// Smart Contract Security Assessment 10.25.2024 - 10.28.2024

Safety Module

Moonwell

HALBORN

Safety Module - Moonwell

Prepared by: HALBORN

Last Updated 10/31/2024

Date of Engagement by: October 25th, 2024 - October 28th, 2024

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
7	0	0	0	2	5

TABLE OF CONTENTS

1. Introduction

`Moonwell` engaged `Halborn` to conduct a security assessment on their smart contracts **beginning on October 25th and ending on October 28th**. The security assessment was scoped to the smart contracts provided in the moonwell-fi/moonwell-contracts-v2 GitHub repository. Commit hash and further details can be found in the Scope section of this report.

2. Assessment Summary

`Halborn` was provided 2 days for the engagement, and assigned one full-time security engineer to review the security of the smart contracts in scope. The engineer is a blockchain and smart contract security expert with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the smart contracts.
- Ensure that smart contract functionality operates as intended.

In summary, `Halborn` identified informational issues, that were mostly addressed by the `Moonwell team`. The main findings were:

- Missing `address(0)` checks.
- Missing reentrancy lock.
- Unused imports.
- Unchecked return value.
- Remove EIP20 non-compliant functions.
- Use `1e18` constant for Gas optimization.
- Use `++i` instead of `i++` in loops for Gas optimization.

The current report consolidates the **two (2)** Smart Contract Security assessments that were performed in the provided smart contracts in-scope, regarding the `Safety Module` component.

- **Assessment History:**

COMMIT HASH	START DATE
827bbd043ef7b876f06d81044ce17052d4bd820e	February 8th, 2021
9bae493e5df757f5f222ff8884f3b712bd7e4688	October 25th, 2024

3. Test Approach And Methodology

`Halborn` performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (`solgraph`).
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Static Analysis of security for scoped contract, and imported functions (`slither`).
- Testnet deployment (`Foundry`).

4. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4

SEVERITY	SCORE VALUE RANGE
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: moonwell-contracts-v2

(b) Assessed Commit ID: 9bae493

(c) Items in scope:

- src/IStakedWell.sol
- src/governance/multichain/MultichainGovernorDeploy.sol
- src/proposals/mips/mip-m40/mip-m40.sol
- src/stkWell/moonbeam/libraries/DistributionTypes.sol
- src/stkWell/moonbeam/interfaces/IDistributionManager.sol
- src/stkWell/moonbeam/DistributionManager.sol
- src/stkWell/moonbeam/StakedToken.sol
- src/stkWell/moonbeam/StakedWellMoonbeam.sol
- src/stkWell/moonbeam/OpenZeppelin/ReentrancyGuardUpgradeable.sol
- src/stkWell/moonbeam/libraries/ERC20.sol
- src/stkWell/moonbeam/libraries/ERC20WithSnapshot.sol

Out-of-Scope:

FILES AND REPOSITORY

(a) Repository: contracts-open-source

(b) Assessed Commit ID: 827bbd0

(c) Items in scope:

- EcosystemReserve.sol
- StakedToken.sol

Out-of-Scope:

REMEDIATION COMMIT ID:

- e23657c

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	2	5

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
MISSING ADDRESS(0) CHECKS	LOW	SOLVED - 03/31/2022
MISSING REENTRANCY LOCK	LOW	SOLVED - 03/31/2022
UNUSED IMPORTS	INFORMATIONAL	ACKNOWLEDGED - 10/31/2024
UNCHECKED RETURN VALUE	INFORMATIONAL	ACKNOWLEDGED - 10/31/2024
REMOVE EIP20 NON-COMPLIANT FUNCTIONS	INFORMATIONAL	ACKNOWLEDGED - 10/31/2024
USE 1E18 CONSTANT FOR GAS OPTIMIZATION	INFORMATIONAL	SOLVED - 03/31/2022

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
USE ++I INSTEAD OF I++ IN LOOPS FOR GAS OPTIMIZATION	INFORMATIONAL	SOLVED - 03/31/2022

7. FINDINGS & TECH DETAILS

7.1 MISSING ADDRESS(0) CHECKS

// LOW

Description

Safety Module contracts have address fields on multiple functions. These functions have missing address validations. Every address should be validated and checked that is different from zero. This is also considered a best practice.

During the test, it has seen some of these inputs are not protected against using the `address(0)` as the target address.

– StakedToken.sol

```
44      function __StakedToken_init(
45          IERC20 stakedToken,
46          IERC20 rewardToken,
47          uint256 cooldownSeconds,
48          uint256 unstakeWindow,
49          address rewardsVault,
50          address emissionManager,
51          uint128 distributionDuration,
52          string memory name,
53          string memory symbol,
54          uint8 decimals,
55          address governance
56      ) internal initializer {
57          __ReentrancyGuard_init();
58          __ERC20_init_unchained(name, symbol, decimals);
59          __DistributionManager_init_unchained(emissionManager, distributio
60          __StakedToken_init_unchained(
61              stakedToken,
62              rewardToken,
63              cooldownSeconds,
64              unstakeWindow,
65              rewardsVault,
66              governance
67          );
68      }
```

– EcosystemReserve.sol

```
32      function initialize(address reserveController) external initializer {
33          require(reserveController != address(0), "ZERO_ADDRESS");
34          __ReentrancyGuard_init();
35          _setFundsAdmin(reserveController);
36      }
```

– EcosystemReserve.sol

```
54      function setFundsAdmin(address admin) external override onlyFundsAdmi
55          _setFundsAdmin(admin);
56      }
```

BVSS

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:C (3.1)

Recommendation

It is recommended to validate that every address input is different from zero.

Remediation

SOLVED: The **Moonwell team** has solved this issue as recommended. The commit hash for reference is `e23657c5fbeb12c7393fa49da6f350dc0bd5114e`.

Remediation Hash

https://github.com/moonwell-fi/contracts-open-source/commit/e23657c5fbeb12c7393fa49da6f350dc0bd5114e

7.2 MISSING REENTRANCY LOCK

// LOW

Description

To protect against cross-function re-entrancy attacks, it may be necessary to use a **mutex**. By using this lock, an attacker can no longer exploit the withdrawal function with a recursive call.
OpenZeppelin has its own mutex implementation called `ReentrancyGuard` which provides a modifier to any function called `nonReentrant` that guards the function with a mutex against re-entrancy attacks.

– EcosystemReserve.sol

```
46      function transfer(
47          IERC20 token,
48          address recipient,
49          uint256 amount
50      ) external override onlyFundsAdmin {
51          token.transfer(recipient, amount);
52      }
```

– StakedToken.sol

```
87      function stake(address onBehalfOf, uint256 amount) external override
88          require(amount != 0, 'INVALID_ZERO_AMOUNT');
89          require(onBehalfOf != address(0), 'STAKE_ZERO_ADDRESS');
90          uint256 balanceOfUser = balanceOf(onBehalfOf);
91
92          uint256 accruedRewards = _updateUserAssetInternal(
93              onBehalfOf,
94              address(this),
95              balanceOfUser,
96              totalSupply()
97          );
98          if (accruedRewards != 0) {
99              emit RewardsAccrued(onBehalfOf, accruedRewards);
100             stakerRewardsToClaim[onBehalfOf] = stakerRewardsToClaim[onBeh
101         }
102
103         stakersCooldowns[onBehalfOf] = getNextCooldownTimestamp(0, amount
104
```

```
        _mint(onBehalfOf, amount);
        IERC20(STAKED_TOKEN).safeTransferFrom(msg.sender, address(this),

        emit Staked(msg.sender, onBehalfOf, amount);
    }
```

- StakedToken.sol

```
    function redeem(address to, uint256 amount) external override {
        require(amount != 0, 'INVALID_ZERO_AMOUNT');
        require(to != address(0), 'REDEEM_ZERO_ADDRESS');
        //solium-disable-next-line
        uint256 cooldownStartTimestamp = stakersCooldowns[msg.sender];
        require(
            block.timestamp > cooldownStartTimestamp.add(COOLDOWN_SECONDS
            'INSUFFICIENT_COOLDOWN'
        );
        require(
            block.timestamp.sub(cooldownStartTimestamp.add(COOLDOWN_SECON
            'UNSTAKE_WINDOW_FINISHED'
        );
        uint256 balanceOfMessageSender = balanceOf(msg.sender);

        uint256 amountToRedeem = (amount > balanceOfMessageSender) ? bala

        _updateCurrentUnclaimedRewards(msg.sender, balanceOfMessageSender

        _burn(msg.sender, amountToRedeem);

        if (balanceOfMessageSender.sub(amountToRedeem) == 0) {
            stakersCooldowns[msg.sender] = 0;
        }

        IERC20(STAKED_TOKEN).safeTransfer(to, amountToRedeem);

        emit Redeem(msg.sender, to, amountToRedeem);
    }
```

- StakedToken.sol

```
163     function claimRewards(address to, uint256 amount) external override {
164         uint256 newTotalRewards = _updateCurrentUnclaimedRewards(
165             msg.sender,
166             balanceOf(msg.sender),
167             false
168         );
169         uint256 amountToClaim = (amount == type(uint256).max) ? newTotalR
170 
171         stakerRewardsToClaim[msg.sender] = newTotalRewards.sub(amountToCl
172 
173         IERC20(REWARD_TOKEN).safeTransferFrom(REWARDS_VAULT, to, amountTo
174 
175         emit RewardsClaimed(msg.sender, to, amountToClaim);
176     }
```

BVSS

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:C (3.1)

Recommendation

It is recommended to add OpenZeppelin `ReentrancyGuard` library to the project and use the `nonReentrant` modifier to avoid introducing future re-entrancy vulnerabilities.

Remediation

SOLVED: The **Moonwell team** has solved this issue as recommended. The commit hash for reference is `e23657c5fbeb12c7393fa49da6f350dc0bd5114e`.

Remediation Hash

https://github.com/moonwell-fi/contracts-open-source/commit/e23657c5fbeb12c7393fa49da6f350dc0bd5114e

7.3 UNUSED IMPORTS

// INFORMATIONAL

Description

It was identified unused imports in the `StakedToken` smart contract and `IDistributionManager` interface.

- `moonwell-contracts-v2/src/stkWell/StakedToken.sol`

```
7 | import {IEcosystemReserve} from "./interfaces/IEcosystemReserve.sol";
```

- `moonwell-contracts-v2/src/stkWell/moonbeam/interfaces/IDistributionManager.sol`

```
5 | import {DistributionTypes} from "../libraries/DistributionTypes.sol";
```

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:P/S:C (0.0)

Recommendation

It is recommended to remove unused imports in order to enhance code readability and avoid unnecessary complexity.

Remediation

ACKNOWLEDGED: The **Moonwell team** has acknowledged this finding.

7.4 UNCHECKED RETURN VALUE

// INFORMATIONAL

Description

In the `StakedToken.sol` smart contract, during the execution of the `stake` and `_transfer` functions, the returned value from the call to the `_updateCurrentUnclaimedRewards` is not checked.

– moonwell-contracts-v2/src/stkWell/moonbeam/StakedToken.sol

```
165         _updateCurrentUnclaimedRewards(
166             msg.sender,
167             balanceOfMessageSender,
168             true
169         );
```

```
228     function _transfer(
229         address from,
230         address to,
231         uint256 amount
232     ) internal override {
233         uint256 balanceOfFrom = balanceOf(from);
234         // Sender
235         _updateCurrentUnclaimedRewards(from, balanceOfFrom, true);
236 
237         // Recipient
238         if (from != to) {
239             uint256 balanceOfTo = balanceOf(to);
240             _updateCurrentUnclaimedRewards(to, balanceOfTo, true);
241 
242             uint256 previousSenderCooldown = stakersCooldowns[from];
243             stakersCooldowns[to] = getNextCooldownTimestamp(
244                 previousSenderCooldown,
245                 amount,
246                 to,
247                 balanceOfTo
248             );
249             // if cooldown was set and whole balance of sender was transf
250             if (balanceOfFrom == amount && previousSenderCooldown != 0) {
251                 stakersCooldowns[from] = 0;
```

```
252                 }
253             }
254
255             super._transfer(from, to, amount);
256         }
```

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:P/S:C (0.0)

Recommendation

It is recommended to check the returned value from the `_updateCurrentUnclaimedRewards` function call, attributing a local variable to this value and checking it against undesired values, if any.

Remediation

ACKNOWLEDGED: The **Moonwell team** has acknowledged this finding.

7.5 REMOVE EIP20 NON-COMPLIANT FUNCTIONS

// INFORMATIONAL

Description

The `increaseAllowance` and `decreaseAllowance` functions are non-compliant with the `EIP20` standard and are therefore not recommended for use, unless strictly required by the smart contract's functionality. It is well-known in the community that these functions can enable bad actors to execute less traditional phishing attacks, as opposed to the more common `approve` or `permit` methods.

– moonwell-contracts-v2/src/stkWell/moonbeam/libraries/ERC20.sol

```
139     function increaseAllowance(
140         address spender,
141         uint256 addedValue
142     ) public virtual returns (bool) {
143         _approve(
144             _msgSender(),
145             spender,
146             _allowances[_msgSender()][spender].add(addedValue)
147         );
148         return true;
149     }
```

```
157     function decreaseAllowance(
158         address spender,
159         uint256 subtractedValue
160     ) public virtual returns (bool) {
161         _approve(
162             _msgSender(),
163             spender,
164             _allowances[_msgSender()][spender].sub(
165                 subtractedValue,
166                 "ERC20: decreased allowance below zero"
167             )
168         );
169         return true;
170     }
```

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:P/S:C (0.0)

Recommendation

To address this issue, it is recommended to avoid using the `increaseAllowance` and `decreaseAllowance` functions. Instead, rely on the standard `approve` function for setting allowances. If dynamic allowance adjustments are necessary, ensure that the contract logic is thoroughly reviewed and tested to prevent potential security vulnerabilities.
Removing `increaseAllowance` and `decreaseAllowance` from the code-base is also recommended.

Remediation

ACKNOWLEDGED: The **Moonwell team** has acknowledged this finding.

7.6 USE 1E18 CONSTANT FOR GAS OPTIMIZATION

// INFORMATIONAL

Description

In Solidity, exponentiation operation (`**`) costs up to 10 gas. It is possible to consume less gas to calculate token prices if `DECIMAL` variable is fixed.

It was identified in the `DistributionManager` contract the usage of `**` instead of `1e18`.

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:P/S:C (0.0)

Recommendation

It is recommended to use `1e18` instead of (`10 ** 18`) on price calculations to optimize gas usage.

Remediation

SOLVED: The **Moonwell team** has solved this issue as recommended. The commit hash for reference is `e23657c5fbeb12c7393fa49da6f350dc0bd5114e`.

Remediation Hash

https://github.com/moonwell-fi/contracts-open-source/commit/e23657c5fbeb12c7393fa49da6f350dc0bd5114e

7.7 USE ++I INSTEAD OF I++ IN LOOPS FOR GAS OPTIMIZATION

// INFORMATIONAL

Description

In all the loops of the contracts in-scope, the variable `i` is incremented using `i++`. It is known that, in loops, using `++i` costs less gas per iteration than `i++`. This also affects variables incremented inside the loop code block.

– DistributionManager.sol

```
48 | for (uint256 i = 0; i < assetsConfigInput.length; i ++) {
```

```
146 | for (uint256 i = 0; i < stakes.length; i++) {
```

```
173 | for (uint256 i = 0; i < stakes.length; i++) {
```

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:P/S:C (0.0)

Recommendation

It is recommended to use `++i` instead of `i++` to increment the value of an uint variable inside a loop. This also applies to the variables declared inside the for loop, not just the iterator.
On the other hand, this is not applicable outside of loops.

Remediation

SOLVED: The **Moonwell team** has solved this issue as recommended. The commit hash for reference is `e23657c5fbeb12c7393fa49da6f350dc0bd5114e`.

8. AUTOMATED TESTING

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their ABIs and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

```
WethUnwrapper.send(address,uint256) (src/WethUnwrapper.sol#20-31) sends eth to arbitrary user
        Dangerous calls:
        - (success,returndata) = to.call{value: amount}() (src/WethUnwrapper.sol#22)
WETHRouter.repayBorrowBehalf(address) (src/router/WETHRouter.sol#43-61) sends eth to arbitrary user
        Dangerous calls:
        - weth.deposit{value: borrows}() (src/router/WETHRouter.sol#52)
        - (success,None) = msg.sender.call{value: address(this).balance}() (src/router/WETHRouter.sol#54)
WormholeBridgeAdapter._bridgeOut(address,uint256,uint256,address) (src/xWELL/WormholeBridgeAdapter.sol#168-186) sends eth to arbitrary user
        Dangerous calls:
        - wormholeRelayer.sendPayloadToEvm{value: cost}(targetChainId,targetAddress[targetChainId],abi.encode(to,amount),0,gasLimit) (src/xWELL/WormholeBridgeAdapter.sol#177-183)
xWELLRouter._bridgeToChain(address,uint256,uint16) (src/xWELL/xWELLRouter.sol#76-105) sends eth to arbitrary user
        Dangerous calls:
        - wormholeBridge.bridge{value: bridgeCostGlmr}(wormholeChainId,xwellAmount,to) (src/xWELL/xWELLRouter.sol#97)
        - (success,None) = msg.sender.call{value: address(this).balance}() (src/xWELL/xWELLRouter.sol#100)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
MToken.accrueInterest() (src/MToken.sol#394-472) uses a dangerous strict equality:
        - accrualBlockTimestampPrior == currentBlockTimestamp (src/MToken.sol#400)
MToken.accrueInterest() (src/MToken.sol#394-472) uses a dangerous strict equality:
        - require(bool,string)(mathErr == MathError.NO_ERROR,could not calculate block delta) (src/MToken.sol#416)
MToken.balanceOfUnderlying(address) (src/MToken.sol#200-205) uses a dangerous strict equality:
        - require(bool,string)(mErr == MathError.NO_ERROR,balance could not be calculated) (src/MToken.sol#203)
MToken.borrowBalanceStored(address) (src/MToken.sol#281-285) uses a dangerous strict equality:
        - require(bool,string)(err == MathError.NO_ERROR,borrowBalanceStored: borrowBalanceStoredInternal failed) (src/MToken.sol#283)
CarefulMath.divUInt(uint256,uint256) (src/CarefulMath.sol#41-47) uses a dangerous strict equality:
        - b == 0 (src/CarefulMath.sol#42)
MToken.exchangeRateStored() (src/MToken.sol#338-342) uses a dangerous strict equality:
        - require(bool,string)(err == MathError.NO_ERROR,exchangeRateStored: exchangeRateStoredInternal failed) (src/MToken.sol#340)
MToken.exchangeRateStoredInternal() (src/MToken.sol#349-379) uses a dangerous strict equality:
        - _totalSupply == 0 (src/MToken.sol#351)
MToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (src/MToken.sol#30-66) uses a dangerous strict equality:
        - require(bool,string)(accrualBlockTimestamp == 0 && borrowIndex == 0,market may only be initialized once) (src/MToken.sol#42)
MToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (src/MToken.sol#30-66) uses a dangerous strict equality:
        - require(bool,string)(err == uint256(Error.NO_ERROR),setting comptroller failed) (src/MToken.sol#50)
MToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (src/MToken.sol#30-66) uses a dangerous strict equality:
        - require(bool,string)(err == uint256(Error.NO_ERROR),setting interest rate model failed) (src/MToken.sol#58)
MToken.liquidateBorrowFresh(address,address,uint256,MTokenInterface) (src/MToken.sol#977-1045) uses a dangerous strict equality:
        - require(bool,string)(amountSeizeError == uint256(Error.NO_ERROR),LIQUIDATE_COMPTROLLER_CALCULATE_AMOUNT_SEIZE_FAILED) (src/MToken.sol#1021)
MToken.liquidateBorrowFresh(address,address,uint256,MTokenInterface) (src/MToken.sol#977-1045) uses a dangerous strict equality:
        - require(bool,string)(seizeError == uint256(Error.NO_ERROR),token seizure failed) (src/MToken.sol#1035)
MToken.mintFresh(address,uint256) (src/MToken.sol#507-572) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_EXCHANGE_CALCULATION_FAILED) (src/MToken.sol#546)
MToken.mintFresh(address,uint256) (src/MToken.sol#507-572) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_NEW_TOTAL_SUPPLY_CALCULATION_FAILED) (src/MToken.sol#554)
MToken.mintFresh(address,uint256) (src/MToken.sol#507-572) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_NEW_ACCOUNT_BALANCE_CALCULATION_FAILED) (src/MToken.sol#557)
Exponential.mulExp(ExponentialNoError.Exp,ExponentialNoError.Exp) (src/Exponential.sol#136-155) uses a dangerous strict equality:
        - assert(bool)(err2 == MathError.NO_ERROR) (src/Exponential.sol#152)
CarefulMath.mulUInt(uint256,uint256) (src/CarefulMath.sol#24-36) uses a dangerous strict equality:
        - a == 0 (src/CarefulMath.sol#25)
ExponentialNoError.mul_(uint256,uint256,string) (src/ExponentialNoError.sol#151-158) uses a dangerous strict equality:
        - a == 0 || b == 0 (src/ExponentialNoError.sol#152)
ExponentialNoError.mul_(uint256,uint256,string) (src/ExponentialNoError.sol#151-158) uses a dangerous strict equality:
        - require(bool,string)(c / a == b,errorMessage) (src/ExponentialNoError.sol#156)
MToken.repayBorrowFresh(address,address,uint256) (src/MToken.sol#874-941) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,REPAY_BORROW_NEW_ACCOUNT_BORROW_BALANCE_CALCULATION_FAILED) (src/MToken.sol#923)
MToken.repayBorrowFresh(address,address,uint256) (src/MToken.sol#874-941) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,REPAY_BORROW_NEW_TOTAL_BALANCE_CALCULATION_FAILED) (src/MToken.sol#926)
MToken.seizeInternal(address,address,address,uint256) (src/MToken.sol#1082-1142) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,exchange rate math error) (src/MToken.sol#1110)
MToken.transfer(address,uint256) (src/MToken.sol#145-147) uses a dangerous strict equality:
```

```
MToken.transfer(address,uint256) (src/MToken.sol#145-147) uses a dangerous strict equality:
        - transferTokens(msg.sender,msg.sender,dst,amount) == uint256(Error.NO_ERROR) (src/MToken.sol#146)
MToken.transferFrom(address,address,uint256) (src/MToken.sol#156-158) uses a dangerous strict equality:
        - transferTokens(msg.sender,src,dst,amount) == uint256(Error.NO_ERROR) (src/MToken.sol#157)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in MToken.liquidateBorrowInternal(address,uint256,MTokenInterface) (src/MToken.sol#951-966):
        External calls:
        - error = mTokenCollateral.accrueInterest() (src/MToken.sol#958)
        - liquidateBorrowFresh(msg.sender,borrower,repayAmount,mTokenCollateral) (src/MToken.sol#965)
                - allowed = comptroller.seizeAllowed(address(this),seizerToken,liquidator,borrower,seizeTokens) (src/MToken.sol#1084)
                - allowed = comptroller.repayBorrowAllowed(address(this),payer,borrower,repayAmount) (src/MToken.sol#876)
                - seizeError = mTokenCollateral.seize(liquidator,borrower,seizeTokens) (src/MToken.sol#1031)
        State variables written after the call(s):
        - liquidateBorrowFresh(msg.sender,borrower,repayAmount,mTokenCollateral) (src/MToken.sol#965)
                - totalBorrows = vars.totalBorrowsNew (src/MToken.sol#931)
        MTokenStorage.totalBorrows (src/MTokenInterfaces.sol#53) can be used in cross function reentrancies:
        - MToken.accrueInterest() (src/MToken.sol#394-472)
        - MToken.borrowRatePerTimestamp() (src/MToken.sol#245-247)
        - MToken.exchangeRateStoredInternal() (src/MToken.sol#349-379)
        - MToken.supplyRatePerTimestamp() (src/MToken.sol#253-255)
        - MTokenStorage.totalBorrows (src/MTokenInterfaces.sol#53)
        - liquidateBorrowFresh(msg.sender,borrower,repayAmount,mTokenCollateral) (src/MToken.sol#965)
                - totalReserves = vars.totalReservesNew (src/MToken.sol#1127)
        MTokenStorage.totalReserves (src/MTokenInterfaces.sol#56) can be used in cross function reentrancies:
        - MToken.accrueInterest() (src/MToken.sol#394-472)
        - MToken.borrowRatePerTimestamp() (src/MToken.sol#245-247)
        - MToken.exchangeRateStoredInternal() (src/MToken.sol#349-379)
        - MToken.supplyRatePerTimestamp() (src/MToken.sol#253-255)
        - MTokenStorage.totalReserves (src/MTokenInterfaces.sol#56)
Reentrancy in MToken.redeemFresh(address,uint256,uint256) (src/MToken.sol#624-730):
        External calls:
        - allowed = comptroller.redeemAllowed(address(this),redeemer,vars.redeemTokens) (src/MToken.sol#676)
        State variables written after the call(s):
        - accountTokens[redeemer] = vars.accountTokensNew (src/MToken.sol#712)
        MTokenStorage.accountTokens (src/MTokenInterfaces.sol#62) can be used in cross function reentrancies:
        - MToken.balanceOf(address) (src/MToken.sol#190-192)
        - MToken.balanceOfUnderlying(address) (src/MToken.sol#200-205)
        - MToken.getAccountSnapshot(address) (src/MToken.sol#213-231)
        - totalSupply = vars.totalSupplyNew (src/MToken.sol#711)
        MTokenStorage.totalSupply (src/MTokenInterfaces.sol#59) can be used in cross function reentrancies:
        - MToken.exchangeRateStoredInternal() (src/MToken.sol#349-379)
        - MTokenStorage.totalSupply (src/MTokenInterfaces.sol#59)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
MToken.mintFresh(address,uint256).vars (src/MToken.sol#519) is a local variable never initialized
MToken.repayBorrowFresh(address,address,uint256).vars (src/MToken.sol#886) is a local variable never initialized
MToken._addReservesFresh(uint256).actualAddAmount (src/MToken.sol#1291) is a local variable never initialized
MToken.borrowFresh(address,uint256).vars (src/MToken.sol#776) is a local variable never initialized
MToken.redeemFresh(address,uint256,uint256).vars (src/MToken.sol#627) is a local variable never initialized
MToken.seizeInternal(address,address,address,uint256).vars (src/MToken.sol#1094) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
MErc20.initialize(address,ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (src/MErc20.sol#23-38) ignores return value by EIP20Interface(underlying).totalSupply() (src/MErc20.sol#37)
WETHRouter.constructor(WETH9,MErc20) (src/router/WETHRouter.sol#26-30) ignores return value by _weth.approve(address(_mToken),type()(uint256).max) (src/router/WETHRouter.sol#29)
WormholeBridgeAdapter.bridgeCost(uint16) (src/xWELL/WormholeBridgeAdapter.sol#150-152) ignores return value by (gasCost,None) = wormholeRelayer.quoteEVMDeliveryPrice(dstChainId,0,gasLimit) (src/xWELL/WormholeBridgeAdapter.sol#151)
WormholeBridgeAdapter._bridgeOut(address,uint256,uint256,address) (src/xWELL/WormholeBridgeAdapter.sol#168-186) ignores return value by wormholeRelayer.sendPayloadToEvm{value: cost}(targetChainId,targetAddress[targetChainId],abi.encode(to,amount),0,gasLimit) (src/xWELL/WormholeBridgeAdapter.sol#177-183)
WormholeUnwrapperAdapter._bridgeIn(uint256,address,uint256) (src/xWELL/WormholeUnwrapperAdapter.sol#34-41) ignores return value by IERC20(address(xERC20)).approve(lockbox,amount) (src/xWELL/WormholeUnwrapperAdapter.sol#39)
xWELLRouter._bridgeToChain(address,uint256,uint16) (src/xWELL/xWELLRouter.sol#76-105) ignores return value by well.approve(address(lockbox),amount) (src/xWELL/xWELLRouter.sol#85)
xWELLRouter._bridgeToChain(address,uint256,uint16) (src/xWELL/xWELLRouter.sol#76-105) ignores return value by xwell.approve(address(wormholeBridge),xwellAmount) (src/xWELL/xWELLRouter.sol#94)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

```
Exponential.divScalarByExpTruncate(uint256,ExponentialNoError.Exp).fraction (src/Exponential.sol#125) shadows:
        - ExponentialNoError.fraction(uint256,uint256) (src/ExponentialNoError.sol#193-195) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
MToken._setPendingAdmin(address).newPendingAdmin (src/MToken.sol#1152) lacks a zero-check on :
                - pendingAdmin = newPendingAdmin (src/MToken.sol#1162)
WethUnwrapper.constructor(address)._weth (src/WethUnwrapper.sol#12) lacks a zero-check on :
                - weth = _weth (src/WethUnwrapper.sol#13)
WethUnwrapper.send(address,uint256).to (src/WethUnwrapper.sol#20) lacks a zero-check on :
                - (success,returndata) = to.call{value: amount}() (src/WethUnwrapper.sol#22)
WormholeUnwrapperAdapter.setLockbox(address)._lockbox (src/xWELL/WormholeUnwrapperAdapter.sol#23) lacks a zero-check on :
                - lockbox = _lockbox (src/xWELL/WormholeUnwrapperAdapter.sol#25)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in MToken.borrowFresh(address,uint256) (src/MToken.sol#759-823):
        External calls:
        - allowed = comptroller.borrowAllowed(address(this),borrower,borrowAmount) (src/MToken.sol#761)
        State variables written after the call(s):
        - accountBorrows[borrower].principal = vars.accountBorrowsNew (src/MToken.sol#803)
        - accountBorrows[borrower].interestIndex = borrowIndex (src/MToken.sol#804)
        - totalBorrows = vars.totalBorrowsNew (src/MToken.sol#805)
Reentrancy in MToken.mintFresh(address,uint256) (src/MToken.sol#507-572):
        External calls:
        - allowed = comptroller.mintAllowed(address(this),minter,mintAmount) (src/MToken.sol#509)
        State variables written after the call(s):
        - accountTokens[minter] = vars.accountTokensNew (src/MToken.sol#561)
        - totalSupply = vars.totalSupplyNew (src/MToken.sol#560)
Reentrancy in MToken.repayBorrowFresh(address,address,uint256) (src/MToken.sol#874-941):
        External calls:
        - allowed = comptroller.repayBorrowAllowed(address(this),payer,borrower,repayAmount) (src/MToken.sol#876)
        State variables written after the call(s):
        - accountBorrows[borrower].principal = vars.accountBorrowsNew (src/MToken.sol#929)
        - accountBorrows[borrower].interestIndex = borrowIndex (src/MToken.sol#930)
        - totalBorrows = vars.totalBorrowsNew (src/MToken.sol#931)
Reentrancy in MToken.seizeInternal(address,address,address,uint256) (src/MToken.sol#1082-1142):
        External calls:
        - allowed = comptroller.seizeAllowed(address(this),seizerToken,liquidator,borrower,seizeTokens) (src/MToken.sol#1084)
        State variables written after the call(s):
        - accountTokens[borrower] = vars.borrowerTokensNew (src/MToken.sol#1129)
        - accountTokens[liquidator] = vars.liquidatorTokensNew (src/MToken.sol#1130)
        - totalReserves = vars.totalReservesNew (src/MToken.sol#1127)
        - totalSupply = vars.totalSupplyNew (src/MToken.sol#1128)
Reentrancy in MToken.transferTokens(address,address,address,uint256) (src/MToken.sol#77-137):
        External calls:
        - allowed = comptroller.transferAllowed(address(this),src,dst,tokens) (src/MToken.sol#79)
        State variables written after the call(s):
        - accountTokens[src] = srcTokensNew (src/MToken.sol#122)
        - accountTokens[dst] = dstTokensNew (src/MToken.sol#123)
        - transferAllowances[src][spender] = allowanceNew (src/MToken.sol#127)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in WormholeBridgeAdapter._bridgeOut(address,uint256,uint256,address) (src/xWELL/WormholeBridgeAdapter.sol#168-186):
        External calls:
        - _burnTokens(user,amount) (src/xWELL/WormholeBridgeAdapter.sol#175)
                - xERC20.burn(user,amount) (src/xWELL/xERC20BridgeAdapter.sol#62)
        - wormholeRelayer.sendPayloadToEvm{value: cost}(targetChainId,targetAddress[targetChainId],abi.encode(to,amount),0,gasLimit) (src/xWELL/WormholeBridgeAdapter.sol#177-183)
        External calls sending eth:
        - wormholeRelayer.sendPayloadToEvm{value: cost}(targetChainId,targetAddress[targetChainId],abi.encode(to,amount),0,gasLimit) (src/xWELL/WormholeBridgeAdapter.sol#177-183)
        Event emitted after the call(s):
        - TokensSent(targetChainId,to,amount) (src/xWELL/WormholeBridgeAdapter.sol#185)
```

```
Reentrancy in MToken.borrowFresh(address,uint256) (src/MToken.sol#759-823):
        External calls:
        - allowed = comptroller.borrowAllowed(address(this),borrower,borrowAmount) (src/MToken.sol#761)
        Event emitted after the call(s):
        - Borrow(borrower,borrowAmount,vars.accountBorrowsNew,vars.totalBorrowsNew) (src/MToken.sol#808)
        - Failure(uint256(err),uint256(info),opaqueError) (src/ErrorReporter.sol#136)
                - failOpaque(Error.MATH_ERROR,FailureInfo.BORROW_ACCUMULATED_BALANCE_CALCULATION_FAILED,uint256(vars.mathErr)) (src/MToken.sol#785)
        - Failure(uint256(err),uint256(info),opaqueError) (src/ErrorReporter.sol#136)
                - failOpaque(Error.MATH_ERROR,FailureInfo.BORROW_NEW_ACCOUNT_BORROW_BALANCE_CALCULATION_FAILED,uint256(vars.mathErr)) (src/MToken.sol#790)
        - Failure(uint256(err),uint256(info),opaqueError) (src/ErrorReporter.sol#136)
                - failOpaque(Error.MATH_ERROR,FailureInfo.BORROW_NEW_TOTAL_BALANCE_CALCULATION_FAILED,uint256(vars.mathErr)) (src/MToken.sol#795)
        - Failure(uint256(err),uint256(info),0) (src/ErrorReporter.sol#127)
                - fail(Error.TOKEN_INSUFFICIENT_CASH,FailureInfo.BORROW_CASH_NOT_AVAILABLE) (src/MToken.sol#773)
        - Failure(uint256(err),uint256(info),0) (src/ErrorReporter.sol#127)
                - fail(Error.MARKET_NOT_FRESH,FailureInfo.BORROW_FRESHNESS_CHECK) (src/MToken.sol#768)
        - Failure(uint256(err),uint256(info),opaqueError) (src/ErrorReporter.sol#136)
                - failOpaque(Error.COMPTROLLER_REJECTION,FailureInfo.BORROW_COMPTROLLER_REJECTION,allowed) (src/MToken.sol#763)
Reentrancy in MToken.liquidateBorrowFresh(address,address,uint256,MTokenInterface) (src/MToken.sol#977-1045):
        External calls:
        - (repayBorrowError,actualRepayAmount) = repayBorrowFresh(liquidator,borrower,repayAmount) (src/MToken.sol#1010)
                - allowed = comptroller.repayBorrowAllowed(address(this),payer,borrower,repayAmount) (src/MToken.sol#876)
        Event emitted after the call(s):
        - Failure(uint256(err),uint256(info),0) (src/ErrorReporter.sol#127)
                - (fail(TokenErrorReporter.Error(repayBorrowError),FailureInfo.LIQUIDATE_REPAY_BORROW_FRESH_FAILED),0) (src/MToken.sol#1012)
Reentrancy in MToken.liquidateBorrowFresh(address,address,uint256,MTokenInterface) (src/MToken.sol#977-1045):
        External calls:
        - (repayBorrowError,actualRepayAmount) = repayBorrowFresh(liquidator,borrower,repayAmount) (src/MToken.sol#1010)
                - allowed = comptroller.repayBorrowAllowed(address(this),payer,borrower,repayAmount) (src/MToken.sol#876)
        - seizeError = seizeInternal(address(this),liquidator,borrower,seizeTokens) (src/MToken.sol#1029)
                - allowed = comptroller.seizeAllowed(address(this),seizerToken,liquidator,borrower,seizeTokens) (src/MToken.sol#1084)
        Event emitted after the call(s):
        - Failure(uint256(err),uint256(info),opaqueError) (src/ErrorReporter.sol#136)
                - seizeError = seizeInternal(address(this),liquidator,borrower,seizeTokens) (src/MToken.sol#1029)
        - Failure(uint256(err),uint256(info),0) (src/ErrorReporter.sol#127)
                - seizeError = seizeInternal(address(this),liquidator,borrower,seizeTokens) (src/MToken.sol#1029)
        - ReservesAdded(address(this),vars.protocolSeizeAmount,vars.totalReservesNew) (src/MToken.sol#1135)
                - seizeError = seizeInternal(address(this),liquidator,borrower,seizeTokens) (src/MToken.sol#1029)
        - Transfer(borrower,liquidator,vars.liquidatorSeizeTokens) (src/MToken.sol#1133)
                - seizeError = seizeInternal(address(this),liquidator,borrower,seizeTokens) (src/MToken.sol#1029)
        - Transfer(borrower,address(this),vars.protocolSeizeTokens) (src/MToken.sol#1134)
                - seizeError = seizeInternal(address(this),liquidator,borrower,seizeTokens) (src/MToken.sol#1029)
Reentrancy in MToken.liquidateBorrowFresh(address,address,uint256,MTokenInterface) (src/MToken.sol#977-1045):
        External calls:
        - (repayBorrowError,actualRepayAmount) = repayBorrowFresh(liquidator,borrower,repayAmount) (src/MToken.sol#1010)
                - allowed = comptroller.repayBorrowAllowed(address(this),payer,borrower,repayAmount) (src/MToken.sol#876)
        - seizeError = seizeInternal(address(this),liquidator,borrower,seizeTokens) (src/MToken.sol#1029)
                - allowed = comptroller.seizeAllowed(address(this),seizerToken,liquidator,borrower,seizeTokens) (src/MToken.sol#1084)
        - seizeError = mTokenCollateral.seize(liquidator,borrower,seizeTokens) (src/MToken.sol#1031)
        Event emitted after the call(s):
        - LiquidateBorrow(liquidator,borrower,actualRepayAmount,address(mTokenCollateral),seizeTokens) (src/MToken.sol#1038)
Reentrancy in MToken.mintFresh(address,uint256) (src/MToken.sol#507-572):
        External calls:
        - allowed = comptroller.mintAllowed(address(this),minter,mintAmount) (src/MToken.sol#509)
        Event emitted after the call(s):
        - Failure(uint256(err),uint256(info),opaqueError) (src/ErrorReporter.sol#136)
                - (failOpaque(Error.COMPTROLLER_REJECTION,FailureInfo.MINT_COMPTROLLER_REJECTION,allowed),0) (src/MToken.sol#511)
        - Failure(uint256(err),uint256(info),opaqueError) (src/ErrorReporter.sol#136)
                - (failOpaque(Error.MATH_ERROR,FailureInfo.MINT_EXCHANGE_RATE_READ_FAILED,uint256(vars.mathErr)),0) (src/MToken.sol#523)
        - Failure(uint256(err),uint256(info),0) (src/ErrorReporter.sol#127)
                - (fail(Error.MARKET_NOT_FRESH,FailureInfo.MINT_FRESHNESS_CHECK),0) (src/MToken.sol#516)
        - Mint(minter,vars.actualMintAmount,vars.mintTokens) (src/MToken.sol#564)
        - Transfer(address(this),minter,vars.mintTokens) (src/MToken.sol#565)
```

```
Reentrancy in MErc20.mintWithPermit(uint256,uint256,uint8,bytes32,bytes32) (src/MErc20.sol#63-72):
        External calls:
        - SafeERC20.safePermit(token,msg.sender,address(this),mintAmount,deadline,v,r,s) (src/MErc20.sol#68)
        - (err,None) = mintInternal(mintAmount) (src/MErc20.sol#70)
                - allowed = comptroller.mintAllowed(address(this),minter,mintAmount) (src/MToken.sol#509)
                - token.transferFrom(from,address(this),amount) (src/MErc20.sol#183)
        Event emitted after the call(s):
        - AccrueInterest(cashPrior,interestAccumulated,borrowIndexNew,totalBorrowsNew) (src/MToken.sol#469)
                - (err,None) = mintInternal(mintAmount) (src/MErc20.sol#70)
        - Failure(uint256(err),uint256(info),0) (src/ErrorReporter.sol#127)
                - (err,None) = mintInternal(mintAmount) (src/MErc20.sol#70)
        - Failure(uint256(err),uint256(info),opaqueError) (src/ErrorReporter.sol#136)
                - (err,None) = mintInternal(mintAmount) (src/MErc20.sol#70)
        - Mint(minter,vars.actualMintAmount,vars.mintTokens) (src/MToken.sol#564)
                - (err,None) = mintInternal(mintAmount) (src/MErc20.sol#70)
        - Transfer(address(this),minter,vars.mintTokens) (src/MToken.sol#565)
                - (err,None) = mintInternal(mintAmount) (src/MErc20.sol#70)
Reentrancy in MToken.redeemFresh(address,uint256,uint256) (src/MToken.sol#624-730):
        External calls:
        - allowed = comptroller.redeemAllowed(address(this),redeemer,vars.redeemTokens) (src/MToken.sol#676)
        Event emitted after the call(s):
        - Failure(uint256(err),uint256(info),0) (src/ErrorReporter.sol#127)
                - fail(Error.MARKET_NOT_FRESH,FailureInfo.REDEEM_FRESHNESS_CHECK) (src/MToken.sol#683)
        - Failure(uint256(err),uint256(info),opaqueError) (src/ErrorReporter.sol#136)
                - failOpaque(Error.COMPTROLLER_REJECTION,FailureInfo.REDEEM_COMPTROLLER_REJECTION,allowed) (src/MToken.sol#678)
        - Failure(uint256(err),uint256(info),0) (src/ErrorReporter.sol#127)
                - fail(Error.TOKEN_INSUFFICIENT_CASH,FailureInfo.REDEEM_TRANSFER_OUT_NOT_POSSIBLE) (src/MToken.sol#703)
        - Failure(uint256(err),uint256(info),opaqueError) (src/ErrorReporter.sol#136)
                - failOpaque(Error.MATH_ERROR,FailureInfo.REDEEM_NEW_TOTAL_SUPPLY_CALCULATION_FAILED,uint256(vars.mathErr)) (src/MToken.sol#693)
        - Failure(uint256(err),uint256(info),opaqueError) (src/ErrorReporter.sol#136)
                - failOpaque(Error.MATH_ERROR,FailureInfo.REDEEM_NEW_ACCOUNT_BALANCE_CALCULATION_FAILED,uint256(vars.mathErr)) (src/MToken.sol#698)
        - Redeem(redeemer,vars.redeemAmount,vars.redeemTokens) (src/MToken.sol#716)
        - Transfer(redeemer,address(this),vars.redeemTokens) (src/MToken.sol#715)
Reentrancy in WETHRouter.repayBorrowBehalf(address) (src/router/WETHRouter.sol#43-61):
        External calls:
        - borrows = mToken.borrowBalanceCurrent(borrower) (src/router/WETHRouter.sol#45)
        Event emitted after the call(s):
        - WethRouter(received,received) (src/router/WETHRouter.sol#47)
        - WethRouter(borrows,borrows) (src/router/WETHRouter.sol#48)
        - WethRouter(DEPOSIT BORROWS,true) (src/router/WETHRouter.sol#51)
        - WethRouter(DEPOSIT RECEIVED,true) (src/router/WETHRouter.sol#57)
Reentrancy in MToken.repayBorrowFresh(address,address,uint256) (src/MToken.sol#874-941):
        External calls:
        - allowed = comptroller.repayBorrowAllowed(address(this),payer,borrower,repayAmount) (src/MToken.sol#876)
        Event emitted after the call(s):
        - Failure(uint256(err),uint256(info),0) (src/ErrorReporter.sol#127)
                - (fail(Error.MARKET_NOT_FRESH,FailureInfo.REPAY_BORROW_FRESHNESS_CHECK),0) (src/MToken.sol#883)
        - Failure(uint256(err),uint256(info),opaqueError) (src/ErrorReporter.sol#136)
                - (failOpaque(Error.COMPTROLLER_REJECTION,FailureInfo.REPAY_BORROW_COMPTROLLER_REJECTION,allowed),0) (src/MToken.sol#878)
        - Failure(uint256(err),uint256(info),opaqueError) (src/ErrorReporter.sol#136)
                - (failOpaque(Error.MATH_ERROR,FailureInfo.REPAY_BORROW_ACCUMULATED_BALANCE_CALCULATION_FAILED,uint256(vars.mathErr)),0) (src/MToken.sol#894)
        - RepayBorrow(payer,borrower,vars.actualRepayAmount,vars.accountBorrowsNew,vars.totalBorrowsNew) (src/MToken.sol#934)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

```
MToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (src/MToken.sol#30-66) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(accrualBlockTimestamp == 0 && borrowIndex == 0,market may only be initialized once) (src/MToken.sol#42)
        - require(bool,string)(err == uint256(Error.NO_ERROR),setting comptroller failed) (src/MToken.sol#50)
        - require(bool,string)(err == uint256(Error.NO_ERROR),setting interest rate model failed) (src/MToken.sol#58)
MToken.transferTokens(address,address,address,uint256) (src/MToken.sol#77-137) uses timestamp for comparisons
        Dangerous comparisons:
        - mathErr != MathError.NO_ERROR (src/MToken.sol#104)
        - mathErr != MathError.NO_ERROR (src/MToken.sol#109)
        - mathErr != MathError.NO_ERROR (src/MToken.sol#114)
MToken.transfer(address,uint256) (src/MToken.sol#145-147) uses timestamp for comparisons
        Dangerous comparisons:
        - transferTokens(msg.sender,msg.sender,dst,amount) == uint256(Error.NO_ERROR) (src/MToken.sol#146)
MToken.transferFrom(address,address,uint256) (src/MToken.sol#156-158) uses timestamp for comparisons
        Dangerous comparisons:
        - transferTokens(msg.sender,src,dst,amount) == uint256(Error.NO_ERROR) (src/MToken.sol#157)
MToken.balanceOfUnderlying(address) (src/MToken.sol#200-205) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(mErr == MathError.NO_ERROR,balance could not be calculated) (src/MToken.sol#203)
MToken.getAccountSnapshot(address) (src/MToken.sol#213-231) uses timestamp for comparisons
        Dangerous comparisons:
        - mErr != MathError.NO_ERROR (src/MToken.sol#221)
        - mErr != MathError.NO_ERROR (src/MToken.sol#226)
MToken.borrowBalanceStored(address) (src/MToken.sol#281-285) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(err == MathError.NO_ERROR,borrowBalanceStored: borrowBalanceStoredInternal failed) (src/MToken.sol#283)
MToken.borrowBalanceStoredInternal(address) (src/MToken.sol#292-322) uses timestamp for comparisons
        Dangerous comparisons:
        - mathErr != MathError.NO_ERROR (src/MToken.sol#312)
        - mathErr != MathError.NO_ERROR (src/MToken.sol#317)
MToken.exchangeRateStored() (src/MToken.sol#338-342) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(err == MathError.NO_ERROR,exchangeRateStored: exchangeRateStoredInternal failed) (src/MToken.sol#340)
MToken.exchangeRateStoredInternal() (src/MToken.sol#349-379) uses timestamp for comparisons
        Dangerous comparisons:
        - _totalSupply == 0 (src/MToken.sol#351)
        - mathErr != MathError.NO_ERROR (src/MToken.sol#368)
        - mathErr != MathError.NO_ERROR (src/MToken.sol#373)
MToken.accrueInterest() (src/MToken.sol#394-472) uses timestamp for comparisons
        Dangerous comparisons:
        - accrualBlockTimestampPrior == currentBlockTimestamp (src/MToken.sol#400)
        - require(bool,string)(mathErr == MathError.NO_ERROR,could not calculate block delta) (src/MToken.sol#416)
        - mathErr != MathError.NO_ERROR (src/MToken.sol#434)
        - mathErr != MathError.NO_ERROR (src/MToken.sol#439)
        - mathErr != MathError.NO_ERROR (src/MToken.sol#444)
        - mathErr != MathError.NO_ERROR (src/MToken.sol#449)
        - mathErr != MathError.NO_ERROR (src/MToken.sol#454)
MToken.mintFresh(address,uint256) (src/MToken.sol#507-572) uses timestamp for comparisons
        Dangerous comparisons:
        - accrualBlockTimestamp != getBlockTimestamp() (src/MToken.sol#515)
        - vars.mathErr != MathError.NO_ERROR (src/MToken.sol#522)
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_EXCHANGE_CALCULATION_FAILED) (src/MToken.sol#546)
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_NEW_TOTAL_SUPPLY_CALCULATION_FAILED) (src/MToken.sol#554)
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_NEW_ACCOUNT_BALANCE_CALCULATION_FAILED) (src/MToken.sol#557)
MToken.redeemFresh(address,uint256,uint256) (src/MToken.sol#624-730) uses timestamp for comparisons
        Dangerous comparisons:
        - vars.mathErr != MathError.NO_ERROR (src/MToken.sol#631)
        - vars.mathErr != MathError.NO_ERROR (src/MToken.sol#649)
        - vars.mathErr != MathError.NO_ERROR (src/MToken.sol#662)
        - vars.mathErr != MathError.NO_ERROR (src/MToken.sol#669)
        - allowed != 0 (src/MToken.sol#677)
```

```
        - accrualBlockTimestamp ≠ getBlockTimestamp() (src/MToken.sol#682)
        - vars.mathErr ≠ MathError.NO_ERROR (src/MToken.sol#692)
        - vars.mathErr ≠ MathError.NO_ERROR (src/MToken.sol#697)
        - getCashPrior() < vars.redeemAmount (src/MToken.sol#702)
MToken.borrowFresh(address,uint256) (src/MToken.sol#759-823) uses timestamp for comparisons
        Dangerous comparisons:
        - accrualBlockTimestamp ≠ getBlockTimestamp() (src/MToken.sol#767)
        - vars.mathErr ≠ MathError.NO_ERROR (src/MToken.sol#784)
        - vars.mathErr ≠ MathError.NO_ERROR (src/MToken.sol#789)
        - vars.mathErr ≠ MathError.NO_ERROR (src/MToken.sol#794)
MToken.repayBorrowFresh(address,address,uint256) (src/MToken.sol#874-941) uses timestamp for comparisons
        Dangerous comparisons:
        - accrualBlockTimestamp ≠ getBlockTimestamp() (src/MToken.sol#882)
        - vars.mathErr ≠ MathError.NO_ERROR (src/MToken.sol#893)
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,REPAY_BORROW_NEW_ACCOUNT_BORROW_BALANCE_CALCULATION_FAILED) (src/MToken.sol#923)
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,REPAY_BORROW_NEW_TOTAL_BALANCE_CALCULATION_FAILED) (src/MToken.sol#926)
MToken.liquidateBorrowFresh(address,address,uint256,MTokenInterface) (src/MToken.sol#977-1045) uses timestamp for comparisons
        Dangerous comparisons:
        - accrualBlockTimestamp ≠ getBlockTimestamp() (src/MToken.sol#985)
        - mTokenCollateral.accrualBlockTimestamp() ≠ getBlockTimestamp() (src/MToken.sol#990)
        - repayBorrowError ≠ uint256(Error.NO_ERROR) (src/MToken.sol#1011)
        - require(bool,string)(amountSeizeError == uint256(Error.NO_ERROR),LIQUIDATE_COMPTROLLER_CALCULATE_AMOUNT_SEIZE_FAILED) (src/MToken.sol#1021)
        - require(bool,string)(mTokenCollateral.balanceOf(borrower) ≥ seizeTokens,LIQUIDATE_SEIZE_TOO_MUCH) (src/MToken.sol#1024)
        - require(bool,string)(seizeError == uint256(Error.NO_ERROR),token seizure failed) (src/MToken.sol#1035)
MToken.seizeInternal(address,address,address,uint256) (src/MToken.sol#1082-1142) uses timestamp for comparisons
        Dangerous comparisons:
        - allowed ≠ 0 (src/MToken.sol#1085)
        - vars.mathErr ≠ MathError.NO_ERROR (src/MToken.sol#1102)
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,exchange rate math error) (src/MToken.sol#1110)
        - vars.mathErr ≠ MathError.NO_ERROR (src/MToken.sol#1118)
MToken._setReserveFactorFresh(uint256) (src/MToken.sol#1241-1263) uses timestamp for comparisons
        Dangerous comparisons:
        - accrualBlockTimestamp ≠ getBlockTimestamp() (src/MToken.sol#1248)
MToken._addReservesInternal(uint256) (src/MToken.sol#1270-1280) uses timestamp for comparisons
        Dangerous comparisons:
        - error ≠ uint256(Error.NO_ERROR) (src/MToken.sol#1272)
MToken._addReservesFresh(uint256) (src/MToken.sol#1288-1325) uses timestamp for comparisons
        Dangerous comparisons:
        - accrualBlockTimestamp ≠ getBlockTimestamp() (src/MToken.sol#1294)
        - require(bool,string)(totalReservesNew ≥ totalReserves,add reserves unexpected overflow) (src/MToken.sol#1315)
MToken._reduceReservesFresh(uint256) (src/MToken.sol#1348-1389) uses timestamp for comparisons
        Dangerous comparisons:
        - accrualBlockTimestamp ≠ getBlockTimestamp() (src/MToken.sol#1358)
        - reduceAmount > totalReserves (src/MToken.sol#1368)
        - require(bool,string)(totalReservesNew ≤ totalReserves,reduce reserves unexpected underflow) (src/MToken.sol#1378)
MToken._setInterestRateModelFresh(InterestRateModel) (src/MToken.sol#1413-1440) uses timestamp for comparisons
        Dangerous comparisons:
        - accrualBlockTimestamp ≠ getBlockTimestamp() (src/MToken.sol#1423)
MToken._setProtocolSeizeShareFresh(uint256) (src/MToken.sol#1464-1488) uses timestamp for comparisons
        Dangerous comparisons:
        - accrualBlockTimestamp ≠ getBlockTimestamp() (src/MToken.sol#1474)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
MErc20Delegator.delegateTo(address,bytes) (src/MErc20Delegator.sol#632-643) uses assembly
        - INLINE ASM (src/MErc20Delegator.sol#637-641)
MErc20Delegator.delegateToViewImplementation(bytes) (src/MErc20Delegator.sol#664-676) uses assembly
        - INLINE ASM (src/MErc20Delegator.sol#670-674)
MErc20Delegator.fallback() (src/MErc20Delegator.sol#682-703) uses assembly
        - INLINE ASM (src/MErc20Delegator.sol#691-702)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Exponential.addExp(ExponentialNoError.Exp,ExponentialNoError.Exp) (src/Exponential.sol#38-42) is never used and should be removed
Exponential.divExp(ExponentialNoError.Exp,ExponentialNoError.Exp) (src/Exponential.sol#180-182) is never used and should be removed
Exponential.divScalar(ExponentialNoError.Exp,uint256) (src/Exponential.sol#92-99) is never used and should be removed
Exponential.mulExp(ExponentialNoError.Exp,ExponentialNoError.Exp) (src/Exponential.sol#136-155) is never used and should be removed
Exponential.mulExp(uint256,uint256) (src/Exponential.sol#160-162) is never used and should be removed
Exponential.mulExp3(ExponentialNoError.Exp,ExponentialNoError.Exp,ExponentialNoError.Exp) (src/Exponential.sol#167-173) is never used and should be removed
Exponential.subExp(ExponentialNoError.Exp,ExponentialNoError.Exp) (src/Exponential.sol#47-51) is never used and should be removed
ExponentialNoError.add_(ExponentialNoError.Exp,ExponentialNoError.Exp) (src/ExponentialNoError.sol#88-90) is never used and should be removed
ExponentialNoError.div_(ExponentialNoError.Double,ExponentialNoError.Double) (src/ExponentialNoError.sol#172-174) is never used and should be removed
ExponentialNoError.div_(ExponentialNoError.Double,uint256) (src/ExponentialNoError.sol#176-178) is never used and should be removed
ExponentialNoError.div_(ExponentialNoError.Exp,uint256) (src/ExponentialNoError.sol#164-166) is never used and should be removed
ExponentialNoError.div_(uint256,ExponentialNoError.Double) (src/ExponentialNoError.sol#180-182) is never used and should be removed
ExponentialNoError.greaterThanExp(ExponentialNoError.Exp,ExponentialNoError.Exp) (src/ExponentialNoError.sol#67-69) is never used and should be removed
ExponentialNoError.isZeroExp(ExponentialNoError.Exp) (src/ExponentialNoError.sol#74-76) is never used and should be removed
ExponentialNoError.lessThanOrEqualExp(ExponentialNoError.Exp,ExponentialNoError.Exp) (src/ExponentialNoError.sol#60-62) is never used and should be removed
ExponentialNoError.mul_(ExponentialNoError.Double,ExponentialNoError.Double) (src/ExponentialNoError.sol#135-137) is never used and should be removed
ExponentialNoError.mul_(ExponentialNoError.Double,uint256) (src/ExponentialNoError.sol#139-141) is never used and should be removed
ExponentialNoError.sub_(ExponentialNoError.Double,ExponentialNoError.Double) (src/ExponentialNoError.sol#110-112) is never used and should be removed
ExponentialNoError.sub_(ExponentialNoError.Exp,ExponentialNoError.Exp) (src/ExponentialNoError.sol#106-108) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Version constraint 0.8.19 contains known severe issues (https://solidity.readthedocs.io/en/latest/bugs.html)
        - VerbatimInvalidDeduplication
        - FullInlinerNonExpressionSplitArgumentEvaluationOrder
        - MissingSideEffectsOnSelectorAccess.
It is used by:
        - 0.8.19 (src/4626/Factory4626.sol#1)
        - 0.8.19 (src/4626/Factory4626Eth.sol#1)
        - 0.8.19 (src/4626/LibCompound.sol#2)
        - 0.8.19 (src/4626/MoonwellERC4626.sol#2)
        - 0.8.19 (src/4626/MoonwellERC4626Eth.sol#2)
        - 0.8.19 (src/CarefulMath.sol#2)
        - 0.8.19 (src/Comptroller.sol#2)
        - 0.8.19 (src/ComptrollerInterface.sol#2)
        - 0.8.19 (src/ComptrollerStorage.sol#2)
        - 0.8.19 (src/EIP20Interface.sol#2)
        - 0.8.19 (src/EIP20NonStandardInterface.sol#2)
        - 0.8.19 (src/ErrorReporter.sol#2)
        - 0.8.19 (src/Exponential.sol#2)
        - 0.8.19 (src/ExponentialNoError.sol#2)
        - 0.8.19 (src/IStakedWell.sol#1)
        - 0.8.19 (src/MErc20.sol#2)
        - 0.8.19 (src/MErc20Delegate.sol#2)
        - 0.8.19 (src/MErc20Delegator.sol#2)
        - 0.8.19 (src/MLikeDelegate.sol#2)
        - 0.8.19 (src/MToken.sol#2)
        - 0.8.19 (src/MTokenInterfaces.sol#2)
        - 0.8.19 (src/MWethDelegate.sol#2)
        - 0.8.19 (src/Recovery.sol#1)
        - 0.8.19 (src/SafeMath.sol#2)
        - 0.8.19 (src/Unitroller.sol#2)
        - 0.8.19 (src/WethUnwrapper.sol#2)
        - 0.8.19 (src/governance/IERC20.sol#2)
```

```
        - 0.8.19 (src/governance/ITemporalGovernor.sol#2)
        - 0.8.19 (src/governance/IWormholeTrustedSender.sol#2)
        - 0.8.19 (src/governance/MarketAddChecker.sol#1)
        - 0.8.19 (src/governance/TemporalGovernor.sol#2)
        - 0.8.19 (src/governance/Well.sol#2)
        - 0.8.19 (src/governance/WormholeTrustedSender.sol#2)
        - 0.8.19 (src/interfaces/IArtemisGovernor.sol#2)
        - 0.8.19 (src/interfaces/IStellaSwapRewarder.sol#2)
        - 0.8.19 (src/interfaces/ITimelock.sol#2)
        - 0.8.19 (src/irm/InterestRateModel.sol#2)
        - 0.8.19 (src/irm/JumpRateModel.sol#2)
        - 0.8.19 (src/irm/WhitePaperInterestRateModel.sol#2)
        - 0.8.19 (src/oracles/AggregatorV3Interface.sol#2)
        - 0.8.19 (src/oracles/ChainlinkCompositeOracle.sol#2)
        - 0.8.19 (src/oracles/ChainlinkOracle.sol#2)
        - 0.8.19 (src/oracles/PriceOracle.sol#2)
        - 0.8.19 (src/rewards/IMultiRewardDistributor.sol#2)
        - 0.8.19 (src/rewards/MultiRewardDistributor.sol#2)
        - 0.8.19 (src/rewards/MultiRewardDistributorCommon.sol#2)
        - 0.8.19 (src/router/ERC4626EthRouter.sol#1)
        - 0.8.19 (src/router/IWETH.sol#20)
        - 0.8.19 (src/router/WETHRouter.sol#1)
        - 0.8.19 (src/utils/Address.sol#2)
        - 0.8.19 (src/utils/Bytes.sol#2)
        - 0.8.19 (src/utils/ChainIds.sol#2)
        - 0.8.19 (src/utils/String.sol#2)
        - 0.8.19 (src/wormhole/IWormhole.sol#3)
        - 0.8.19 (src/wormhole/WormholeBridgeBase.sol#1)
        - 0.8.19 (src/xWELL/AxelarBridgeAdapter.sol#1)
        - 0.8.19 (src/xWELL/ConfigurablePause.sol#1)
        - 0.8.19 (src/xWELL/ConfigurablePauseGuardian.sol#1)
        - 0.8.19 (src/xWELL/MintLimits.sol#1)
        - 0.8.19 (src/xWELL/WormholeBridgeAdapter.sol#1)
        - 0.8.19 (src/xWELL/WormholeUnwrapperAdapter.sol#2)
        - 0.8.19 (src/xWELL/XERC20Lockbox.sol#1)
        - 0.8.19 (src/xWELL/axelarInterfaces/AddressString.sol#3)
        - 0.8.19 (src/xWELL/interfaces/IXERC20.sol#1)
        - 0.8.19 (src/xWELL/interfaces/IXERC20Lockbox.sol#1)
        - 0.8.19 (src/xWELL/xERC20.sol#1)
        - 0.8.19 (src/xWELL/xERC20BridgeAdapter.sol#1)
        - 0.8.19 (src/xWELL/xWELL.sol#1)
        - 0.8.19 (src/xWELL/xWELLRouter.sol#1)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in WethUnwrapper.send(address,uint256) (src/WethUnwrapper.sol#20-31):
        - (success,returndata) = to.call{value: amount}() (src/WethUnwrapper.sol#22)
Low level call in WETHRouter.repayBorrowBehalf(address) (src/router/WETHRouter.sol#43-61):
        - (success,None) = msg.sender.call{value: address(this).balance}() (src/router/WETHRouter.sol#54)
Low level call in xWELLRouter._bridgeToChain(address,uint256,uint16) (src/xWELL/xWELLRouter.sol#76-105):
        - (success,None) = msg.sender.call{value: address(this).balance}() (src/xWELL/xWELLRouter.sol#100)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
The following unused import(s) in src/MToken.sol should be removed:
        -import "./EIP20Interface.sol"; (src/MToken.sol#8)
The following unused import(s) in src/MTokenInterfaces.sol should be removed:
        -import "./ErrorReporter.sol"; (src/MTokenInterfaces.sol#7)
The following unused import(s) in src/rewards/IMultiRewardDistributor.sol should be removed:
        -import "../ExponentialNoError.sol"; (src/rewards/IMultiRewardDistributor.sol#5)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-imports
INFO:Detectors:
```

```
ExponentialNoError.halfExpScale (src/ExponentialNoError.sol#14) is never used in Comptroller (src/Comptroller.sol#15-1418)
ExponentialNoError.mantissaOne (src/ExponentialNoError.sol#15) is never used in Comptroller (src/Comptroller.sol#15-1418)
MTokenStorage._notEntered (src/MTokenInterfaces.sol#11) is never used in MErc20Delegator (src/MErc20Delegator.sol#11-704)
MTokenStorage.borrowRateMaxMantissa (src/MTokenInterfaces.sol#23) is never used in MErc20Delegator (src/MErc20Delegator.sol#11-704)
MTokenStorage.reserveFactorMaxMantissa (src/MTokenInterfaces.sol#26) is never used in MErc20Delegator (src/MErc20Delegator.sol#11-704)
MTokenStorage.initialExchangeRateMantissa (src/MTokenInterfaces.sol#41) is never used in MErc20Delegator (src/MErc20Delegator.sol#11-704)
MTokenStorage.accountTokens (src/MTokenInterfaces.sol#62) is never used in MErc20Delegator (src/MErc20Delegator.sol#11-704)
MTokenStorage.transferAllowances (src/MTokenInterfaces.sol#65) is never used in MErc20Delegator (src/MErc20Delegator.sol#11-704)
MTokenStorage.accountBorrows (src/MTokenInterfaces.sol#78) is never used in MErc20Delegator (src/MErc20Delegator.sol#11-704)
ExponentialNoError.halfExpScale (src/ExponentialNoError.sol#14) is never used in MultiRewardDistributor (src/rewards/MultiRewardDistributor.sol#43-1249)
ExponentialNoError.mantissaOne (src/ExponentialNoError.sol#15) is never used in MultiRewardDistributor (src/rewards/MultiRewardDistributor.sol#43-1249)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
INFO:Detectors:
Loop condition i < allMarkets.length (src/Comptroller.sol#1040) should use cached array length instead of referencing `length` member of the storage array.
 Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#cache-array-length
INFO:Detectors:
MErc20Storage.underlying (src/MTokenInterfaces.sol#173) should be constant
MTokenStorage._notEntered (src/MTokenInterfaces.sol#11) should be constant
MTokenStorage.accrualBlockTimestamp (src/MTokenInterfaces.sol#47) should be constant
MTokenStorage.borrowIndex (src/MTokenInterfaces.sol#50) should be constant
MTokenStorage.comptroller (src/MTokenInterfaces.sol#35) should be constant
MTokenStorage.decimals (src/MTokenInterfaces.sol#20) should be constant
MTokenStorage.initialExchangeRateMantissa (src/MTokenInterfaces.sol#41) should be constant
MTokenStorage.interestRateModel (src/MTokenInterfaces.sol#38) should be constant
MTokenStorage.name (src/MTokenInterfaces.sol#14) should be constant
MTokenStorage.pendingAdmin (src/MTokenInterfaces.sol#32) should be constant
MTokenStorage.protocolSeizeShareMantissa (src/MTokenInterfaces.sol#81) should be constant
MTokenStorage.reserveFactorMantissa (src/MTokenInterfaces.sol#44) should be constant
MTokenStorage.symbol (src/MTokenInterfaces.sol#17) should be constant
MTokenStorage.totalBorrows (src/MTokenInterfaces.sol#53) should be constant
MTokenStorage.totalReserves (src/MTokenInterfaces.sol#56) should be constant
MTokenStorage.totalSupply (src/MTokenInterfaces.sol#59) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
MTokenStorage.admin (src/MTokenInterfaces.sol#29) should be immutable
xWELLRouter.wormholeBridge (src/xWELL/xWELLRouter.sol#30) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
INFO:Slither:. analyzed (138 contracts with 94 detectors), 145 result(s) found
```

All issues identified by Slither were proved to be false positives or have been added to the issue list in this report.

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.