OpenZeppelin | security

# Mellow Simple DVT Strategy Audit

October 1, 2024

# Table of Contents

# Summary

| | |
|---|---|
| Type | DeFi |
| Timeline | From 2024-08-12<br>To 2024-09-06 |
| Languages | Solidity |

| | |
|---|---|
| Total Issues | 23 (0 resolved) |
| Critical Severity Issues | 0 (0 resolved) |
| High Severity Issues | 0 (0 resolved) |
| Medium Severity Issues | 0 (0 resolved) |
| Low Severity Issues | 8 (0 resolved) |
| Notes & Additional Information | 15 (0 resolved) |

# Scope

We audited the mellow-finance/mellow-lrt repository at commit 1c885ad.

In scope were the following files:

```
src
├── Vault.sol
├── VaultConfigurator.sol
├── modules
│   ├── DefaultModule.sol
│   ├── erc20
│   │   └──ERC20TvlModule.sol
│   ├── obol
│       └── StakingModule.sol
├── oracles
│   ├── ChainlinkOracle.sol
│   ├── ConstantAggregatorV3.sol
│   ├── ManagedRatiosOracle.sol
│   └── WStethRatiosAggregatorV3.sol
├── security
│   ├── AdminProxy.sol
│   ├── DefaultProxyImplementation.sol
│   └── Initializer.sol
├── strategies
│   └── SimpleDVTStakingStrategy.sol
├── utils
│   └──DefaultAccessControl.sol
└── validators
    └── ManagedValidator.sol
```

# System Overview

The Mellow protocol consists of a set of modular smart contracts, including an ERC-20-compliant Vault and peripheral contracts such as strategies for managing the assets deposited in the Vault. This system allows for the creation of custom Vaults with tailored strategies to manage the Vault's assets and generate yield for depositors. The audit focused on the Mellow DVstETH Vault, a collaborative effort among Mellow, Lido, Obol Network, and SSV Network. DVstETH is a Liquid Staking Token that wraps a Mellow Vault to facilitate ETH staking directly to Lido's Simple DVT (Distributed Validator Technology) staking module. The Simple DVT staking module employs node operators using distributed validator technologies from Obol Network and SSV Network. This Vault is upgradeable and uses a Transparent Proxy.

To participate in the system and acquire DVstETH, users deposit WETH into the Vault using the `deposit` function. The only token accepted for deposits is WETH, and the only token allowed for withdrawals is wstETH. In return, users receive a minted amount of LP shares (DVstETH), representing their stake in the Vault. The Vault securely holds the assets and uses the `call` and `delegateCall` functions to manage the distribution of these assets. Strategy contracts act as operators, invoking `delegateCall` to execute specific logic, while module contracts define the logic for how and where the funds are allocated. In the DVstETH Vault, the `SimpleDVTStrategy` contract is responsible for interacting with the Vault, targeting specific functions in the `StakingModule` contract which handles the logic for direct deposits to Lido.

The primary objective of the DVstETH Vault is to facilitate direct deposits into the Simple DVT module. To achieve this, Lido modified its Depositor Bot which is responsible for managing calls to the Lido Deposit Security Module and providing guardian signatures. If certain conditions are met, the Depositor Bot triggers the Vault's deposit to Lido by calling the `convertAndDeposit` function of the `SimpleDVTStrategy`, which then uses `delegateCall` on the Vault to invoke `convertAndDeposit` in the `StakingModule` contract. The `StakingModule` contract logic ensures that deposits to the Simple DVT module are feasible and are, for example, not being used to cover Lido withdrawals, reverting the transaction otherwise. During the deposit process, the Vault receives stETH which is automatically converted to wstETH. If deposits to the staking module are not possible due to reasons such as the Lido withdrawal queue not being covered or the Simple DVT module not having available validators to deposit into, the curator admin is responsible for converting any WETH that remains in the Vault for more than seven days into wstETH. One important thing to

note is that to prevent inflation attacks, the initial allocation must be made to the Vault and handled by an operator.

Users who deposit into the DVstETH Vault will earn rewards from the appreciation of wstETH, along with three additional rewards: Mellow Points, SSV Network Points, and Obol Network Points. The distribution of these points has not yet been implemented by the protocol, but the first distribution is expected to occur in the near future. To withdraw funds, users must first submit a request to the Vault using `registerWithdrawal()`, specifying, among other details, the amount of LP shares they wish to withdraw. At the time of the request, the LP shares are transferred to the Vault and the transaction sender is added to the list of pending withdrawers. It is important to note that a user can only have one active withdrawal request at a time. Therefore, if they want to modify their request, they must cancel the previous one and submit a new request with the updated amounts.

The withdrawal process is managed by the `CuratorOperator`, who, under normal circumstances, processes withdrawals once per day. The `CuratorOperator` calls `processWithdrawals()` in `SimpleDVTStakingStrategy`, specifying an array of users to be processed and an `uint256 amountForStake` parameter. This parameter is used if the wstETH available in the Vault is insufficient to cover the withdrawals, which can occur if the WETH has not yet been staked in Lido. The necessary amount to cover the withdrawals, as specified by `amountForStake`, will be automatically converted to wstETH using `StakingModule.convert()` without going through the `convertAndDeposit()` logic.

Once the process is completed, each user will receive the corresponding amount of wstETH based on the LP amount specified in their withdrawal request minus a fee and their LP shares will be burned. The fee is currently 0, but if it becomes non-zero, the fee will be distributed among the Vault's depositors. There is an emergency withdrawal mechanism to prevent funds from being stuck in the contract if the operator fails to process user withdrawals. The `emergencyWithdraw()` method can be called by a user if their request does not get processed even after the time period (90 days in the current deployment) set in the `VaultConfigurator` contract has passed.

The `VaultConfigurator` contract is created alongside the Vault and controls many Vault parameters. These include the Vault's withdrawal fees, maximum LP supply, transfer lock status, and the addresses for the ratio oracle, price oracle, validator, and deposit or withdrawal callback addresses.

# Security Model and Trust Assumptions

During the audit, the following trust assumptions were made about the project:

- Withdrawals will be monitored and processed properly.
- The actors described in the following section (Privileged Roles) are trusted and behave correctly according to their functions.
- 1 stETH is equivalent to 1 ETH in value.
- The wstETH function `getStETHByWstETH` returns an accurate stETH/wstETH ratio.
- The Lido system will not be compromised and admin actors will behave correctly.
- The Lido Depositor Bot will function correctly.
- The SimpleDVT staking module will increase its target share in Lido and will continuously add available validators for deposits.
- The TVL modules will not have any overlap. For example, there will not be two different TVL modules accounting for the same tokens in the Vault.
- The funds allocated to the Vault in the first deposit to prevent inflation attacks remain in the Vault.
- The ratios within the Vault are correctly set and will be updated if the underlying tokens change.

## Privileged Roles

The system has the following different actors:

- The `VaultAdmin`, a Lido-and-Mellow-owned 5-of-8 Gnosis Safe multisig.
- The `ProxyVaultAdmin`, a Lido-and-Mellow-owned 5-of-8 Gnosis Safe multisig.
- The `CuratorAdmin`, a 3-of-6 Gnosis Safe multisig operated by Steakhouse.
- The `CuratorOperator`, an EOA operated by Steakhouse.
- The Lido Depositor Bot.

The addresses of the above-given actors can be found in the mellow protocol documentation.

There are two types of role mechanics in the system:

- `DefaultAccessControl` role mechanics
- `ManagedValidator` role mechanics

### `DefaultAccessControl`

`DefaultAccessControl` has 3 roles: `ADMIN`, `ADMIN_DELEGATE`, and `OPERATOR`, and is used in three contracts, `Vault`, `VaultConfigurator`, and `SimpleDVTStakingStrategy`. `Vault` and `SimpleDVTStakingStrategy` have independent sets of roles, `VaultConfigurator` inherits its role model from `Vault`. `ADMIN` and `ADMIN_DELEGATE` both have admin power, that is, to assign `OPERATOR` roles and access admin methods (specific to a contract). The only additional power of `ADMIN` over `ADMIN_DELEGATE` is to assign `ADMIN` and `ADMIN_DELEGATE` roles.

The following contract functions are available to the `OPERATOR` role:

- Within the `Vault` contract:
  - `externalCall` (subject to `ManagedValidator` role mechanics)
  - `delegateCall` (subject to `ManagedValidator` role mechanics)
  - `processWithdrawals`
- Within the `VaultConfigurator` contract:
  - `stageDepositsLock`
  - `commitDepositsLock`
  - `rollbackDepositsLock`
  - `revokeDepositsLock`
- Within the `SimpleDVTStakingStrategy` contract:
  - `processWithdrawals`

The `ADMIN` and `ADMIN_DELEGATE` roles have access to all of the above functions along with the following:

- Within the `Vault` contract:
  - `addToken`
  - `removeToken`
  - `addTvlModule`
  - `removeTvlModule`
- Within the `SimpleDVTStakingStrategy` contract:
  - `setMaxAllowedRemainder`
- Within the `VaultConfigurator` contract:
  - All the non-`view` functions (not listed here for brevity). These functions allow staging, committing, and rolling back various system parameters. These parameters include the addresses for callbacks on deposit and withdrawal, the

maximum total supply of LP tokens, the addresses for the ratio and price oracles, and the withdrawal fee for the Vault. It should be noted that the Admin of the Vault Configurator has near-total control over the Vault and a malicious admin may use the Vault Configurator to lock the Vault or steal funds.

The `ADMIN` role of the `Vault` has been assigned to the `VaultAdmin` multisig and the `OPERATOR` role to both the `SimpleDVTStakingStrategy` contract and the `CuratorAdmin` multisig. The `SimpleDVTStakingStrategy` `ADMIN` role has been assigned to the `VaultAdmin` and the `OPERATOR` role to the `CuratorOperator`.

#### Managed Validator

The Managed Validator adds an additional layer of role-based access control to methods used for actions on behalf of the Vault, specifically `delegateCall` and `externalCall`. The Managed validator has a custom set of roles each represented by a `uint8` number. Roles are assigned to EOAs and contracts. Each role has a set of permissions which specify the set of methods the role can call in a specific contract. For example, `delegateCall` and `externalCall` have two additional checks beyond requiring the sender to have the `OPERATOR` role. Both functions check that the sender of the transaction is authorized to call the target function selector and they perform some validations on the `data` that will be used to execute the EVM `call` or `delegatecall`. There is a special `ADMIN_ROLE` assigned to the `VaultAdmin` that has permission to do anything within the `ManagedValidator`, including granting and revoking roles.

#### Vault Configurator

Currently, calling `delegateCall` in the Vault is only allowed when the target address is the `StakingModule`. This is controlled by the `VaultConfigurator` contract function `isDelegateModuleApproved`. As stated above, the `ADMIN` role has the ability to call functions within `VaultConfigurator` which can add or remove addresses from the list of approved delegate modules, allowing them to be `delegatecall`ed from the Vault.

## Operational Concerns

The protocol's extensibility brings added risks once it is deployed and operational. Many of these risks can be mitigated by being prepared for them. Some of such preparatory steps have been listed below.

- Perform all admin actions in the correct order and with knowledge of the system. For example, such TVL modules will not be added that overlap with other TVL modules, and

`delegateCall` permissions will only be given to contracts which require them. For sensitive actions that require elevated permissions:

- Monitor for all calls to `delegateCall` and `externalCall` within the Vault as these could potentially expose the system to very dangerous behavior.
- Simulate all changes to system parameters ahead of time, validating all state changes and event emissions. In the simulation, ensure that the withdrawals and deposits still function as desired.
- Monitor for any calls which require admin privileges to ensure that none are happening without the community's knowledge. This can be done by scanning for event emissions in most cases.
- Simulate updates and check for storage layout compatibility by listing the entire storage layout for any upgraded contracts both before and after the upgrade.
- Monitor for suspicious transactions from any multisig participant addresses. These include transactions which do not involve the multisig and transactions which interact with other protocols.
- Occasionally conduct drills where multisig key holders demonstrate their ability to sign messages with their keys as fast as possible. These simulations should have a clear chain of command and the message to sign should not be known ahead of time by any of the signers. The intent of these drills is to understand any pain points in the event of an emergency where a multisig must quickly sign some transaction.

- Monitor the protocol health and the health of integrations while the system is running.
  - Monitor all deposits and withdrawals, ensuring that they are conducted in the correct ratios. In addition, ensure that the value of users' assets before and after remains largely the same.
  - Monitor Uniswap and other DEXs for liquidity of any involved tokens in the Vault, including LPs and both wrapped and unwrapped stETH. Significant liquidity changes quickly may indicate ecosystem-wide problems and should warrant further investigation.
  - Monitor for very large deposits and very large withdrawals from the Vault. These may indicate malicious activity or the existence of an undetected bug. They may also indicate a need to raise the `_maximalTotalSupply` parameter.
  - Monitor for large transfers of LP tokens for the same reasons as above.
  - Monitor for large or frequent approvals of LP tokens to the same addresses for the same reasons as above.
  - Monitor the stETH contract for large deposits, withdrawals, or transfers. These may indicate an undetected bug in their system which could affect Mellow.

    - Monitor the stETH contract's deposit limits to ensure that the Vault will be able to deposit user funds into it.
    - Monitor for potential depegs between stETH and ETH by monitoring the prices of both assets across various exchanges, and asserting that they are fairly close to equal.

Consider maintaining good system documentation while it is running. Make this information easy to access from the main website for Mellow and ensure that it is clear to readers how they can submit a report if the documentation is incorrect. Ensure that all contract addresses are accessible. Furthermore, document all system parameters while the system is live. This should include lists of all admin and operator role holders, all tokens and their associated TVL modules, and all parameters in the `VaultConfigurator` contract. Monitor for event emissions related to changing `VaultConfigurator` parameters, tokens, or TVL modules.

Explain and document the various permissions defined in the `ManagedValidator` contract. The index of each bit for a given role should be very clear to prevent accidentally giving the wrong role to the wrong individual. While this already exists in the system documentation, care should be taken to ensure that the information does not become outdated.

Periodically check all Admin or Operator role holders in all contracts, as well as all permissions in the `ManagedValidator`, and ensure that any unneeded roles or permissions are revoked. Ensure that for all addresses with special privileges, the keys are accessible by periodically ensuring that key holders can sign messages, similar to the drills mentioned above. If any roles or permissions are no longer needed, inform the community and revoke them.

Monitor for calls to the various Lido functions which the protocol uses, even if the protocol does not initiate these calls. Ensure expected return values and behaviors to identify smart contract bugs or changes to external systems. These calls include:

- The `get_deposit_root` function of the Beacon Chain Deposit Contract
- The `getStakingModuleMaxDepositsCount` function of the Lido Staking Router
- The `getMaxDeposits` and `depositBufferedEther` functions of the Deposit Security Module
- The `unfinalizedStETH` function of the Withdrawal Queue contract
- The `getBufferedEther` and `submit` functions of the stETH contract
- The `wrap` and `unwrap` functions of the wstETH contract
- The `deposit` and `withdraw` functions of the wETH contract

# Known Issues

- If the vault fee is non-zero, an issue can arise in certain specific cases when batches of withdrawals are processed: a malicious user can front-run the `Vault.processWithdrawals` function to gain rewards from the fees. Moreover, the users in the batch will not receive fees from previously processed requests in the batch.
- In the `SimpleDVTStrategy`, the `processWithdrawal` function does not check the return value of `vault.delegateCall()` (used to deposit `amountForStake` into Lido). As a result, the actual staked amount may be inconsistent with the value reflected in the events.

- There is a case where deposits will not be fully deposited into the SimpleDVT staking module. This issue arises from how Lido allocates deposits across staking modules, where another staking module might take priority over the SimpleDVT staking module. The case can be exemplified as follows:

    - Assume Lido has 0 depositable ETH, Mellow has 64 WETH, and 32 ETH will be allocated to another staking module first, with the remaining 32 allocated to Obol:
        1. `maxDepositsCount` will be 1.
        2. The amount to deposit will be 32 ETH, which will be fully used for the other staking module.

  This issue occurs because `getStakingModuleMaxDepositsCount` is called with `(wethBalance + bufferedEther - unfinalizedStETH)`, which in this case is 64. However, the amount deposited into Lido is later capped to 32 ETH * `maxDepositsCount` = 32.

# Low Severity

## L-01 Missing Error Message in `revert` Statement

Within the `initialize` function of `Initializer.sol`, there is a `revert` statement in line 19 that lacks an error message.

Consider including specific, informative error messages in `revert` statements to improve the overall clarity of the codebase and to avoid potential confusion when a function reverts.

***Update:*** *Acknowledged, not resolved.*

## L-02 DoS Attack Vector in `processWithdrawals()` When Tokens Have a Blacklist

In cases where a Vault has, among its underlying tokens, any token with blacklist functionality (e.g., USDC), the withdrawal process can be subject to a DoS (Denial of Service) attack.

To withdraw funds from the vault, users must first submit a withdrawal request specifying, among other details, a `to` address that will receive the funds once the withdrawal is executed. A malicious user could specify the address of a blacklisted user. Consequently, when the vault processes the batch of withdrawal requests, the transaction will revert if it attempts to transfer USDC to a blacklisted address. The malicious user can exploit this by repeating the process from different addresses, causing a DoS for the withdrawals. The attacker will be able to do this without risking their funds as they can simply cancel the request and resubmit it with a different recipient address.

Consider implementing logic that prevents the entire transaction from failing if only one of the transfers fails (e.g., a try-catch block). Alternatively, consider documenting this possibility.

***Update:*** *Acknowledged, not resolved.*

## L-03 No Deposit "Pause Guardian" Implemented

The functions which affect the deposit lock within `VaultConfigurator.sol` are the only functions which are marked with the `atLeastOperator` modifier. This indicates that deposit pausing has been set up to be called quickly, potentially from a special address that has the `OPERATOR` role rather than the more powerful `ADMIN_ROLE`.

However, the only addresses with the `OPERATOR` role are the `SimpleDVTStakingStrategy` contract and a multisig deployed at Ethereum address `0x2E93913A796a6C6b2bB76F41690E78a2E206Be54`. The `SimpleDVTStakingStrategy` contract cannot call any of these functions. Noting that the value of `_isDepositLockedDelay` is currently set to 1 hour, it may significantly increase the time needed to pause deposits in an emergency if any of the members of the multisig are unreachable even for a few hours. In emergency situations, it is crucial to react quickly and limit risk by stopping deposits.

Consider implementing the planned "security fuse" feature as soon as possible. Alternatively, consider implementing a contract which is only able to call the deposit lock setter functions and granting that contract the `OPERATOR` role. This contract should be callable by a single EOA which has high uptime for fast responses in case of emergency.

***Update:*** *Acknowledged, not resolved.*

## L-04 Missing Zero Address Checks

When assigning an address from a user-provided parameter, it is crucial to ensure that the address is not set to zero. Setting an address to zero is a sensitive operation because it has special burn/renounce semantics. Instead, this action should be handled by a separate function to prevent accidental loss of access during value or ownership transfers.

Throughout the codebase, multiple instances of assignments missing zero checks were identified:

- The `newAcceptor` assignment operation within the `AdminProxy` contract in `AdminProxy.sol`.
- The `newEmergencyOperator` assignment operation within the `AdminProxy` contract in `AdminProxy.sol`.
- The `newProposer` assignment operation within the `AdminProxy` contract in `AdminProxy.sol`.

Consider adding a zero address check before assigning a state variable.

***Update:*** *Acknowledged, not resolved.*

## L-05 Protocol Admins May Circumvent Configurator Delays

Within the `VaultConfigurator` contract, the delays to update various different parameters are defined. The `_baseDelay` defines the delay needed to wait before changing many other parameters, including delays themselves.

For any delays greater than `_baseDelay`, the effective delay to update the corresponding parameter is actually just `_baseDelay`. Currently, the `_areTransfersLockedDelay` is `365 days`. However, to update this value, the protocol admins only need to wait `_baseDelay`, which is `30 days`. They may do this by staging a new value of 0 for `areTransfersLockedDelay` which can be committed after `_baseDelay`, which is `30 days`. Then, they can instantly stage and commit an update to `_areTransfersLocked`. Thus, the minimum delay for changing `_areTransfersLocked` is actually only 30 days.

Consider utilizing a `max` function for the delay for updating various `...Delay` parameters. For example, consider changing `_baseDelay` to `max(_baseDelay, _areTransfersLockedDelay)` within `commitTransferLockedDelay()`. This should be repeated for all functions matching the naming pattern of `commit...Delay()`. This will ensure that the delay cannot be reduced without waiting for at least that amount of time. In addition, consider implementing a `MIN_DELAY` parameter to match the `MAX_DELAY` parameter within the `stage...Delay()` functions. This will help ensure that a delay cannot be set to 0, which may be abused to update parameters instantly and harm users.

***Update:*** *Acknowledged, not resolved.*

## L-06 Token Removal Can Be Griefed

Within the `Vault` contract, the `removeToken` function is designed to enable the admins of the Vault to remove one of its configured tokens. However, there is a check on the balance of this token: if the balance is not 0, the call to `removeToken()` will revert. A malicious actor may abuse this by sending 1 token unit to the contract if they detect a call to `removeToken` in the mempool.

However, since the balance check is performed by calling `underlyingTvl()` which utilizes external TVL modules, the balance can be forced to be 0 for the sake of this check by simply removing the relevant TVL module. With the current state of the deployed code, there is only a single TVL module that is responsible for both tokens.

Consider implementing a contract or function which allows the Vault admins to call `removeTvlModule()`, then `addTvlModule()`, and optionally `removeToken()` atomically. This can be utilized to remove a token from the Vault without needing to use Flashbots to avoid frontrunning. It would also update the list of tokens and TVL modules atomically, preventing any mishaps from misconfigurations due to "partial removals" of tokens, such as removing a token from the Vault but not from the TVL list.

Note that this issue is similar to **5.12 Possible to Block Token Removal** from the ChainSecurity audit report.

***Update:*** *Acknowledged, not resolved.*

## L-07 Potential Fee Change Between Withdrawal Request and Processing

In the current deployment of the DVstETH system, `VaultConfigurator.emergencyWithdrawalDelay()` has been set to `90 days`. This means that if a withdrawal request is not processed by an Operator within this period, a user can trigger an emergency withdrawal to recover funds. On the other hand, the withdrawal fee can be changed by an admin through the `VaultConfigurator` but there is a system for staging and committing which introduces a delay of `_baseDelay` (currently set to `30 days`) until the new fee takes effect.

The issue arises when the fee changes between the moment a user requests a withdrawal and the time the withdrawal is processed. This behavior can be abused by the admin or happen unintentionally, potentially affecting the user negatively. For example, a user's withdrawal request may intentionally be ignored until after the fee has been increased.

To prevent this issue, consider adding a field to the `WithdrawalRequest` struct to indicate the fee that is applicable to the request at the moment it was submitted.

***Update:*** *Acknowledged, not resolved.*

## L-08 Missing Docstrings

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

- The `convertAndDeposit` function in `StakingModule.sol` contains many parts of the logic that are not fully explained, such as the interactions with Lido and the deposit contract
- The `receive` function in `Vault.sol`.
- The `_update` function in `Vault.sol`
- The `AllowAllValidator` contract in `AllowAllValidator.sol`
- The `ConstantAggregatorV3` contract in `ConstantAggregatorV3.sol`
- The `Initializer` contract in `Initializer.sol`

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. 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:*** *Acknowledged, not resolved.*

# Notes & Additional Information

## N-01 Inconsistent Order Within Contracts

Throughout the codebase, multiple instances of contracts having inconsistent function ordering were identified:

- The `ChainlinkOracle` contract in `ChainlinkOracle.sol`
- The `DefaultAccessControl` contract in `DefaultAccessControl.sol`
- The `ManagedValidator` contract in `ManagedValidator.sol`
- The `Vault` contract in `Vault.sol`
- The `VaultConfigurator` contract in `VaultConfigurator.sol`

To improve the project's overall legibility, consider standardizing ordering throughout the codebase as recommended by the Solidity Style Guide (Order of Functions).

***Update:*** *Acknowledged, not resolved.*

## N-02 Unused Import

Having unused imports can negatively affect code clarity. The `import "../utils/DefaultAccessControl.sol";` import in `ManagedValidator.sol` is unused and can be removed.

Consider removing unused imports to improve the overall readability and maintainability of the codebase.

***Update:*** *Acknowledged, not resolved.*

## N-03 Unnecessary Casts

Throughout the codebase, multiple instances of unnecessary casts were identified:

- The `address(weth)` cast in the `DepositWrapper` contract
- The `address(wsteth)` cast in the `StakingModule` contract

To improve the overall clarity, intent, and readability of the codebase, consider removing any unnecessary casts.

***Update:*** *Acknowledged, not resolved.*

## N-04 Magic Numbers in the Code

Throughout the codebase, multiple instances of unexplained literals being used were identified:

- The `0x20` literal number in `Initializer.sol`
- The `0x20` literal number in `Initializer.sol`
- The `0x4` literal number in `ManagedValidator.sol`

Consider adding in-line comments for any literal numbers being used.

***Update:*** *Acknowledged, not resolved.*

## N-05 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is quite beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for their maintainers to contact the appropriate person about the problem and provide mitigation instructions.

Within the audit scope, none of the contracts have a security contact.

Consider adding a NatSpec comment containing a security contact above each contract definition. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

***Update:*** *Acknowledged, not resolved.*

## N-06 Missing Named Parameters in Mappings

Since Solidity 0.8.18, developers can utilize named parameters in mappings. This means mappings can take the form of `mapping(KeyType KeyName? => ValueType ValueName?)`. This updated syntax provides a more transparent representation of a mapping's purpose.

Throughout the codebase, multiple instances of mappings without any named parameters were identified:

- The `baseTokens` state variable in the `ChainlinkOracle` contract
- The `_aggregatorsData` state variable in the `ChainlinkOracle` contract
- The `vaultToData` state variable in the `ManagedRatiosOracle` contract
- The `vaultParams` state variable in the `ManagedTvlModule` contract
- The `_withdrawalRequest` state variable in the `Vault` contract
- The `_isUnderlyingToken` state variable in the `Vault` contract
- The `_isDelegateModuleApproved` state variable in the `VaultConfigurator` contract

Consider adding named parameters to mappings in order to improve the readability and maintainability of the codebase.

***Update:*** *Acknowledged, not resolved.*

## N-07 `registerWithdrawal` Will Not Revert When `lpAmount > Balance`

Within the `registerWithdrawal` function, whenever the requested withdrawal amount exceeds `balanceOf(msg.sender)`, `lpAmount` will be overwritten with `balance`. This behavior is not intuitive and may cause problems for users or external integrations which assume that `registerWithdrawal` succeeding indicates that the specified `lpAmount` will be withdrawn. In addition, this non-standard behavior may complicate things for users because the withdrawal is not immediately processed until much later. So, users may assume that they will receive some higher amount once their withdrawal is processed.

Consider making this behavior extremely clear to users and reminding them to only approve addresses they trust. In addition, consider adding inline documentation for third-party developers and reminding them to check their LP balance in the same transaction before performing any calls to `registerWithdrawal`.

***Update:*** *Acknowledged, not resolved.*

## N-08 Inconsistent Use of Named Returns

To enhance the readability of the contract, some functions could benefit from using named returns. One such function is the `analyzeRequest` function of the `Vault` contract. It has one named return variable while the other two returned `bool` values do not have names, making them less intuitive.

Consider using named returns in all relevant functions throughout the codebase. This will help improve code clarity and maintainability.

***Update:*** *Acknowledged, not resolved.*

## N-09 Impossible to Recover Stuck ETH in Vault

Within the `Vault` contract, the `receive` function allows native ETH to be sent to the Vault contract. However, there is no way to recover this ETH. While it is possible to use the `delegateCall` function to trigger a call to a contract which could send ETH to the

`DepositWrapper.deposit()` function, doing this would require choosing a `to` account to receive the deposit.

If the `receive` function is intended to only allow receiving ETH for use withdrawals from the wrapped ether (wETH) contract, consider adding a `require` statement which reverts when the sender is not the wETH contract. Alternatively, consider implementing a function which converts stuck ETH to wstETH within the Vault without minting any LP tokens. This will effectively credit the deposit as a "donation" to the pool. Ensure this function is authorized, `nonreentrant`, and cannot perform the first deposit to the pool.

Note that this finding matches issue **7.6 Vault Accepts Ether Transfers** of the ChainSecurity audit

***Update:*** *Acknowledged, not resolved.*

## N-10 Initializable Contract Implementation Not Locked Down

An implementation of `Initializer` contract in a proxy pattern allows anyone to call its `initialize` function. While not a direct security concern, preventing the implementation contract from being initialized is important, as this could allow an attacker to take over the contract. This would not affect the proxy contract's functionality as only the implementation contract's storage would be affected. In `Initializer.sol`, the constructor does not call anything to inhibit later calls to `initialize()`. Typically, this is accomplished using a `_disableInitializers()` function, but in this case, calling `initialize()` successfully once should prevent re-initialization via use of the `configurator` parameter.

Consider calling `initialize()` in the `Initializer` contract's `constructor` to prevent malicious actors from front-running initialization and to adhere to the best practices for proxy implementations.

Note that this issue is similar to **5.10 Initialization of Implementations** in the ChainSecurity Audit Report.

***Update:*** *Acknowledged, not resolved.*

## N-11 Functions Are Updating the State Without Event Emissions

Throughout the codebase, multiple instances of functions updating the state without an event emission were identified:

- The `constructor` function in `ConstantAggregatorV3.sol`
- The `initialize` function in `Initializer.sol`
- The `constructor` function in `SimpleDVTStakingStrategy.sol`
- The `constructor` function in `StakingModule.sol`
- The `constructor` function in `Vault.sol`
- The `constructor` function in `VaultConfigurator.sol`
- The `revokeDelegateModuleApproval` function in `VaultConfigurator.sol`
- The `constructor` function in `WStethRatiosAggregatorV3.sol`

Consider emitting events whenever there are state changes to make the codebase less error-prone and improve its readability.

***Update:*** *Acknowledged, not resolved.*

## N-12 Confusing or Nondescriptive Naming

Throughout the codebase, multiple instances of potential naming improvements were identified:

- There are many functions named `rollback...()` within the `VaultConfigurator` contract. These functions do not change the value back to some previous value but instead cancel a staged update. Consider renaming these functions to `cancelStaged...()`.
- There are many functions using the string "tvl" or "Tvl", such as `Vault.baseTvl()`, `Vault.underlyingTvl()`, [`Vault._calculateTvl()](https://github.com/mellow-finance/mellow-lrt/blob/1c885ad9a2964ca88ad3e59c3a7411fc0059aa34/src/Vault.sol#L86)`, `Vault._tvls`, and the `ITvlModule` function `tvl()`. Consider renaming these functions to clarify the differences between them.
- The `delegateCall` function of the `Vault` contract is remarkably similar to the solidity built-in function `delegatecall`. Consider renaming this function to something else (e.g., "doDelegateCall") to eliminate any confusion between the two.

- The `IVaultConfigurator`, `ITvlModule`, and `IManagedRatiosOracle` contracts all have `struct`s called `Data`, which differ from each other. Consider renaming these structs to be more descriptive of what data is encapsulated and to differentiate each instance.

Consider implementing the aforementioned renaming suggestions to improve code clarity and readability.

***Update:*** *Acknowledged, not resolved.*

## N-13 Binary Search Optimization in `removeToken()`

Since the `_underlyingTokens` array is sorted, this can be leveraged to save gas and optimize the code in the removeToken function of the `Vault` contract. If the array contains more than 4 tokens, a binary search could be implemented to more efficiently find the index of the token being removed.

Consider implementing a binary search to take advantage of the sorted `_underlyingTokens` array.

***Update:*** *Acknowledged, not resolved.*

## N-14 Code Simplification in `Vault.removeTvlModule`

Lines 243 and 244 of the `removeTvlModule` function of the `Vault` contract can be merged into the following line of code:
`if (!_tvlModules.remove(module)) revert InvalidState();`, following the same pattern as in `addTvlModule`.

Consider making the above change to simplify the code and make it easier to understand for future reviewers.

***Update:*** *Acknowledged, not resolved.*

## N-15 `processWithdrawals` Calculations May Be Simplified To Save Gas

Within the `processWithdrawals` function, there are many subcalls which perform calculations. With the currently deployed Vault, these calculations can be simplified. This is because the "ratios" for the Vault are set at `0` and `Q96` (the maximum allowed value) for the two tokens in the vault.

- Inside the loop in `calculateStack`, if `amounts[i] == 0`, then the `s.totalValue` computation can be skipped.
- Inside same loop, the `s.ratiosX96Value` computation can be skipped if `s.ratiosX96[i] == 0` or simplified to `+= priceX96` if `s.ratiosX96[i] == Q96`.
- Inside the first loop in `analyzeRequest`, the calculation of `expectedAmounts[i]` can be simplified to `= coefficientX96` when `ratiosX96 == Q96`.
- Iterations of the second loop within `processWithdrawals` can be skipped when `expectedAmounts[j] == 0`. This may be common when the ratio for some asset is `0`.

Consider implementing the aforementioned code simplifications to improve code clarity and maintainability.

***Update:*** *Acknowledged, not resolved.*

# Conclusion

The audited codebase is an extensible Vault that manages deposits into the Lido DVT system which is already deployed on the Ethereum mainnet. The Vault contains many modifiable parameters such as which assets are included in the Vault, what ratios deposits and withdrawals are conducted in, and setups for hooks on deposit and withdrawal. The Vault and its associated contracts may be re-implemented for different strategies in the future. As such, the contracts were investigated considering both current deployments as well as potential changes in the future.

Several low-severity issues and informational findings were identified, while recommendations were made to improve the overall quality of the codebase. Generally, the codebase is well thought out, and has been designed for generality and extensibility. It may benefit from increased inline documentation and simplification of unneeded complexity in specific cases (e.g., simplification when dealing with Vaults that only hold 2 tokens with ratios of `100%` and `0%`). We appreciate the Mellow team's prompt responses to our questions throughout the audit process.