STATE MIND

Mellow Multi Vault

06-01-2025 – 31-01-2025

Table of contents

1. Project brief 3

2. Finding severity breakdown 5

3. Summary of findings 6

4. Conclusion 6

5. Findings report 7

Critical

- Funds locked in EigenLayerWithdrawalQueue due to incomplete EigenLayerWithdrawalQueue.claimableAssetsOf() accounting 7

High

- Broken deposit functionality in IsolatedEigenLayerVault 8
- Funds will be stuck if the operator undelegated the staker 8

Medium

- Missing access control in the IsolatedEigenLayerVault.delegateTo() function 8
- Missing memory allocation in EigenLayerWithdrawalQueue.transferedWithdrawalsOf() 9
- Wrong manager contract in EigenLayerAdapter.areWithdrawalsPaused() 9
- The function IsolatedEigenLayerVaultFactory.getOrCreate() can't be called twice with the same parameter 10
- MAX_CLAIMING_WITHDRAWALS limit accounting Inconsistency 10

Informational

- Sanity checks 11
- Wrong value is passed to Transfer event in SymbioticWithdrawalQueue 11
- Gas optimisations 12
- Subvault removing may cause blocking of staked funds 12

Informational	Unused function in EigenLayerAdapter	13
	Code refactor for withdrawal process from EigenLayer strategy	13
	delegationApprover should know address of IsolatedEigenLayerVault before its creation	13
	Missing events in EigenLayer contracts	14
	The return value from EigenLayerWithdrawalQueue._pull() is not used	14
	Gas optimization in the EigenLayerWithdrawalQueue.transferPendingAssets() function	14
	Typos	15
	Gas optimizations in the EigenLayerWithdrawalQueue contract	15
	Extra check in the function EigenLayerWithdrawalQueue.transferPendingAssets()	16

1. Project brief

Title	Description
Client	Mellow
Project name	Mellow Multi Vault
Timeline	06-01-2025 - 31-01-2025

Project Log

Date	Commit Hash	Note
10-01-2025	733df8430261584a7bfb53fcd5ce9d11b4e7123a	Initial Commit
13-01-2025	1876687cd64e6269494de3264eefa97909e8b2e5	New commit with fixes
30-01-2025	84a28875947006a5aa0f17d83d9e19faec6f354c	Reaudit
16-02-2025	65ce8dde861678f5d00bd3ac5bd458a4bf523f7f	Reaudit with library bump

Short Overview

Mellow LRT functions as an LRT constructor, enabling users to deploy and manage their LRTs securely. Key features include robust access control and strategic asset management through modules and strategies.

Mellow Multi Vault allows you to operate directly with multiple subvaults. A subvault is a code-level abstraction for interacting with different restaking platforms and ERC4626 tokens. To interact with the EigenLayer contract system, IsolatedVaults are used, which allow restaking into different strategies.

Project Scope

The audit covered the following files:

- EigenLayerFactoryHelper.sol
- ERC4626Adapter.sol
- EigenLayerAdapter.sol
- EigenLayerWstETHAdapter.sol
- IsolatedEigenLayerVault.sol
- IsolatedEigenLayerVaultFactory
- IsolatedEigenLayerWstETHVault.sol
- IsolatedEigenLayerWstETHVaultFactory.sol
- SymbioticAdapter.sol
- EigenLayerWithdrawalQueue.sol
- SymbioticWithdrawalQueue.sol
- RatiosStrategy.sol
- Claimer.sol
- WhitelistedEthWrapper.sol
- MultiVault.sol
- MultiVaultStorage.sol

2. Finding severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds to be transferred to any party.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss of funds.
Informational	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Client regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Client is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

3. Summary of findings

Severity	# of Findings
Critical	1 (1 fixed, 0 acknowledged)
High	2 (2 fixed, 0 acknowledged)
Medium	5 (5 fixed, 0 acknowledged)
Informational	13 (6 fixed, 7 acknowledged)
Total	21 (14 fixed, 7 acknowledged)

4. Conclusion

During the audit of the codebase, 21 issues were found in total:

- 1 critical severity issue (1 fixed)
- 2 high severity issues (2 fixed)
- 5 medium severity issues (5 fixed)
- 13 informational severity issues (6 fixed, 7 acknowledged)

The final reviewed commit is 65ce8dde861678f5d00bd3ac5bd458a4bf523f7f

5. Findings report

CRITICAL-01	Funds locked in EigenLayerWithdrawalQueue due to incomplete EigenLayerWithdrawalQueue.claimableAssetsOf() accounting	Fixed at: 9e2032c

Description

Line: EigenLayerWithdrawalQueue.sol#L75

The **EigenLayerWithdrawalQueue.claimableAssetsOf()** does not take into account **accountData_.claimableAssets**. That is, after **EigenLayerWithdrawalQueue.handleWithdrawals()** is called and assets moved into **_accountData[account].claimableAssets**, the function will return zero, despite the fact that the remaining funds can still be claimed.

Impact: **RatiosStrategy** contract calls the **EigenLayerWithdrawalQueue.claimableAssetsOf()** to calculate withdrawal amounts. But the queue will account assets only from **WithdrawalData** and the remaining part from **_accountData[account].claimableAssets** will be locked on the **EigenLayerWithdrawalQueue** contract.

Since the current implementation of the **MultiVault** contract does not support calling the **EigenLayerWithdrawalQueue.claim()** function directly, the remaining funds will no longer be claimable.

This vulnerability is not critical for the user, as he can call the **EigenLayerWithdrawalQueue.claim()** from the queue himself, bypassing the **EigenLayerWithdrawalQueue.claimableAssetsOf()** function.

Recommendation

We recommend accounting **_accountData[account].claimableAssets** in **EigenLayerWithdrawalQueue.claimableAssetsOf()**.

HIGH-01	Broken deposit functionality in IsolatedEigenLayerVault	Fixed at: d11e531

Description

Line: IsolatedEigenLayerVault.sol#L40
When depositing into **MultiVault**, after receiving data from the strategy, the deposit amount is distributed across the subvaults. Depending on the subvault type, the corresponding adapter's logic is invoked via **delegateCall**. In the **EigenLayerAdapter**, an allowance is granted to the **IsolatedEigenLayerVault** during the deposit call, followed by a call to its respective function at Lines 124-125. Within the deposit function of the **IsolatedEigenLayerVault**, a direct call is made to the **StrategyManager**. This call will revert because **safeTransferFrom** is not invoked beforehand, resulting in the balance of **IsolatedEigenLayerVault** always remaining zero.

Recommendation

We recommend transferring funds from **MultiVault** to **IsolatedEigenLayerVault** before interacting with **StrategyManager**.

HIGH-02	Funds will be stuck if the operator undelegated the staker	Fixed at: 9e2032c

Description

DelegationManager allows the operator to undelegate the staker. If the operator calls **DelegationManager.undelegate()**, the shares delegated to the operator will be queued for withdrawal to the staker address. In this case withdrawal happened outside of **EigenLayerWithdrawalQueue** and this withdrawal will not be handled. Due to **EigenLayerWithdrawalQueue** will not have this withdrawal request there will be no way to claim assets from EigenLayer.
Impact: assets of **IsolatedVault** will stuck in the EigenLayer, calculations in **MultiVault.totalAssets()** will be incorrect.

Recommendation

We recommend adding implementation to handle cases when request withdrawal happens outside of **EigenLayerWithdrawalQueue**. For example, using **EigenLayerAdapter.claimWithdrawal()**.

MEDIUM-01	Missing access control in the IsolatedEigenLayerVault.delegateTo() function	Fixed at: 9e2032c

Description

Line: IsolatedEigenLayerVault.sol#L25
When the factory creates **IsolatedEigenLayerVault** in the **IsolatedEigenLayerVaultFactory.sol#L42** contract, it calls **IsolatedEigenLayerVault.delegateTo()**. This function doesn't have access control and anyone can call it. EigenLayer doesn't allow to call **DelegationManager.delegateTo()** again if the staker already delegated shares. But the delegated operator can call **DelegationManager.undelegate()** and then anyone can call **IsolatedEigenLayerVault.delegateTo()** and delegate the vault's shares to the wrong operator.
Impact: The vault's shares will be delegated to the wrong operator who can commit illegal acts.

Recommendation

We recommend adding an access check in the **IsolatedEigenLayerVault.delegateTo()** function:

```
require(msg.sender == factory)
```

MEDIUM-02	Missing memory allocation in EigenLayerWithdrawalQueue.transferedWithdrawalsOf()	Fixed at: 9e2032c

Description

Line: EigenLayerWithdrawalQueue.sol#L100

The function declares a return value **uint256[] memory withdrawals** but never allocates memory for this array. When the function tries to write to **withdrawals[i]** in the loop, it will cause a revert: index out of bounds.

Recommendation

We recommend allocating memory for the return value.

```
function transferedWithdrawalsOf(address account, uint256 limit, uint256 offset)
    public
    view
    returns (uint256[] memory withdrawals)
{
    ...
    uint256 count = (length - offset).min(limit);
    withdrawals = new uint256[](count); // allocation
    for (uint256 i = 0; i < count; i++) {
        withdrawals[i] = transferedWithdrawals.at(i + offset);
    }
}
```

MEDIUM-03	Wrong manager contract in EigenLayerAdapter.areWithdrawalsPaused()	Fixed at: 9e2032c

Description

Line: EigenLayerAdapter.sol#L143

In the **EigenLayerAdapter.areWithdrawalsPaused()** function, the wrong contract is checked for a pause. The function is using **strategyManager** when it should be using **delegationManager** to check the withdrawal queue pause state. **PAUSED_ENTER_WITHDRAWAL_QUEUE** and **PAUSED_EXIT_WITHDRAWAL_QUEUE** operations are managed by the **DelegationManager** contract, not the **StrategyManager**.

Impact: This could lead to withdrawals being allowed when they should be paused or vice versa.

Recommendation

We recommend calling **DelegationManager** contract instead of **StrategyManager**.

```
- IPausable manager = IPausable(address(strategyManager));
+ IPausable manager = IPausable(address(delegationManager));
```

MEDIUM-04	The function IsolatedEigenLayerVaultFactory.getOrCreate() can't be called twice with the same parameter	Fixed at: 9e2032c

Description

Lines: IsolatedEigenLayerVaultFactory.sol#L33-L37

At the beginning of the function **IsolatedEigenLayerVaultFactory.getOrCreate()**, it checks if a corresponding **isolatedVault** has been already created and returns its Withdrawal queue:

```
bytes32 key_ = key(owner, strategy, operator);
isolatedVault = isolatedVaults[key_];
if (isolatedVault != address(0)) {
    return (isolatedVault, instances[isolatedVault].withdrawalQueue);
}
```

If it hasn't the function creates a new vault and queue.

After that, it should have added the new vault and queue the **isolatedVaults** mapping to return it the next time the function is called with the same parameters. The second call will lead to an error instead because it will try to create **IsolatedEigenLayerVault** at the same address where it already exists.

Recommendation

We recommend saving the vault and its corresponding queue to the storage.

MEDIUM-05	MAX_CLAIMING_WITHDRAWALS limit accounting Inconsistency	Fixed at: 84a2887

Description

Line: EigenLayerWithdrawalQueue.sol#L193

Problem scenario:

1. **MultiVault** has 6 claimable withdrawals
2. **EigenLayerWithdrawalQueue.claimableAssetsOf()** and **EigenLayerWithdrawalQueue.pendingAssetsOf()** would be calculated based on: the first 5 as "claimable" and 6th as "pending"
3. We call **MultiVault.withdraw()** and via **MultiVault._withdraw()** call **EigenLayerWithdrawalQueue.claim()** and **EigenLayerWithdrawalQueue.transferPendingAssets()** consequently
4. First **handleWithdrawals()** call processes 5 withdrawals in **EigenLayerWithdrawalQueue.claim()**
5. Second **handleWithdrawals()** call processes the last withdrawal in **EigenLayerWithdrawalQueue.transferPendingAssets()** and convert **pendingAssets** into **claimableAssets**. Now we have 0 pending withdrawals and a non-zero amount for transfer
6. Transaction reverts with **revert("EigenLayerWithdrawalQueue: insufficient pending assets")**

Recommendation

We recommend reconsidering logic when the claimable **withdrawal[i]** is considered pending due to **i > MAX_CLAIMING_WITHDRAWALS** or changing call order in MultiVault._withdraw():

```
Subvault memory subvault = subvaultAt(subvaultIndex);
address this_ = address(this);
if (pending != 0) {
    IWithdrawalQueue(subvault.withdrawalQueue).transferPendingAssets(receiver, pending);
}
if (claimable != 0) {
    IWithdrawalQueue(subvault.withdrawalQueue).claim(this_, receiver, claimable);
}
```

INFORMATIONAL-01	Sanity checks	Acknowledged

Description

Lines:

- MultiVault.sol#L150
- MultiVault.sol#L156
- MultiVault.sol#L162

The only restriction when setting adapters for different protocols is the restriction on **msg.sender**—it must have the **SET_ADAPTER_ROLE**. There are no checks inside the setters, which means that it can add adapters configured for another vault.

Recommendation

We recommend adding sanity checks in setter functions for adapters to ensure that the vault address in the adapter is the MultiVault address.

INFORMATIONAL-02	Wrong value is passed to Transfer event in SymbioticWithdrawalQueue	Fixed at: d11e531

Description

Line: SymbioticWithdrawalQueue.sol#L145

In the if condition at Lines 141-145 shares from the **nextEpoch** are transferred. In the **Transfer** event **nextPending** value is passed for the **amount** field, but the **nextPending** is the actual amount of assets that were transferred not the withdrawal queue shares.

Recommendation

We recommend passing the **nextSharesToClaim** value for the **amount** field of the **Transfer** event.

INFORMATIONAL-03	Gas optimisations	Acknowledged

Description

Lines:

- MultiVault.sol#L236
- MultiVault.sol#L323
- EigenLayerWithdrawalQueue.sol#L97
- IsolatedEigenLayerVault.sol#L62

1. All calculations performed with **curatorFee** are necessary only If **data.curatorFeeD6** doesn't equal zero. If it is zero, then **curatorFee** will be zero and won't affect further calculations.
2. At Line 323 where **data[i].deposit** is subtracted from the variable **assets**, the value of **assets** is neither used nor returned afterward. As a result, this subtraction is a waste of gas and serves no purpose.
3. The **transferedWithdrawalsOf** function is declared as **public**, but it is not called within the **EigenLayerWithdrawalQueue** contract itself. Therefore, to save gas, it can be declared as **external**.
4. In the **RewardsCoordinator** function **processClaim()**, the second parameter specifies the **recipient**'s address. In **IsolatedEigenLayerVault**, the **recipient** address is set to **address(this)**, meaning the rewards are first transferred to the **IsolatedEigenLayerVault** and then forwarded to the **MultiVault**. This adds an unnecessary transfer since the **processClaim()** function can only be called by the **MultiVault**, which already has a mechanism to check the number of transferred assets. To optimize this process the **recipient** address in **processClaim** should be directly set to **the MultiVault** address instead of **address(this)** in the **IsolatedEigenLayerVault**.

Recommendation

We recommend accounting for **curatorFee** only if **data.curatorFeeD6** equals zero.

```
...

if (data.curatorFeeD6 != 0) {
    uint256 curatorFee = rewardAmount.mulDiv(data.curatorFeeD6, D6);
    rewardAmount = rewardAmount - curatorFee;

    rewardToken.safeTransfer(data.curatorTreasury, curatorFee);
}
...
```

Also, we recommended resolving all other gas optimisation issues

INFORMATIONAL-04	Subvault removing may cause blocking of staked funds	Acknowledged

Description

Line: MultiVault.sol#L97

The function for removing a subvault from the set in **MultiVault** is restricted and can only be called by the role **REMOVE_SUBVAULT_ROLE**. In the internal function **_removeSubvault**, data is simply deleted from the array and the mapping. However, there are no checks to ensure that there is no active stake in the given subvault. This could lead to a situation where, due to careless or malicious removal, MultiVault's funds could become stuck in the subvault, as it will no longer be possible to make a withdrawal request after removal.

Recommendation

We recommend adding a sanity check when removing a subvault, that **MultiVault** doesn't have an active stake in this subvault or making a withdrawal of this stake before removal.

INFORMATIONAL-05	Unused function in EigenLayerAdapter	Fixed at: 9e2032c

Description

Line: EigenLayerAdapter.sol#L129

Function **EigenLayerAdapter.claimWithdrawal()** can be invoked via **delegateCall** and from **MultiVault**. However, there is no logic in **MultiVault** to call this function. Additionally, within this function, it calls **IsolatedEigenLayerVault.claimWithdrawal()**, but this function can only be invoked by the **EigenLayerWithdrawalQueue**. Consequently, the call will revert.

Recommendation

We recommend adding possibility to call this function from **MultiVault** contract to handle unexpected withdrawals.

INFORMATIONAL-06	Code refactor for withdrawal process from EigenLayer strategy	Acknowledged

Description

Line: EigenLayerAdapter.sol#L109

During the withdrawal process, the adapter calls the **withdraw()** function in **IsolatedEigenLayerVault**, which in turn internally calls the **request()** function in **EigenLayerWithdrawalQueue**. Within this function, the required **calldata** is generated, and it calls back to **IsolatedEigenLayerVault** at Line 138. This approach is gas-intensive, complicates the logic, and increases the risk of unpredictable behavior.

Instead, the parameters for the **delegationManager**'s function calls can be formed directly within the **IsolatedEigenLayerVault**. When calling **withdraw** in **IsolatedEigenLayerVault**, the process can be optimized as follows:

1. Generate the parameters for the **queueWithdrawals()** function and call it in the **delegationManager**.
2. Call **request** in **WithdrawalQueue** and make the necessary changes only to the state of the queue. This approach eliminates reentrancy during calls between **IsolatedEigenLayerVault** and **WithdrawalQueue**, reduces the risk of unpredictable behavior, and optimizes gas costs.

Recommendation

We recommend refactoring the code to eliminate reentrancy calls and remove excessive logic from the **EigenLayerWithdrawalQueue**.

INFORMATIONAL-07	delegationApprover should know address of IsolatedEigenLayerVault before its creation	Fixed at: 84a2887

Description

Line: IsolatedEigenLayerVaultFactory.sol#L42

When creating an **IsolatedEigenLayerVault**, along with the addresses of the operator, strategy, and owner, a special signature from the **delegationApprover** is also provided. Immediately after the **Vault** is created, the **delegateTo()** function in the **delegationManager** is called. This function receives the provided signature, which is validated if the operator has a configured **delegationApprover**.

The **delegationApprover** signs the addresses of the staker, operator, and salt, thereby authorizing the staker to delegate for a specific operator. However, to generate this signature, the delegationApprover must know the address of the IsolatedEigenLayerVault being created.

Recommendation

We recommend adding a **view** function that returns the address of the planned **IsolatedEigenLayerVault**.

INFORMATIONAL-08	Missing events in EigenLayer contracts	Fixed at: 9e2032c

Description

None of EigenLayer's contracts includes any **events**, which makes it challenging for off-chain observers and other services to efficiently track actions and interactions.

Recommendation

We recommend introducing relevant events in functions to improve transparency and enable better monitoring.

INFORMATIONAL-09	The return value from EigenLayerWithdrawalQueue._pull() is not used	Fixed at: 84a2887

Description

Line: EigenLayerWithdrawalQueue.sol#L258

The function **EigenLayerWithdrawalQueue._pull()** returns a **bool** value, which is not used anywhere.

Recommendation

We recommend removing the returned value.

INFORMATIONAL-10	Gas optimization in the EigenLayerWithdrawalQueue.transferPendingAssets() function	Fixed at: 84a2887

Description

Line: EigenLayerWithdrawalQueue.sol#L179

If **accountAssets == amount** then we should remove **withdrawalIndex** from **accountData_.withdrawals**, change **balances** and exit the loop, but we don't exit the loop. This will waste more gas.

Recommendation

We recommend adding extra condition:

```
if (accountAssets <= amount) {
    delete balances[from];
    balances[to] += accountShares;
    accountData_.withdrawals.remove(withdrawalIndex);
    amount -= accountAssets;
    if (amount == 0) {
        return;
    }
    pendingWithdrawals--;
} else {
    uint256 shares_ = accountShares.mulDiv(amount, accountAssets);
    balances[from] -= shares_;
    balances[to] += shares_;
    return;
}
```

INFORMATIONAL–11	Typos	Acknowledged

Description

Lines:

SymbioticWithdrawalQueue.sol#L217 – **If claimEpoch is zero, no claims are made for claimEpoch – 1.**

MultiVault.sol#L370 – **// emitting event with transferred + new pending assets**

IsolatedEigenLayerVault.sol#L56 – **IRewardsCoordinator coordinator**

Typo in parameter name **receiver**:

- EigenLayerAdapter.sol#L112
- ERC4626Adapter.sol#L52
- IsolatedEigenLayerWstETHVault.sol#L36

Typo in the word **transferred**:

- EigenLayerWithdrawalQueue.sol#L97
- EigenLayerWithdrawalQueue.sol#L103
- EigenLayerWithdrawalQueue.sol#L153
- EigenLayerWithdrawalQueue.sol#L177
- EigenLayerWithdrawalQueue.sol#L242
- EigenLayerWithdrawalQueue.sol#L245
- EigenLayerWithdrawalQueue.sol#L299

Recommendation

We recommend correcting the typos.

INFORMATIONAL–12	Gas optimizations in the EigenLayerWithdrawalQueue contract	Acknowledged

Description

Lines:

- EigenLayerWithdrawalQueue.sol#L58
- EigenLayerWithdrawalQueue.sol#L75
- EigenLayerWithdrawalQueue.sol#L277

The **EigenLayerWithdrawalQueue** contract contains functions that iterate through **AccountData.withdrawals** values. It can be done more efficiently by accessing the elements directly by the index in the set.

Recommendation

To save gas, we recommend changing the **AccountData.withdrawals** reading method to an index reference.

INFORMATIONAL-13	Extra check in the function EigenLayerWithdrawalQueue.transferPendingAssets()	Acknowledged

Description

Lines:

- EigenLayerWithdrawalQueue.sol#L216
- EigenLayerWithdrawalQueue.sol#L203
- EigenLayerWithdrawalQueue.sol#L283-L290

The function **EigenLayerWithdrawalQueue.transferPendingAssets()** checks if an account's **withdrawal** has the **isClaimed** flag set to **true** but all **withdrawal**s of the account marked as claimed have been removed by calling the function **handleWithdrawals()**. Therefore, this check may be removed to save the gas.

Recommendation

We recommend considering all account's **withdrawal** as not claimed.

STATE
MIND