certora

Security Assessment
Final Report

Volo Vault

June 2025

Prepared for Navi

Table of contents

Project Summary

Project Scope

Project Name	Repository (link)	Audit Commit Hash	Fix Review Commit Hash	Platform
Volo Vault	https://github.com/Sui-Volo/volo-vault	f5fcf80	523c82e	Sui

Project Overview

This document describes the specification and verification of **Volo Vault** using manual code review findings. The work was undertaken from **June 2** to **June 13, 2025**

The following contract list is included in our scope:

```
{
      sources/utils.move
      sources/volo_vault.move
      sources/manage.move
      sources/operation.move
      sources/user_entry.move
      sources/vault_receipt_info.move
      sources/receipt.move
      sources/requests/withdraw_request.move
      sources/requests/deposit_request.move
      sources/adaptors/momentum.adaptor.move
      sources/adaptors/cetus_adaptor.move
      sources/adaptors/suilend_adaptor.move
      sources/adaptors/receipt_adaptor.move
      sources/adaptors/navi_adaptor.move
      sources/reward_manager.move
      sources/oracle.move
}
```

The team performed a manual audit of all the Move modules**.** During the manual audit, the Certora team discovered bugs in the Move contracts code, as listed on the following page.

Protocol Overview

Volo Vault is a vault where users can deposit their funds in exchange for shares, while assigned operators manage the investment strategy of those funds.
The vault has built in mechanism to prevent operators from acting maliciously and stealing the funds they manage.
The yield from the vault is distributed equally between the share holders.
On top of that, share holders are also eligible for rewards which are distributed equally as well.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	2	2	2
High	4	4	4
Medium	2	2	2
Low	5	5	5
Informational	5	5	5
Total	18	18	18

Severity Matrix

Impact		Likelihood: Low	Likelihood: Medium	Likelihood: High
	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium

Detailed Findings

ID	Title	Severity	Status
C-01	Vault evaluation can be manipulated by calling update_coin_type_asset_value() on the principal coin	Critical	Fixed
C-02	Receipt's reward isn't accumulated during deposit/withdraw execution, leading to accounting issues	Critical	Fixed
H-01	Operator can steal funds using a nested operation and execute a deposit in between	High	Fixed
H-02	Receipt adapter wrongly accounts for claimable principal and pending deposits	High	Fixed
H-03	Operator might underestimate the vault's value by using a receipt from a vault that is during an operation	High	Fixed
H-04	Operator can overestimate Cetus and Momentum positions by manipulating the liquidity in the pool	High	Fixed
M-01	Lack of assert_vault_receipt_matched() check in several functions	Medium	Fixed
M-02	Interest isn't compounded in Suilend and Navi adapters, leading to a misestimation of the position	Medium	Fixed
L-01	Tolerance limit uses the current total value while the loss is calculated based on the previous total value	Low	Fixed
L-02	Malicious users can spam always-fail requests leading to saturation of request queue	Low	Fixed

L-03	Admin can set vault's status to normal during operation by calling set_enabled()	Low	Fixed
L-04	Some vault setter functions don't have a public function to access them	Low	Fixed
L-05	Missing version checks in some functions	Low	Fixed

Critical Severity Issues

C-01 Vault evaluation can be manipulated by calling `update_coin_type_asset_value()` on the principal coin

Severity: **Critical**	Impact: **High**	Likelihood: **High**
Files: sources/volo_vault.move	Status: Fixed	

Description: The free principal evaluation should be done by calling `update_free_principal_value()`.
However, the update can also be done by calling `update_coin_type_asset_value()` with the CoinType set to the principal coin type. In that case the evaluation of the principal coin would be zero, since the function would read the zero balance that's stored in the bag during initialization.

This can cause a significant underestimation of the vault's value, as if the free principal doesn't exist.

Exploit Scenario:

- Vault has an evaluation of 100K USD, where 50K out of that is free principal and 100K shares
- Bob requests a deposit of 1 USD worth of the principal coin
- Bob calls `update_coin_type_asset_value()` right before the operator executes the request
    - This would bring the evaluation down to 50K
- The operator executes the request
    - The evaluation after would come back to 100K, meaning the difference evaluation before and after the deposit would be 50K
    - Bob would get 100K shares, which are worth 50K USD now
- Bob was able to get 50K USD worth of shares at the expense of the other share holders

Recommendations: In `update_coin_type_asset_value()` add a check that the coin isn't the principal coin.

Customer's response: Fixed in 5696908

Fix Review: Fix looks good

C-02 Receipt's reward isn't accumulated during deposit/withdrawal execution, leading to accounting issues

Severity: **Critical**	Impact: **High**	Likelihood: **High**
Files: sources/vault_receipt_info.move	Status: Fixed	

Description: Reward is calculated proportionally to the shares the user/recipient holds.
In order for the accounting to be accurate `VaultReceiptInfo.update_reward()` has to be called for every reward every time the shares in the receipt change.

However, this isn't done during deposit and withdrawal execution, when the shares in the receipt change, leading to accounting errors. Some errors would get more rewards while others would get less, and the total amount of claimable rewards would be out of sync with the total available rewards.

Exploit Scenario:

- Bob deposits and gets 10 at day 1
- Bob deposits another 90 shares after 5 days
- Bob claims rewards on day 6
    - Bob gets rewards as if they've deposited 100 shares on day 1, which can be significantly more then they're eligible for

Recommendations: Update the reward during deposit/withdraw execution

Customer's response: Fixed in 7ad9d85

Fix Review: Fix looks good

High Severity Issues

H-01 Operator can steal funds using a nested operation and execute a deposit in between

Severity: **High**	Impact: **High**	Likelihood: **Medium**
Files: sources/operation.move	Status: Fixed	

Description: During an operator's operation, the vault attempts to prevent execution of normal functions by setting the vault's status to 'during operation'

However, an operator can bypass this by starting 2 operations concurrently, this would set the status back to normal after the first operation ends, allowing the operator to execute normal functions before the second operation ends.

Exploit Scenario:

- Bob requests a deposit of 10K USD worth of the principal coin
- Operator starts the first operation with no assets
- Operator starts a second operation, borrowing the free principal
- Operator ends the first operation
    - This sets the status back to normal
- Operator executes Bob's deposit request
- Operator returns the free principal, minus the amount that Bob has deposited
- The operator has successfully stolen funds from the shares holders

Recommendations: Prevent starting another operation when the previous operation hasn't ended yet by calling `assert_normal()` instead of `assert_enabled()` at `pre_vault_check()`

Customer's response: Fixed in 5696908

Fix Review: Fix looks good

H-02 Receipt adapter wrongly accounts for claimable principal and pending deposits

Severity: **High**	Impact: **High**	Likelihood: **Medium**
Files: sources/adaptors/receipt_adaptor.move	Status: Fixed	

Description: When evaluating the value of the receipt we're adding the pending deposit and claimable principal to the value, but we don't multiply it by the price of the asset.
This would cause a wrong evaluation of the receipt and, subsequently, the vault.

```Rust
    let value =
         vault_utils::mul_d(shares, share_ratio) +  (vault_receipt.pending_deposit_balance()
as u256) + (vault_receipt.claimable_principal() as u256);
```

Recommendations: Account for the value of the claimable principal and pending deposits using the oracle price for the principal coin.

Customer's response: Fixed in a3af9fe

Fix Review: Fix looks good

H-03 Operator might underestimate the vault's value by using a receipt from a vault that is during an operation

Severity: **High**	Impact: **High**	Likelihood: **Medium**
Files: sources/adaptors/momentum.adaptor.move	Status: Fixed	

Description: The receipt adapter doesn't check that the receipt's vault status is normal. This would allow an operator to underestimate the value of the receipt by evaluating it while the vault is undergoing an operation.

Exploit Scenario:

- Bob is the operator of both vaults A and B
- Bob adds a receipt from vault B as one of the assets of vault A
- Bob starts an operation on vault B, removing free principal
- Bob calls `update_free_principal_value()`, lowering the evaluation of the vault
- Bob starts an operation on vault A, which underestimates the receipt value
- Bob finishes op on vault B, returning free principal and getting the evaluation back to normal
- Bob finishes op on vault A, while keeping some assets to themselves.
    - This would pass since the evaluation diff on the receipt would cover this

Recommendations: Assert that the receipt's vault status is normal and not during an operation.

Customer's response: Fixed in 9c539de

Fix Review: Fix looks good

H-04 Operator can overestimate Cetus and Momentum positions by manipulating the liquidity in the pool

Severity: **High**	Impact: **High**	Likelihood: **Medium**
Files: cetus_adaptor.move momentum.adaptor.move	Status: Fixed	

Description: The value of Cetus and Momentum positions can be manipulated and overestimated by large swaps.
Both work on the basic formula of Uniswap (x*y=k) where large swaps increase the value of the liquidity in the pool (this is supposed to keep the pool balanced and close to the real price).
An operator can utilize that and sandwich an operation-end with a large swap, overestimating the position's value and stealing funds in the same value.

Exploit Scenario:

- The vault holds a Momentum position with 10% of the liquidity in the pool
  - The pool has 100K USDC and 100K wUSDT, so the position's value is $200K * 10\% = 20K\ USD$
- The operator starts an operation, taking 22.5K USD worth of free principal
- In the same tx the operator swaps 300K USDC for 75K USDT
  - Pool now has 400K USDC and 25K wUSDT
  - Position is now worth $425K * 10\% = 42.5K\ USD$
- The operator ends the operation without returning the free principal
  - This would pass because the position overestimation would cover the cost
- Operator swaps back the 75K wUSDT for 300 USDC
- The swap fees are 0.01% of the input, so that would be $375K * 0.01\% = 37.5\ USD$ for both swaps
- The operator has successfully stolen 22.5K USD from the vault

Recommendations: Check that the liquidity in the pool isn't manipulated by comparing the pool's price with the oracle's price.

Customer's response: Fixed in 9c539de..a0e19da

Fix Review: Fix looks good

Medium Severity Issues

M-01 Lack of `assert_vault_receipt_matched()` check in several functions

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: sources/user_entry.move	Status: Fixed	

Description: The deposit and claim principle functions lack the `assert_vault_receipt_matched()` check, allowing users to use the same receipt across different vaults.

Recommendations: Add the check to those functions.

Customer's response: Fixed in f753bb1

Fix Review: Looks good

M-02 Interest isn't compounded in Suilend and Navi adapters, leading to a misestimation of the position

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: sources/adaptors/suilend_adaptor.move sources/adaptors/navi_adaptor.move	Status: Fixed	

Description: The Suilend adapter estimates the value of a Suilend position/obligation by summing up the value of all of the deposits and borrows.
However, we don't compound the interest before doing the estimation. This would cause an underestimation of both deposits and borrows, causing an under- or overestimation of the value of the obligation.

Recommendations: Compound the interest for relevant reserves before estimating the value of the position.

Customer's response: Fixed in a65f95e (Suiled) and ff72b5f (Navi - here we use the dynamic calculator to estimate the compounded interest)

Fix Review: Fixes look good

Low Severity Issues

L-01 Tolerance limit uses the current total value, while the loss is calculated based on the previous total value

Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: sources/volo_vault.move	Status: Fixed	

Description: In order to enforce the loss limit we store the accumulated loss in the epoch in `cur_epoch_loss` in terms of total value lost, then we compare it with the limit, which is calculated as a percentage of the *current* total value in the vault.
The issue is that in case of a decrease in the total value due to withdrawal, the comparison wouldn't consider the loss as a percentage at the time of the loss, but rather as a percentage of the current value.

```rust
self.cur_epoch_loss = self.cur_epoch_loss + loss;
let loss_limit = usd_value_before * (self.loss_tolerance as u256) / (RATE_SCALING as u256);
assert!(loss_limit >= self.cur_epoch_loss, ERR_EXCEED_LOSS_LIMIT);
```

Exploit Scenario:

1. 1M USD worth of assets in the vault
2. We have a loss of 900 USD in an operation at the beginning of the epoch (below the tolerance limit, which is 0.1%)
3. Somebody withdraws 0.5M USD
4. Now the operator does an operation that has a loss of 1 USD, this would revert because the total loss during the epoch (901 USD) is higher than the new loss limit (`0.5M * 0.1% = 500 USD`)

Recommendations: Consider storing the loss as a percentage of total value, this way it wouldn't revert in the above case.

Customer's response: Under commit bf549dd we've modified the way the loss limit is calculated, instead of using the current total value we use the total value at the time of the first operation during an epoch.

Fix Review: Looks good.

L-02 Malicious users can spam always-fail requests, leading to saturation of request queue

Severity: **Low**	Impact: **Low**	Likelihood: **Medium**
Files: sources/user_entry.move	Status: Fixed	

Description: Malicious actors can overwhelm the vault's deposit and withdraw queue by repeatedly calling `request_deposit` with parameters that guarantee execution will fail - such as a zero deposit amount, an amount so small it rounds to zero shares, or an unrealistically high `expected_shares` value.

Each invocation creates a unique `request_id` and buffers the (zero) coins, and because there is no merging or deduplication, these bad requests accumulate one by one in `request_buffer.deposit_requests`. A very similar queue-saturation attack works on withdrawals.

After depositing to obtain shares, a malicious user can spam `request_withdraw` with parameters guaranteed to fail execution - for example, requesting a small number of shares but setting `expected_amount` too high.

Once a request is in the buffer, neither the user nor the operator can cancel it until five minutes have elapsed, thanks to the enforced cancel-lock. During that window, an off-chain operator process that continuously scans and executes pending deposits will be forced to attempt each invalid request, hit the `ERR_ZERO_SHARE` or `ERR_UNEXPECTED_SLIPPAGE` abort, and possibly move on - yet the queue remains clogged. With enough spam, honest deposits and cancellations become impractically slow or require manual cleanup. It should be highlighted that the request executor is likely to be an off-chain script, which might become non-functional as a result of such action.

Exploit Scenario:

- There is an off-chain component running the execution of requests
- A malicious user starts to create requests that are guaranteed to fail
- Even if the component properly handles reverting transactions, it may become saturated, as the always-reverting requests are inexpensive to create
- The cancellations can only start after 5 minutes grace period, which might be enough to fill up the requests queue
- Cancelling those requests requires manual intervention and may effectively disrupt protocol operations.

Recommendations: To defend against this, the protocol should enforce a nonzero minimum deposit amount and reject tiny or impossible slippage parameters at request time. It could also merge multiple pending requests from the same user into a single buffered entry or run basic sanity checks (e.g. “expected_shares ≤ max reasonable value”) on creation.

Customer’s response: Fixed by adding zero checks

Fix Review: Looks good

L-03 Admin can set vault's status to normal during operation by calling `set_enabled()`

Severity: **Low**	Impact: **High**	Likelihood: **Low**
Files: sources/volo_vault.move	Status: Fixed	

Description: The function `set_enabled()` allows the admin to set the vault status to normal or disabled, even when the vault's status is 'during operation'

This would allow an admin to set the status back to normal during operation and execute deposits, which can be used to steal funds from the vault.

Recommendations: In `set_enabled()` check that the current status isn't 'during operation'.

Customer's response: Fixed in 5696908

Fix Review: Looks good

L-04 Some vault setter functions don't have a public function to access them

Severity: **Low**	Impact: **Low**	Likelihood: **Medium**
Files: sources/volo_vault.move	Status: Fixed	

Description: The following functions don't have a public function in the manager/operator module to access them:

- set_locking_time_for_withdraw()
- set_locking_time_for_cancel_request()

Recommendations: Add functions in the manager or operator module to access them.

Customer's response: Fixed in a3af9fe

Fix Review: Looks good

L-05 Missing version checks in some functions		
Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: sources/oracle.move	Status: Fixed	

Description: The functions `set_update_interval()` and `set_coin_decimals()` in `Oracle.move` are missing a version check.

Recommendations: Add a version check to those functions

Customer's response: Fixed in 5696908

Fix Review: Fix looks good

Informational Issues

I-01. Coin support cannot be revoked

Description: In `volo_vault::vault`, operators can call `add_new_coin_type_asset<PrincipalCoinType, AssetType>()` to register a new coin-type asset, but there is no corresponding `remove_coin_type_asset` function.

By contrast, DeFi assets can be unregistered via `remove_defi_asset_support` in the same module. As a result, once a coin type is added to `vault.assets` and `asset_types`, it remains on-chain forever. This decreases the maintainability of the protocol.

Recommendation: Add a corresponding removal routine, similar to how defi assets support can be revoked.

Customer's response: Fixed. Added function to remove coin type asset.

Fix Review: Looks good

I-02. Division by zero error if rewards are added before users join the vault

Description: In `RewardManager::update_reward_indices`, the code is as follows when called from inside `add_reward_balance`:

```rust
    let add_index = vault_utils::div_d(
        reward_amount as u256,
        vault.total_shares(),
    );
```

If `vault.total_shares() == 0`, this integer division by zero aborts. Because there's no guard, any call to `add_reward_balance` before the first deposit will always revert. While this does not impact the protocol, the revert seems unexpected, hence a graceful error handler should be implemented to account for this case.

Recommendation: Add an explicit check before division such as `assert!(vault.total_shares() > 0, ERR_NO_SHARES_YET);`

Customer's response: Fixed. Added the check for total shares.

Fix Review: Looks good

I-03. Unused fields in RewardManager

Description: `receipt_reward_indices` and `receipt_unclaimed_rewards` fields in the `RewardManager` aren't used

Recommendation: Remove those fields

Customer's response: Removed

Fix Review: Looks good

I-04. Lending positions adapters might underflow under bad debt

Description: For both the Navi and Suilend adatpors we estimate the value of the position by subtracting the total debts from the total positions.
However, under a case of a severe bad debt this might underflow and revert.

Recommendation: In case that the total borrow is more than the total supply return 0

Customer's response: Fixed in a568dcd and a65f95e

Fix Review: Looks good

I-05. `OracleConfig.coin_decimals` isn't used

Description: The `OracleConfig.coin_decimals` isn't used in the package, instead we're using the `PriceInfo.decimals` field to normalize the price. Therefore, this field can be removed.

Recommendation: Remove this unused field

Customer's response: Removed

Fix Review: Looks good

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.