OpenZeppelin | security

Very Liquid Vaults Audit

September 11, 2025

Table of Contents

Summary

Type	DeFi
Timeline	From 2025-08-18 To 2025-08-22
Languages	Solidity

Total Issues	10 (10 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	1 (1 resolved)
Low Severity Issues	6 (6 resolved)
Notes & Additional Information	3 (3 resolved)

Scope

OpenZeppelin audited the SizeCredit/very-liquid-vaults repository at commit d5d781c.

In scope were the following files:

```
src
├── Auth.sol
├── IVault.sol
├── SizeMetaVault.sol
├── strategies
│   ├── AaveStrategyVault.sol
│   ├── CashStrategyVault.sol
│   └── ERC4626StrategyVault.sol
└── utils
    ├── BaseVault.sol
    ├── NonReentrantVault.sol
    └── PerformanceVault.sol
```

System Overview

The Very Liquid Vault (as audited: *Size Meta Vault*) is a modular "meta" vault that allocates its assets to underlying vaults called strategies. The meta vault is ERC-4626-compliant and allows users to deposit and withdraw an underlying asset such as USDC. The meta vault automatically allocates the underlying assets to the strategies during deposits. The strategies are also ERC-4626-compliant.

Currently, there are three types of strategy vaults. An AAVE strategy vault, a generic ERC-4626 strategy vault, and a cash strategy vault for the reserve store. The meta vault itself can also be used as a strategy vault. It can have up to 10 strategy vaults, and mix and match different types of strategy vaults. However, the meta vault and its strategy vaults must all have the same underlying asset.

The meta vault features a protocol fee system. A configurable percentage of the profit generated is allocated to the fee recipient by way of minting new shares. The protocol also includes a role-based access-control contract. Instead of each vault having its own access-control management system, they all share a single contract for unified management of roles.

Security Model and Trust Assumptions

The protocol uses certain privileged roles to manage the allocation of the assets for its users.

Privileged Roles

The protocol is governed by a role-based system managed by a single access-control contract. Below are the roles and their expected timelock setups.

- `DEFAULT_ADMIN_ROLE` (7-day timelock): It has ultimate control over the system. It can upgrade contracts, grant and revoke any role, and modify critical parameters such as the performance fee percentage.

- `VAULT_MANAGER_ROLE` (1-day timelock): It manages the vault's strategies and operational state. It can add new strategies, set total asset caps, and unpause the system.
- `GUARDIAN_ROLE` (no timelock): It is a trusted role for incident response. It can pause the system, remove a strategy in an emergency, and cancel any pending timelock proposals.
- `STRATEGIST_ROLE` (no timelock): It is responsible for tactical fund management. It can execute rebalances between strategies and reorder the deposit/withdrawal priority of strategies.

Trust Assumptions

During the audit, the following trust assumptions were made:

- All privileged roles are trusted to act in the best interests of the users. However, there are certain limitations on each role regarding any potential malicious actions.
- The Admin has absolute control over the protocol through its ability to upgrade the contracts. However, the 7-day timelock reduces the required trust as it allows users to act upon any harmful upgrades.
- The Vault Manager has also a lot of control as it can add arbitrary strategies. Similarly, the 1-day timelock reduces the required trust as it allows an exit window for the users, albeit a short one.
- The Strategist has the lowest trust required as it can only rebalance between existing strategies. Additionally, the potential harm a Strategist can do is limited by the maximum slippage configuration that must be respected during rebalancing. The guardian is perhaps the role that requires the highest trust.
- The guardian does not have a timelock, and it can permanently cause DoS for the protocol by pausing the contract and blocking any proposals to unpause it. The guardian can also remove a strategy with unlimited slippage, bypassing the maximum slippage limit configuration. Furthermore, the guardian can forfeit all assets during a strategy removal. Although re-adding the strategy would reclaim those assets, anyone who deposits to the vault before the re-adding of the strategy would benefit from the re-inclusion of the assets at the expense of the other users. Therefore, the guardian must be fully trusted to act non-maliciously.

Design Considerations

Deposits and withdrawals are processed sequentially according to a priority list of strategies set by the Strategist. For example, withdrawals are fulfilled from the first strategy in the list with sufficient liquidity, then the second, and so on. This design choice requires the Strategist to frequently rebalance and reorder the strategies.

In their documentation, the team also acknowledges the following design choices:

- The performance fee might stop being applied following major downturns as the price per share might never reach the high-water mark again.
- Assets sent directly to vaults may be lost (except the cash strategy vault, which treats them as donations).
- The vaults are incompatible with fee-on-transfer assets.
- An ERC-4626 strategy vault only supports fully ERC-4626-compliant vaults that do not take any fees on deposits/withdrawals.

Medium Severity

M-01 Incorrect Order of Operations Causes Permanent Reduction in Protocol Fees

In the `_mintPerformanceFee` function of the `PerformanceVault` contract, protocol fees are based on the difference between the current price-per-share (PPS) and the high-water-mark (HWM), which tracks the highest PPS reached and is only updated when the current PPS exceeds it. Following the fee shares calculation, the HWM is set to the current PPS and then the fee shares are minted.

This order of operations introduces dilution. Once fee shares are minted, the total supply increases while the amount of assets remains the same, which reduces PPS. Since the HWM was already updated to the pre-dilution PPS, it now sits above the post-dilution PPS. The vault will not charge another performance fee until the PPS surpasses this inflated HWM, resulting in a permanent reduction of fee revenue.

Consider updating the HWM with the current PPS value *after* the fee shares have been minted to ensure accurate accounting and prevent permanent fee revenue loss.

Update: *Resolved in pull request #38 at commit dd634d4.*

Low Severity

L-01 Share Transfers Blocked During Pause

The `_update` function in `BaseVault.sol` has a `notPaused` modifier which blocks all ERC-20 operations (`transfer`, `transferFrom`, `mint`, and `burn`). This is overly restrictive as it prevents user-to-user share transfers during incidents. Pausing should gate vault I/O (deposit, mint, withdraw, and redeem), not secondary-market movement of existing shares. This may cause secondary liquidity freezes, collateral issues and integration fails, and will result in the governance gaining de facto control over transferability. While this issue does not entail direct fund loss, users can still be locked into positions during pauses.

Consider removing the `notPaused` modifier from the `_update` function and applying pause checks only to vault I/O functions.

Update: *Resolved in pull request #39 at commits 91a1fcf and e847e3f.*

L-02 Malicious or Faulty Strategies Cannot Be Removed

The `removeStrategy` function of the `SizeMetaVault` contract calls the `convertToAssets` function of the strategy being removed. This call is performed even when the guardian forfeits all assets by setting the rebalance amount to zero. A malicious or faulty strategy can cause DoS by reverting in its `convertToAssets` function. This would prevent the removal of the strategy. Such a strategy could also be causing DoS in the core function of the meta vault, completely bricking the protocol.

Consider not calling the strategy and rebalancing if the rebalance amount is zero.

Update: *Resolved in pull request #40 at commit 7cb122a.*

L-03 Missing, Incomplete, and Misleading Documentation

Throughout the codebase, multiple instances of misleading documentation were identified:

- In `SizeMetaVault.sol`, the NatSpec for `removeStrategy` incorrectly states `VAULT_MANAGER_ROLE` can call it, but the implementation correctly uses `GUARDIAN_ROLE`.
- In `BaseVault.sol`, a comment above `setTotalAssetsCap` claims only the `Auth` contract can call it, but the code allows any address with `VAULT_MANAGER_ROLE`.

Throughout the codebase, multiple instances of missing or incomplete documentation were identified:

- `Auth.sol`: `initialize`
- `IVault.sol`: `auth`, `totalAssetsCap`
- `SizeMetaVault.sol`: `initialize`, `maxDeposit`, `maxMint`, `maxWithdraw`, `maxRedeem`, `totalAssets`, `setPerformanceFeePercent`, `setFeeRecipient`, `setRebalanceMaxSlippagePercent`, `addStrategy`, `removeStrategy`, `reorderStrategies`, `rebalance`, `strategies`, `strategies (index)`,

`strategiesCount`, `rebalanceMaxSlippagePercent`, `isStrategy`, `MAX_STRATEGIES`, Events (Lines 42 to 47)
- `AaveStrategyVault.sol`: `initialize`, `maxDeposit`, `maxMint`, `maxWithdraw`, `maxRedeem`, `totalAssets`, `pool`, `aToken`, Events (Lines 61 to 62)
- `ERC4626StrategyVault.sol`: `initialize`, `maxDeposit`, `maxMint`, `maxWithdraw`, `maxRedeem`, `vault`, `VaultSet` Event
- `BaseVault.sol`: `initialize`, `setTotalAssetsCap`, `decimals`, `maxDeposit`, `maxMint`, `maxWithdraw`, `maxRedeem`, `auth`, `totalAssetsCap`, Events (`52-54`)
- `NonReentrantVault.sol`: `deposit`, `mint`, `withdraw`, `redeem`
- `PerformanceVault.sol`: `deposit`, `mint`, `withdraw`, `redeem`, `highWaterMark`, `performanceFeePercent`, `feeRecipient`, State Variables (Lines 15 to 16), Events (Lines 39 to 42)

There are also two other minor issues:

- The default max rebalance slippage is assigned as a literal (`0.01e18`). This variable could instead be a named constant or there could be an inline comment specifying that the default max rebalance slippage is 1%.
- Import paths contain unnecessary double slashes in `BaseVault.sol` and `SizeMetaVault.sol`.

Consider updating the documentation to match the code, completing the NatSpec documentation, adding inline comments where relevant, and fixing the double slashes. The `@inheritdoc` tag can be used in the NatSpec documentation where relevant to prevent duplicate comments.

Update: *Resolved in pull request #47 at commit dbba5d6. The Size Credit team stated:*

> *The NatSpec for events and errors was purposefully left out as we consider these self-describing. Moreover, other well-known Solidity projects such as OpenZeppelin's `openzeppelin-contracts` repository follow the same practice.*

L-04 Incorrect Return Values for Nested Meta Vaults Sharing the Same Strategy

The `_maxWithdrawFromStrategies` and `_maxDepositToStrategies` functions of the `SizeMetaVault` contract can return incorrect values in case of nested meta vaults. For example, the issue occurs if a meta vault has two strategies, one of which is an ERC-4626 strategy and the other is a meta vault strategy that has the same ERC-4626 strategy. In this scenario, if the ERC-4626 vault has a `maxDeposit` limit of 100 tokens remaining, the top-

level meta vault would double count this value and return 200 tokens. However, in practice, trying to deposit 200 tokens would cause a revert, because only 100 can be deposited.

Consider documenting this behavior and either acknowledging the issue or explicitly stating that such nested strategy setups are not supported. Otherwise, updating the code to return correct values appears to be unfeasible.

Update: *Resolved in pull request #41 at commit ed2c2a8.*

L-05 Incomplete Reentrancy Protection

The `SizeMetaVault` contract does not have the `nonReentrant` modifier for some of its state-changing functions. Most of these functions can be argued to function properly without any reentrancy guards because they are only callable by trusted actors behind a timelock. However, since the `setRebalanceMaxSlippagePercent` function is not behind a timelock, it would definitely benefit from having a reentrancy guard.

In addition, the `NonReentrantVault` and `PerformanceVault` contracts do not have the `nonReentrant` modifier for their parent ERC-20 contract's functions. Specifically, the `transfer` and `transferFrom` functions should be guarded. Furthermore, the vaults are also susceptible to read-only reentrancy. Applying the `nonReentrant` modifier to the state-changing functions alone does not protect against read-only reentrancy, which could make third-party contracts vulnerable if they read from a `view` function of these vaults mid-state change.

Consider adding reentrancy guards to applicable functions of the vaults, including the functions from the parent contracts.

Update: *Resolved in pull request #46 at commits 7d4e04c and a811e04. The `nonReentrant` and `nonReentrantView` modifiers were added where possible and the documentation was expanded to clarify the remaining read-only reentrancy vectors. The update also removed the `notPaused` modifier from access-controlled functions which simplifies admin intervention during emergencies. The Size Credit team stated:*

> *Because of how these contracts are inherited from OpenZeppelin's `openzeppelin-contracts-upgradeable` library, practically all ERC-20 and ERC-4626 `view` functions cannot be guarded with a `nonReentrantView` modifier, since they are used internally in state-changing functions which themselves are non-reentrant. If we applied `nonReentrantView` to `public` `view` functions that are used by state-changing `public` functions, they would revert.*

L-06 Incorrect Return Value in `maxRedeem` and `maxMint` Functions

The `SizeMetaVault` and `ERC4626StrategyVault` contracts derive the return values of their `maxRedeem` and `maxMint` functions from the `maxWithdraw` and `maxDeposit` functions, respectively. This results in a precision loss during the conversion of these values. For example, in both contracts, the `maxRedeem` function first invokes `maxWithdraw` which gets the total withdrawable assets from the underlying vault(s). The underlying vault(s) would most likely calculate the withdrawable assets by converting users' total shares to assets. The `maxRedeem` function then converts these assets back to shares. The conversions from shares to assets to shares again all use floor division causing precision loss. This means that the resulting share amount would most of the time be *1 Wei* less than the actual share amount of the user. This could badly affect the user experience and potentially lead to a DoS of the dependent contracts that expect the entire balance to be withdrawable.

Consider documenting this behavior to ensure users do not falsely assume the entire share amount is withdrawable.

Update: *Resolved in pull request #49 at commit 7e2c800.*

Notes & Additional Information

N-01 `reorderStrategies` Function is Unnecessarily Expensive

The `reorderStrategies` function of the `SizeMetaVault` contract removes each existing strategy via `_removeStrategy` and re-adds each new one via `_addStrategy`, wasting gas due to redundant work.

Each `_removeStrategy` operation loops through and shifts existing strategies to preserve order, and then pops the last strategy. Trying to preserve the order of strategies in the process of updating the entire order is unnecessary work. Each `_addStrategy` operation following the `_removeStrategy` operation performs checks on the validity of the strategy, then pushes the strategy to the array. The validation loop within `reorderStrategies` already ensures that `newStrategiesOrder` only contains existing, non-duplicate strategies, making

the `_addStrategy` checks unnecessary. This wastes gas, emits redundant `StrategyAdded` and `StrategyRemoved` events, and enlarges the reversion surface without added safety.

Consider updating the existing strategies directly in a single loop instead of invoking the `internal` `_removeStrategy` and `_addStrategy` functions.

Update: *Resolved in pull request #42 at commit fffe0e9.*

N-02 Avoidable External Call

In the `initialize` function of the `AaveStrategyVault` contract, the AAVE pool is called (external call) twice to get the same `aToken` address. Performing external calls is an expensive operation that should be avoided if possible.

Consider caching the `aToken` in a local variable by calling the AAVE pool once and using this local variable the second time.

Update: *Resolved in pull request #44 at commit 9548b93.*

N-03 Opportunity to Break Loops Early

The looping of the strategies in the `_deposit` and `_withdraw` functions of the `SizeMetaVault` can be safely broken if the `assetsToDeposit` and `assetsToWithdraw` values, respectively, are fully consumed.

Consider breaking the loops in the `_deposit` and `_withdraw` functions if the `assetsToDeposit` and `assetsToWithdraw` values, respectively, are zeroed out.

Update: *Resolved in pull request #45 at commit 553cc8f.*

Conclusion

The audited codebase implements a modular ERC-4626-compliant meta vault system that allocates assets across underlying strategy vaults. The audit identified several medium- and low-severity issues related to precision loss in ERC-4626 functions, pause functionality, and strategy management.

The codebase was well-written, thoroughly tested, and comprehensively documented with clear design considerations and acknowledged limitations. The Size Credit team was very responsive throughout the audit process, answering all questions satisfactorily and providing extensive documentation about the project.