PUBLIC

Code Assessment

of the Symbiotic Vaults Smart Contracts

January 24, 2025

Produced for

by

Contents

1 Executive Summary 3
2 Assessment Overview 5
3 Limitations and use of report 11
4 Terminology 12
5 Open Findings 13
6 Resolved Findings 16
7 Informational 22
8 Notes 25

1 Executive Summary

Dear all,

Thank you for trusting us to help Mellow Finance with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Symbiotic Vaults according to Scope to support you in forming an opinion on their security risks.

Mellow Finance implements simple LRTs to tokenize deposits in Symbiotic.

The most critical subjects covered in our audit are function correctness, access control and integration with Symbiotic. The general subjects covered are gas efficiency, documentation and upgradeability. Security regarding the aforementioned subjects is good but improvable.

The most notable issues found were:

- Broken Queue Accounting
- Withdrawal Request Claiming Manipulation
- Bank Run on Excess Funds in Vault Prior to Slashing Event
- Migration can be DoSed

Note that the first two issues have been resolved through code correction. For the third and the fourth item, the risk has been accepted. Note that some other issues have been only partially corrected or their risk has been accepted.

Further, we provide some considerations for migration in Migration Considerations. See also the Notes for other considerations.

In summary, we find that the codebase provides a good but improvable level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	4
• Code Corrected	2
• Risk Accepted	2
Low-Severity Findings	6
• Code Corrected	4
• Code Partially Corrected	1
• Risk Accepted	1

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Symbiotic Vaults repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	21 Aug 2024	77a30bcf9d3fd9d24c46c33fed81e76afac02282	Initial Version
2	23 Aug 2024	564aa9d79cf46514b54a4acb29e0cb2bc0410d21	Preliminary Fixes
3	29 Aug 2024	fcdb2fff64d95cd71a9f61836356b7dbc9559b29	After Intermediate Report
4	30 Aug 2024	c2216cd0059abe6a5e08c4e19c07d92729e2ecdf	Further Changes
5	02 Sep 2024	62e676f1ef4acadf185108ae77ee52f2704b7b0f	Approval Migrations
6	11 Sep 2024	164dcc755cc45d129981f9c0488721ac3c099ae9	Migrator
7	19 Sep 2024	f1fbb71af935fead832a2c63f7188bd61a11197b	Fixes
8	23 Sep 2024	3eae981b480c0060405c4001a447a0075960dbd1	Add Account Data Getter
9	16 Dez 2024	8135ecc1ec5c8fcfa0311f57b0d9e8903dcc700e	Sync with Symbiotic Changes
10	23 Jan 2025	7f99510648be262f26184e87285884efa84e6af4	Dependency Bump

For the solidity smart contracts, the compiler version `0.8.25` was chosen.

As of Version 1, the following files were in scope:

```
src:
    ERC4626Vault.sol
    IdleVault.sol
    MellowSymbioticVault.sol
    MellowSymbioticVaultFactory.sol
    MellowSymbioticVaultStorage.sol
    SymbioticWithdrawalQueue.sol
    VaultControl.sol
    VaultControlStorage.sol
    interfaces:
        utils:
```

```
            ISymbioticWithdrawalQueue.sol
            IWithdrawalQueue.sol
        vaults:
            IERC4626Vault.sol
            IIdleVault.sol
            IMellowSymbioticVault.sol
            IMellowSymbioticVaultFactory.sol
            IMellowSymbioticVaultStorage.sol
            IVaultControl.sol
            IVaultControlStorage.sol
```

As of (Version 2), the following files were added to the scope:

```
src:
    EthWrapper.sol
    MellowVaultCompat.sol
    interfaces:
        tokens:
            ISTETH.sol
            IWETH.sol
            IWSTETH.sol
        utils:
            IEthWrapper.sol
```

In (Version 6), the following files were added to the scope for the vault migration:

```
src:
    Migrator.sol
    interfaces:
        utils:
            IMigrator.sol
```

The migration should be triggered on correctly configured vaults. Here is a non-exhaustive list of configuration assumptions:

1. Configurator: The vault and its configurator are wired correctly.
2. Vault:
    1. The Vault is solvent and should have only one expected underlying token.
    2. It only uses the DefaultBondStrategy and should only have two Tvl modules: ERC20TvlModule and DefaultBondTvlModule.
3. DefaultBondStrategy:
    1. The DefaultBondStrategy is the deposit callback registered at the vault configurator and it is wired correctly to the vault.
    2. The `tokenToData` of the underlying token should encodes 100% ratio allocation to the corresponding symbiotic default collateral.
4. DefaultBondTvlModule: The `vaultParams` should contain only one bond contract (the corresponding symbiotic default collateral) for this Vault.
5. Oracles: Oracle is configured correctly (i.e. for vault with wstETH as underlying token, the WStethRatiosAggregatorV3 should be used. For other vaults, the corresponding chainlink oracle should be used.)

6. Roles: The privileged roles on the Vault and peripheral contracts are honest and trusted (i.e. No unexpected storage slots are written with vault delegatecalls by malicious admins or operators).
7. Allowance: The vault should have no remaining allowance to any addresses.

2.1.1 Excluded from scope

All other contracts are out of scope. The correctness of Symbiotic is out of scope and is expected to work correctly as documented and according to the given state of the codebase.

Some steps for the migration are out of scope. Only the ERC-20 balance accounting is in scope. Please consider Migration Considerations.

2.2 System Overview

This system overview describes the second received version (Version 2) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Mellow Finance implements ERC-4626 like vaults to tokenize and integrate with Symbiotic Finance's vaults.

2.2.1 Vaults

Vaults allows users to deposit their funds and tokenize them in form of vault shares (as in ERC-4626). The general and abstract vault on which other vault build on is `ERC4626Vault` which adds the following features on top of the ERC-4626 functionality:

- Deposit Limit: A local limit on the managed assets by the vault can be set (only considered for deposits) with `setLimit`.
- Pausing functionality: Deposits and withdrawals can be paused `pauseDeposits` / `unpauseDeposits` and `pauseWithdrawals` / `unpauseWithdrawals`, respectively. Note that a pauser loses their respective role on pausing actions.
- Deposit Whitelist: A depositor whitelist can be set with `setDepositWhitelist`. In case the whitelist is active, addresses can be (un-)whitelisted with `setDepositorWhitelistStatus`.

Getters for all variables are provided. Note that these settings are accounted for in regard to the ERC-4626 functionality.

Note that `IdleVault` is equal to `ERC4626Vault` but is not abstract.

2.2.1.1 Mellow Symbiotic Vault

The `MellowSymbioticVault` inherits from `ERC4626Vault` and integrates with Symbiotic's vaults.

The Mellow Symbiotic Vaults may hold the underlying `asset` locally and in Symbiotic. Namely, depositing tries to deposit as much as possible into Symbiotic. However, in some instances (e.g. Symbiotic's deposit limit would be exceeded) funds can be held locally in the contract. However, note that anyone can try to push unpushed funds to Symbiotic with `pushIntoSymbiotic` if it becomes possible (or in case of other deposits). Note that `totalAssets` also accounts for that. Also, `getBalances` retrieves a user's number of shares, the number of assets this represents, and how much of these two could be immediately burned with the buffered funds.

Note that withdrawals are incompatible with ERC-4626. More specifically, the `withdraw` function tries to use as much as possible from the previously mentioned buffered funds. However, in case the buffered

funds are not sufficient, the remainder of the withdrawal will request a withdrawal from Symbiotic. The recipient of the withdraw request will be Mellow's `SymbioticWithdrawalQueue`. The queue is notified immediately through its `request` function.

The withdrawal queue essentially tracks all users' withdrawals (since the withdrawals in Symbiotic will be tracked for the queue). More specifically, it implements a share mechanism per epoch to split the claimed amounts fairly among withdrawers. Note that on every user interaction it first tries to handle all pending epochs (last tracked epoch and the one prior to that) to ensure that funds from Symbiotic have been claimed, and to account for the user's share. Note that with `claim` users can claim their unwithdrawn funds from the queue.

Further, the queue implements `handlePendingEpochs` which allows to handle the pending epochs for a user, without creating a request or transferring funds to the user. Additionally, epochs can be claimed independently with `pull`. Note that the queue implements the getters

- `getCurrentEpoch` to get the current epoch
- `pendingAssets` to compute how many assets are pending in total (not claimable yet)
- `pendingAssetsOf` to compute the pending assets for a user
- `claimableAssetsOf` to compute the claimable assets of a user (assets a user can receive)

The Mellow Symbiotic Vault additionally implements wrappers for `claimableAssetsOf`, `pendingAssetsOf` and `claim`.

Last, the Mellow Symbiotic Vault implements interactions with Symbiotic's reward contracts, referred to as farms. With `setFarm`, privileged addresses can whitelist farms. Then, arbitrary addresses can call `pushRewards` to claim the rewards from the default staker rewards v1. These rewards are then pushed to so-called distribution farms which shall allocate to Mellow's users (undefined).

2.2.1.2 ETH Compat Vaults

The ETH compat vaults are equal to the Mellow Symbiotic Vaults. They are intended to be used as an intermediate upgrade step. More specifically, the contract additionally migrates the user balances from the previous storage slot to the new one (similar with the total supply counter). Note that the getter functions for the balances and the supply also account for both storage slots. Last, `migrate` allows to migrate the balance for one user while `migrateMultiple` allows it for multiple. `compatTotalSupply` is offered as an indicator of the migration progress (at zero, the balance migration has been completed and the contract could be upgraded to the regular Mellow Symbiotic Vault).

Please consider Migration Considerations.

2.2.2 ETH Wrapper

The ETH wrapper supports `deposit` which allows inputting ETH, WETH, stETH or wstETH to convert it to wstETH to deposit it into an ERC4626 compatible vault (typically Mellow Symbiotic Vault).

2.2.3 Migrator

A migration process from previous implementation Mellow LRT has been implemented and added to the scope as of **Version 6**.

The process is intended to work as follows:

1. The `Migrator` contract is deployed.
2. `stageMigration` is called by the `Migrator.admin`. This derives the vault from the default bond strategy, retrieves the token to be used and stores this information along with the proxy admin and the proxy admin owner address as parameters for the `migrate` function. Note that some sanity checks are performed - however, they might be incomplete. Note that the `InitParams` for the call to `initialize` are additionally prepared (which includes also the deployment of the withdrawal queue).

3. After the migration is staged, no administrative changes should be performed as otherwise the migration might break.
4. The only exception for this is that the `Migrator` needs to receive the required roles for the old vault. Namely, that is `ProxyAdmin.owner` to allow upgrading the contract and an operator role for default bond strategy to process withdraw requests. Note that by transferring these rights, the governance set for the vault fully agrees to the parameters set during staging.
5. `migrate` can be called when `migrationDelay` has passed after staging. This processes all pending withdrawals (`processAll`) (in case no malicious actions have been taken by the governance, this should process all). Further, the implementation is upgraded and initialized. Last, this transfers back the ownership of the proxy admin. Note that some checks on the parameters stored are performed - however, they might be incomplete.

Note that the `admin` of the `Migrator` can cancel a migration with `cancelMigration`. Also note that there might be other revert reasons in case weird actions are taken. Also, other scenarios might lead to reverts (e.g. stale price feeds). In such cases, `admin` should cancel and eventually restart the migration.

In addition, the migration disregards the transfer lock on the configurator. In case the transfer lock is enabled, upon a successful migration, the transfer will be immediately enabled. 3rd-party systems that integrate with Mellow Vault should be careful of this kind of abrupt changes.

Any deviation from the process above could lead to undesirable results. Also, note that while sanity checks are performed, the parameters should nevertheless be validated.

2.2.4 Changes in Version 4

Mellow Finance now also uses Symbiotic's default collateral token as a deposit location, as Symbiotic will potentially incentivize deposits to the default collateral contracts. The deposits, now, try to deposit as much of the buffered asset into Symbiotic. If more can be deposited and the contract holds the default collateral token, the asset is withdrawn and moved to Symbiotic, too. However, if no full deposit to Symbiotic can be made, the remainder is deposited into the default collateral token. In case not all can be deposited there, the undepositable assets are buffered locally.

Similarly, withdrawals now additionally withdraw from the default collateral for instant withdrawals.

2.2.5 Changes in Version 5

The ETH compat vaults now also migrate user's approvals.

2.2.6 Changes in Version 6

Some formatting in regards to the compat vaults has been performed. Additionally, a Migrator was added to the scope. Note that as of this version the migration has been added to the scope. Please see Migrator for further details.

2.2.7 Changes in Version 7

The `EthWrapper` contract has received some minor adjustments. Namely, the `receive` function can now only receive ETH from the `WETH` contract. Further, the `MellowSymbioticVaultFactory` now deploys with `CREATE2`.

Upon discussion with Mellow Finance, the following adjustments have been made:

- `Migrator` now implements an interface equal to the factory for validating migrated entities. That also disallows re-staging migrated vaults.
- A call to `pushIntoSymbiotic` to allocate funds accordingly is made after a successful migration.

2.2.8 Changes in Version 8

The queue has now a function `getAccountData` that returns the stored values for the claimable assets, the pending shares to claim (for both relevant epochs) and the claim epoch. Last, upon discussing with Mellow Finance, the EVM version for the compilation has been fixed.

2.2.9 Changes in Version 9

Symbiotic changed the deposit limit logic. In this Version, the codebase was adapted to account for that.

2.2.10 Roles and Trust Model

The following roles are only defined for the vaults:

- `DEFAULT_ADMIN_ROLE`: Trusted. Has all the capabilities of the roles below.
- `SET_FARM_ROLE`: Trusted to honestly whitelist farms and not to steal rewards.
- `SET_LIMIT_ROLE`: Trusted to set reasonable limits. Note that the limit can be interpreted as a risk parameter - especially given the buffering mechanism.
- `PAUSE_WITHDRAWALS_ROLE`: Trusted to behave reasonably and to not pause withdrawals without reason.
- `UNPAUSE_WITHDRAWALS_ROLE`: Trusted to behave reasonably and to not unpause withdrawals in case of emergency.
- `PAUSE_DEPOSITS_ROLE`: Trusted to behave reasonably and to not pause deposits without reason.
- `UNPAUSE_DEPOSITS_ROLE`: Trusted to behave reasonably and to not unpause deposits in case of emergency.
- `SET_DEPOSIT_WHITELIST_ROLE`: Trusted to behave reasonably and to not break any integrations or similar.
- `SET_DEPOSITOR_WHITELIST_STATUS_ROLE`: Trusted to behave reasonably and to not break any integrations or similar.
- `Migrator.admin`: Fully trusted to execute the actions responsibly. However, to migrate, its actions need to be validated by the governance of the vault (according to the previous implementation). **Note** it is assumed that the admin private keys are properly managed, otherwise, in case of a private key loss after a migration is staged, the `proxyAdmin` ownership can not be retrieved back from the Migrator.
- `Vault` governance: Fully trusted (note that this relates to the previous version). Namely, otherwise migration could be manipulated.

Note that addresses with pausing capabilities lose their rights in case they pause the respective functionality.

Further

- Users are untrusted.
- The proxy admin is completely trusted and could steal all funds.
- The whitelisted addresses are all trusted.
- Tokens are trusted to be typical ERC-20 tokens (including tokens with fees, and the common reentrant token types)

Further, note that the migration from the old vaults requires careful consideration and requires preliminary steps. Please consider Migration Considerations.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Open Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Security: Related to vulnerabilities that could be exploited by malicious actors
- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	2

- Migration Can Be DoSed Risk Accepted
- Bank Run on Excess Funds in Vault Prior to Slashing Event Risk Accepted

Low-Severity Findings	2

- ERC-4626 Violations Code Partially Corrected
- Integration of Weird ERC20 Tokens Risk Accepted

5.1 Migration Can Be DoSed

Security Medium Version 6 Risk Accepted

CS-MLW-SYMV-012

A staged migration can be accomplished by the admin with `migrate()`, which will process all the pending withdraw requests (`processAll()`). This could be DoSed by maliciously registering many withdraw requests, consequently the call to `migrate()` will run out of gas and revert.

Risk Accepted:

Mellow Finance accepts the risk.

5.2 Bank Run on Excess Funds in Vault Prior to Slashing Event

Design Medium Version 1 Risk Accepted

CS-MLW-SYMV-001

The vault holds liquid assets when not all funds can be deposited into Symbiotic (e.g. because the deposit limit has been reached). The idle assets can be used to convert shares for underlying assets at the current exchange rate (balance/shares) and withdraw them from the vault without any delay.

This can lead to a bank run on the excess funds in the vault before a slashing event occurs.

Slashing events are predictable, especially if the vault uses a `VetoSlasher` that enforces a delay between the slash request and the execution of the slash. During the delay, users can withdraw their funds at the (pre-slash) exchange rate until all excess funds are used up. Consequently, the remaining share owners are the only ones affected by the slash.

Risk accepted:

Mellow Finance is aware of the issue and accepts the risk.

5.3 ERC-4626 Violations

Design | Low | Version 1 | Code Partially Corrected

CS-MLW-SYMV-006

The contracts implement the ERC-4626 tokenized vault standard. However, the standard is violated on multiple occasions:

1. According to the EIP, the ERC-4626 `max...` functions must take pausing into account and return zero if the deposits/withdrawals have been paused. Note that this is not the case. Note that this would also pause the actions accordingly. Further, `pushIntoSymbiotic` will revert when a deposit whitelist is set and the vault is not whitelisted and `max...` should return zero in that case.
2. `maxMint` computes its result as `convertToShares(maxDeposit(account))` which rounds down. However, in case that no `limit` was set, `maxDeposit` will return `uint256.max`. In that case, `maxMint` will not return `uint256.max` due to rounding. That violates the standard since, if no limit is imposed, the function must return `uint256.max`.
3. Note that any combination of `ERC20VotesUpgradeable` and `ERC4626Upgradeable` violate ERC-4626. Namely, the ERC-20 with votes imposes a supply limit `_maxSupply` which the `maxDeposit` and the `maxRedeem` functions do not account for. For example, that is the case for `IdleVault`.
4. The `MellowSymbioticVault` is not compatible with EIP-4626 as for example `withdraw` does not necessarily transfer the assets to the withdrawer and can emit event `Withdraw` without assets being transferred.
5. As of Version 2, the `maxDeposit` does not return `uint256.max` if the limit is set to the maximum which violates EIP-4626.

The following changes have been done in Version 3:

1. `pushIntoSymbiotic` will not revert when a deposit whitelist is set and the vault is not whitelisted in Symbiotic.

Code partially corrected:

1. Corrected.
2. Corrected.
3. Corrected. It does not inherit from the votes contract anymore.
4. Not corrected.
5. Corrected.

5.4 Integration of Weird ERC20 Tokens

Design | Low | Version 1 | Risk Accepted

CS-MLW-SYMV-003

The `MellowSymbioticVaultFactory` allows anyone to deploy their own `MellowSymbioticVault` around a Symbiotic Vault. Note that Symbiotic supports fee-on-transfer and reentrant tokens, but using these tokens can lead to the following issues with Mellow:

Issues with Fee-on-Transfer Tokens:

1. In `SymbioticWithdrawalQueue._pullFromSymbioticForEpoch()`, the contract uses the return value to determine the total claimable assets. If the token has a fee-on-transfer, the contract will receive less than expected, and not all assets can be claimed (insolvency).
2. In `MellowSymbioticVault.deposit()`, the depositor receives shares based on the amount of assets sent to the Mellow Vault. If these funds are further pushed to the Symbiotic Vault and incur transfer fees, these fees are socialized across all depositors.

Issues with Reentrant Tokens:

1. In `SymbioticWithdrawalQueue._pullFromSymbioticForEpoch()`, the call to `symbioticVault.claim()` can fail due to the reentrancy guard.
2. In `MellowSymbioticVault.safeTransfer()`, a malicious `curatorTreasury` can re-enter another contract and read deprecated state from the `MellowSymbioticVault` (Read-only reentrancy). If the reward token is the asset, `totalAssets()` will return an inflated value that includes the pending rewards that will be sent to the `distributionFarm`.

Version 3:

Issues with Fee-on-Transfer Tokens:

3. In `MellowSymbioticVault._withdraw()`, the contract uses withdraw amount from the default collateral to determine the amount of liquid assets. When the tokens has fees-on-transfer, the contract will receive less then expected.

Risk accepted:

Mellow Finance is aware of the issue and does not plan to use such tokens. Note that the factory, however, is permissionless.

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Open Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	2

- Broken Queue Accounting Code Corrected
- Withdrawal Request Claiming Manipulation Code Corrected

Low-Severity Findings	4

- Incorrect Deposit Limit During Migration Code Corrected
- Broken Pending Asset Computation Code Corrected
- Migration Code Corrected
- Whitelisting Might Block Pushing to Symbiotic Code Corrected

Informational Findings	4

- Unusual Unchecked Usage Code Corrected
- migrations Is Unused Code Corrected
- Gas Optimizations Code Corrected
- msg.sender Used in VaultControlStorage Code Corrected

6.1 Broken Queue Accounting

Security Medium Version 1 Code Corrected

CS-MLW-SYMV-002

The queue's shares are based on the underlying asset amounts withdrawn. However, that can create unfair distributions and might allow attacker to reduce the impact of slashing. More specifically, for the current epoch, Mellow vault shares withdrawn before a slashing event (in the current epoch) will be slashed less than shares withdrawn in the same epoch but after the slashing event.

Consider the following example:

1. The current epoch is `X`. Assume 1 share equals 1 asset.
2. Alice performs `redeem(100)` to redeem 100 Mellow Vault shares. That will, ultimately, withdraw 100 assets from Symbiotic and call `SymbioticWithdrawalQueue.request` with 100 assets as input.
3. Alice will have claimable shares of 100 for `X+1`.
4. Now a slashing event occurs. All the vault's stake is reduced by 50%.
5. As a result, Alice at the moment should only be able to receive 50 assets.

6. Now, Bob does `redeem(100)` to redeem 100 Mellow Vault shares. That will withdraw 50 assets from Symbiotic and call `SymbioticWithdrawalQueue.request` with 50 assets as input. Note that the difference is due to the change in exchange rate of the vault.
7. Alice will have claimable shares of 50 for `X+1`.
8. The queue has 100 assets (50 from Alice (recall that slashing reduced her withdrawn amount) and 50 from Bob). However, Alice will receive 66.66% of the claimable assets while Bob will receive only 33.33%. That is due to their queue shares are not equal (100 and 50).

Finally, there is an unfair distribution of claimed assets in an epoch. Note that Symbiotic distributes funds in a fair way. Further, note that slashing could be predictable. Ultimately, some users might reorder transactions so that their slashing penalty is reduced.

Code corrected:

In Version 3 the queue's shares are based on the withdrawal shares computed in Symbiotic. Any slashing event reduces the value of a given share so in the above example Alice and Bob would receive the same amount of shares.

6.2 Withdrawal Request Claiming Manipulation

Security | Medium | Version 1 | Code Corrected

CS-MLW-SYMV-011

The queue creates withdrawal requests on Symbiotic. Then, once the queue's request is claimable, the queue tries to claim it. However, it handles all reverts as legitimate reverts, i.e. as epochs without claims:

```
epochData.isClaimed = true;
try symbioticVault.claim(address(this), epoch) returns (uint256 claimedAssets) {
    epochData.claimableAssets = claimedAssets;
    emit EpochClaimed(epoch, claimedAssets);
} catch {
    // if we failed to claim epoch we assume it is claimed
    // most likely low funds in epoch got additionally slashed so we can't claim
    it (error becase of 0 amounts)
    emit EpochClaimFailed(epoch);
}
```

Note that this is not necessarily the case. Below are two examples of other reverts:

- If reentrant tokens are used, the reentrancy lock will trigger a revert (even if the epoch would require claiming).
- There might be a scenario where `claim` would run out of gas but the call could complete (even if the epoch would require claiming).

In both instances, the request claiming would be skipped and impossible thereafter. Ultimately, users could lose funds.

Code corrected:

The code has been fully corrected. It now considers all valid revert reasons that would allow for skipping of claiming rewards (especially rewards zero). However, the `isWithdrawalsClaimed` is not strictly necessary since if it were `true`, `isClaimed` would be `true`, too.

6.3 Incorrect Deposit Limit During Migration

Correctness | Low | Version 6 | Code Corrected

CS-MLW-SYMV-014

In Migrator, when a migration is staged (`stageMigration()`), the `maximalTotalSupply` will be retrieved from the vault configurator as part of the initialization parameters (`InitParams.limit`). However, it has different semantic from `limit`:

- `maximalTotalSupply` restricts the total supply of lp shares.
- `limit` restricts the total assets that can be deposited into the vault.

Consequently, if the conversion rate between shares and assets is not `1:1`, a wrong `limit` will be used.

Code corrected:

Code has been corrected: the migrator now converts the `maximalTotalSupply` of shares to the deposit limit with the current total shares and underlying tvl.

Note: The limit can be inflated by donations (i.e. transferring underlying tokens to the vault), hence the limit may be inconsistent with the previous `maximalTotalSupply`. However, there is no obvious incentives to do this.

6.4 Broken Pending Asset Computation

Correctness | Low | Version 1 | Code Corrected

CS-MLW-SYMV-010

The `pendingAssetsOf` computes the pending asset of a user as follow:

```
pendingAssets_ = sharesToClaim.mulDiv(activeStake, activeShares);
```

Note that `sharesToClaim` as well as `activeStake` is in the units of the underlying assets while `activeShares` are in the units of the Symbiotic shares. Ultimately, the computation has no purposeful meaning.

Code corrected:

Now the computation is based on the ratio of the user's shares and the total shares being multiplied with the current withdrawals for the epoch

6.5 Migration

Design | Low | Version 1 | Code Corrected

CS-MLW-SYMV-004

The migration aims to convert migrate the ERC-20 balance accounting to new storage slots. However, the design could be improved:

1. Other storage slots should be cleared (e.g. configurator, name, ...). Future upgrades could potentially break this. Alternatively, the storage slots could be marked as reserved with a static array.

2. Old approvals are lost (and also will remain in storage forever). That, however, might break integrations (in case the integrating contract only approves the token once with `uint256.max`). However, they could be automatically migrated for pairs of addresses.
3. The events emitted during `migrate` might lead to wrong accounting on front-ends. More specifically, during the `_mint` an event will be emitted making it seem like the balance doubled.

Code corrected:

The code has been corrected.

6.6 Whitelisting Might Block Pushing to Symbiotic

Correctness | Low | Version 1 | Code Corrected

CS-MLW-SYMV-005

Note the Symbiotic Vaults might implement a whitelist. In case the Mellow Vault is not whitelisted, the `pushIntoSymbiotic` will revert. However, given the design, it is indirectly implied that `pushIntoSymbiotic` should not revert under such circumstances.

Namely that is due to two reasons:

- No ERC-4626 accounting for such a revert
- The handling of Symbiotic's deposit limits (buffering of funds)

Ultimately, such reverts are improperly handled.

Code corrected:

In Version 3 the code has been updated to return early if the Mellow Vault is not whitelisted.

```
if (symbioticVault.depositWhitelist() && !symbioticVault.isDepositorWhitelisted(this_)) {
    return 0;
}
```

That behavior is in line with the contracts' design.

6.7 Unusual Unchecked Usage

Informational | Version 6 | Code Corrected

CS-MLW-SYMV-016

The migration of token balances to the new storage slots is defined as follows:

```
delete compatStorage._balances[user];
upgradeableStorage._balances[user] += balance;
unchecked {
    compatStorage._totalSupply -= balance;
    upgradeableStorage._totalSupply += balance;
}
```

Note that typically for ERC-20 operations, the balance update is performed in the `unchecked` block while the total supply update is performed in an `checked` block. That is to prevent overflows and trigger a revert in such scenarios (since the total supply is greater than the balance).

Note to trigger an overflow, deposits are required to increase the new total supply slot additionally (since the previous total supply was not overflowing). Ultimately, the storage value for the new total supply could overflow.

However, note that in case the total supply increase would be moved outside of the block and the balance update inside the unchecked block, the migration could revert and DoS user balance migrations.

However, both scenarios are extremely unlikely. Nevertheless, the unchecked block is used in an uncommon fashion.

Code corrected:

The code has been adjusted. Namely, the balance update has been moved to the unchecked block. However, it is of utmost importance that no overflow occurs. The value should be monitored and measures should be taken in case the an overflow could be possible (however, it is very unlikely).

6.8 `migrations` Is Unused

Informational **Version 6** **Code Corrected**

CS-MLW-SYMV-017

The NatSpec specifies that `Migrator.migrations` should indicate the number of staged migrations. However, the variable is never updated and remains unused.

Code corrected:

The variable was removed.

6.9 Gas Optimizations

Informational **Version 1** **Code Corrected**

CS-MLW-SYMV-009

Below possible gas optimizations are listed:

1. Storage slots are computed as follows. Note that the right-most expression is redundant.

```
keccak256(...) & ~bytes32(uint256(0xff)) & ~bytes32(uint256(0xff))
```

2. `withdrawalPause` is checked twice for `withdraw` and `redeem`.

As of compiler version 0.8.22 incrementing the loop counter in the unchecked() block (i.e. in `migrateMultiple`) does not lead to lower gas consumption, as the optimization is already performed by the compiler: https://soliditylang.org/blog/2023/10/25/solidity-0.8.22-release-announcement/

Code corrected:

The code has been adjusted.

6.10 `msg.sender` Used in VaultControlStorage

Informational | Version 1 | Code Corrected

CS-MLW-SYMV-008

`VaultControl` inherits from `ContextUpgradeable`. Thus, it is best practice to use `_msgSender()` in case of future upgrades. Note that `VaultControlStorage` is not inheriting from `ContextUpgradeable`. Thus, it is not possible to use `_msgSender()`. Consequently, the events emitted in `VaultControlStorage` use `msg.sender` which might become inaccurate in case of future upgrades.

Note that similarly `pushIntoSymbiotic` uses `msg.sender` in its event emission.

Code corrected:

The abstract contract inherits now from `ContextUpgradeable` and uses `_msgSender()`. `pushIntoSymbiotic` uses `_msgSender()` now, too.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Potential Checks and Remarks on Parameters and Related Values

Informational **Version 6** **Code Partially Corrected**

CS-MLW-SYMV-015

The checks on the parameters and contracts' configurations could be further improved:

1. It could be validated that the DefaultBondStrategy is at least an operator on the Vault. Otherwise, the admin could specify an arbitrary strategy contract. In case ownership is transferred, the migration could maliciously succeed without processing any withdrawal requests.
2. In defaultBondStrategy, the length of `tokenToData(token)` could be validated. While it is ensured during staging, it is not during migration. However, before migration, the data could have been changed to add more bonds.
3. The underlying token of Symbiotic's vault is not validated against the expected value.
4. The `proxyAdminOwner` is passed as an input, however, it could be retrieved from contract ProxyAdmin. This can ensure the ownership is returned correctly back to the same address.
5. The `vaultAdmin` could be validated to be an actual admin of the vault.
6. The `maximalTotalSupply` could change between staging and migration. However, this is not retrieved and checked in `migrate()`.
7. `depositPause` will always be set to `false`, however, the previous version of vault has a `_isDepositLocked` flag in the configurator, which is not retrieved and checked.
8. Technically, the vault admin could perform malicious actions that could lead to bad outcomes. For example, adding another TVL module to make shares worth more on withdraw. While that is part of the old trust model, in the migrator it seems that this might lead to a scenario where not all withdrawal requests can be processed due to insufficient funds. Hence it could be validated that there is no pending withdrawal requests after calling `processAll()`.
9. Similarly, it could be validated that the size TVL modules is correct (there should be only 2 TVL module: one for ERC-20 Token and the other for Default Collateral).
10. No sanity checks are performed on Symbiotic's vault (as in the factory, see Sanity Checks for reference).

In addition:

11. The `Parameters` in `stageMigration()` contains a parameter `vault`, which is unnecessary as the vault will be the key in the `_migration` mapping.

Code partially corrected:

1. *Corrected*: It is now validated that the default bond strategy has the corresponding rights on the vault, indicating a correct setup. However, it is only done on `migrate`.

2. *Corrected*: The size of `tokenToData(token)` is now always validated to ensure only one bond is used.
3. *Corrected*: The underlying token of Symbiotic's vault is now validated against the expected value.
4. *Corrected*: The owner of the proxy admin is now retrieved on-chain.
5. *Corrected*: The `vaultAdmin` is validated to be an admin of the vault. However, it is only done on `migrate`.
6. *Corrected*: The `maximalTotalSupply` is now retrieved on-chain during `migrate` and converted to `limit`.
7. *Changed*: The `depositPause` and `withdrawalPause` will be set initially to `true` and require admin actions to undo after discussion with Mellow Finance. However, due to the fragility of such migrations, the value seems more appropriate.
8. *Partially corrected*: Note that while the size is validated, modules could be replaced by malicious ones. Similarly, other modules could be replaced.
9. *Corrected*: The size of the TVL modules is ensured to be 2.
10. *Not corrected*

In addition:

11. *Corrected*: The parameter has been removed.

7.2 Sanity Checks

Informational **Version 4** **Acknowledged**

CS-MLW-SYMV-013

While generally, many sanity checks are not required, some of them can help prevent errors. Namely, the codebase could validate following important properties:

1. Check whether default collateral has been deployed through the factory.
2. Check whether the default collateral's underlying matches the vault's underlying.
3. Check whether the used symbiotic vault has been deployed through Symbiotic's factory.
4. Check whether the rewards contract has been deployed through the corresponding factory.
5. **Check that the default collateral token is not equal to the asset of the vault**. Otherwise, double counting can occur.

Acknowledged:

Mellow Finance acknowledged the issue.

7.3 Privileged Read-Only Reentrancy

Informational **Version 1** **Risk Accepted**

CS-MLW-SYMV-007

The `SET_FARM_ROLE` can escalate its privileges to temporarily manipulate the ERC-4626 exchange rate. While it can manipulate it with reentrant tokens, it is also possible to specify a reentrant farm. More specifically, one could setup a farm with the `asset` as the reward token. Then, a simple donation attack could be possible during the `claimRewards`. The rewards would be sent to the Mellow vault to then

increase the balance. Then, any integration with the vault could see a wrong rate. After, the distribution farm could be set to the privileged address so that no losses from the donation are made.

While this requires a privileged address, the role is not intended to hold such power. Ultimately, external protocols integrating with the contract could be attacked.

Risk accepted:

Mellow Finance accepts the risk.

8 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

8.1 Idle Funds Can Be Pushed to Soften Slashing for SymbioticFi Users

Note Version 1

The contract can hold idle funds that are not currently pushed into Symbiotic due to a previously reached deposit limit. It is expected that other users will start to withdraw their funds from SymbioticFi if the Vault experiences a slashing event. This would allow Mellow to push the idle funds into SymbioticFi again. Another SymbioticFi user can decide to withdraw funds from SymbioticFi to lower the limit and push more funds from Mellow into SymbioticFi to soften the impact of an upcoming slashing event on their own position.

The impact of this strategy can be calculated as follows:

Assume the attacker has *x* funds in SymbioticFi, the total funds in SymbioticFi are *X*, and the idle funds in Mellow are *I*. With an upcoming slash of *S*, the attacker can reduce their slashed amount from *x * S / X* to *x * S / (X + min(I, x))* by pushing idle funds into SymbioticFi.

8.2 Limit Considerations

Note Version 1

Governance and users should be aware that Mellow has local limits. Below are some considerations:

- Note that the vaults implement a limit on the managed assets. Note that the limit can be trivially violated by reducing it below the managed assets or by donating large sums to the vault.
- Further, note that this limit ignores the assets managed by the withdrawal queue.

8.3 Migration Considerations

Note Version 1

The migration from the old vaults requires careful planning and execution. Note that not all of the migration is in scope but only the accounting of ERC-20 balances. Below is a list of additional considerations (potentially incomplete) that governance and users should and must consider:

1. It is necessary that the size of underlying tokens is 1.
2. It is further necessary that only one token is held (e.g. default collateral tokens should be converted back).
3. Other changes, that might not be in our knowledge and are not directly evident (e.g. token approvals or other held tokens), should be undone optimally.
4. The contract should be initialized during the upgrade (specifying a delegatecall to the intializer function).
5. The upgrade might be sandwiched which could allow attackers to leverage an arbitrage opportunity between the old share price and the new share price.

6. It is of utmost importance that no withdrawals are pending. Before the upgrade, all withdrawals must be processed or cancelled so that users receive the funds before the upgrade.
7. While the formerly used storage slots are reserved, there could have been storage writes by modules into which a delegatecall had been made. These are not reserved and should be considered for all future upgrades.

Note that we considered only the implementation on Etherscan for address `0xaf108ae0AD8700ac41346aCb620e828c03BB8848` which is used for most of the contracts. The DVstETH vaults cannot be migrated.

Further, note that the full migration would require a more elaborate review.

Follow-up:

Note that a `Migrator` contract has been added that ensures 1, 4 and 6 are handled. 2 is resolved due to the added support for default collateral tokens and the constraints that only 1 token is an underlying and that no other module than the default bond module is used. 3 must be ensured and undone manually if necessary. 5, given the set of vaults to migrate at the time of writing, seems negligible.

Note that integrations might be broken due to the changing interface.

8.4 No Information About Rewards Shared With Farms

Note Version 1

Note that no information about historic or current balances is shared with the distribution farms. Thus, there is no simple and fair possibility of sharing rewards based on on-chain computations.

8.5 Potentially Non-Undoable Ownership Transfer

Note Version 6

Note that the proxy admin ownership needs to be transferred to the migrator. However, in case `Migrator.admin` loses the private keys (or the wallet contract is broken), the ownership can never be transferred back to the original owner since all functions on the `Migrator` can only be called by the `admin`.

8.6 Reaching the Local Limit Can Be Delayed

Note Version 1

Note that Mellow Vaults implement a custom limit on the assets to deposit. However, `pushIntoSymbiotic` does not take that limit into account, which means that it will take longer to reach the limit if it is lower than the currently managed limit.

Consider the following example:

1. The local limit is 10 and Symbiotic's limit is 10. 20 assets are managed (10 buffered and 10 in Symbiotic).
2. Now the local limit is reduced to 5 and Symbiotic increases its limit to 20.
3. `pushIntoSymbiotic` pushes 10 assets to Symbiotic.

4. Withdrawal requests cannot occur instantly as there is no buffer. Thus, reaching the limit can be delayed purposefully.

Governance and users should be aware of such possibilities

8.7 Rounding in Symbiotic Vault

Note Version 1

The `MellowSymbioticVault` queries its active balance from Symbiotic via `symbioticVault().activeBalanceOf` to calculate the total assets in the Vault. Note that the Symbiotic Vault stores the active balance in shares, converts it to assets, and rounds down the result.

This rounding can lead to undesired effects in the Mellow Vault. Specifically, the `_convertToShares()` function uses the rounded-down value of `totalAssets` in the denominator. As a result, more shares are minted from `deposit`, and more assets are withdrawn by `withdraw`.