PUBLIC

Code Assessment

of the Wrapped M Token Smart Contracts

August 13, 2024

Produced for

M^ZERO LABS_

by

CHAINSECURITY

Contents

1 Executive Summary

Dear all,

Thank you for trusting us to help M^ZERO Labs with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Wrapped M Token according to Scope to support you in forming an opinion on their security risks.

M^ZERO Labs implements an upgradable, non-rebasing wrapper for the `M` token, supporting yield accrual while respecting the original whitelist of earners.

The most critical subjects covered in our audit are asset solvency, functional correctness and security. Security regarding all the aforementioned subjects is high.

The general subjects covered are documentation, gas efficiency and the integration of the wrapper into the existing system. All reported issues have been addressed in the latest version of the codebase.

In summary, we find that the codebase provides a good 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	3
• Code Corrected	3
Low-Severity Findings	4
• Code Corrected	3
• 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 Wrapped M Token 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	16 Jul 2024	0dd7b4553009e3d87298a2f9e6aec2a89ba308ad	Initial version
2	05 Aug 2024	39fe8f7a7501042432d1e131865ea2488620011c	Version with fixes
3	08 Aug 2024	88c54bb06728e58d9e061997297876687dd9fa55	Further changes
4	13 Aug 2024	55896c2f37a13a39fae933d52a2b6687ff4c9408	Final Version

For the solidity smart contracts, the compiler version `0.8.23` and EVM version `shanghai` were chosen.

The following contracts are in scope of the review:

```
Migratable.sol
WrappedMToken.sol
Proxy.sol
libs:
    IndexingMath.sol
interfaces:
    IMigratable.sol
    IRegistrarLike.sol
    IWrappedMToken.sol
    IMTokenLike.sol
```

2.1.1 Excluded from scope

Any contracts that are not explicitly listed above are out of the scope of this review. The WrappedMToken is an implementation contract intended for use behind the Proxy. Direct interaction with the implementation is not a valid use case and is out of scope.

2.2 System Overview

This system overview describes the initially received version (**Version 1**) 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.

M^ZERO Labs implements a wrapper for the rebalancing `M` token to facilitate interaction with various decentralized finance systems. The wrapper is subject to the original whitelist of eligible earners. The token implementation is deployed behind an upgradable proxy and the governance can override yield recipients for addresses.

2.2.1 Proxy

Minimal EIP-1967 type proxy: All calls are forwarded using delegatecall to the implementation stored at storage slot `keccak256('eip1967.proxy.implementation') - 1`.

2.2.2 WrappedMToken

The wrapped `M` token is ERC-20 compliant with 6 decimals and supports ERC-3009 (transfer of fungible assets via signed authorization, signatures are compliant to ERC-712).

The contract is operational after deployment behind the proxy. The wrapper should be added by the governance into the earners list before the earning can be enabled:

`enableEarning()`: Registers the contract to start earning in the `M` token, stores the current `` `M `` index. `disableEarning()`: Halts earning in the wrapper. Important to be executed should the wrapper contract ever be excluded from earning in the `M` token.

`M` token holders can wrap their tokens using `wrap()` and receive wrapped `M` tokens in exchange. These tokens can be unwrapped using `unwrap()`. Rewards (accrued only if started explicitly) are claimed automatically prior to changes of balance, or if triggered directly using `claimFor()`. Governance can override the recipient of yield for any address, in this case the claimed yield will be transferred to the specified recipient. For an address holding wrapped `wM` tokens to start earning, this must be activated explicitly:

`startEarningFor()`: Registers the address to start earning. Only possible if earning is globally enabled and the account is eligible to earn. Switches the balance account for this account to the earning mode.

`stopEarningFor()`: Stops an address from earning. Can be called permissionlessly on the condition that the account has been removed from the earners list. The function claims the outstanding yield of this account up until the time of execution, and then converts the balance accounting to non-earning mode. This function is important to be executed when conditions are fulfilled, otherwise the account will continue to earn yield until being stopped.

`claimExcess()`: The wrapper contract earns yield for its full `M` token balance, however since not all wrapped token holders may earn there can be a surplus. This function calculates and transfers the surplus to a predefined address. The current implementation transfers the surplus to the distribution vault, however, this behavior might change in future versions.

The balances are stored in different formats depending on whether the account has earning activated or not. For accounts without earning account, their balance is stored in amounts of `wM` token. For earning-enabled accounts, the balance is stored in the form of index and principal. The conversion is done automatically. The contract keeps track of the amount of earning/non-earning tokens.

Several view functions exist to inspect the state of the contract. The main view functions:

`balanceOf()`: Returns the stored balance of an address and does not account its unrealized yield. `totalSupply()`: Returns the approximated sum of `wM` balances for all accounts in earning and non-earning modes. This function overestimates the supply of accounts in the earning mode and does not consider the unrealized yield. `accruedYieldOf()`: Returns the pending yield of an account. This yield is realized if function `claimFor()` is called or an operation that changes the account's balance is triggered, e.g., `transfer()`, `wrap()`, or `unwrap()`.

2.2.3 Migration

The proxy scheme used requires the implementation to support upgradability. This contract offers two different methods to trigger an upgrade:

- `migrate(address migrator_)`: Allows the `migrationAdmin` to initiate a migration.

- `migrate()`: allows anyone to initiate the migration if a migrator address is set in the registrar by the governance.

During migration, the proxy executes a delegatecall to the migrator contract which must set the new implementation address at storage slot `keccak256('eip1967.proxy.implementation') - 1`. The migrator can implement additional logic to ensure compatibility between different versions. Each wrapper implementation must use a unique key prefix to avoid collisions with other keys in the Registrar or with old migration keys. The migrator contract and the new implementation should be carefully evaluated before the upgrade and the storage layout of implementations should match.

2.2.4 Trust Model & Roles

Any interaction with the WrappedMToken is expected to happen via the proxy.

The following role exist within this contract:

MigrationAdmin: Fully trusted role within this implementation, allows upgrades bypassing the governance. If this account becomes malicious or gets compromised, it can set arbitrary implementations for the proxy and drain all tokens held by the wrapper.

Registrar: Source of truth for various information such as the earners list or whether the whitelist is to be ignored, the migrator role and claim overrides. Fully trusted, expected to be the TTG Registrar contract of the Mzero system.

Mtoken: Rebasing token this wrapper is for. Expected to work according to specifications, including rebasing correctly according to the index report, and valid values returned only. Expected to be the Mzero token.

Vault: Recipient of the surplus yield, set in the constructor, queried from the registrar.

Users: untrusted, can call public functions including functions allowing to wrap/unwrap `M` tokens.

2.2.5 Changes in Version 2:

- Two new functions `wrap(address)` and `unwrap(address)` were introduced that allow users to wrap their whole balance in `M` token, or unwrap their whole balance in `wM` token.
- The struct `Account` is used for internal accounting of user balances instead of the type `BalanceInfo`.

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 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	0
Low-Severity Findings	1

- Discrepancy of Yield When Earning Is Disabled Risk Accepted

5.1 Discrepancy of Yield When Earning Is Disabled

Correctness Low Version 1 Risk Accepted

CS-MZEROWT-002

If the governance removes the wrapper contract from the earners list, anyone can trigger a call to stop the wrapper from earning more yield in `M` token in two ways:

1. Call `disableEarning()` in the wrapper contract.
2. Call `stopEarning(wrapperAddr)` in `M` token contract.

The first option is the correct way to stop the wrapper earning more yield and ensure that the accounting in the wrapper is correct. However, since anyone can call `stopEarning()` in the `M` token contract, the second option is also possible. In this case, the accounting of the wrapper will be off depending on the delay that `disableEarning()` is executed. Theoretically, any delay creates solvency issues for the wrapper as the yield distributed to `wM` holders is larger than the yield earned by the wrapper, hence last users cannot unwrap their tokens.

Rick accepted:

M^ZERO Labs has acknowledged the issue but has decided to keep the respective codebase unchanged. The function `disableEarning()` in the wrapper contract is permissionless and can be triggered by anyone to ensure that the deviation on the accounting is not significant.

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the 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	3

- Excess Amount Is Overestimated Code Corrected
- Effects of Roundings in Wrapped M Code Corrected
- Name and Symbol Are Not Properly Set Code Corrected

Low-Severity Findings	3

- Accumulated Dust Amounts for Earners Are Considered as Excess Code Corrected
- Rounding Errors on M Token Transfers Code Corrected
- Specifications About BalanceInfo Encoding Code Corrected

Informational Findings	4

- Gas Optimizations Code Corrected
- Order of Events Code Corrected
- Foundry.toml: Mismatch of Solidity and EVM Version Code Corrected
- Unwrapping Might Leave Dust Amounts for Earners Code Corrected

6.1 Excess Amount Is Overestimated

Correctness Medium Version 2 Code Corrected

CS-MZEROWT-001

The function `claimExcess()` transfers to the distribution vault the excess amount of `M` tokens that the wrapper contract holds. The excess amount must be underestimated to ensure that the wrapper contract is always solvent.

However, the function `excess()` computes `earmarked_` as:

```
uint240 earmarked_ = totalNonEarningSupply + _projectedEarningSupply(currentIndex());
```

and the function `_projectedEarningSupply()` rounds down the estimated supply for earners if all yield is claimed:

```
function _projectedEarningSupply(uint128) ... {
    return IndexingMath.getPresentAmountRoundedDown(_principalOfTotalEarningSupply, currentIndex_);
}
```

Furthermore, the principal is rounded down when earners wrap their tokens or a non-earner switches to earner mode:

```
function _addTotalEarningSupply(uint240 amount_, uint128 currentIndex_) internal {
    unchecked {
        ...
        _principalOfTotalEarningSupply += IndexingMath.getPrincipalAmountRoundedDown(amount_, currentIndex_);
    }
}
```

The rounding errors accumulate over time and result in excess amounts being larger than intended.

Note that the function `_subtractTotalEarningSupply()` also rounds down the principal being removed when earners burn or transfer out their `wM` tokens, however it does not offset the rounding errors described above.

Acknowledged:

The codebase in Version 3 has been revised to round down the amount of `M` tokens that the wrapper transfers out, which compensate for the rounding errors in the excess amount. When unwrapping `wM` tokens and transferring the excess tokens to the vault, the transferred amounts are rounded down in the function `_getSafeTransferableM()`.

M^ZERO Labs is aware that the rounding errors are still possible and the excess amount is slightly overestimated.

Code corrected:

In Version 4 `_addTotalEarningSupply()` rounds up the principal delta before adding it to `principalOfTotalEarningSupply`. This will lead to a slight over-estimate of `projectedEarningSupply`, which can potentially lead to unresolvable dust of `principalOfTotalEarningSupply`, but guarantees the safety of underestimating of excess.

6.2 Effects of Roundings in Wrapped M

Design Medium Version 1 Code Corrected

CS-MZEROWT-008

This issue was initially reported as a Note but was upgraded to an Issue after M^ZERO Labs independently discovered during testing that an intended use case breaks.

Similar to `M` token, the wrapper contract uses rounding up or down when operating on balances of earning accounts. This has the following effects:

- for earners, the wrapping of `X` `M` tokens will effectively mint `Y` `wM` tokens, with $Y \leq X$.
- for earners with initial token balances `A` and `B`, in-kind transfers of `X` tokens will result in updated balances `A'` and `B'`, with $A' \leq A - X$ and $B' \leq B + X$, and $A + B \geq A' + B'$.
- for out-of-kind transfers of `X` tokens and initial balances `A` and `B`, the updated balances `A'` and `B'` yield $A + B \geq A' + B'$.
- as soon as earners are active in the system, the invariant:

  $$\sum_{i \in nonEarningBalances} balanceOf(i) = totalSupply$$

  is relaxed to

  $$\sum_{i \in nonEarners} balanceOf(i) + \sum_{j \in earners} balanceOf(j) \leq totalSupply$$

- when non earners become earners, their balances go from `A` to `A'`, with $A' \leq A$.
- when earners stop earning, their balances go from `A` to `A'`, with $A' \leq A$.

Functions `transfer()`, `transferFrom()`, `wrap()` and `unwrap()` update balances with amounts that might be different from those specified by callers due to rounding errors. These specifics of token `wM` should be taken into consideration when integrating with 3rd-party protocols.

Code corrected:

The implementation of the wrapper has been simplified and now keeps track of exact balances, resolving rounding issues when querying balances. Notably, transfers of tokens now result in the expected balance increase, which is crucial for DeFi protocols that check actual balance increases.

6.3 Name and Symbol Are Not Properly Set

Correctness Medium Version 1 Code Corrected

CS-MZEROWT-012

The wrapper contract of the `M` token is deployed behind a proxy that uses `WrappedMToken` as its implementation. However, the constructor of the contract `WrappedMToken` inherits `ERC20Extended` which sets state variables:

```
constructor(address mToken_, address migrationAdmin_)
    ERC20Extended("WrappedM by M^0", "wM", 6) { ... }
```

The constructor of `ERC20Extended` sets the state variable `symbol` to the input `"vM"`. Similarly, the contract `ERC712Extended` stores `_name` as state variable.

Note that any state variable set in the constructor of the implementation contract is stored in the storage of the implementation itself instead of the proxy storage. Therefore, the storage slots for `symbol` and `_name` in the proxy remain empty and the external functions `symbol()` and `name()` do not return the intended values. This creates integration issues with wallets and third-party apps.

Code corrected:

The project now uses a newer version of the library `common`. The state variables `symbol` and `_name` are now stored as immutables of type `bytes32`.

6.4 Accumulated Dust Amounts for Earners Are Considered as Excess

Design Low Version 3 Code Corrected

CS-MZEROWT-014

Function `_subtractTotalEarningSupply()` has been revised in Version 3 and now it's is possible that `_principalOfTotalEarningSupply` goes to `0`, while the variable `totalEarningSupply` remains non-zero due to rounding errors. If such a scenario happens, the projected earnings become 0 (`_principalOfTotalEarningSupply == 0`), hence any `M` token accounted by the wrapper for earners would be transferred to the vault by `claimExcess()`.

Code corrected:

As final fix for Excess amount is overestimated, in Version 4, `_addTotalEarningSupply()` rounds up the principal delta before adding it to `principalOfTotalEarningSupply`. This leads to a slight over-estimate of `projectedEarningSupply`. As a result, it's no longer possible for `_principalOfTotalEarningSupply``to go to zero while ``totalEarningSupply` remains non-zero, hence the issue described above no longer exists.

6.5 Rounding Errors on M Token Transfers

Security Low Version 1 Code Corrected

CS-MZEROWT-011

The `wrap()` function mints to users the amount of `wM` tokens specified in the input parameter, however, the wrapper might receive less tokens due to the roundings that happen in `mToken.transferFrom()`. This creates solvency issues for the wrapper in extreme scenarios.

Consider the following scenario: a non-earner user wraps `10 M` tokens, he receives `10 wM` tokens, but the `M` balance of the wrapper might increase with `9.999999` instead of `10`. The same user can unwrap `10 wM` tokens that will decrease the `M` balance of wrapper by `10`, which causes a negative net flow for the wrapper.

Code corrected:

The `wrap()` function has been revised to mint to users an amount of `wM` that matches the actual increase of the wrapper's balance in `M` token:

```
function _wrap(address account_, address recipient_, uint240 amount_) internal {
    uint256 startingBalance_ = IMTokenLike(mToken).balanceOf(address(this));
    ...
    IMTokenLike(mToken).transferFrom(account_, address(this), amount_);
    ...
    _mint(recipient_, UIntMath.safe240(IMTokenLike(mToken).balanceOf(address(this)) - startingBalance_));
}
```

6.6 Specifications About BalanceInfo Encoding

Correctness Low Version 1 Code Corrected

CS-MZEROWT-006

The inline comments in the internal function `_getBalanceInfo` state that the index value in a `BalanceInfo` type are in the 128 bits that are next to the earning flag:

```
// If the account is an earner, the next 128 bits are the index of the last
    interaction and the last 112 bits
```

Similarly, the inline comments in `_setBalanceInfo()` state that index is stored in the 128 bits next to the earning flag:

```
//   - If the account is an earner:
//     - The most significant 8 bits is a flag for whether the account is earning
          or not.
//     - The next 128 bits are the index of the last interaction
```

However, the implementation of `BalanceInfo` type has a gap of 8 bits between the earning flag and the index. `BalanceInfo` type encodes the following values:

```
flag (8 bits) | gap (8 bits) | last_index (128 bits) | principal (112 bits)
or
flag (8 bits) | gap (8 bits) | amount (240 bits)
```

Code corrected:

The type `BalanceInfo` has been replaced with the struct `Account` in the codebase (Version 2), therefore the respective inline natspec are not present anymore.

6.7 Foundry.toml: Mismatch of Solidity and EVM Version

Informational | Version 1 | Code Corrected

CS-MZEROWT-013

In Foundry.toml, Solidity 0.8.23 and evm_version `cancun` are specified. However support for `cancun` was only introduced in Solidity 0.8.24. Compilation succeeds, the resulting build info shows the code was compiled for EVM version `shanghai`.

Code corrected:

EVM version has been changed to `shanghai` in `foundry.toml`.

6.8 Gas Optimizations

Informational | Version 1 | Code Corrected

CS-MZEROWT-007

- The function `startEarningFor` can be more gas efficient if the internal function `_currentMIndex()` is used instead of `currentIndex()` which checks redundantly if the earning is enabled.
- The function `_getBalanceInfo()` returns the variable `isEarning` if the account is not an earner. Using `false` hardcoded is more gas efficient.
- Duplicate check on the sender's balance in `_transfer()`: When the earn statuses differ, the balance check is done in either `_subtractEarningAmount()` or `_subtractNonEarningAmount()`, making the check redundant in these cases. This check is necessary only when both the sender and receiver share the same earn status and their balances are updated directly.

Code corrected:

All points have been addressed or are no longer relevant in the new code.

6.9 Order of Events

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

CS-MZEROWT-009

The order of events emitted by the wrapper contract is sometimes inconsistent and the events might be misinterpret by third party apps or integrators that parse such events.

Function `wrap` emits events related to minting of new `wM` tokens and afterwards emits the event transfer of `M` tokens into the contract. So, there is an intermediary state during which the wrapper mints the new tokens although it has not received a transfer.

Function `_mint` emits the event `Transfer` with the user-specified input `amount_` as the amount of `wM` tokens being minted. However, due to rounding errors for earning accounts, the real minted about might be lower.

The function `_claim` which is triggered whenever the balance of an account is updated, emits an event similar to minting when yield is realized. If the recipient for yield claiming is set, a transfer from the account to the recipient is emitted. This might be confusing for DeFi pools such as Uniswap as these additional transfer events will be emitted whenever a swap happens.

Code corrected:

Functions `wrap` and `unwrap` have been revised to emit the events in the correct order. Due to the simplified balance accounting which tracks exact balances, the function `_mint` now emits the correct amount unaffected by rounding in the event.

6.10 Unwrapping Might Leave Dust Amounts for Earners

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

CS-MZEROWT-010

Holders of `wM` token that earn yield have no easy way to unwrap their whole balance and leave no dust in the contract. While smart contracts can perform `claimFor()` first and then unwrap everything, EOAs have no way to unwrap their whole balance as yield depends on the exact time the transaction is included in a block.

Code corrected:

A new version of the `unwrap()` function has been implemented in **Version 2** that enables a user to unwrap their whole balance and their yield:

```
function unwrap(address recipient_) external {
    _unwrap(msg.sender, recipient_, uint240(balanceWithYieldOf(msg.sender)));
}
```

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 No Claimed Event on 0 Yield

Informational **Version 2**

CS-MZEROWT-003

`_claim()` may update an accounts `_lastIndex` but return early without emitting the `Claimed` event if `_getAccruedYield()` resulted in 0, e.g. due to rounding.

7.2 Re-enabling of Earning in Future Versions

Informational **Version 1**

CS-MZEROWT-004

The current implementation of `WrappedMToken` allows enabling of earnings for the wrapper only once, however, this behavior might change in the future versions as noted:

```
// NOTE: This is a temporary measure to prevent re-enabling earning after it has
    been disabled.
//       This line will be removed in the future.
if (wasEarningEnabled()) revert EarningCannotBeReenabled();
```

It must be emphasized that this requires further changes in the codebase, this line cannot simply be removed. For example, the internal function `_lastDisableEarningIndex()` always returns the index of the first event where earnings are disabled. Additionally, re-enabling earnings breaks the accounting for users who do not claim their yield after earnings are disabled.

7.3 Wrapper Contract Can Earn WrappedM Yield

Informational **Version 1**

CS-MZEROWT-005

One can donate `vM` to the wrapper contract and trigger the start of earnings, allowing yield to accumulate over time. Currently, the wrapper contract does not rely on its own balance, so this is not problematic. Governance could redirect the yield to an arbitrary address by overriding the recipient for the wrapper's yield. Since the current implementation of the wrapper contract lacks functionality to transfer out such tokens, they would be locked.

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 Migrator Prefix Must Not Collide With Other Keys

Note **Version 1**

The WrappedMToken can be upgraded to a new implementation if there is a value in registrar for a key that matches a hash of a constant prefix and the wrapper address:

```
function _getMigrator() internal view override returns (address migrator_) {
    ...
        uint256(IRegistrarLike(registrar).get(
            keccak256(abi.encode(_MIGRATOR_V1_PREFIX, address(this)))))
    ...
}
```

Although unlikely, it is possible that the prefix can be chosen such that it collides with another key in the registrar. For instance, a prefix set to `"wm_claim_override_recipient"` or `"IN_LISTearners"` would trick the function `_getMigrator()` to read a value from a key that is meant to store another value and not the implementation address.

8.2 Removed Earners Must Be Stopped Earning Yield

Note **Version 1**

When governance removes an account from the eligible earning list, the holder of `wM` continues to earn yield until `stopEarningFor()` is called on the wrapper contract for that account. This is similar to the `M` token, where an account removed from the eligible earning list must also be stopped from earning additional yield.

8.3 Storage Layout of the Proxy

Note **Version 1**

The wrapper contract is deployed behind a proxy scheme and implementation upgrades are possible. The storage layout of the new implementation should be carefully checked to be aligned with the existing implementation to ensure compatibility. Note that `WrappedMToken` inherits multiple contracts that declare their own variables.