounted = usdc.expected_ata_balance()?;
43
44      // unaccounted USDC from donations and rounding.
45      let unaccounted = actual_balance.checked_sub(accounted).ok_or(VaultError::InsufficientFund
46
47      // surplus due to NAV-difference inside owed_to_users
48      let total_surplus = unaccounted.checked_add(pending_withdrawal_surplus).ok_or(VaultError::
49      require!(total_surplus > 0, VaultError::NoDiscrepancyDetected);
50
51      usdc.owed_to_users = usdc.owed_to_users.checked_sub(pending_withdrawal_surplus).ok_or(Vaul
52      usdc.reserve = usdc.reserve.checked_add(total_surplus).ok_or(VaultError::MathOverflow)?;
```

programs/trubill-vault/src/instructions/claim_withdrawal.rs

Copy Code

```
22      require!(
23          usdc.last_settlement_epoch >= user_redeem_request.earliest_redeem_epoch,
24          VaultError::WithdrawalNotClaimable
25      );
26      require!(usdc.owed_to_users >= usdc_to_redeem, VaultError::InsufficientFunds);
27
28      usdc.owed_to_users = usdc.owed_to_users.saturating_sub(usdc_to_redeem);
```

Proof of Concept

Scenario

In the first test Alice has a single open `RedeemRequest` for 600 USDC and `owed_to_users` equals 600 with no actual NAV surplus present. In the second test Alice and Bob each hold an open `RedeemRequest` for 600 USDC and `owed_to_users` equals 1200. Both tests then attempt the affected users' `claim_withdrawal`.

Test Code

tests/litesvm/reconcile-usdc.test.ts

Copy Code

```
249  describe("POC: reconcile drains user obligations", () => {
250    const TRUBILL_BURNED = trubill(5);
251    const ULTRA_TO_REDEEM = usdc(10);
252    const NAV_AT_REDEEM = new BN(10_000);
253    const REDEEM_EPOCH = ZERO_BN;
254    const SETTLEMENT_EPOCH = new BN(1);
255    const RESERVE = usdc(50_000);
256
257    describe("scenario 1: full drain", () => {
258      const ALICE_USDC_OWED = usdc(600);
259      const ALICE_REQ_ID = ZERO_BN;
260
261      beforeEach(async () => {
262        await ctx.setUp({ init: true });
263        await setWhitelistedUserStatus(ctx.provider, ctx.alice.keys.publicKey, WhitelistedUser
264
265        // alice has a redeem request for 600 usdc, settlement epoch already past so its claim
266        await setClaimRedeemRequestFields(ctx, ALICE_REQ_ID, {
267          user: ctx.alice.keys.publicKey,
268          trubillBurned: TRUBILL_BURNED,
269          usdcOwed: ALICE_USDC_OWED,
270          ultraToRedeem: ULTRA_TO_REDEEM,
271          redeemRequestId: ALICE_REQ_ID,
272          sharePriceAtRedeem: NAV_AT_REDEEM,
273          epoch: REDEEM_EPOCH,
274          earliestRedeemEpoch: REDEEM_EPOCH,
275        });
276
277        // owed_to_users matches exactly what alice is owed. no real surplus here.
278        await setUsdcAccountingFields(ctx, {
279          reserve: RESERVE,
280          pendingDeposits: ZERO_BN,
281          owedToUsers: ALICE_USDC_OWED,
282          sentForMinting: ZERO_BN,
283          lastSettlementEpoch: SETTLEMENT_EPOCH,
284        });
285        await setUltraAccountingFields(ctx, { sentForRedemption: ZERO_BN });
286
287        // vault ata holds reserve + alice's owed amount. no donation, nothing extra
288        fundMint(ctx.provider, ctx.usdcMint, ctx.vaultPDAs.vaultAuthority, RESERVE.add(ALICE_U
289      });
290
291      it("owner drains the full owed_to_users and alice cant claim anymore", async () => {
292        // pass the full owed_to_users as "surplus" even though all of it is backing alice.
293        await reconcileUsdc(ctx, {
294          ownerKeys: ctx.owner.keys,
295          pendingWithdrawalSurplus: ALICE_USDC_OWED,
296        });
297
298        const stateAfterReconcile = (await ctx.fetchState()).usdcAcct;
299        expect(stateAfterReconcile.owedToUsers).toEqual(ZERO_BN);
300        expect(stateAfterReconcile.reserve).toEqual(RESERVE.add(ALICE_USDC_OWED));
301
302        await expectError(claimWithdrawal(ctx, { redeemRequestId: ALICE_REQ_ID }), "Insufficie
303      });
304    });
305
306    describe("scenario 2: partial drain locks out the second claimer", () => {
307      const USDC_OWED_EACH = usdc(600);
308      const ALICE_REQ_ID = ZERO_BN;
309
```

```
      const BOB_REQ_ID = new BN(1);
      // only drain 100 so alice's 600 claim still fits but bob's wont
      const PARTIAL_SURPLUS = usdc(100);
      let bob: ReturnType<TestContext["createUser"]>;

      beforeEach(async () => {
        await ctx.setUp({ init: true });
        bob = ctx.createUser();

        await setWhitelistedUserStatus(ctx.provider, ctx.alice.keys.publicKey, WhitelistedUser
        await setWhitelistedUserStatus(ctx.provider, bob.keys.publicKey, WhitelistedUserStatus

        // both alice and bob have a 600 usdc redeem request waiting to be claimed
        for (const [user, reqId] of [
          [ctx.alice.keys.publicKey, ALICE_REQ_ID],
          [bob.keys.publicKey, BOB_REQ_ID],
        ] as const) {
          await setClaimRedeemRequestFields(ctx, reqId, {
            user,
            trubillBurned: TRUBILL_BURNED,
            usdcOwed: USDC_OWED_EACH,
            ultraToRedeem: ULTRA_TO_REDEEM,
            redeemRequestId: reqId,
            sharePriceAtRedeem: NAV_AT_REDEEM,
            epoch: REDEEM_EPOCH,
            earliestRedeemEpoch: REDEEM_EPOCH,
          });
        }

        const totalOwed = USDC_OWED_EACH.muln(2);
        await setUsdcAccountingFields(ctx, {
          reserve: RESERVE,
          pendingDeposits: ZERO_BN,
          owedToUsers: totalOwed,
          sentForMinting: ZERO_BN,
          lastSettlementEpoch: SETTLEMENT_EPOCH,
        });
        await setUltraAccountingFields(ctx, { sentForRedemption: ZERO_BN });

        fundMint(ctx.provider, ctx.usdcMint, ctx.vaultPDAs.vaultAuthority, RESERVE.add(totalOw
      });

      it("partial overstate, first claim works, second reverts forever", async () => {
        // overstate by 100. owed_to_users goes 1200 -> 1100. total owed across both users is
        await reconcileUsdc(ctx, {
          ownerKeys: ctx.owner.keys,
          pendingWithdrawalSurplus: PARTIAL_SURPLUS,
        });

        expect((await ctx.fetchState()).usdcAcct.owedToUsers).toEqual(USDC_OWED_EACH.muln(2).s

        // alice gets in first. her claim works fine, owed_to_users drops to 500
        await claimWithdrawal(ctx, { redeemRequestId: ALICE_REQ_ID });
        expect((await ctx.fetchState()).usdcAcct.owedToUsers).toEqual(USDC_OWED_EACH.sub(PARTI

        // bob's request still says he's owed 600 but only 500 is left. his claim reverts and
        await expectError(
          claimWithdrawal(ctx, { redeemRequestId: BOB_REQ_ID, userKeys: bob.keys }),
          "InsufficientFunds",
        );
      });
    });
  });
```

Results

```
(pass) reconcile usdc > POC: reconcile drains user obligations > scenario 1: full drain > owner drains the full ow
ed_to_users and alice cant claim anymore [23.06ms]
(pass) reconcile usdc > POC: reconcile drains user obligations > scenario 2: partial drain locks out the second cl
aimer > partial overstate, first claim works, second reverts forever [22.73ms]

 10 pass
 0 fail
 22 expect() calls
Ran 10 tests across 1 file. [431.00ms]
```

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:C/I:N/D:C/Y:C (3.0)

Recommendation

To address this finding, it is recommended to track the sum of unclaimed `RedeemRequest.usdc_owed` in a dedicated field on `UsdcAccounting`, incremented in `request_redeem` and decremented in `claim_withdrawal`, and require the post-reconcile `owed_to_users` to remain at or above that sum, so the owner cannot drain USDC that is still backing open user redemptions.

Remediation Comment

SOLVED: The **Trufin** team resolved the finding by introducing a dedicated field on `UsdcAccounting` to track the total USDC owed across all open redemption requests, and enforcing that `owed_to_users` never falls below that sum after reconciliation.

Remediation Hash

50a7a327b46b9930e8d05dd85c34cbe1ef0bc3da

7.3 INSTANT REDEEM FEE CAN BE BYPASSED VIA SMALL REDEEM AMOUNTS

// INFORMATIONAL

Description

The `process_instant_redeem` instruction lets a whitelisted user burn TruBill shares and immediately receive USDC from the vault's reserve, charging an `instant_redeem_fee` (basis points) that is forwarded to the treasury. The fee is computed via `mul_div`, which performs an unsigned integer divide and truncates the result toward zero.

However, whenever `usdc_amount * instant_redeem_fee < BPS_PRECISION`, the truncated fee rounds to zero and the user receives the computed `usdc_amount`. As a result, the configured fee can be silently bypassed at any share-price ratio, causing the treasury to under-collect on every such redemptions.

programs/trubill-vault/src/instructions/instant_redeem.rs

Copy Code

```
32      let fee_amount: u64 = mul_div(usdc_amount, ctx.accounts.vault_config.instant_redeem_fee as
33      let usdc_to_user = usdc_amount.checked_sub(fee_amount).ok_or(VaultError::MathOverflow)?;
```

programs/trubill-vault/src/math.rs

Copy Code

```
10  pub fn mul_div(a: u64, b: u64, denominator: u64) -> Result<u64> {
11      require!(denominator != 0, VaultError::MathOverflow);
12      let result = (a as u128)
13          .checked_mul(b as u128)
14          .ok_or(VaultError::MathOverflow)?
15          .checked_div(denominator as u128)
16          .ok_or(VaultError::MathOverflow)?;
17      u64::try_from(result).map_err(|_| error!(VaultError::MathOverflow))
18  }
```

Proof of Concept

Scenario

The fee bypass is independent of share price, it depends only on whether the integer product `usdc_amount * instant_redeem_fee` stays below `BPS_PRECISION`. The three unit tests below exercise the same 1% fee at three different vault states (1:1 share price, share price = 2, share price ≈ 0.333) and confirm `fee_amount` rounds to 0 in all three.

Test Code

Copy Code

```
26      #[test]
27      fn mul_div_fee_bypass() {
28          let redeem_fee = 100; // 1%
29
```

```
        let usdc_amount = usdc_for_redeem(99, 1_000_000, 1_000_000).unwrap();
        let fee_amount: u64 = mul_div(usdc_amount, redeem_fee, BPS_PRECISION).unwrap();
        let usdc_to_user = usdc_amount.checked_sub(fee_amount).unwrap();
        assert_eq!(usdc_amount, 99);
        assert_eq!(fee_amount, 0);
        assert_eq!(usdc_to_user, 99);
    }

    #[test]
    fn mul_div_fee_bypass_share_price_above_one() {
        // Share price = 2 USDC per share (vault appreciated).
        // Redeeming 49 shares yields usdc_amount = 49 * 2 = 98 → still under the
        // 1% fee precision floor, so fee rounds to 0 despite share price ≠ 1.
        let redeem_fee = 100; // 1%
        let usdc_amount = usdc_for_redeem(49, 1_000_000_000, 2_000_000_000).unwrap();
        let fee_amount: u64 = mul_div(usdc_amount, redeem_fee, BPS_PRECISION).unwrap();
        let usdc_to_user = usdc_amount.checked_sub(fee_amount).unwrap();
        assert_eq!(usdc_amount, 98);
        assert_eq!(fee_amount, 0);
        assert_eq!(usdc_to_user, 98);
    }

    #[test]
    fn mul_div_fee_bypass_share_price_below_one() {
        // Share price ≈ 0.333 USDC per share (vault depreciated).
        // Redeeming 297 shares yields usdc_amount = floor(297 * 1e9 / 3e9) = 99
        // → fee still rounds to 0 under a 1% rate.
        let redeem_fee = 100; // 1%
        let usdc_amount = usdc_for_redeem(297, 3_000_000_000, 1_000_000_000).unwrap();
        let fee_amount: u64 = mul_div(usdc_amount, redeem_fee, BPS_PRECISION).unwrap();
        let usdc_to_user = usdc_amount.checked_sub(fee_amount).unwrap();
        assert_eq!(usdc_amount, 99);
        assert_eq!(fee_amount, 0);
        assert_eq!(usdc_to_user, 99);
    }
```

Result

All three tests pass: `fee_amount == 0` on each assertion. The user receives the full `usdc_amount` while the treasury receives nothing, despite a configured 1% fee. Repeating these dust redemptions until the desired total is drained yields a 0% effective fee at any nonzero share price.

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:N/I:N/D:L/Y:N (1.7)

Recommendation

To address this finding, it is recommended to enforce a minimum fee of one base unit whenever both `usdc_amount` and `instant_redeem_fee` are non-zero, so floor truncation cannot reduce the configured fee to zero.

Remediation Comment

SOLVED: The **Trufin** team resolved the finding by rounding the instant redeem fee up via a new `mul_div_ceil` function.

Remediation Hash

fc099039deeccdf2d04e884e38003192515b40b0

7.4 RISK OF INITIALIZATION FRONT-RUNNING

// INFORMATIONAL

Description

The `initialize` instruction allows to initialize the program and create various role assignment accounts.

However the instruction does not enforce the signer to be a specific address allowing anyone to initialize the program and gain admin access to critical protocol instructions.

programs/trubill-vault/src/instructions/initialize.rs

Copy Code

```
#[derive(Accounts)]
#[event_cpi]
pub struct Initialize<'info> {
    #[account(mut)]
    pub payer: Signer<'info>,

    #[account(
        init,
        payer = payer,
        seeds = [seeds::VAULT_CONFIG],
        bump,
        space = ANCHOR_DISCRIMINATOR + VaultConfig::INIT_SPACE,
    )]
    pub vault_config: Box<Account<'info, VaultConfig>>,

    #[account(
        init,
        payer = payer,
        seeds = [seeds::USDC_ACCOUNTING],
        bump,
        space = ANCHOR_DISCRIMINATOR + UsdcAccounting::INIT_SPACE,
    )]
    pub usdc_accounting: Box<Account<'info, UsdcAccounting>>,

    #[account(
        init,
        payer = payer,
        seeds = [seeds::ULTRA_ACCOUNTING],
        bump,
        space = ANCHOR_DISCRIMINATOR + UltraAccounting::INIT_SPACE,
    )]
    pub ultra_accounting: Box<Account<'info, UltraAccounting>>,

    #[account(
        init,
        payer = payer,
        seeds = [seeds::VAULT_ACCESS],
        bump,
        space = ANCHOR_DISCRIMINATOR + VaultAccess::INIT_SPACE,
    )]
    pub vault_access: Box<Account<'info, VaultAccess>>,

    /// CHECK: Vault authority PDA, used as signer for CPI calls
    #[account(seeds = [seeds::VAULT_AUTHORITY], bump)]
    pub vault_authority: UncheckedAccount<'info>,

    /// CHECK: Treasury account
    #[account()]
    pub treasury_info: UncheckedAccount<'info>,

    /// Vault's collateral token account (USDC)
    #[account(
        init,
```

```
        payer = payer,
        associated_token::mint = usdc_mint,
        associated_token::authority = vault_authority,
        associated_token::token_program = token_program,
    )]
    pub vault_collateral_ata: Box<InterfaceAccount<'info, TokenAccount>>,

    /// Vault's ULTRA token account
    #[account(
        init,
        payer = payer,
        associated_token::mint = ultra_mint,
        associated_token::authority = vault_authority,
        associated_token::token_program = token_program_2022,
    )]
    pub vault_ultra_ata: Box<InterfaceAccount<'info, TokenAccount>>,

    #[account(address = USDC_MINT @ VaultError::InvalidMint)]
    pub usdc_mint: Box<InterfaceAccount<'info, Mint>>,

    #[account(address = ULTRA_MINT @ VaultError::InvalidMint)]
    pub ultra_mint: Box<InterfaceAccount<'info, Mint>>,

    #[account(
        init,
        payer = payer,
        seeds = [seeds::TRUBILL_MINT],
        bump,
        mint::decimals = TRUBILL_DECIMALS,
        mint::authority = vault_authority,
        mint::token_program = token_program_2022,
        extensions::permanent_delegate::delegate = vault_authority,
    )]
    pub trubill_mint: InterfaceAccount<'info, Mint>,

    /// Treasury's TruBILL ATA, pre-created so `update_total_assets` can mint
    /// performance-fee shares without requiring a first-time init.
    #[account(
        init,
        payer = payer,
        associated_token::mint = trubill_mint,
        associated_token::authority = treasury_info,
        associated_token::token_program = token_program_2022,
    )]
    pub treasury_trubill_ata: Box<InterfaceAccount<'info, TokenAccount>>,

    /// CHECK: Metaplex metadata PDA — validated by the Metaplex program during CPI
    #[account(mut)]
    pub metadata_pda: UncheckedAccount<'info>,

    /// CHECK: Owner account for vault access control
    #[account()]
    pub owner_info: UncheckedAccount<'info>,

    /// CHECK: Operator account
    #[account()]
    pub operator_info: UncheckedAccount<'info>,

    pub token_program_2022: Program<'info, Token2022>,

    /// CHECK: Metaplex Token Metadata program
    #[account(address = mpl_token_metadata::ID)]
    pub metadata_program: UncheckedAccount<'info>,

    /// CHECK: Sysvar instructions account required by Metaplex
    #[account(address = anchor_lang::solana_program::sysvar::instructions::ID)]
    pub sysvar_instructions: UncheckedAccount<'info>,

    pub system_program: Program<'info, System>,
    pub token_program: Program<'info, Token>,
    pub associated_token_program: Program<'info, AssociatedToken>,
}
```

BVSS

AO:A/AC:L/AX:H/R:F/S:U/C:N/A:C/I:N/D:C/Y:N (1.0)

Recommendation

To address this finding, it is recommended to restrict the signer of the `initialize` instruction to be a specific known address.

Remediation Comment

ACKNOWLEDGED: The **Trufin** team acknowledged the risk of the finding.

7.5 REDEEMER USDC CAN BE PERMANENTLY LOCKED BY WHITELIST REVOCATION AFTER REQUEST_REDEEM

// INFORMATIONAL

Description

The `process_claim_withdrawal` instruction completes the slow-path redemption flow. The user's `RedeemRequest` PDA is consumed and the corresponding USDC is transferred from `vault_usdc_ata` to `user_usdc_ata`.

However, when the staker program revokes the user's whitelist between `request_redeem`, which already burned the user's TruBill shares and credited `usdc.owed_to_users`, and `claim_withdrawal`, every claim attempt fails with `NotWhitelisted` and the corresponding USDC remains permanently in `usdc.owed_to_users`.

As a result, the user's burned shares cannot be redeemed for USDC at any later point, the funds can be absorbed into `usdc.reserves` via `reconcile_usdc` instruction.

programs/trubill-vault/src/instructions/claim_withdrawal.rs

Copy Code

```
62      #[account(
63          constraint = matches!(user_whitelist.status, WhitelistUserStatus::Whitelisted) @ Vault
64          seeds = [seeds::USER, user.key().as_ref()],
65          bump,
66          seeds::program = STAKER_PROGRAM_ID,
67      )]
68      pub user_whitelist: Account<'info, UserStatus>,
```

programs/trubill-vault/src/instructions/request_redeem.rs

Copy Code

```
46      token_2022::burn(
47          CpiContext::new_with_signer(
48              ctx.accounts.token_program_2022.to_account_info(),
49              token_2022::Burn {
50                  mint: ctx.accounts.trubill_mint.to_account_info(),
51                  from: ctx.accounts.user_trubill_ata.to_account_info(),
52                  authority: ctx.accounts.vault_authority.to_account_info(),
53              },
54              &[signer_seeds],
55          ),
56          trubill_amount,
57      )?;
```

Step-By-Step Scenario

1. A whitelisted user holds TruBill shares in the vault.
2. The user calls `process_request_redeem`. The vault authority's permanent delegate burns the user's TruBill shares, `usdc.owed_to_users` is incremented by the corresponding USDC amount, and a `RedeemRequest` PDA is created.

3. While the redeem queue waits for ULTRA settlement, the staker program admin updates the user's `UserStatus` PDA to a non-whitelisted variant.
4. The user calls `process_claim_withdrawal`. The `user_whitelist` constraint fails with `NotWhitelisted` and the USDC transfer reverts.

BVSS

AO:A/AC:L/AX:L/R:F/S:U/C:N/A:L/I:N/D:N/Y:N (0.6)

Recommendation

To address this finding, it is recommended to document the whitelist behavior, as re-whitelisting a user after redemption may conflict with compliance requirements.

Remediation Comment

SOLVED: The **Trufin** team resolved the finding by documenting revoked whitelist behavior.

Remediation Hash

2b90dee8363da163e36cbdb17493b8abc66fb077

7.6 MISSING INPUT VALIDATION ACROSS MULTIPLE INSTRUCTIONS

// INFORMATIONAL

Description

Multiple instances of missing input validation were observed throughout the codebase.

Privileged-Role Setters Do Not Validate Against The Default Pubkey

The `process_set_pending_owner`, `process_set_operator`, and `process_set_treasury` instructions each store the supplied Pubkey directly into vault state without checking that it is not `Pubkey::default()`. Setting any of these roles to the default address produces a value that no real signer can match: pending ownership transfer is wedged in `process_claim_ownership`, operator-gated instructions fail because the system program cannot sign, and treasury-bound performance-fee mints in `process_update_total_assets` are sent to a wallet that nobody controls. Recovery for each requires the current owner to reissue the corresponding setter with a valid key.

programs/trubill-vault/src/instructions/setters.rs

Copy Code

```
16 pub fn process_set_pending_owner(ctx: Context<SetPendingOwner>, new_pending_owner: Pubkey) ->
17     let access = &mut ctx.accounts.vault_access;
18
19     access.pending_owner = Some(new_pending_owner);
20
21     emit_cpi!(PendingOwnerSet { owner: access.owner, pending_owner: new_pending_owner });
22     Ok(())
23 }
```

programs/trubill-vault/src/instructions/setters.rs

Copy Code

```
70 pub fn process_set_operator(ctx: Context<SetOperator>, new_operator: Pubkey) -> Result<()> {
71     let access = &mut ctx.accounts.vault_access;
72
73     access.operator = new_operator;
74
75     emit_cpi!(OperatorSet { new_operator });
76     Ok(())
77 }
```

programs/trubill-vault/src/instructions/setters.rs

Copy Code

```
94 pub fn process_set_treasury(ctx: Context<SetTreasury>, new_treasury: Pubkey) -> Result<()> {
95     let vault_config = &mut ctx.accounts.vault_config;
96     let old_treasury = vault_config.treasury;
97     vault_config.treasury = new_treasury;
98
99     emit_cpi!(TreasurySet { new_treasury, old_treasury });
100    Ok(())
101 }
```

Process_initialize Accepts Unbounded Min_deposit_amount And Reserve_ratio_bps

The `process_initialize` instruction validates `nav_expiry_time` via `require_valid_nav_expiry_time`, but writes `min_deposit_amount` and `reserve_ratio_bps` verbatim into `VaultConfig` without bounds checks. A misconfigured deployment can set `min_deposit_amount` to a value that effectively blocks deposits or `reserve_ratio_bps` above `BPS_PRECISION` (10,000), which breaks downstream reserve-ratio math. The intended bounds are documented but not enforced.

programs/trubill-vault/src/instructions/initialize.rs

Copy Code

```rust
pub fn process_initialize(
    ctx: Context<Initialize>,
    min_deposit_amount: u64,
    reserve_ratio_bps: u16,
    nav_expiry_time: i64,
) -> Result<()> {
    require_valid_nav_expiry_time(nav_expiry_time)?;

    // --- Vault Config ---
    let config = &mut ctx.accounts.vault_config;
    config.vault_authority_bump = ctx.bumps.vault_authority;
    config.is_paused = false;
    config.min_deposit_amount = min_deposit_amount;
    config.instant_redeem_fee = 0;
    config.treasury = ctx.accounts.treasury_info.key();
    config.last_snapshot_epoch = u64::MAX;
    config.nav_expiry_time = nav_expiry_time;
    config.fee_bps = 0;
    config.last_fee_share_price = 0;
    config.reserve_ratio_bps = reserve_ratio_bps;
```

Setter Caps Use Strict Less-Than Instead Of Less-Than-Or-Equal

The `process_set_instant_redeem_fee`, `process_set_reserve_ratio_bps`, and `process_set_fee` instructions all gate their input on `new_value < BPS_PRECISION as u16`. This rejects the inclusive boundary at exactly 10,000 bps (100%), which is a legitimate configuration for the reserve ratio (a fully reserved vault) and an intended ceiling for the fee setters. The check should be `<=` so the documented maximum is reachable.

programs/trubill-vault/src/instructions/setters.rs

Copy Code

```rust
pub fn process_set_reserve_ratio_bps(ctx: Context<SetReserveRatioBps>, new_reserve_ratio_bps:
    require!(new_reserve_ratio_bps < BPS_PRECISION as u16, VaultError::InvalidReserveRatioBps)

    let vault_config = &mut ctx.accounts.vault_config;
    let old_reserve_ratio_bps = vault_config.reserve_ratio_bps;
    vault_config.reserve_ratio_bps = new_reserve_ratio_bps;

    emit_cpi!(ReserveRatioBpsSet { new_reserve_ratio_bps, old_reserve_ratio_bps });
    Ok(())
}
```

BVSS

AO:A/AC:L/AX:L/R:F/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

To address this finding, it is recommended to validate that the new Pubkey is not `Pubkey::default()` in `process_set_pending_owner`, `process_set_operator`, and `process_set_treasury`, enforce the documented bounds on `min_deposit_amount` and `reserve_ratio_bps` in `process_initialize`, and change the comparison in the three bps setters from `<` to `<=` so the inclusive 10,000 bps maximum is reachable.

Remediation Comment

SOLVED: The **Trufin** team resolved the finding by validating input parameters across setter functions.

Remediation Hash

b040489419c6b0ac5dd1cfc3abe47f0e84786b79

8. AUTOMATED TESTING

Static Analysis Report

Description

Halborn used automated security scanners to assist with detection of well-known security issues and vulnerabilities. Among the tools used was `cargo audit`, a security scanner for vulnerabilities reported to the RustSec Advisory Database. All vulnerabilities published in `https://crates.io` are stored in a repository named The RustSec Advisory Database. `cargo audit` is a human-readable version of the advisory database which performs a scanning on Cargo.lock. Security Detections are only in scope. All vulnerabilities shown here were already disclosed in the above report. However, to better assist the developers maintaining this code, the auditors are including the output with the dependencies tree, and this is included in the cargo audit output to better know the dependencies affected by unmaintained and vulnerable crates.

Cargo Audit Results

The program contains no vulnerabilities detected by `cargo audit`.
