// Security Assessment

03.25.2026 - 03.30.2026

Stellar Vaults

Upshift

HALBORN

Stellar Vaults - Upshift

Prepared by: HALBORN

Last Updated 04/28/2026

Date of Engagement: March 25th, 2026 - March 30th, 2026

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
14	0	0	2	5	7

TABLE OF CONTENTS

1. Introduction

`Upshift` engaged Halborn to conduct a security assessment of the **Stellar Vault** smart contracts, beginning on March 25th, 2026, and ending on March 30th, 2026. This security assessment was scoped to the core vault contract (`august-vault`) and the reference strategy implementation (`xlm-strategy`). Commit hashes, audited components, and further details can be found in the Scope section of this report.

The Upshift Stellar Vault is a share-based tokenized vault built on Stellar Soroban following the ERC-4626 Tokenized Vault Standard pattern, implemented on top of the OpenZeppelin Stellar Contracts library. It accepts any SEP-41 compliant token as the underlying asset and issues proportional share tokens to depositors. An operator role manages capital allocation by deploying assets to external subaccounts (either Strategy contracts or plain Wallet addresses) enabling yield generation through external DeFi protocols. The `xlm-strategy` contract is a reference implementation of the vault's strategy interface, wrapping XLM and providing controller-gated functions to deploy and recall funds from external protocol addresses.

2. Assessment Summary

The team at Halborn assigned a full-time security engineer to verify the security of the smart contracts. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

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

- `Add get_balance() -> i128 to the IStrategy trait so the vault can compute NAV on-chain from actual subaccount balances.`
- `Track vault-originated idle balances in a dedicated local_balance storage variable so that any unsolicited token transfers to the strategy address are isolated and cannot influence the NAV reported by get_balance().`
- `Define a standard IProtocol trait with withdraw() and get_balance() to replace the raw token transfer in recall_from_protocol and enable on-chain reconciliation in remove_subaccount.`
- `Addition of a get_asset function to the IStrategy interface and explicit asset validation to prevent silent cross-token NAV corruption.`
- `Enforcement of a minimum decimals_offset in the constructor to mitigate first-depositor share inflation attacks.`
- `Use of balance-differencing in recall operations to prevent bookkeeping drift with fee-on-transfer tokens.`
- `Automatically subtract token.balance(wallet) from deployed_assets when removing a Wallet subaccount.`
- `Consistent instance TTL management to avoid unexpected automatic restoration fees.`
- `Ordering change to call authorization (require_auth) before identity equality checks for defensive consistency with Soroban conventions.`
- `Rename of update_total_assets to update_deployed_assets (or equivalent) to avoid operator and integrator confusion that could cause double-counting.`

3. Test Approach And Methodology

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. Manual testing was emphasized to uncover flaws in logic, process, and contract interaction, while automated tools supported the detection of unsafe coding patterns and dependency vulnerabilities.

The following phases and associated tools were used during the assessment:

- Research into the architecture, purpose, and operational model of the Upshift vault system, including the ERC-4626 vault standard and its adaptation to the Soroban execution model.
- Manual code review and walk-through of all contracts (`august-vault`, `xlm-strategy`).
- Verification of role-based access control: admin, operator, and controller authorization flows.
- Analysis of share/asset conversion arithmetic and ERC-4626 rounding compliance.
- Review of NAV accounting logic, AUM rate limits, and the `deployed_assets` bookkeeping model.
- Analysis of the subaccount lifecycle: registration, capital deployment, withdrawal, and removal.
- Review of upgrade and migration flows, including the `UpgradeableInternal` hook.
- Assessment of instance storage TTL management and the impact of CAP-046 automatic restoration.
- Review of cross-contract interactions between vault, strategy, and external protocol addresses.
- Evaluation of the custodial operator model and its trust assumptions relative to user fund safety.
- Scanning of Rust code for unsafe patterns, integer overflow risks, and arithmetic edge cases.
- Review of existing integration tests and addition of new test cases where required.

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: stellar-vaults

(b) Assessed Commit ID: 18c3605

(c) Items in scope:

- august-vault/src/contract.rs
- august-vault/src/errors.rs
- august-vault/src/events.rs
- august-vault/src/lib.rs
- august-vault/src/storage.rs
- august-vault/src/strategy.rs
- xlm-strategy/src/lib.rs

FILE

(a) Submitted File: stellar-vaults-47207d768bc39c6fa6379f46f399744818eda662.zip

(b) Items in scope:

- /stellar-vaults-47207d768bc39c6fa6379f46f399744818eda662/contracts/august-vault/src/contract.rs
- /stellar-vaults-47207d768bc39c6fa6379f46f399744818eda662/contracts/august-vault/src/errors.rs
- /stellar-vaults-47207d768bc39c6fa6379f46f399744818eda662/contracts/august-vault/src/events.rs
- /stellar-vaults-47207d768bc39c6fa6379f46f399744818eda662/contracts/august-vault/src/lib.rs
- /stellar-vaults-47207d768bc39c6fa6379f46f399744818eda662/contracts/august-vault/src/storage.rs
- /stellar-vaults-47207d768bc39c6fa6379f46f399744818eda662/contracts/august-vault/src/strategy.rs
- /stellar-vaults-47207d768bc39c6fa6379f46f399744818eda662/contracts/xlm-strategy/src/lib.rs

REMEDIATION COMMIT ID:

- 9f549ae
- d19a8e3

- 6ff1456
- 9d9860d

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	2	5

INFORMATIONAL
7

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
OPERATOR-ATTESTED NAV WITH NO ON-CHAIN VERIFICATION PATH	MEDIUM	PARTIALLY SOLVED - 04/14/2026
STRATEGY DONATION ATTACK / UNTRACKED EXTERNAL TRANSFERS INFLATE NAV	MEDIUM	SOLVED - 04/23/2026
ABSENCE OF IPROTOCOL INTERFACE PREVENTS DEFI INTEGRATION	LOW	RISK ACCEPTED - 04/14/2026
ISTRATEGY SHOULD ENFORCE GET_BALANCE() SEMANTICS	LOW	SOLVED - 04/23/2026
STRATEGY ASSET NOT VALIDATED AGAINST VAULT ASSET AT REGISTRATION	LOW	SOLVED - 04/07/2026
FIRST-DEPOSITOR SHARE INFLATION ATTACK POSSIBLE WHEN DECIMALS_OFFSET IS ZERO	LOW	SOLVED - 04/02/2026

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
RECALL_FROM_PROTOCOL DOES NOT USE BALANCE-DIFFERENCING	LOW	SOLVED - 04/07/2026
REMOVE_SUBACCOUNT DOES NOT RECONCILE WALLET DEPLOYED_ASSETS ON REMOVAL	INFORMATIONAL	SOLVED - 04/07/2026
NO RATE LIMIT ON SETTLE_PROTOCOL_RETURNS	INFORMATIONAL	SOLVED - 04/23/2026
REMOVE_SUBACCOUNT DOES NOT ENFORCE ACCOUNTING PRECONDITIONS ON WALLET REMOVAL	INFORMATIONAL	SOLVED - 04/23/2026
AUTHORIZATION CHECK OCCURS AFTER IDENTITY EQUALITY CHECK	INFORMATIONAL	SOLVED - 04/07/2026
INCONSISTENT TTL MANAGEMENT ACROSS VAULT AND STRATEGY	INFORMATIONAL	SOLVED - 04/07/2026
UPDATE_TOTAL_ASSETS MISLEADINGLY NAMED	INFORMATIONAL	SOLVED - 04/02/2026
STRATEGY REMOVAL DOES NOT RECORD NAV IMPACT ON-CHAIN	INFORMATIONAL	SOLVED - 04/23/2026

7. FINDINGS & TECH DETAILS

7.1 OPERATOR-ATTESTED NAV WITH NO ON-CHAIN VERIFICATION PATH

// MEDIUM

Description

The vault's Net Asset Value (NAV) is computed as `local_balance + deployed_assets`, where `deployed_assets` is set exclusively via `update_total_assets`, a function that accepts an `i128` input and it is controlled exclusively by the operator with no on-chain verification. The vault has no mechanism to query the actual value of capital held by registered subaccounts at any point in time. Neither `IStrategy` nor any lower-level interface exposes a `get_balance` function, making on-chain NAV computation structurally impossible regardless of how many strategies are registered.

This design creates a permanent decoupling between the NAV the vault reports and the value that actually exists on-chain. Three concrete scenarios illustrate this:

- **Scenario A — Yield accrual:** A strategy deploys 100,000 XLM into a lending protocol. The protocol accrues 10,000 XLM in yield. The vault continues reporting `deployed_assets = 100,000` until the operator manually calls `update_total_assets(110,000)`. During this window, depositors receive more shares than they should (NAV is understated) and redeemers receive fewer assets than they are entitled to.
- **Scenario B — Loss or partial recovery:** A strategy suffers a 30% loss. The vault has no way to detect this. If the operator delays calling `update_total_assets`, users who monitor the protocol directly and detect the loss can redeem at the pre-loss NAV at the expense of users who do not have this information.
- **Scenario C — Frontrunning NAV updates:** Since `update_total_assets` changes NAV instantly with no time delay or smoothing, any actor who can observe a pending NAV update can deposit just before a positive update and redeem immediately after, extracting value from existing shareholders. Deposits and redemptions have no slippage protection parameters, making this extraction straightforward.

Beyond these exploitation scenarios, the operator dependency also causes operational failures: if capital deployed to a strategy has accrued yield and the operator has not reconciled this before a withdrawal is requested, the vault panics with an underflow error, blocking fund recovery until multiple manual reconciliation transactions are executed. Similarly, when a strategy is removed from the whitelist, the vault cannot automatically adjust `deployed_assets` because it has no way to know how much of the global counter was attributable to that specific strategy.

Code Location

Code of `update_total_assets` function from `contracts/august-vault/src/contract.rs` file:

Copy Code

```
599 pub fn update_total_assets(e: &Env, operator: Address, amount: i128) {
600     Self::bump_instance(e);
601
```

```
    storage::require_operator(e, &operator);    // ← only check: is caller the operator?

    if amount < 0 {
        panic_with_error!(e, VaultError::InvalidAmount);
    }

    let local = Self::local_balance(e);
    let _ = local
        .checked_add(amount)
        .unwrap_or_else(|| panic_with_error!(e, VaultError::MathOverflow));

    let old_deployed = storage::get_deployed_assets(e);
    let new_deployed = amount;                  // ← operator-provided, no on-chain verificati

    if new_deployed == old_deployed {
        return;
    }

    if storage::is_paused(e) && new_deployed > old_deployed {
        panic_with_error!(e, VaultError::VaultPaused);
    }

    if old_deployed > 0 {
        // Rate limit exists but has known bypasses
        ...
    }

    storage::set_deployed_assets(e, new_deployed);
}
```

Code of `IStrategy` trait from `contracts/august-vault/src/strategy.rs` file:

Copy Code

```
#[contractclient(name = "StrategyClient")]
pub trait IStrategy {
    fn deposit(e: Env, from: Address, amount: i128);
    fn withdraw(e: Env, to: Address, amount: i128) -> i128;
    // ← no get_balance: vault cannot query the real value of deployed capital
}
```

Impact

The vault's NAV is structurally decoupled from on-chain reality. Any divergence between `deployed_assets` and the true value of deployed capital (whether due to yield, losses, or external interactions with the underlying protocols) is invisible to the vault until the operator manually corrects it. This creates persistent windows of incorrect share pricing that benefit informed actors at the expense of uninformed users. The operator, by controlling both the timing and value of NAV updates, holds unchecked influence over the share price of the vault with no on-chain counterbalance.

More broadly, this design makes the operator the single most powerful actor in the system by a significant margin. The operator controls NAV attestation (`update_total_assets`), capital movement (`deposit_to_subaccount` , `withdraw_from_subaccount`), and the timing of both. This means the operator simultaneously determines the share price and decides when funds move in and out of yield-generating positions, two levers that, combined, allow systematic value extraction from depositors with no on-chain counterbalance.

An honest operator is expected to act in the interest of shareholders, but the protocol offers no enforcement mechanism: there is no multi-sig requirement, no timelock on NAV changes, no independent price feed to cross-check the attested value, and no

separation between the role that controls capital allocation and the role that controls price reporting. In traditional finance, these functions are deliberately separated precisely because their combination in a single actor constitutes a structural conflict of interest. As currently designed, the vault's safety is entirely a function of operator trust, not protocol guarantees.

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:N/I:C/D:N/Y:N (6.7)

Recommendation

It is recommended to implement a `get_balance` function chain across all layers of the architecture, using the same `#[contractclient]` pattern already established by `IStrategy`. This would allow the vault to query the real value of deployed capital directly from each strategy, and each strategy to query its real value from each registered protocol, without relying on the operator to report it.

The proposed chain is:

1. `IStrategy::get_balance`, added to `contracts/august-vault/src/strategy.rs`:

```
#[contractclient(name = "StrategyClient")]
pub trait IStrategy {
    fn deposit(e: Env, from: Address, amount: i128);
    fn withdraw(e: Env, to: Address, amount: i128) -> i128;
    fn get_balance(e: Env) -> i128;  // real value in vault's underlying token
}
```

2. `IProtocol::get_balance`, new trait in `contracts/xlm-strategy/src/`:

```
#[contractclient(name = "ProtocolClient")]
pub trait IProtocol {
    fn deposit(e: Env, amount: i128);
    fn withdraw(e: Env, amount: i128) -> i128;
    fn get_balance(e: Env) -> i128;  // real value in strategy's underlying token
}
```

3. Example of `XlmStrategy::get_balance` that aggregates local balance and all registered protocol positions:

```
pub fn get_balance(e: &Env) -> i128 {
    let local = token_client.balance(&e.current_contract_address());
    local + Self::get_protocol_balances(e)  // sum of IProtocol::get_balance per registered protocol
}
```

4. `update_total_assets` rewritten to use `IStrategy::get_balance` as its primary source, with operator attestation retained only as an explicit fallback for protocols that cannot implement `IProtocol::get_balance`.

With this architecture, NAV becomes on-chain verifiable for all protocols that implement the interface. The operator's role shifts from being the sole source of truth for NAV to being a fallback for protocols that cannot expose a balance query. This eliminates the frontrunning surface, removes the operational failures caused by unreconciled yield, and enables atomic `deployed_assets` reconciliation on strategy removal, all without changing the vault's external interface.

Remediation Comment

PARTIALLY SOLVED: The `Upshift team` partially addressed this finding by adding `IStrategy::get_balance()` to the strategy trait and integrating it into the vault's NAV computation. The `total_assets` formula now aggregates the vault's local balance, live on-chain balances from all registered strategies via `strategy.get_balance()`, and the operator-reported `deployed_assets` for wallet-subaccount AUM.

The `IProtocol` interface recommended to bring external protocol positions fully on-chain was not implemented. The client has acknowledged this and accepted the residual risk, noting that their initial deployment scope comprises utility custody wallets and a single token wrapper strategy with an EOA-addressable counterparty, where a fully on-chain adapter is not yet necessary. The client further notes that pricing of off-chain or cross-chain positions cannot always be handled by a fully on-chain adapter, and that `IProtocol` integration will be addressed when a DeFi venue capable of reporting positions on-chain is integrated in a future release.

Remediation Hash

https://github.com/fractal-protocol/stellar-vaults/commit/9f549ae7158aed8f370eebd88cc2cc147e1747b3

7.2 STRATEGY DONATION ATTACK / UNTRACKED EXTERNAL TRANSFERS INFLATE NAV

// MEDIUM

Description

This vulnerability was identified in commit `18c36059acd4581b837d7b094eac9e66225cefd2`.

The vault's `total_assets()` includes strategy balances queried live via `IStrategy::get_balance()`. In the `XlmStrategy` implementation, `get_balance()` was defined as:

```
get_balance() = token.balance(strategy_address) + deployed_total
```

The idle component, `token.balance(strategy_address)`, is the raw on-chain token balance of the strategy contract. Because SEP-41 transfers are permissionless, any third party can transfer tokens directly to the strategy contract address without going through any vault-controlled flow. This produces two distinct attack scenarios that share the same root cause.

Scenario 1: Donation Attack

A malicious actor deliberately sends tokens to the strategy to inflate `idle`. The share price appears higher than it actually is, causing new depositors to receive fewer vault shares than they are entitled to. The attack requires no authorization and can be triggered by anyone holding the underlying asset.

Scenario 2: Uncontrolled Protocol Push

An external protocol returns funds to the strategy via a direct SEP-41 transfer outside the `recall_from_protocol` flow. The result is identical: `idle` increases while `deployed_total` remains unchanged, double-counting that capital in the NAV until the controller manually calls `settle_protocol_returns`. Note that `settle_protocol_returns` does not mitigate the pure donation scenario: it only adjusts `deployed_total`, which is already zero in that case.

Code Location

Code of `get_balance` function from `contracts/xlm-strategy/src/lib.rs`:

```
130 pub fn get_balance(e: &Env) -> i128 {
131     Self::bump_instance(e);
132     let self_addr = e.current_contract_address();
133     let idle = Self::token_client(e).balance(&self_addr);
134     let deployed = Self::get_deployed_total(e);
135     idle
136         .checked_add(deployed)
137
138
```

```
            .unwrap_or_else(|| panic_with_error!(e, StrategyError::MathOverflow))
    }
```

Impact

Any external actor can inflate the vault's reported NAV without authorization by transferring tokens to the strategy address. This affects share pricing for all vault participants: new depositors receive fewer shares than their fair entitlement, and the vault's `withdraw` path can inadvertently distribute donated tokens to the vault, causing an accounting discrepancy. The attack is fully permissionless and persistent until the controller intervenes.

BVSS

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

Recommendation

It is recommended to introduce a `local_balance` storage variable in the strategy contract that tracks exclusively the vault-originated idle balance, decoupled from the raw `token.balance()`. This variable must be updated atomically with every vault-controlled flow.

Remediation Comment

SOLVED: The `Upshift` team addressed this finding by introducing a `StorageKey::LocalBalance` tracker in `XlmStrategy`, updated across all vault-controlled flows. `get_balance()` now returns `local_balance + deployed_total` instead of `token.balance(self) + deployed_total`.

A `DonationDetected` event is emitted whenever `token.balance(strategy) > local_balance`. A controller-only `recover_donation()` function sweeps excess tokens to a designated recipient. One-shot migration functions `seed_local_balance()` and `seed_deployed_total()` were added for post-upgrade state initialization.

Remediation Hash

https://github.com/fractal-protocol/stellar-vaults/commit/d19a8e382f02ecbf65d00823c961681ecd61d5ad

7.3 ABSENCE OF IPROTOCOL INTERFACE PREVENTS DEFI INTEGRATION

// LOW

Description

The strategy architecture lacks a standardized `IProtocol` interface for interacting with external DeFi protocols. This single architectural gap produces two distinct operational failures that share the same root cause.

- **Scenario 1 — recall_from_protocol is structurally incompatible with DeFi contracts**

`deploy_to_protocol` transfers strategy funds to any address passing `require_valid_protocol_target`, including Soroban smart contracts. The outgoing transfer is authorized by the strategy itself and always succeeds. However, `recall_from_protocol` (its intended counterpart) retrieves funds using a raw SEP-41 `token_client.transfer(&protocol, &self_addr, &amount)`. This mechanism requires the `protocol` address to authorize an outbound transfer via Soroban auth, which is compatible with EOAs (that can co-sign the transaction) but is structurally incompatible with DeFi protocol contracts, which expose their own withdrawal interfaces and never authorize arbitrary outbound transfers.

The function's own documentation acknowledges this explicitly:

> "For protocols requiring a custom withdrawal call, replace the transfer with the protocol's client interface."

This comment reveals the core architectural problem: the intended extension mechanism is a manual source code edit. Every new DeFi integration requires a developer to modify the strategy source, locate the `token_client.transfer` line, and replace it with a protocol-specific client call. This is architecturally unhygienic, the strategy cannot support multiple protocol targets simultaneously without branching logic hardcoded per integration, and any modification introduces regression risk for existing integrations.

`deploy_to_protocol` transfers strategy funds to any address that passes `require_valid_protocol_target`, including Soroban smart contracts (except from Vault or Strategy addresses). The outgoing transfer is authorized by the strategy itself and always succeeds regardless of the target type.

`recall_from_protocol` is the intended counterpart, it retrieves funds from an external target using a raw SEP-41 `token_client.transfer(&protocol, &self_addr, &amount)`. This works correctly for EOA targets, which can authorize the sub-invocation by signing the transaction. For DeFi protocol contracts, this is structurally incompatible: those contracts expose their own withdrawal interfaces and never authorize arbitrary outbound transfers.

The function's own documentation acknowledges this explicitly:

> *"For protocols requiring a custom withdrawal call, replace the transfer with the protocol's client interface."*

This comment reveals the core architectural problem: **the intended extension mechanism is a manual source code edit.** Every new DeFi integration requires a developer to open the strategy source, locate the `token_client.transfer` line, and replace it with a protocol-specific client call. There is no extensibility point, no protocol registry, and no dispatch mechanism. This is architecturally unhygienic, the strategy cannot support multiple protocol targets simultaneously without branching logic hardcoded per integration, and any modification introduces regression risk for existing integrations.

- **Scenario 2 — remove_subaccount cannot reconcile deployed_assets**

`remove_subaccount` removes a strategy from the vault's whitelist but explicitly does not reconcile `deployed_assets`. The docstring acknowledges the reason: on-chain balance checks are unreliable for strategies that have funds deployed to external protocols. Without an `IProtocol::get_balance()` interface, there is no on-chain mechanism to determine how much capital a strategy currently holds across all its external protocol positions. The admin must manually call `update_total_assets` after removal with an operator-computed off-chain value.

Code Location

Code of `recall_from_protocol` function from `contracts/xlm-strategy/src/lib.rs` file:

Copy Code

```
221 pub fn recall_from_protocol(e: &Env, controller: Address, protocol: Address, amount: i128) {
222     Self::bump_instance(e);
223     controller.require_auth();
224     Self::require_controller(e, &controller);
225     Self::require_valid_protocol_target(e, &protocol);
226 
227     if amount <= 0 {
228         panic_with_error!(e, StrategyError::InvalidAmount);
229     }
230 
231     let token_client = Self::token_client(e);
232     let self_addr = e.current_contract_address();
233     token_client.transfer(&protocol, &self_addr, &amount);   // ← requires protocol to authori
234 
235     let deployed = Self::get_deployed_total(e);
236     let new_deployed = deployed.saturating_sub(amount).max(0);
237     e.storage()
238         .instance()
239         .set(&StorageKey::DeployedTotal, &new_deployed);
240 
241     RecalledFromProtocol { controller, protocol, amount }.publish(e);
242 }
```

Code of `remove_subaccount` function from `contracts/august-vault/src/contract.rs` file:

Copy Code

```
429 /// Does NOT reconcile `deployed_assets`. If the removed subaccount had
430 /// deployed capital, the operator should call `update_total_assets`
431 /// with the new total that excludes the removed subaccount's funds.
432 pub fn remove_subaccount(e: &Env, admin: Address, subaccount: Address) {
433     Self::bump_instance(e);
434     storage::require_admin(e, &admin);
435 
436     let subs = storage::get_subaccounts(e);
437 
438
```

```
    let idx = match Self::find_subaccount_index(&subs, &subaccount) {
        Some(i) => i,
        None => panic_with_error!(e, VaultError::SubaccountNotWhitelisted),
    };

    let new_subs = Self::subaccounts_without_index(e, &subs, idx);
    storage::set_subaccounts(e, &new_subs);
    storage::remove_subaccount_type(e, &subaccount);

    let deployed_assets = storage::get_deployed_assets(e);
    events::emit_subaccount_removed(e, &admin, &subaccount, deployed_assets);
    // ← deployed_assets is NOT updated; reconciliation delegated to operator off-chain
}
```

Impact

Both scenarios are symptoms of the same missing abstraction. In Scenario 1, no direct fund loss occurs at the recall step since the transaction reverts safely when the protocol does not authorize the transfer. The operational consequence is that any integration with a real DeFi protocol (lending pool, AMM, staking contract) requires redeploying the strategy contract with a hardcoded protocol-specific client, making the strategy non-extensible and unsuitable for multi-protocol deployment without branching logic per integration. In Scenario 2, the absence of an `IProtocol::get_balance()` function forces `deployed_assets` reconciliation to be entirely off-chain and operator-attested, creating a window of accounting drift between strategy removal and the subsequent `update_total_assets` call. During this window, `total_assets` (and thus the vault share price) may be overstated or understated, affecting all depositors.

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:N/I:L/D:N/Y:M (3.8)

Recommendation

It is recommended to define a standard `IProtocol` trait with at minimum two functions: `withdraw(amount: i128) -> i128` (returning actual amount received) and `get_balance() -> i128` (returning the strategy's current balance at the protocol). Each external protocol integration would implement this trait in its own adapter contract. `recall_from_protocol` should invoke `IProtocol::withdraw` via cross-contract call instead of a raw token transfer. `remove_subaccount` should call `IStrategy::get_balance()` (which in turn aggregates idle balance and all `IProtocol::get_balance()` values) to perform an exact, on-chain, automatically-reconciled `deployed_assets` update at the moment of removal, eliminating both the off-chain dependency and the accounting drift window.

Remediation Comment

RISK ACCEPTED: The `Upshift team` has explicitly accepted this risk. In the short term, the custody layer will comprise predominantly EOAs via MPC providers, making the raw `token_client.transfer()` pattern in `recall_from_protocol` operationally sufficient. Fully on-chain strategy contracts implementing an `IProtocol`-compatible interface are planned for subsequent releases.

7.4 ISTRATEGY SHOULD ENFORCE GET_BALANCE() SEMANTICS

// LOW

Description

This vulnerability was identified in commit `18c36059acd4581b837d7b094eac9e66225cefd2`.

The `IStrategy` trait defines the interface that all vault-registered strategies must implement. In the previous version, the trait exposes only four functions:

```
#[contractclient(name = "StrategyClient")]
pub trait IStrategy {
    fn deposit(e: Env, from: Address, amount: i128);
    fn withdraw(e: Env, to: Address, amount: i128) -> i128;
    fn get_balance(e: Env) -> i128;
    fn get_asset(e: Env) -> Address;
}
```

A strategy that implements `get_balance()` as `token.balance(self) + deployed_total` is fully compliant with this interface as defined. However, as described in HAL-02, this implementation is vulnerable to NAV inflation from direct token transfers.

The whitelist review at `add_subaccount` is the only enforcement boundary, but without a `get_local_balance()` function in the trait, the vault cannot probe whether a candidate strategy implements the safe pattern even at registration time.

Impact

Any future strategy implementation that uses `token.balance(self)` for the idle component will be silently vulnerable to the donation attack described in HAL-02. The interface provides no in-band signal to detect this at registration time, placing the full burden of correctness on off-chain whitelist review. A single incorrectly reviewed strategy can inflate the vault's NAV without any on-chain guardrail.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:L/D:N/Y:L (3.1)

Recommendation

It is recommended to extend `IStrategy` with a `get_local_balance()` function that makes the vault-controlled idle tracking an explicit, observable, and required part of the interface:

```
#[contractclient(name = "StrategyClient")]
pub trait IStrategy {
    fn deposit(e: Env, from: Address, amount: i128);
```

```
    fn withdraw(e: Env, to: Address, amount: i128) -> i128;
    fn get_balance(e: Env) -> i128;
    fn get_asset(e: Env) -> Address;
    fn get_local_balance(e: Env) -> i128;
}
```

The `add_subaccount` function should probe `get_local_balance()` at registration time and reject strategies that do not implement it or that return a negative value. This converts a previously off-chain-only enforcement boundary into an on-chain gate for the most common implementation error.

Remediation Comment

SOLVED: The `Upshift` team extended the `IStrategy` trait with a `get_local_balance()` function, with a docstring specifying that it must be incremented on `deposit`, decremented on `withdraw`, and must not include tokens received outside the vault-controlled flow. The `add_subaccount` function now probes `get_local_balance()` at registration time and rejects strategies that do not implement it or return a negative value.

Remediation Hash

https://github.com/fractal-protocol/stellar-vaults/commit/d19a8e382f02ecbf65d00823c961681ecd61d5ad

7.5 STRATEGY ASSET NOT VALIDATED AGAINST VAULT ASSET AT REGISTRATION

// LOW

Description

Both the vault and the strategy are token-agnostic: the vault is initialized with any SEP-41 `asset`, and each strategy instance is independently initialized with its own `asset`. These two assets must match for the vault-strategy interaction to be semantically correct, the vault transfers its own token to the strategy expecting the strategy to hold and manage exactly that token.

However, `add_subaccount` performs no validation that the strategy's configured asset matches the vault's asset. The only check performed for `SubaccountType::Strategy` is a smoke-test that calls `strategy.deposit(vault, 0)` and `strategy.withdraw(vault, 0)` with amount zero. Since both functions exit immediately for zero-amount calls without performing any token transfer, the smoke-test only verifies that the function signatures exist, it provides no information about which token the strategy actually manages.

`IStrategy` does not declare an `asset()` function, so the vault has no standardized way to query the strategy's configured token. Notably, `XlmStrategy` does expose a `get_asset()` public function, but since it is not part of the `IStrategy` trait, the vault cannot call it through the typed `StrategyClient`.

The result is that a strategy configured with token B can be whitelisted in a vault configured with token A, and `add_subaccount` will succeed without any error or warning.

Code Location

Code of `add_subaccount` function from `contracts/august-vault/src/contract.rs` file:

Copy Code

```
391 pub fn add_subaccount(
392     e: &Env,
393     admin: Address,
394     subaccount: Address,
395     subaccount_type: SubaccountType,
396 ) {
397     Self::bump_instance(e);
398     storage::require_admin(e, &admin);
399     storage::require_not_paused(e);
400 
401     let mut subs = storage::get_subaccounts(e);
402 
403     if Self::find_subaccount_index(&subs, &subaccount).is_some() {
404         panic_with_error!(e, VaultError::SubaccountAlreadyRegistered);
405     }
406 
407     if subs.len() >= storage::MAX_SUBACCOUNTS {
408         panic_with_error!(e, VaultError::MaxSubaccountsReached);
409     }
410 
411     // Smoke-test: only verifies function signatures exist.
412     // Does NOT verify that strategy.asset() == vault.asset().
413     if subaccount_type == SubaccountType::Strategy {
414         let vault_addr = e.current_contract_address();
415         let strategy = StrategyClient::new(e, &subaccount);
416         strategy.deposit(&vault_addr, &0);   // ← zero-amount: no token transfer, no asset che
```

```
            strategy.withdraw(&vault_addr, &0);  // ← zero-amount: no token transfer, no asset che
        }

        subs.push_back(subaccount.clone());
        storage::set_subaccounts(e, &subs);
        storage::set_subaccount_type(e, &subaccount, subaccount_type.clone());

        events::emit_subaccount_added(e, &admin, &subaccount, &subaccount_type);
    }
```

Code of `IStrategy` trait from `contracts/august-vault/src/strategy.rs` file:

Copy Code

```
#[contractclient(name = "StrategyClient")]
pub trait IStrategy {
    fn deposit(e: Env, from: Address, amount: i128);
    fn withdraw(e: Env, to: Address, amount: i128) -> i128;
    // ← no get_asset() function: vault cannot query strategy's configured token
}
```

Impact

Beyond the operational failure of a mismatched strategy slot, the deeper risk is silent NAV corruption. `total_assets` is computed as `local_balance + deployed_assets`, where both operands are assumed to be denominated in the vault's configured asset. If a registered strategy manages a different token, `deployed_assets` accumulates values in a different unit, but the vault's share pricing formula treats them as equivalent.

The result is that share prices are computed by summing magnitudes of economically disparate tokens as if they were the same currency. Depending on the relative price of the mismatched token, NAV can be systematically overstated or understated by an arbitrary factor for the lifetime of the misconfigured strategy. This corruption is invisible on-chain since all values are stored as plain i128 with no denomination metadata, and it propagates through every deposit, redemption, and NAV update until the strategy is manually removed.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (2.5)

Recommendation

It is recommended to add an `get_asset` function to the `IStrategy` trait so that the vault can query the strategy's configured token during registration, then `add_subaccount` should verify the match explicitly before whitelisting.

Remediation Comment

SOLVED: The `Upshift team` addressed this finding by adding `get_asset()` and `get_balance()` functions to the `IStrategy` trait. The `add_subaccount` function now calls both during strategy registration: it verifies the strategy's configured asset matches the vault's asset and validates that the returned balance is non-negative.

Remediation Hash

https://github.com/fractal-protocol/stellar-vaults/commit/9f549ae7158aed8f370eebd88cc2cc147e1747b3

7.6 FIRST-DEPOSITOR SHARE INFLATION ATTACK POSSIBLE WHEN DECIMALS_OFFSET IS ZERO

// LOW

Description

The vault uses OpenZeppelin's virtual-shares mechanism to mitigate the classic ERC-4626 first-depositor **share inflation attack**. The protection works by adding `10^decimals_offset` virtual shares to the supply denominator in the share conversion formula:

```
shares = assets × (totalSupply + 10^offset) / (totalAssets + 1)
```

The larger `10^offset` is, the harder and more expensive it becomes for an attacker to manipulate the share price via a direct token donation to the vault. However, the constructor accepts any deployer-provided `decimals_offset` including `0`, with no minimum enforcement.

The attack proceeds as follows: the attacker becomes the first depositor by minting 1 share for 1 token, then donates a large amount of tokens directly to the vault address (not via `deposit`). This inflates `total_assets` without minting new shares, making each share worth a proportionally larger amount. Any subsequent depositor whose deposit amount is smaller than the inflated price per share receives 0 shares due to integer truncation, losing their tokens entirely. If the victim deposits a large enough amount to receive at least 1 share, the attacker redeems their single share for approximately half of all vault assets, extracting value proportional to the donation.

Code Location

Code of `__constructor` function from `contracts/august-vault/src/contract.rs` file:

```
153 pub fn __constructor(
154     e: &Env,
155     name: String,
156     symbol: String,
157     asset: Address,
158     decimals_offset: u32,   // ← accepts 0 with no minimum check
159     admin: Address,
160 ) {
161     Self::bump_instance(e);
162
163     token::TokenClient::new(e, &asset).decimals();
164
165     Vault::set_asset(e, asset);
166     Vault::set_decimals_offset(e, decimals_offset);   // ← stored as-is, no minimum enforced
167     Base::set_metadata(e, Self::decimals(e), name, symbol);
168
169     storage::set_admin(e, &admin);
170     storage::set_deployed_assets(e, 0);
171
172     events::emit_admin_set(e, &admin);
173 }
```

Impact

With `decimals_offset = 0`, a first-depositor attacker can drain approximately 50% of any subsequent large deposit with a donation cost that is proportional to the victim's deposit size. For smaller victim deposits, the attack causes a complete loss via integer truncation (victim receives 0 shares). The attack is fully trustless and requires no privileged access, any external actor can execute it against a freshly deployed vault before any legitimate user deposits.

The economic viability threshold is determined by `10^offset`: with `offset = 0`, the required donation equals the victim's deposit amount, making the attack profitable at any scale. With `offset = 3`, the required donation is 1,000× the victim's deposit, making the attack economically irrational. With `offset = 6` (the value used in tests), the required donation is 1,000,000× the deposit, effectively infeasible.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:L (2.5)

Recommendation

It is recommended to enforce a minimum `decimals_offset` in the constructor, rejecting values below 3 and preferring a default of 6.

Remediation Comment

SOLVED: The `Upshift team` solved this issue by adding a minimum enforcement check in `__constructor`. If `decimals_offset < 3`, the constructor panics with `VaultError::InvalidDecimalsOffset`.

Remediation Hash

https://github.com/fractal-protocol/stellar-vaults/commit/6ff1456ddb9bd1e8e3d4b64fcefcd6e507181c04

7.7 RECALL_FROM_PROTOCOL DOES NOT USE BALANCE-DIFFERENCING

// LOW

Description

`recall_from_protocol` decrements `deployed_total` by the caller-provided `amount` parameter directly, without measuring the actual tokens received. The vault's own `deposit_to_subaccount` and `withdraw_from_subaccount` both use balance-before/after differencing as a defensive pattern to ensure bookkeeping reflects actual token movement rather than declared intent. `recall_from_protocol` is the only point in the system that skips this pattern and trusts the `amount` parameter directly.

Since the strategy constructor accepts any SEP-41 token address with no restrictions on token behavior, a future deployment using a fee-on-transfer token would cause `deployed_total` to be decremented by more than the tokens actually received.

Code Location

Code of `recall_from_protocol` function from `contracts/xlm-strategy/src/lib.rs` file:

Copy Code

```
221 pub fn recall_from_protocol(e: &Env, controller: Address, protocol: Address, amount: i128) {
222     Self::bump_instance(e);
223     controller.require_auth();
224     Self::require_controller(e, &controller);
225     Self::require_valid_protocol_target(e, &protocol);
226 
227     if amount <= 0 {
228         panic_with_error!(e, StrategyError::InvalidAmount);
229     }
230 
231     let token_client = Self::token_client(e);
232     let self_addr = e.current_contract_address();
233     token_client.transfer(&protocol, &self_addr, &amount);  // ← actual received may differ fr
234 
235     let deployed = Self::get_deployed_total(e);
236     let new_deployed = deployed.saturating_sub(amount).max(0);  // ← trusts amount, not actual
237     e.storage()
238         .instance()
239         .set(&StorageKey::DeployedTotal, &new_deployed);
240 
241     RecalledFromProtocol { controller, protocol, amount }.publish(e);
242 }
```

Impact

The risk is forward-looking: if the strategy is deployed with a fee-on-transfer token, `deployed_total` will undercount the actual balance held by the strategy after each recall, causing the vault's reported NAV to be lower than reality. This results in depositors receiving more shares than their fair entitlement during the undercount window.

BVSS

AO:A/AC:L/AX:H/R:N/S:U/C:N/A:N/I:N/D:N/Y:H (2.5)

Recommendation

It is recommended to apply the same balance-differencing pattern used by the vault in `deposit_to_subaccount` and `withdraw_from_subaccount` into `recall_from_protocol` function.

Remediation Comment

SOLVED: The `Upshift team` solved this issue by applying balance-differencing in `recall_from_protocol`. The function now records `balance_before` and `balance_after` the token transfer and decrements `deployed_total` by the actual received delta, not the caller-provided `amount` parameter.

Remediation Hash

https://github.com/fractal-protocol/stellar-vaults/commit/9d9860db36898f4e8f2b65c893522c39805db880

7.8 REMOVE_SUBACCOUNT DOES NOT RECONCILE WALLET DEPLOYED_ASSETS ON REMOVAL

// INFORMATIONAL

Description

`remove_subaccount` removes a subaccount from the vault's whitelist but does not update `deployed_assets` to reflect the removal of that subaccount's capital from the accounting.

The docstring explicitly acknowledges this limitation and delegates reconciliation to a subsequent manual `update_total_assets` call by the operator. However, the justification given applies only to `Strategy` subaccounts with funds deployed to external protocols. For `Wallet` subaccounts, the balance is always directly and atomically queryable via the standard SEP-41 `token.balance(wallet)` call, making the reconciliation entirely feasible on-chain at the moment of removal.

The function already reads `deployed_assets` at the end to include it in the emitted event, but performs no update.

BVSS

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

Recommendation

It is recommended to add subaccount-type branching in `remove_subaccount` so that `Wallet` subaccounts trigger an automatic `deployed_assets` reconciliation using `token.balance(wallet)` at the moment of removal. Full automatic reconciliation for `Strategy` subaccounts with externally deployed capital requires the `IProtocol::get_balance()` interface described in HAL-02.

Remediation Comment

SOLVED: The `Upshift` team added a new `remove_wallet_and_reconcile(admin, operator, subaccount, new_deployed_total)` function that atomically reconciles `deployed_assets` and removes the wallet subaccount in a single transaction, eliminating the pricing window that existed between the previous two-step sequence (`update_deployed_assets` → `remove_subaccount`).

Remediation Hash

https://github.com/fractal-protocol/stellar-vaults/commit/9d9860db36898f4e8f2b65c893522c39805db880

7.9 NO RATE LIMIT ON SETTLE_PROTOCOL_RETURNS

// INFORMATIONAL

Description

This vulnerability was identified in commit `18c36059acd4581b837d7b094eac9e66225cefd2`.

`settle_protocol_returns` allows the controller to set `deployed_total` to any non-negative value in a single call, with no per-call limit and no temporal window. Since `deployed_total` is a direct component of `get_balance()`, which is queried live by the vault in every `total_assets()` computation, a compromised controller can inflate the vault's NAV to an arbitrary value in a single transaction.

The vault already implements AUM rate limits on `deployed_assets` changes via `apply_deployed_assets_change`, but no analogous protection exists for `deployed_total` in the strategy.

Impact

A compromised controller key can inflate the vault's `total_assets()` to an arbitrary value in a single transaction by setting `deployed_total` to a large number. This results in the vault issuing disproportionately few shares on the next deposit and allowing an attacker-controlled address to redeem at the inflated NAV, extracting value from existing depositors.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:H (1.5)

Recommendation

It is recommended to introduce a per-call rate limit on `settle_protocol_returns`, analogous to the vault's `apply_deployed_assets_change` mechanism.

Remediation Comment

SOLVED: The `Upshift` team introduced per-call rate limits on `settle_protocol_returns`, returning a new `SettleRateLimitExceeded` error when exceeded. A `DeployedTotalBootstrapped` latch closes both the cyclic bypass and the first-call-on-pristine bypass, ensuring the bootstrap exemption can only be used once. The `recall_from_protocol` path is deliberately not rate-limited as it represents actual on-chain token movement.

Remediation Hash

https://github.com/fractal-protocol/stellar-vaults/commit/d19a8e382f02ecbf65d00823c961681ecd61d5ad

7.10 REMOVE_SUBACCOUNT DOES NOT ENFORCE ACCOUNTING PRECONDITIONS ON WALLET REMOVAL

// INFORMATIONAL

Description

This vulnerability was identified in commit `18c36059acd4581b837d7b094eac9e66225cefd2`.

`remove_subaccount` can be called on wallet subaccounts without verifying that their attributed value has been fully reconciled first. The vault provides two paths to remove a wallet subaccount: (1) `remove_wallet_and_reconcile`, which is atomic, dual-auth, applies AUM rate limits, and adjusts `deployed_assets` in a single transaction; and (2) `remove_subaccount`, which is admin-only, applies no AUM rate limit, and does not check or update `deployed_assets`.

Path 2 can be used to remove a wallet whose `WalletNetDeployed` tracker is non-zero without any corresponding adjustment to `deployed_assets`. The vault over-reports NAV until a subsequent manual reconciliation via `update_deployed_assets`.

Code Location

Code of `remove_subaccount` function from `contracts/august-vault/src/contract.rs`:

Copy Code

```
820     pub fn remove_subaccount(e: &Env, admin: Address, subaccount: Address) {
821         Self::bump_instance(e);
822         storage::require_admin(e, &admin);
823         Self::do_remove_subaccount(e, &admin, &subaccount);
824     }
```

Impact

An admin removing a wallet with a non-zero `WalletNetDeployed` tracker via `remove_subaccount` leaves `deployed_assets` overstated by the tracker amount, inflating the vault's NAV and share price until manually corrected.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:H (1.5)

Recommendation

It is recommended to add a precondition check in `remove_subaccount` that rejects wallet removal when the `WalletNetDeployed` tracker is non-zero.

Remediation Comment

SOLVED: The `Upshift` team added a precondition check in `remove_subaccount` that rejects wallet removal when `get_wallet_net_deployed(subaccount) != 0`.

Remediation Hash

https://github.com/fractal-protocol/stellar-vaults/commit/d19a8e382f02ecbf65d00823c961681ecd61d5ad

7.11 AUTHORIZATION CHECK OCCURS AFTER IDENTITY EQUALITY CHECK

// INFORMATIONAL

Description

`require_admin` and `require_operator` check whether the caller's identity matches the stored role before calling `caller.require_auth()`.

This is not exploitable in the current Soroban auth model, `require_auth()` is correctly enforced for the matched identity. However, the pattern is atypical: the standard Soroban convention is to call `require_auth()` first, then check roles.

Code Location

Code of `require_admin` function from `contracts/august-vault/src/storage.rs` file:

Copy Code

```
54 pub fn require_admin(e: &Env, caller: &Address) {
55     let admin = get_admin(e);
56     if *caller != admin {                  // ← identity check first
57         panic_with_error!(e, VaultError::Unauthorized);
58     }
59     caller.require_auth();                 // ← auth registered after
60 }
```

BVSS

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

Recommendation

It is recommended to swap the order so that `require_auth()` is always called first, consistent with the standard Soroban defensive pattern.

Remediation Comment

SOLVED: The `Upshift team` swapped the authorization order in both `require_admin` and `require_operator`.

Remediation Hash

https://github.com/fractal-protocol/stellar-vaults/commit/9d9860db36898f4e8f2b65c893522c39805db880

7.12 INCONSISTENT TTL MANAGEMENT ACROSS VAULT AND STRATEGY

// INFORMATIONAL

Description

Both the vault and the strategy use Soroban Instance storage for all operational state. Instance storage carries a TTL that must be actively extended to prevent the entry from being archived. With the Stellar protocol upgrade CAP-046, archived entries are automatically restored when an operation that requires them is executed, eliminating the permanent loss risk that existed in earlier protocol versions.

However, automatic restoration carries a higher ledger fee than a proactive `bump_instance` call, since it includes both the restoration cost and the standard operation fee. This cost is paid by whoever triggers the restoring transaction (typically an end user performing a deposit or withdrawal) without any indication that a higher-than-normal fee will be charged.

Three specific gaps in the current implementation increase the likelihood of hitting automatic restoration and its associated costs:
1. `XlmStrategy` **constructor does not call** `bump_instance`: The constructor sets all instance storage keys but never extends the TTL. The contract starts with the network minimum TTL meaning the strategy could expire very quickly after deployment if no user interaction occurs.
2. Zero-amount paths skip `bump_instance`: `XlmStrategy.deposit()` and `withdraw()` return early for `amount == 0` before calling `bump_instance`.
3. TTL constants are inconsistent between vault and strategy: `XlmStrategy` uses locally defined constants equivalent to 7 days (`INSTANCE_EXTEND_AMOUNT = 120,960 ledgers`), while `AugustVault` uses the OpenZeppelin `stellar_tokens` library constants equivalent to approximately 30 days. A strategy holding deployed capital can expire four times faster than the vault that manages it.

BVSS

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

Recommendation

It is recommended to add `Self::bump_instance(e)` as the first line of `XlmStrategy::__constructor`, mirroring the pattern already used in `AugustVault::__constructor`. It is also recommended to move the `bump_instance` call before the zero-amount early returns in `deposit()` and `withdraw()`, and to align `INSTANCE_EXTEND_AMOUNT` with the vault's 30-day constant to ensure both contracts operate under a consistent TTL policy.

Remediation Comment

SOLVED: The `Upshift team` resolved all three gaps identified in this finding:

- `Self::bump_instance(e)` added as the first line of `XlmStrategy::__constructor`,
- `bump_instance` moved before the zero-amount early returns in `deposit()` and `withdraw()`,
- `INSTANCE_EXTEND_AMOUNT` in `XlmStrategy` aligned from 7 days to 30 days.

Remediation Hash

https://github.com/fractal-protocol/stellar-vaults/commit/9d9860db36898f4e8f2b65c893522c39805db880

7.13 UPDATE_TOTAL_ASSETS MISLEADINGLY NAMED

// INFORMATIONAL

Description

The public function `update_total_assets` only updates the `deployed_assets` storage value, it does not modify `local_balance`, nor does it compute or write `total_assets` as a whole.

This naming inconsistency creates two concrete risks:
1. Operator confusion: An operator who reads the function name without examining the implementation may call `update_total_assets(total_vault_NAV)`, passing the sum of local and deployed capital, instead of `update_total_assets(deployed_capital_only)`. This would overstate `deployed_assets` by exactly `local_balance`, causing the vault to double-count the locally held tokens in `total_assets()` and inflate the share price proportionally.
2. Auditor and integrator confusion: External parties reviewing or integrating with the vault may assume the function reconciles both components of NAV. The discrepancy between the name and the behavior is a documentation gap that increases the likelihood of misuse over time.

BVSS

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

Recommendation

It is recommended to rename the function to `update_deployed_assets` to accurately reflect its effect, and update all off-chain tooling, scripts, and documentation accordingly.

Remediation Comment

SOLVED: The `Upshift team` renamed `update_total_assets` to `update_deployed_assets` across the project, the new name accurately reflects the purpose of the function.

Remediation Hash

https://github.com/fractal-protocol/stellar-vaults/commit/6ff1456ddb9bd1e8e3d4b64fcefcd6e507181c04

7.14 STRATEGY REMOVAL DOES NOT RECORD NAV IMPACT ON-CHAIN

// INFORMATIONAL

Description

This vulnerability was identified in commit `18c36059acd4581b837d7b094eac9e66225cefd2`.

The gap is one of auditability: the `SubaccountRemoved` event does not record the strategy's balance at the time of removal. There is therefore no on-chain record linking the NAV change to the removal transaction. Off-chain indexers that track the vault's share price will observe a step-change in NAV with no associated event explaining the cause or magnitude.

Code Location

`SubaccountRemoved` event definition from `contracts/august-vault/src/events.rs`:

Copy Code

```
// SubaccountRemoved event (events.rs)
pub struct SubaccountRemoved {
    #[topic]
    pub admin: Address,
    pub subaccount: Address,
    pub deployed_assets: i128,
    // strategy_balance field absent
}
```

Impact

The absence of the strategy balance in the `SubaccountRemoved` event makes it impossible to reconstruct the NAV impact of a strategy removal from on-chain data alone. Off-chain monitoring systems cannot distinguish a removal of a zero-balance strategy from one with significant capital, making the event insufficient for audit trail purposes.

BVSS

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

Recommendation

It is recommended to enrich the `SubaccountRemoved` event for strategy subaccounts with the result of a `try_get_balance()` call at the time of removal.

Remediation Comment

SOLVED: The `Upshift` team enriched the `SubaccountRemoved` event with a `strategy_balance: Option<i128>` field, populated via a `try_get_balance()` probe at removal time. A companion event `StrategyBalanceProbeFailed` with a `StrategyProbeFailure` enum (`InvokeError` / `ConvertError`)

is emitted when the probe fails, allowing off-chain indexers to correlate failures by `(tx_hash, subaccount)`.

Remediation Hash

https://github.com/fractal-protocol/stellar-vaults/commit/d19a8e382f02ecbf65d00823c961681ecd61d5ad

---

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.