STATE MIND

Mellow Simple LRT

05-09-2024 - 20-09-2024

Table of contents

1. Project brief

Title	Description
Client	Mellow
Project name	Mellow Simple LRT
Timeline	05-09-2024 - 20-09-2024

Project Log

Date	Commit Hash	Note
09-09-2024	164dcc755cc45d129981f9c0488721ac3c099ae9	First commit for audit
27-09-2024	3eae981b480c0060405c4001a447a0075960dbd1	Reaudit
24-01-2025	7f99510648be262f26184e87285884efa84e6af4	Final commit

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.

The Mellow Simple LRT centers around the MellowSymbioticVault which is specifically designed for Symbiotic's Vault and Symbiotic's Collateral.

Project Scope

The audit covered the following files:

- MellowSymbioticVault.sol
- SymbioticWithdrawalQueue.sol
- Migrator.sol
- ERC4626Vault.sol
- MellowSymbioticVaultStorage.sol
- VaultControlStorage.sol
- MellowVaultCompat.sol
- EthWrapper.sol
- VaultControl.sol
- MellowSymbioticVaultFactory.sol
- IdleVault.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	0 (0 fixed, 0 acknowledged)
High	0 (0 fixed, 0 acknowledged)
Medium	2 (1 fixed, 1 acknowledged)
Informational	13 (3 fixed, 10 acknowledged)
Total	15 (4 fixed, 11 acknowledged)

4. Conclusion

During the audit of the codebase, 15 issues were found in total:

- 2 medium severity issues (1 fixed, 1 acknowledged)
- 13 informational severity issues (3 fixed, 10 acknowledged)

The final reviewed commit is 7f99510648be262f26184e87285884efa84e6af4

Deployment

Mellow

Contract	Address
MellowVaultCompat	0x09bBa67C316e59840699124a8DC0bBDa6A2A9d59
Migrator	0x643ED3c06E19A96EaBCBC32C2F665DB16282bEaB
EthWrapper	0x7A69820e9e7410098f766262C326E211BFa5d1B1
MellowSymbioticVault (Implementation)	0x04e0581F5C7B1F760a5245FB58600840f03A3db9
MellowSymbioticVaultFactory	0x6EA5a344d116Db8949348648713760836D60fC5a

Symbiotic

Contract	Address
Symbiotic Vault (Implementation)	0xDd649AdaB2e67cAdC2EC29d75ABe73f3Df08065c
NetworkRestakeDelegator (Implementation)	0xc583e3E488C3CD8738850e2E7B19eF3f23e82e8A
VetoSlasher (Implementation)	0xAE5Bb0C1b2f5d8fc077b2451E23439Ed88c458cc
BurnerRouter (Implementation)	0x42dD40dC2130c658AB32d9989FF8aBe6c36463c0
RockX Vault (Proxy)	0x575d6DD4EA8636E08952Bd7f8AF977081754B1B7
RockX NetworkRestakeDelegator (Minimal Proxy)	0xCe12a04884c1466AfCd64A7A14f5f24Fa8aB2Ec8
RockX VetoSlasher (Minimal Proxy)	0xcf2c14E178eD99D9F93c8790E6852AA6230D8F10
RockX BurnerRouter (Minimal Proxy)	0x3D22d55E46a8682b0B1A6e6379aEa172A3a59Df2
Amphor Vault (Proxy)	0x446970400e1787814CA050A4b45AE9d21B3f7EA7
Amphor NetworkRestakeDelegator (Minimal Proxy)	0xA6851E43FA955753ee90a72c59030e0423F27E41
Amphor VetoSlasher (Minimal Proxy)	0x04a216411317A334C234C6ABDD589bB94d303d5b
Amphor BurnerRouter (Minimal Proxy)	0x8622EB8A33A5B13D77D4316D31BBeE2BCdCaC247
Renzo Vault (Proxy)	0xa88e91cEF50b792f9449e2D4C699b6B3CcE1D19F
Renzo NetworkRestakeDelegator (Minimal Proxy)	0x6d636B070Cd59B17B47931E9E0108869A310FB39
Renzo VetoSlasher (Minimal Proxy)	0xCF42f1B5E0ABB6f1521D8D523715781F9cd933C5
Renzo BurnerRouter (Minimal Proxy)	0x32cA52928e572684c54737A268AF7D8E1900C1b5
Steakhouse Vault (Proxy)	0xf7Ce770AbdD1895f2CB0989D7cf2A26705FF37a7
Steakhouse NetworkRestakeDelegator (Minimal Proxy)	0x510Bdf01886c7899b39c77012fFd10102A513732
Steakhouse VetoSlasher (Minimal Proxy)	0x6DD18a9c62F809bbE6b5Cc101e36FD0bce49D0d8
Steakhouse BurnerRouter (Minimal Proxy)	0xA7280514B8C8Ef206dc06d4649344Ec5eCB1f2d3
Restaking Vault (Proxy)	0x7b276aAD6D2ebfD7e270C5a2697ac79182D9550E
Restaking NetworkRestakeDelegator (Minimal Proxy)	0xdc439a51AB1C1D4DB0CD09A009f94eC4D127D93c
Restaking VetoSlasher (Minimal Proxy)	0x295F8c41eA17B330853AC74D1477a6F83B36ee31

Restaking BurnerRouter (Minimal Proxy)	0x04B01F1778A645f814E2555aE41BFF01e967c6f5
Re7 Vault (Proxy)	0x3D93b33f5E5fe74D54676720e70EA35210cdD46E
Re7 NetworkRestakeDelegator (Minimal Proxy)	0xdE43dade5D05B31b1e3524A268b32314Dee51DA0
Re7 VetoSlasher (Minimal Proxy)	0x31Ebe6E0bec2fa2167903D9C4487c9F0F496b64E
Re7 BurnerRouter (Minimal Proxy)	0x3ee393318CeD722CDecf997798437BfB25C9ABD9

-
  - Verification of BurnerRouter contracts was performed on commit c5840718d3ebf79500b420e5f082e4f8a4363d48. This version is up-to-date with Symbiotic and has undergone audits.

5. Findings report

MEDIUM-01	Incorrect deposit limit during vault migration	Fixed at: 3eae981

Description

Line: Migrator.sol#L84

The **IMellowSymbioticVault.InitParams.limit** is the amount of the underlying asset for **MellowSymbioticVault**.
IMellowLRTConfigurator.maximumTotalSupply() returns the limit for the LP token amount for **Vault**.
Using **IMellowLRTConfigurator.maximumTotalSupply()** as **IMellowSymbioticVault.InitParams.limit** is correct if the exchange rate of the LP to the underlying asset is 1:1.
Most vaults have an exchange rate close to 1:1. However, some vaults have significantly different exchange rates.
For example, Re7 Labs Restaked wBTC Vault

```
totalSupply() = 16814928020000000000

baseTVL()
tokens_0 = 0x2260FAC5E5542a773Aa44fBCfeDf7C193bc2C599
amount_0 = 911461455
tokens_1 = 0x971e5b5D4baa5607863f3748FeBf287C7bf82618
amount_1 = 770031347

exchangeRate = 16814928020000000000 / (911461455 + 770031347) = 10 ^ 10 : 1
```

Therefore, **IMellowLRTConfigurator.maximumTotalSupply()** cannot be used as a deposit limit.

Recommendation

We recommend adding the **depositLimit** external parameter for **Migrator.stageMigration()**. Additionally, the contract can check that **depositLimit** >= the cumulative sum of **Vault.baseTVL()** amounts.

MEDIUM-02	Frontrun Symbiotic slashing allows user to avoid losses	Acknowledged

Description

Mellow Vault deposits users' assets into Symbiotic Vault. Slashing may occur in Symbiotic Vault and Mellow Vault users will have some losses. In Mellow Vault exists **liquidAsset**, which cannot be slashed, and a non-liquid asset(amount deposited to Symbiotic Vault), which can be slashed. If the user requests withdrawal and there are enough **liquidAssets** then Mellow Vault doesn't withdraw assets from Symbiotic Vault, but sends the assets to the user immediately. If there is not enough **liquidAsset** then Mellow Vault has to withdraw assets from Symbiotic Vault and a user has to wait for 2 epochs and some part of tokens can be slashed.
Due to this functionality, if there are enough **liquidAssets**, a user can front-run slashing transaction and request a withdrawal, he will immediately receive the assets and avoid slashing, but other users of the Mellow Vault will lose more funds because the loss will be distributed among the protocol users.

Recommendation

As one of possible solution, we recommend disallowing LP from withdrawing funds from liquid assets immediately.
The contract makes a withdrawal from liquid assets and Symbiotic Vault based on the proportions of assets between the liquid asset amount and symbiotic deposited amount. During withdrawal, the contract calculates the amount from the liquid asset, as well as a share from the Symbiotic Vault.
Part of the funds from the liquid asset goes into the pending state and is not available for immediate withdrawal or reverse deposit to Symbiotic.
The LP receives the pending liquid asset amount + Symbiotic withdraw amount when calling **SymbioticWithdrawalQueue.claim()**. During **MellowSymbioticVault.claim()**, Mellow Vault deposits available assets into the Symbiotic Vault and collateral.
Example:

```
The ratio between assets and shares is 1:1
bobShares = 100
aliceShares = 100
totalSupply = bobShares + aliceShares = 200
liquidAssets = 100 (Symbiotic Vault has a limit of 100 tokens)
depositedInSymbiotic = 100
totalAssets = liquidAssets + depositInSymbiotic = 200

Bob requests the withdrawal of 100 shares

// This asset goes into the pending state and is not available for immediate withdrawal, Bob has to claim them
bobWithdrawFromLiquidAssets = liquidAssets * bobShares / totalSupply = 100 * 100 / 200 = 50
// We request these tokens from Symbiotic, Bob has to claim this asset, it is slashable
bobWithdrawFromSymbiotic = symbioticAssets * bobShares / totalSupply = 100 * 100 / 200 = 50
// Request withdrawal from Symbiotic
symbioticPendingAssets = 50
pendingLiquidAssets = 50
newSymbioticAssets = 50
newLiquidAssets = 50


After this, Bob has a withdrawal request from Mellow Vault. He can claim his assets after 2 epochs, 50 tokens from liquid assets (non-slashable) and 50 tokens from Symbiotic (slashable).
```

Client's comments

We are aware of this issue, but from our point of view, the possibility of frontrunning is less critical for us than the complete absence of instant withdrawals. In practice, the limits of Symbiotic vaults will be determined by the needs of

the networks, which means that a significant portion of our vault's TVL may lie outside of the Symbiotic vault. If we restrict all this TVL to 'slow' withdrawals (up to two epochs of Symbiotic), it will lead to reduced utilization of our vault in various DeFi integrations, such as lending protocols.

INFORMATIONAL-01	Redundant depositToken change	Fixed at: 3eae981

Description

Lines:

- EthWrapper.sol#L67
- EthWrapper.sol#L75

The change is redundant because the variable is not used after the change.

Recommendation

We recommend removing the redundant **depositToken** change.

INFORMATIONAL-02	There is no msg.sender check in EthWrapper.receive()	Fixed at: 3eae981

Description

Line: EthWrapper.sol#L81

EthWrapper.receive() doesn't check **msg.sender**. Therefore, any ETH sent is accepted on the balance of the contract. However, the receive function is only needed to call **WETH.withdraw()**.

Recommendation

We recommend accepting ETH only if **msg.sender == WETH** in **EthWrapper.receive()** function.

INFORMATIONAL-03	Contracts have no protection against the takeover implementation attack.	Acknowledged

Description

Lines:

- IdleVault.sol#L9
- MellowSymbioticVault.sol#L23

OpenZeppelin initialization pattern documentation says:
Do not leave an implementation contract uninitialized. An uninitialized implementation contract can be taken over by an attacker, which may impact the proxy. To prevent the implementation contract from being used, you should invoke the _disableInitializers function in the constructor to automatically lock it when it is deployed:
Current implementation contracts don't contain a **SELFDESTRUCT** opcode and don't make a **delegatecall**, therefore, it has no negative impact.

Recommendation

We recommend adding a **_disableInitializers()** call to vaults constructors.

INFORMATIONAL-04	Reorg attack during IMellowSymbioticVault, SymbioticWithdrawalQueue deployment	Acknowledged

Description

Lines:

- MellowSymbioticVaultFactory.sol#L25
- MellowSymbioticVaultFactory.sol#L27

The **MellowSymbioticVaultFactory** uses **create** instead of **create2**. The create uses the address of the factory and nonce to compute an address where the contract will be deployed. This is susceptible to reorg attacks.
While reorgs on Ethereum are mostly of depth 1.

Recommendation

We recommend using **create2** to deploy contracts and the proxy constructor, initialization arguments, and nonce for the salt creation.

Client's comments

We are aware of this issue, but due to its low applicability and significance, we will not be making any changes to the code.

INFORMATIONAL-05	The curatorFeeD6 update may affect undistributed rewards.	Acknowledged

Description

MellowSymbioticVault.setFarm() allows the **SET_FORM_ROLE** owner to update the **FarmData**. If **FarmData.symbioticFarm** already has accumulated rewards, then before updating **FarmData.curatorFeeD6**, the contract should collect the rewards for the previous period. Otherwise, the role owner can claim the rewards for the previous periods at a new rate.
Also, when resetting **FarmData**, the contract should collect the rewards for the previous period, otherwise they may get stuck in the **IStakerRewards** contract.

Recommendation

We recommend forcibly collecting rewards before updating the **FarmData** in existing data.

INFORMATIONAL-06	Lack of ERC4626Vault.mint() with referral parameter	Acknowledged

Description

There are two ways to deposit assets into Mellow Vault, **ERC4626Vault.deposit()** and **ERC4626.mint()**. The user can specify **referral** in the **ERC4626.deposit()** function, but can't do it in **ERC4626.mint()**, although it is also a deposit function.

Recommendation

We recommend adding another **ERC4626Vault.mint()** function that supports **referral**.

INFORMATIONAL-07	Unused event in SymbioticWithdrawalQueue	Fixed at: 3eae981

Description

Line: ISymbioticWithdrawalQueue.sol#L142

The **SymbioticWithdrawalQueue** contract interface contains an unused event **EpochClaimFailed**.

Recommendation

We recommend deleting or implementing an unused event.

INFORMATIONAL-08	Migrator.cancelMigration() doesn't reset the role owner	Acknowledged

Description

If the migration is canceled, **Migrator** returns the owner for the **ProxyAdmin** Migrator.sol#L103-L105 . However, a **Migrator** can have the role of **OPERATOR** and if the migration is canceled, the role is not renounced.

Recommendation

We recommend resetting the **OPERATOR** owner using **AccessControl.renounceRole()** during cancel migration.

INFORMATIONAL-09	Migration doesn't revoke or validate other roles	Acknowledged

Description

DefaultAccessControl.sol#L11-L13 has **OPERATOR**, **ADMIN_ROLE**, **ADMIN_DELEGATE_ROLE** roles, but after the migration is completed, the roles are not revoked and remain at the assigned addresses. The general recommendation is to reset the previous state so that only the slots of the current implementation have data. These slots may be used again in future migrations, but they already store data.

Recommendation

We recommend granting the **ADMIN** role to the **Migrator** contract. Before calling **IDefaultBondStrategy.processAll()**, grant the **OPERATOR** role to the **Migrator** contract. Revoke all owners for all roles before calling **ProxyAdmin.upgradeAndCall()**.

INFORMATIONAL-10	Missed events in the Migrator contract	Acknowledged

Description

The **Migrator** contract lacks event emissions in the following key functions: **Migrator.stageMigration()**, **Migrator.cancelMigration()**, and **Migrator.migrate()**.

Recommendation

We recommend adding event emissions to the **Migrator.stageMigration()**, **Migrator.cancelMigration()**, and **Migrator.migrate()** functions.

INFORMATIONAL–11	Stealing vault ownership by frontrun attack	Acknowledged

Description

There is a potential scenario where the admin of the **Migrator** contract can steal vault ownership during a migration. This vulnerability arises from the multi-step process of migration:

- The **Migrator** admin initiates **Migrator.stageMigration()** with the correct **proxyAdminOwner** parameter.
- Next, the **ProxyAdmin** owner sends a transaction to **ProxyAdmin.transferOwnership()** to transfer ownership to the **Migrator** contract.
- Before this transaction is included, the **Migrator** admin frontruns it with a **Migrator.cancelMigration()** transaction. Normally, this function transfers ownership back if the **Migrator** already owns the vault, but in this case, since the frontrun happens before the ownership transfer, it doesn't transfer ownership back.
- The **Migrator** contract now has ownership in the vault and calls **Migrator.stageMigration()** again, but this time with the admin's own **proxyAdminOwner**.
- Finally, the **Migrator** admin calls **Migrator.cancelMigration()**, transferring vault ownership to their own **proxyAdminOwner**.

Recommendation

We recommend using a private mempool for **ProxyAdmin.transferOwnership()** transaction to prevent possible frontrun attack.

INFORMATIONAL–12	Vault ownership may be stuck in Migrator	Acknowledged

Description

Line: Migrator.sol#L99

There is a risk that the vault ownership may be stuck in the **Migrator** contract after transferring the ownership of **ProxyAdmin** to the **Migrator**. If the **Migrator** contract does not call either the **Migrator.migrate()** or **Migrator.cancelMigration()**, the vault ownership could be permanently locked. This occurs because the **proxyAdminOwner** is unable to call **Migrator.cancelMigration()**.

Recommendation

We recommended allowing the **proxyAdminOwner** to call **Migrator.cancelMigration()** as an additional security measure.

INFORMATIONAL-13	Redundant super._approve() call	Acknowledged

Description

In the current implementation, **ERC20Upgradeable._allowances** migrate after **MellowVaultCompat.migrateApproval()**. **MellowVaultCompat.approve()** migrates **ERC20Upgradeable._allowances** and then writes new approval, but it calls **super._approve()** twice if **allowances_[from][to]** is not migrated. This will occur every time a user approves tokens and **[from][to]** pair is not migrated. To save gas, the contract should call **super._approve()** only once.

Recommendation

We recommend calling **super._approve()** only once to save gas on storage write.

```
function _approve(address owner, address spender, uint256 value, bool emitEvent)
    internal
    virtual
    override(ERC20Upgradeable)
{
    // replace migrateApproval(owner, spender) with code
    ERC20Storage storage compatStorage = _getERC20CompatStorage();
    uint256 allowance_ = compatStorage._allowances[from][to];
    if (allowance_ == 0) {
        return;
    }
    delete compatStorage._allowances[from][to];
    super._approve(owner, spender, value, emitEvent);
}
```

STATE
MIND