OpenZeppelin | security

# Pods Finance Ethereum Volatility Vault Audit #2


**December 2nd, 2022**

This security assessment was prepared by OpenZeppelin.

# Table of Contents

# Summary

| | |
|---|---|
| Type | DeFi |
| Timeline | From 2022-11-09<br>To 2022-11-18 |
| Languages | Solidity |

| | |
|---|---|
| Total Issues | 23 (16 resolved, 1 partially resolved) |
| Critical Severity Issues | 0 (0 resolved) |
| High Severity Issues | 0 (0 resolved) |
| Medium Severity Issues | 8 (4 resolved, 1 partially resolved) |
| Low Severity Issues | 8 (6 resolved) |
| Notes & Additional Information | 7 (6 resolved) |

# Scope

We audited the pods-finance/yield-contracts repository at the c4b401ce674c24798de5f9d02c82e466ee0a2600 commit.

In scope were the following contracts:

```
contracts/
├── configuration
│   └── ConfigurationManager.sol
├── interfaces
│   ├── IConfigurationManager.sol
│   ├── ICurvePool.sol
│   └── IVault.sol
├── libs
│   └── CastUint.sol
├── mixins
│   └── Capped.sol
├── proxy
│   ├── ETHAdapter.sol
│   └── Migration.sol
└── vaults
    ├── BaseVault.sol
    └── STETHVault.sol
```

# Update from Audit 1

A number of critical and high severity issues were found during the first audit of Pods Finance's contracts. As a result, architectural changes were made to the contracts. This necessitated a second audit in order to fully evaluate the new architectural changes and re-evaluate existing code. The following is a high-level overview of the major changes that were made in response to the results of the first audit:

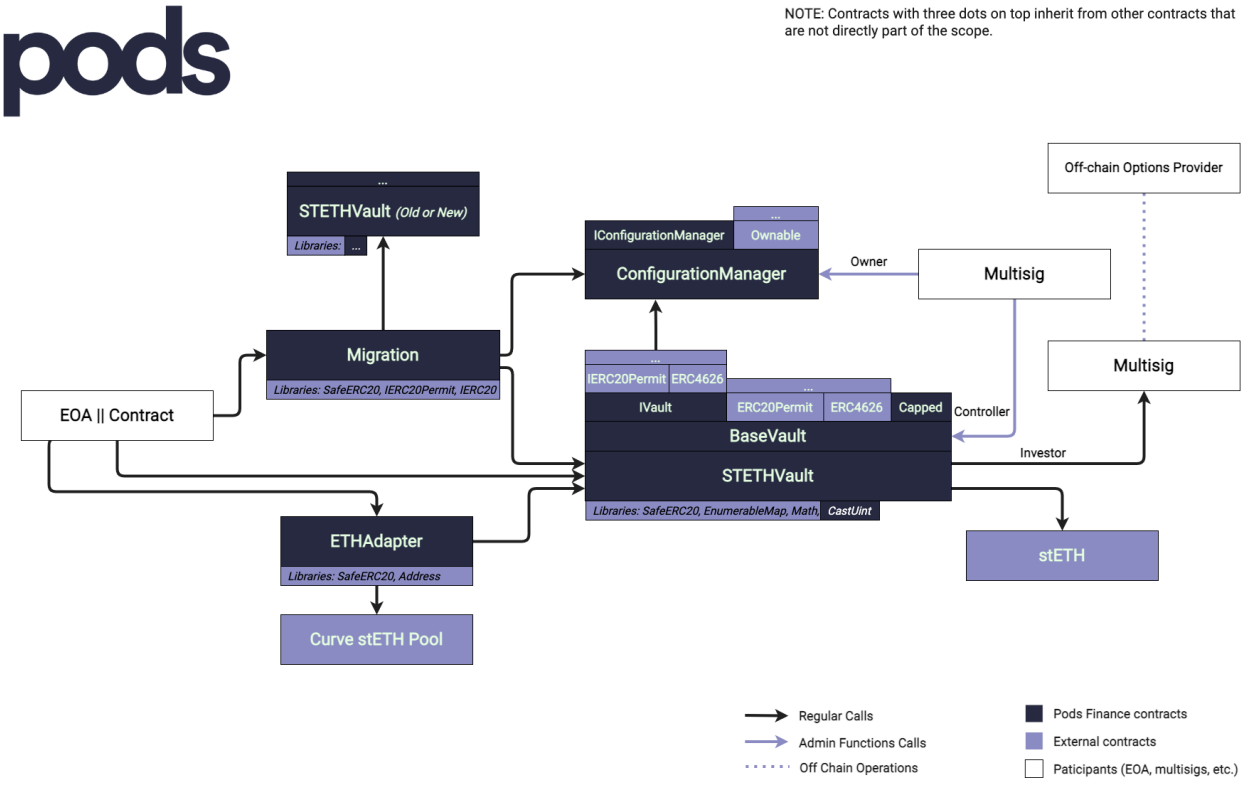
- The custom ERC-4626 implementation was replaced with OpenZeppelin's ERC-4626 implementation.
- The underlying data structure of the `DepositQueue` was changed from an array to OpenZeppelin's EnumerableMap.
- The custom `AuxMath` library was replaced with OpenZeppelin's Math library.

What follows is a system overview of the Pods Finance contracts, followed by notable privileged roles, trust assumptions, and our findings from the second audit.

# System Overview

The Pods Finance staked Ethereum Volatility Vault is a one-click deposit investment product that features an options strategy to generate principal-protected returns when the ETH market is volatile. The underlying technique uses a portion of the yield from stETH assets to invest in the same number of call and put ETH options in an off-chain market. Profits from these investments are then returned to the vault and shared among participants who provided their yield to the investor.

The following diagram gives an overview of the system:




# stETH Volatility Vault

The staked Ethereum Volatility Vault, `stETHvv` for short, is a tokenized vault on the Ethereum mainnet that implements the EIP-4626 standard. It integrates with the Lido `stETH` token as the single underlying asset. To participate in the investment strategy, users deposit the underlying asset into the vault in return for `stETHvv` vault shares.

The investment strategy is executed in rounds. At the start of each round, users can join a deposit queue where they will transfer underlying `stETH` assets to the vault. These assets will contribute yield for the vault; however, `stETHvv` vault shares will not be minted to a user until the queued deposit is processed after the round ends.

Users can withdraw their queued deposits before they are processed, but any `stETH` yield generated by the queued deposit will not be returned. Users that have already had their deposits processed can withdraw their `stETHvv` vault shares for the corresponding `stETH` equivalent, and will have a withdraw fee taken from the `stETH` amount.

When a round ends, only a portion of the `stETH` yield (not the principal) will be transferred to a centralized `investor` account, where the yield is manually used to participate in the off-chain option strategy. In the same transaction, the strategy profit denominated in `stETH` from the previous round will be transferred to the vault from the `investor` account.

After the round ends, deposits to and withdrawals from the vault are temporarily paused while processing of queued deposits is enabled. Anyone can process queued deposits by specifying an array of receiving addresses. Each processed deposit will mint new `stETHvv` vault shares to the receivers, as computed based on the total assets in the vault. Any unprocessed deposit will remain in the queue, which can be either processed or withdrawn in the next round.

When the next round starts, new deposits and withdrawals to the queue may resume, while queued deposits can no longer be processed.

## ConfigurationManager

Parameters such as the controller of the vault, the percentage of fees paid out to the vault, the cap on shares that can be issued, and migration-capable vaults are set in this contract. In the future, this contract may be used by different vaults of the protocol to query globalized parameters. One particularity of this contract is that parameters may be set through a function that allows storing any name-value pair for a specific target on a nested mapping. This allows more values to be added in the future without the need to replace the contract, but adds more complexity to managing contract storage.

## ETHAdapter

In order to allow vault deposits to be made directly with ETH, this contract acts as an intermediary so that users can participate in the protocol without needing to perform the swap themselves. This contract swaps ETH for stETH through a Curve Finance pool. After the exchange, the `ETHAdapter` deposits the resulting tokens to the vault on behalf of the user within the same transaction. The `ETHAdapter` can also be used to withdraw a user's stETH from the vault, exchange stETH for ETH via the Curve Finance pool, and transfer the ETH back to the user in a single transaction.

## Future Migration

The `stETHvv` vault implementation intentionally lacks upgrade functionality, and instead opts to use a migration pattern. In the case of a migration, a `stETHvv` vault shareholder can call `migrate()` on the current vault contract to redeem their shares from this vault and join the deposit queue in the new vault. Users can also leverage the standalone migration contract to join the deposit queue in a new vault. In addition to the `migrate()` function available on the

current vault, the migration contract also has a permit feature that allows users to perform the migration in one transaction.

### Permit

In order to fulfill its value proposition of being a one-click deposit investment product, the `stETHvv` allows users to leverage the permit function on deposits and withdrawals. With this, users can sign a permit off-chain and then use the functions with permit integrations to perform the approval in the same transaction.

## Privileged Roles

Some privileged roles exercise powers over the vault and periphery contracts:

- `Owner` of the `ConfigurationManager`
  - Can add, remove, and modify system parameters such as the cap, the controller, and the vault fees.
  - Can allow any address to be a destination in case of a migration.
- `Controller` of the `stETHvv`
  - After a previous round ends, the `controller` can start a new round anytime.
  - Anyone can start a new round if a week has passed since the last round ended. After a round starts, only the `controller` can end a round.
  - The `controller` receives fees collected on withdrawals.
- `Investor` of the `stETHvv`
  - When a round ends, the investor account collects a portion of the yield from the deposited assets and transfers any profit obtained from the previous round of investment to the vault. The investor account holds the assets used for the off-chain option investment strategy.

# Trust Assumptions

There are a few integrations within the system that contain implicit assumptions, which may hinder protocol functionality if broken:

- Several vault calculations are performed using balances of a rebasing token.
  - In many of these places, an implicit assumption is made that the token will rebase positively. While Lido suggests they have yet to experience a negative rebase, this could occur if they were to experience an event, such as slashing, that lowered their stake.
- `ETHAdapter` gathers the conversion rate for stETH/ETH from a curve pool and provides slippage as input.
  - The curve pool must maintain a certain level of liquidity for the `ETHAdapter` to work appropriately. Additionally, there is an assumption that slippage protection will be appropriately assessed before it is passed in as a parameter to the functions that exchange stETH and ETH. Any transaction constructed without appropriate slippage protection will be susceptible to MEV.
- `ETHAdapter` makes external calls on a user-provided vault parameter.
  - Since the vault parameter is passed in as an argument, there is an implicit assumption that this input is trusted. Users should be aware of this potential phishing avenue, as transactions constructed containing a malicious vault parameter could lead to lost funds.
- Permit functions consume signatures based on variable parameters.
  - There is an assumption that between the time an off-chain signature is gathered and when it is used, the conversion rate for vault shares and `stETH` has not changed. In the event that the conversion has updated, an off-chain signature may need to be regathered from the user for it to be consumed.

# Findings

Here we present our findings.

# Medium Severity

## M-01 Funds held in `ETHAdapter` can be drained by anyone

The `ETHAdapter` contract is used as a proxy to allow users to interact with the vault through sending and receiving ETH instead of stETH. In the course of a normal withdrawal or redemption transaction, the `ETHAdapter` will pull the funds out of the vault before passing them onto the designated receiver. During the moment the `ETHAdapter` is holding the funds, it first converts all of its stETH to ETH, and then sends its entire ETH balance to the receiving address.

Consequently, the `ETHAdapter` sends its full balance to the receiver each time, meaning any ETH or stETH that is mistakenly sent to it can be drained by any user who performs a withdrawal or redemption on the `ETHAdapter`. This is exacerbated by the fact that the vault is passed in as a parameter, potentially allowing a user to perform withdrawals and redemptions without interacting with the actual stETH vault.

Consider transferring the exchanged balance from the Curve pool to the receiver instead of the entire balance of the `ETHAdapter`. Also consider implementing a rescue or sweep function to allow the recovery of funds that are accidentally sent to the `ETHAdapter`.

*Update: Acknowledged, not resolved. Pods Finance team stated:*

> *For now, we do not want to take action in case of funds sent by mistake to our contract. We will prioritize this issue in a future version.*

## M-02 Incorrect `assetsOf` calculation

The `assetsOf` function within `BaseVault` is used to calculate the combination of a user's idle and withdrawable assets. This should depict the number of underlying assets a user could potentially receive from the vault at any given time. The formula it uses, however, incorrectly sums three separate items:

- The number of assets the user's entire share balance would currently convert to
- The number of idle assets a user has in the queue

- The number of assets the user's entire share balance would convert to if the conversion rate factored in all idle assets

During testing, it was observed that the sum of these three items were at least double the amount of assets a user could actually receive from the protocol.

Consider changing the `assetsOf` calculation to match the behavior if a user were to perform both a `redeem` of their total balance of shares and a `refund` of all of their idle assets in the same transaction.

**Update:** *Resolved in PR#98, with commit `602122efed209eed23f14b5eb906be1fab01cec5` being the last one added.*

## M-03 Incorrect recalculation of shares during deposit

When a user deposits to the `STETHVault`, the amount of shares they are expected to receive is recalculated after stETH is transferred from the depositor to the vault. Since the vault's idle assets have not yet been updated, the transferred stETH is incorrectly included in the conversion calculation. This may cause the `spentCap` to be incorrectly updated and the `Deposit` event to emit a smaller value of shares than what will actually be issued when the deposit is processed.

Consider performing the `previewDeposit` and `_spendCap` calls after the `totalIdleAssets` have already been updated. This ensures any stETH that is transferred during the deposit will be correctly accounted for. Additionally, consider swapping the order of the `totalIdleAssets` decrement and `_restoreCap` logic in the `refund` function in `BaseVault` to accommodate for the changes in `_deposit`.

**Update:** *Resolved in PR#118, with commit `db0c86daa2d52d1f92e59ae852cade40be351b82` being the last one added.*

## M-04 Negative rebase of stETH could prevent a round from ending

When a round ends, the amount of underlying assets currently in the vault is subtracted from the amount of assets the vault contained in the previous round. This calculation assumes a positive yield, but the underlying asset stETH is able to rebase in both a positive and negative direction due to the potential for slashing. In the case where Lido is slashed, `totalAssets`

can be less than `lastRoundAssets`. Consequently, the subtraction would cause an underflow, which would prevent the controller from being able to end the round until `totalAssets` is greater than `lastRoundAssets`.

Consider placing the logic that calculates and transfers any investment yield generated by the vault in an `if` statement that only occurs when `totalAssets` is greater than `lastRoundAssets`. This allows any yield generated from the options strategy to still be transferred to the vault and prevents the accrued interest arithmetic from reverting.

**Update:** *Resolved in PR#100, with commit* `a756b4d8ead7fce109e9daf51c218eb952454487` *being the last one added.*

## M-05 Non-standard ERC-4626 vault functionality

There are multiple locations in the ERC-4626 `BaseVault` that do not conform to ERC-4626 specifications:

- `previewWithdraw` does not include withdrawal fees
- `maxDeposit` does not return 0 when deposits are disabled
- `maxMint` does not return 0 when withdrawals are disabled
- `maxWithdraw` does not return 0 when withdrawals are disabled

Consider correcting the above issues to meet the ERC-4626 specifications, allowing future vault developers to expect certain protocol behaviors.

**Update:** *Partially resolved in PR#132, with commit* `87e7de33e0a7a699263624641305a8e06ec178b2` *being the last one added. The issues regarding* `maxDeposit`, `maxMint`, `maxWithdraw` *and* `maxRedeem` *have been resolved and these functions now return 0 when deposits or withdrawals are disabled. The* `previewWithdraw` *function does not include withdrawal fees. However, we note that there is ambiguity in how fees on withdrawal should be implemented according to EIP-4626, and that* `previewWithdraw` *does return the correct number of shares that would be burned in a* `withdraw` *call. Docstrings have been added to* `previewWithdraw` *and* `withdraw` *to inform integrators of the presence of withdrawal fees.*

## M-06 Phantom permit functions callable from child vault

In the previous audit, we described an issue where the `BaseVault` implements a `mintWithPermit` and `depositWithPermit` function, even though the vault's underlying asset is intended to be stETH, which does not have a `permit` function. The Pods Finance team acknowledged this issue and explained that they kept this functionality because they may create other vaults in the future with underlying assets that do contain `permit` functions.

In this case, consider overriding the functions that call `permit` in the `STETHVault` to explicitly revert when called. In the `STETHVault` contract, the calls to `mintWithPermit` and `depositWithPermit` will currently have their transaction reverted due to the stETH contract's fallback function. However, this both provides a potentially inaccurate error message and is subject to different behavior if the stETH contract undergoes an upgrade. Further, tokens containing phantom permit functions are known to exist, which would not revert on failure. Setting a precedent of explicitly reverting when calling unsupported `permit` functions can help ensure future vaults do not suffer from calling phantom permits on their underlying assets.

**Update:** *Resolved in PR#123, with commit* `fd3ca22cedc414f9e6f5fbed34ad725ca509bff2` *being the last one added.*

## M-07 Refunds will be over-credited in a negative yield event

Deposits added to the queue are point-in-time stETH balance amounts. The stETH token rebases to account for yield, and in the event of slashing, may be subject to a negative yield. In the event that a stETH token rebase is negative between the time a user deposits and calls for a refund, the vault will over credit the user by the rebase difference.

Consider handling the deposits in the queue in stETH share amounts to account for rebase changes on refunds.

*Update: Acknowledged, not resolved. Pods Finance team stated:*

> *Although we agree with the issue, we won't prioritize it right now. It would require us to implement a secondary share queue system that would require few parts of the code to change. We will prioritize this issue in a future version.*

## M-08 Vault can be placed back into vulnerable low supply state

The `BaseVault` contract inherits an issue common to ERC-4626 implementations known as the donation attack. When shares are minted to users, they are calculated by multiplying the deposited assets by a ratio of the existing shares to current assets. The result of this calculation is then rounded down, following the ERC specifications. The problem arises when the value of `totalAssets` is manipulated to induce unfavorable rounding for new users depositing into the vault at the benefit of users who already own shares of the vault. Since `totalAssets` is calculated using the balance of underlying assets, it can be manipulated via direct transfers or donations of stETH to the vault. The most extreme example of this attack could occur when a user is the first to enter the vault, because when supply is sufficiently low, the capital requirement to perform an economically beneficial donation attack is also low.

The Pods Finance team is aware of this issue and has taken measures to make the attack more difficult to execute, such as by creating a minimum initial deposit requirement. While this does successfully enforce a minimum initial deposit for the first round, it does not prevent the total supply of shares from falling below the safety threshold throughout the life of the vault. For example, if an early user put forth the minimum initial deposit and then the rest of the queue held dust amounts, the same user could withdraw far below the minimum initial deposit amount in the next round and the vault would be back in a vulnerable state.

Consider taking steps to ensure the supply of vault shares does not go below the minimum initial deposit amount. One way to do this would be for the Pods Finance team to contribute their own initial deposit to the vault that they can guarantee will not be withdrawn. Discussions around other techniques used to protect from this attack can be seen in issue 3706 of the openzeppelin-contracts repository.

**Update:** *Acknowledged, will resolve. Pods Finance team stated:*

> *Pods Finance will follow the recommendations and will contribute their own initial deposit to the vault that they can guarantee will not be withdrawn.*

# Low Severity

## L-01 Magic numbers are used

Although constants are generally used correctly throughout the codebase, there are a few occurrences of literal values being used with unexplained meaning inside of `ETHAdapter`. For example, the following blocks use hardcoded values:

- In the constructor, the pool's `coins` function is called with arguments `0` and `1`.
- In the functions `convertToETH` and `convertToSTETH`, the pool's `get_dy` function is called with arguments `0` and `1` with no explanation.
- In the `deposit` function, the pool's `exchange` function is called with arguments `0` and `1` without explanation.

To improve the code's readability and facilitate refactoring, consider defining a constant for every magic number, giving it a clear and self-explanatory name. Consider adding an inline comment explaining how the magic numbers are calculated or why they are chosen for complex values.

**Update:** *Resolved in PR#103, with commit* `87e7de33e0a7a699263624641305a8e06ec178b2` *being the last one added.*

## L-02 Misplaced `totalAssets` function implementation

As part of the `STETHVault` contract implementation, the `totalAssets` function is overridden and updated in order to account for the assets that have been transferred to the vault contract but have not been processed yet. However, the accounting for idle assets within a vault is primarily implemented in the `BaseVault` contract. Consequently, if a vault were to inherit the `BaseVault` contract and not override `totalAssets`, any calculation involving `totalAssets` would most likely be off due to the default implementation of `totalAssets` not taking into account idle assets.

Consider moving the implementation of `totalAssets` to the `BaseVault` contract. `totalAssets` can additionally be marked `virtual` so contracts that inherit `BaseVault` can optionally choose to implement their own functionality.

**Update:** *Resolved in PR#117, with commit* `2a82f1126c14c713b19a08cdf32690e52bb4c43d` *being the last one added.*

## L-03 Missing docstrings

Throughout the codebase, there are several parts that do not have docstrings. For instance:

- Line 5 in `IConfigurationManager`
- Lines 11-28 in `IConfigurationManager`
- Line 5 in `ICurvePool`
- Lines 12-19 in `ICurvePool`
- Line 8 in `IVault`
- Line 5 in `CastUint`
- Line 7 in `Capped`
- Line 11 in `ETHAdapter`
- Line 28 in `ETHAdapter`
- Line 32 in `ETHAdapter`
- Line 40 in `ETHAdapter`
- Line 52 in `ETHAdapter`
- Line 66 in `ETHAdapter`
- Line 77 in `ETHAdapter`
- Line 91 in `ETHAdapter`
- Line 11 in `Migration`
- Line 26 in `Migration`
- Line 47 in `Migration`

This hinders reviewers' understanding of the code's intention, which is fundamental to correctly assess not only security but also correctness. Additionally, docstrings improve readability and maintenance.

Consider thoroughly inserting documentation above each contract, interface, library, function, state variable, event, and custom error. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

**Update:** *Resolved in PR#122, with commit* `32df444b20be010b5f1da7f42a69fcbeec61fdda` *being the last one added.*

## L-04 Missing `ConfigurationManager` check during vault initialization

As part of the `constructor` in the `BaseVault` contract, the `configuration` state variable is set. However, there is no check to ensure the passed-in `ConfigurationManager` address instance has the `VAULT_CONTROLLER` parameter set for the vault contract that is being created. Having the `VAULT_CONTROLLER` parameter be the default value of the zero address for the new vault could result in a loss of the `fee` taken as part of the withdraw control flow, since that fee will be transferred to the `VAULT_CONTROLLER` address designated by the `ConfigurationManager`.

Consider adding a check in the `constructor` of the `BaseVault` contract to ensure calling `getParameter` with the address of the vault being created, and `VAULT_CONTROLLER` does not return the zero address.

*Update: Acknowledged, not resolved. Pods Finance team stated:*

> *Instead of enforcing it at the code level, that would require us to deploy the vault using CREATE2 in order to know in advance the vault address, we will monitor the VAULT_CONTROLLER variable to make sure that it was set after the deploy.*

## L-05 Share price calculations should never result in a zero value

The share price represents the amount of stETH needed to mint one vault share. When the number of vault shares that has been minted thus far is zero, the share price should simply be the ratio of the underlying asset's decimals to the vault's decimals. There were multiple share price calculations in the `STETHVault` contract that incorrectly returned zero in this scenario:

- the `lastSharePrice` instantiation in the `_afterRoundStart` function
- the `currentSharePrice` calculation in the `_afterRoundStart` function
- the lack of `endSharePrice` calculation when `supply` is zero in the `_afterRoundEnd` function
- the `startSharePrice` calculation in the `_afterRoundEnd` function

Consider using the `convertToAssets` function that is inherited from OpenZeppelin's `ERC-4626` contract instead of the manual calculations, as it contains logic to handle this edge case. Additionally, when instantiating `lastSharePrice` in the `_afterRoundStart`

function, consider setting the numerator to `10 ** underlying asset decimals` and the denominator to `10 ** sharePriceDecimals` when the number of vault shares is zero.

**Update:** *Resolved in PR#109, with commit `88e0b18fa497ce86194ab5ac8b9cc346ecfb3b75` being the last one added.*

## L-06 The `sharePrice` function should not revert

The `sharePrice` function in the `STETHVault` contract calculates the amount of stETH needed in order to mint one vault share. However, the calculation does not take into account that `totalSupply` can return zero when no deposits have been made to the vault and therefore no vault shares have been minted. Consequently, when the `totalSupply` function returns zero the `mulDiv` operation will revert.

Consider calling `convertToAssets(10**sharePriceDecimals)` in the `sharePrice` function instead of using the manual calculation, as `convertToAssets` handles the zero shares edge case.

**Update:** *Resolved in PR#104, with commit `9264f9dc5a5676da381dc6e86768971ffd5415d8` being the last one added.*

## L-07 `spentCap` can be skewed due to rebasing tokens

`spentCap` is a local state variable in the `Capped` contract that is used to indicate how many shares have been minted. This variable is first updated when a user deposits. However, shares are not issued for a user until their deposit has been processed. The share conversion rate is subject to change during that time-frame due to stETH being a rebasing token. Therefore, when `_restoreCap` is called during a withdraw, the number of shares accounted for in `spentCap` may differ from the number of shares that were actually issued for the same deposit. This skew can accumulate over time, to the point where a vault could mistakenly restrict vault deposits due to the `spentCap` incorrectly reaching the available cap limit.

Consider adding `spentCap` to the `ConfigurationManager` as a parameter, and updating calculations in `Capped` to add and subtract to that value instead of using a local state variable. This would allow the owner of the `ConfigurationManager` contract to adjust `spentCap` if the skew becomes too large.

*Update: Acknowledged, not resolved. Pods Finance team stated:*

> *The cap is a temporary feature. As of now, it is possible to workaround this issue and increase the cap through the ConfigurationManager if needed.*

## L-08 Use of deprecated function for setting token allowances

Throughout the codebase there were multiple locations where the deprecated `safeApprove` function was used to set allowances:

- Line 352 in `BaseVault`
- Line 35 in `Migration`
- Line 57 in `Migration`
- Line 43 in `ETHAdapter`
- Line 112 in `ETHAdapter`

If in any of these locations an allowance is carried over for more than the duration of a transaction, `safeApprove` may revert in subsequent transactions. Consider replacing the instances of `safeApprove` with the recommended `safeIncreaseAllowance` or `safeDecreaseAllowance` instead.

**Update:** *Resolved in PR#105, with commit* `7e6bdbcb3d1fff657c81810750521fa0278e4a07` *being the last one added.*

# Notes & Additional Information

## N-01 Inconsistent use of named return variables

Named return variables are a way to declare variables that are meant to be used inside a function body and returned at the end of the function. This is an alternative to the explicit `return` statement to provide function outputs.

Throughout the codebase, there are instances of inconsistent use of named return variables:

- The `deposit` function in `ETHAdapter` has a named return variable declared that is not used.
- The `startRound` function in `IVault` declares a named return variable, while its implementation in `BaseVault` does not.
- The `handleMigration` function in `IVault` declares a named return variable, while its implementation in `BaseVault` does not.

Consider improving consistency and either using or removing any unused named return variables.

**Update:** *Resolved in PR#110, with commit* `f2459432bbdc423c15c501cd277d72ae4fdefd70` *being the last one added.*

## N-02 Inflexible initial deposit

The current minimum initial deposit is defined as `10**decimals` of the underlying token. This strategy may be inflexible for certain edge-case tokens. For example, a vault that has an underlying asset of wBTC would require an initial deposit of $16,600 at today's market price. Additionally, if a vault were to have an underlying asset with very few decimals, it is possible that the capital requirement intended to prevent a donation attack may be insufficient.

If the Pods Finance team ever intends to offer vaults with wBTC or other edge case tokens as underlying assets, consider allowing the minimum initial deposit to be configurable at deployment.

*Update: Acknowledged, not resolved. Pods Finance team stated:*

> *As stated above, Pods Finance will contribute their own initial deposit to the vault that they can guarantee will not be withdrawn. Depositing one whole wBTC won't be a problem. For the small decimals tokens, Pods Finance team will deposit an initial deposit bigger than the minimum.*

## N-03 Lack of indexed parameters

The `VaultAllowanceSet` event in `IConigurationManager` does not have the `newVault` parameter indexed.

Consider indexing event parameters to avoid hindering the task of off-chain services searching and filtering for specific events.

**Update:** *Resolved in PR#105, with commit* `7e6bdbcb3d1fff657c81810750521fa0278e4a07` *being the last one added.*

## N-04 Lack of zero address checks

Throughout the codebase, there are several functions where the zero address could be passed as an argument but is not explicitly checked:

- `migrate` function in `BaseVault.sol`
- `migrate` function in `Migration.sol`
- `migrateWithPermit` function in `Migration.sol`
- `setVaultMigration` function in `ConfigurationManager.sol`

Zero address checks prevent operations from potentially occurring on the zero address, and allows developers to return clear error messages instead of having a function fail in a less-obvious way.

Consider adding zero address checks to the above functions.

**Update:** *Resolved in PR#124 and PR#129, with commits* `bb7be233253bb96589486cc700a9889f716e34a1` *and* `771aa5bf7453405d91310035be65eab698ccfdb1` *being the last ones added.*

## N-05 Mismatched event parameters

Multiple event definitions contain `uint256` parameters for `roundId`, but the value emitted is a `uint32`:

- `RoundStarted` in `IVault` declares a `uint256` parameter for `roundId` but emits the `uint32` `vaultState.currentRoundId`.
- `RoundEnded` in `IVault` declares a `uint256` parameter for `roundId` but emits the `uint32` `vaultState.currentRoundId`.
- `DepositProcessed` in `IVault` declares a `uint256` parameter for `roundId` but emits the `uint32` `vaultState.currentRoundId`.
- `DepositRefunded` in `IVault` declares a `uint256` parameter for `roundId` but emits the `uint32` `vaultState.currentRoundId`.

For clearer code and potential gas savings, consider aligning variable types within event definitions and event emissions.

**Update:** *Resolved in PR#113, with commit `135c912b843b64937f9eb4069e1c0fc8b312cd80` being the last one added.*

## N-06 Non-explicit imports are used

Non-explicit imports are used inside the codebase, which reduces code readability and could lead to conflicts between the names defined locally and the ones imported. This is especially important if many contracts are defined within the same Solidity files or the inheritance chains are long.

Within `ConfigurationManager`, global imports are being used. For instance:

- Line 5 in `ConfigurationManager`
- Line 6 in `ConfigurationManager`

Following the principle that clearer code is better code, consider using named import syntax (`import {A, B, C} from "X"`) to explicitly declare which contracts are being imported.

**Update:** *Resolved in PR#102, with commit `b059e07c90ae62b4a37ffd596ba05bcd997a16b8` being the last one added.*

## N-07 Use of `uint32` for round timestamp limits vault lifetime

The `BaseVault` keeps track of when rounds end by placing the current `block.timestamp` inside of a `uint32` value. This value is then used to determine whether a round can begin. When `block.timestamp` produces a number near the max `uint32` value, the vault will lose the ability to end and start rounds due to integer overflow. Currently this may occur within approximately 83 years.

Consider expanding the round timestamp to a `uint40` or larger to drastically increase the shelf life of the `BaseVault`.

**Update:** *Resolved in PR#107, with commit* `d26b0ee274f4cec28fad2a9f0df3e340ee8fc887` *being the last one added.*

# Conclusions

No critical or high severity issues were found in this second audit. Several medium severity issues were discovered, many of which stemmed from complications related to directly integrating with the rebasable stETH token. Overall, the codebase shows a marked improvement from the first audit. The Pods Finance team was able to leverage pre-existing OpenZeppelin contract implementations and other libraries as part of their fixes for the first audit, which helped reduce the attack surface of their contracts. Additional recommendations have been proposed to further reduce the attack surface of the contracts.

# Appendix

## Monitoring Recommendation

While audits help in identifying code-level issues in the current implementation and potentially the code deployed to production, we encourage the Pods Finance team to consider incorporating monitoring activities in the production environment. Ongoing monitoring of deployed contracts helps in identifying potential threats and issues that may affect the protocol. With the goal of providing a complete security assessment, we want to raise several actions addressing trust assumptions and out-of-scope components that may benefit from on-chain monitoring:

- Minimum balance invariant of the `stETHvv` contract: consider monitoring the underlying balance of stETH in the vault contract to ensure it is always greater than or equal to the `MIN_INITIAL_ASSETS` balance. This can be done on a per-block basis by checking the stETH balance of the vault at a previous block number with the stETH balance of the vault at the most recent block number.
- Low liquidity in the ETH/stETH Curve pool: consider monitoring the liquidity in the ETH/stETH Curve pool to ensure that the `ETHAdapter` contract does not encounter a situation where exchanges are failing or large slippage is occurring due to low liquidity in the corresponding Curve pool.
- Negative rebase event of stETH due to the potential for Lido to be slashed: consider monitoring the balance of stETH in the `stETHvv` contract along with `Withdraw`, `EndRoundData`, and `DepositRefunded` events. The respective events indicate when stETH is transferred from the vault. If the balance of the contract were to decrease not as a direct result of any of the aforementioned events, it could be an indication of a negative stETH rebase event.
- Privileged role activities such as those performed by the Owner of the `ConfigurationManager` contract, the controller of the vault, and the investor of the vault: consider monitoring events related, but not limited to, ownership changes, vault configuration modifications, and transfers initiated by the investor address to somewhere other than the vault as these may signal private key compromise of a privileged role.

Additionally, the team may wish to explore options for monitoring Lido node status so that they may be able to predict if a negative stETH rebase will occur in the near future.