Moonwell Finance - Contracts V2 Updates

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **July 16th, 2023 - August 16th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	07/19/2023	Gokberk Gulgun
0.2	Document Updates	07/23/2023	Gokberk Gulgun
0.3	Document Updates	08/16/2023	Gokberk Gulgun
1.0	Remediation Plan	08/16/2023	Gokberk Gulgun
1.1	Remediation Plan Review	08/16/2023	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Moonwell Finance engaged Halborn to conduct a security assessment on their smart contracts beginning on July 16th, 2023 and ending on August 16th, 2023. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided four weeks for the engagement and assigned a full-time security engineer to verify the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that smart contract functions operate as intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified some security risks that were mostly addressed by the Moonwell Finance team.

1.3 TEST APPROACH & 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.
- Scanning of solidity files for vulnerabilities, security hot-spots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment. (Foundry)

2. 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.

2.1 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:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
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:

$$E = \prod m_e$$

2.2 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	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:H)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

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

2.3 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.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N)	1
	Partial (R:P)	0.5
	Full (R:F)	0.25
Scope (s)	Changed (S:C)	1.25
	Unchanged (S:U)	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
Informational	0 - 1.9

2.4 SCOPE

1. IN-SCOPE TREE & COMMIT :

The security assessment was scoped to the following contract:

- moonwell-contracts-v2

ASSESSMENTS :

1. ASSESSED COMMIT ID:

COMMIT ID : c39f98bdc9dd4e448ba585923034af1d47f74dfa

REMEDIATION COMMIT ID : 17fce574c46259cb22b8b6215b8b982169eb40e7

- MultiRewardDistributor.sol.
- WETHRouter.sol.
- TemporalGovernor.sol.
- ChainlinkCompositeOracle.sol.
- ChainlinkOracle.sol.

2. ASSESSED PULL REQUEST:

- moonwell-contracts-v2/pull/18

COMMIT ID : 08b984680f722fca1c60aab27a7a2715877638a7

REMEDIATION COMMIT ID : 8d1848c5344c12ffb0978721efcf7d44bf250867

- src/MWethDelegate.sol.
- src/MWethDelegate.sol.
- src/router/WETHRouter.sol.

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	2	3	5	12

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) SILENT FAILURE DURING TOKEN MINTING ON THE ROUTER CONTRACT	High (8.2)	SOLVED - 07/23/2023
(HAL-02) SILENT FAILURE DURING TOKEN REDEMPTION ON THE ROUTER CONTRACT	High (8.2)	SOLVED - 07/23/2023
(HAL-03) MINT WITH PERMIT CAN BE BROKEN WHEN USING TOKENS THAT DO NOT FOLLOW THE ERC2612 STANDARD	Medium (6.7)	SOLVED - 07/28/2023
(HAL-04) LACK OF END TIME VALIDATION LEADS TO WRONG MARKET INDEX CALCULATION ON THE NEW MARKETS	Medium (6.2)	SOLVED - 07/28/2023
(HAL-05) MISSING CHAIN ID AND RECEIVER ADDRESS VERIFICATION IN EXECUTEPROPOSAL() FUNCTION	Medium (5.9)	SOLVED - 07/23/2023
(HAL-06) WRONG EVENT IS EMITTED IN THE UPDATE BORROW SPEED FUNCTION	Low (3.4)	SOLVED - 07/28/2023
(HAL-07) EMISSIONCAP LACKS AN UPPER BOUND, LEADING TO POTENTIAL OVERFLOWS	Low (3.4)	RISK ACCEPTED
(HAL-08) UNRESTRICTED RECEIVE IN WETHROUTER ENABLES EXCESS REDEMPTIONS	Low (3.1)	SOLVED - 07/23/2023
(HAL-09) IMPLEMENTATIONS CAN BE INITIALIZED	Low (2.5)	SOLVED - 07/20/2023
(HAL-10) HARD-CODED MTOKEN ADDRESS IN WETHUNWRAPPER CONTRACT	Low (2.5)	SOLVED - 08/16/2023
(HAL-11) EVENT IS MISSING INDEXED FIELDS	Informational (0.0)	SOLVED - 07/28/2023
(HAL-12) FLOATING PRAGMA	Informational (0.0)	SOLVED - 07/24/2023
(HAL-13) USE CUSTOM ERRORS INSTEAD OF REVERT STRINGS TO SAVE GAS	Informational (0.0)	ACKNOWLEDGED
(HAL-14) INCREMENT/DECREMENT FOR LOOP VARIABLE IN AN UNCHECKED BLOCK	Informational (0.0)	ACKNOWLEDGED

(HAL-15) LACK OF A DOUBLE-STEP TRANSFEROWNERSHIP PATTERN	Informational (0.0)	ACKNOWLEDGED
(HAL-16) CACHE ARRAY LENGTH	Informational (0.0)	ACKNOWLEDGED
(HAL-17) REDUNDANT SAFE CAST	Informational (0.0)	SOLVED - 07/24/2023
(HAL-18) REVERT STRING SIZE OPTIMIZATION	Informational (0.0)	ACKNOWLEDGED
(HAL-19) NO NEED TO INITIALIZE VARIABLES WITH DEFAULT VALUES	Informational (0.0)	ACKNOWLEDGED
(HAL-20) RETURN VALUE NOT STORED	Informational (0.0)	SOLVED - 07/28/2023
(HAL-21) UNCONVENTIONAL IMPLEMENTATION OF SAFECAST	Informational (0.0)	ACKNOWLEDGED
(HAL-22) REQUIRE() / REVERT() STATEMENTS SHOULD HAVE DESCRIPTIVE REASON STRINGS	Informational (0.0)	SOLVED - 07/28/2023

FINDINGS & TECH DETAILS

4.1 (HAL-01) SILENT FAILURE DURING TOKEN MINTING ON THE ROUTER CONTRACT - HIGH (8.2)

Description:

The mTokenmint(msg.value); function, originating from Compound's ERC20 mToken contracts, is a call that does not revert on failure but returns an error code as a uint value instead. This behavior deviates from the standard expected of typical Solidity functions that revert on failure.

This non-standard behavior makes it difficult for calling contracts (like the one above) to correctly handle failures. As the above contract does not check the return value of mToken.mint(), failures in this function will not cause the overall transaction to revert.

This could lead to serious imbalances between the perceived balance of mTokens on the router contract and the actual supply of minted mTokens.

Code Location:

Listing 1

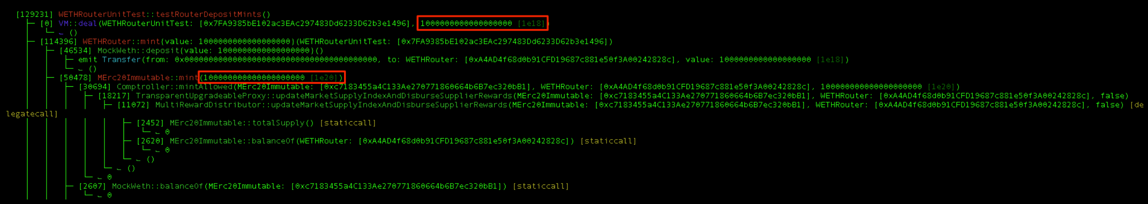
```
/// @notice Deposit ETH into the Moonwell protocol
/// @param recipient The address to receive the mToken
function mint(address recipient) external payable {
    weth.deposit{value: msg.value}();

    mToken.mint(msg.value);

    IERC20(address(mToken)).safeTransfer(
        recipient,
        mToken.balanceOf(address(this))
    );
}
```

Proof Of Concept:

Step 1 : An external actor calls the mint() function, sending some ETH along with the transaction.
Step 2 : The function attempts to convert the sent ETH to WETH by calling weth.deposit{value: msg.value}();.
Step 3 : The contract calls mToken.mint(msg.value);, but this operation fails for some reason. However, instead of reverting the transaction, mToken.mint() returns an error code.
Step 4 : Ignoring the failure from mToken.mint(), the contract proceeds to IERC20(address(mToken)).safeTransfer(recipient, mToken.balanceOf(address(this)));.



```
[129231] WETHRouterUnitTest::testRouterDepositMints()
  ├─ [0] VM::deal(WETHRouterUnitTest: [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496], 1000000000000000000 [1e18])
  │   └─ ← ()
  ├─ [114396] WETHRouter::mint{value: 1000000000000000000}(WETHRouterUnitTest: [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496])
  │   ├─ [46534] MockWeth::deposit{value: 1000000000000000000}()
  │   │   ├─ emit Transfer(from: 0x0000000000000000000000000000000000000000, to: WETHRouter: [0xA4AD4f68d0b91CFD19687c881e50f3A00242828c], value: 1000000000000000000 [1e18])
  │   │   └─ ← ()
  │   ├─ [50478] MErc20Immutable::mint(100000000000000000000 [1e20])
  │   │   ├─ [30694] Comptroller::mintAllowed(MErc20Immutable: [0xc7183455a4C133Ae270771860664b6B7ec320bB1], WETHRouter: [0xA4AD4f68d0b91CFD19687c881e50f3A00242828c], 100000000000000000000 [1e20])
  │   │   │   ├─ [18217] TransparentUpgradeableProxy::updateMarketSupplyIndexAndDisburseSupplierRewards(MErc20Immutable: [0xc7183455a4C133Ae270771860664b6B7ec320bB1], WETHRouter: [0xA4AD4f68d0b91CFD19687c881e50f3A00242828c], false)
  │   │   │   │   ├─ [11072] MultiRewardDistributor::updateMarketSupplyIndexAndDisburseSupplierRewards(MErc20Immutable: [0xc7183455a4C133Ae270771860664b6B7ec320bB1], WETHRouter: [0xA4AD4f68d0b91CFD19687c881e50f3A00242828c], false) [de
legatecall]
  │   │   │   │   │   ├─ [2452] MErc20Immutable::totalSupply() [staticcall]
  │   │   │   │   │   │   └─ ← 0
  │   │   │   │   │   ├─ [2620] MErc20Immutable::balanceOf(WETHRouter: [0xA4AD4f68d0b91CFD19687c881e50f3A00242828c]) [staticcall]
  │   │   │   │   │   │   └─ ← 0
  │   │   │   │   │   └─ ← ()
  │   │   │   │   └─ ← ()
  │   │   │   └─ ← 0
  │   │   ├─ [2607] MockWeth::balanceOf(MErc20Immutable: [0xc7183455a4C133Ae270771860664b6B7ec320bB1]) [staticcall]
  │   │   │   └─ ← 0
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:H/A:H/D:H/Y:H/R:P/S:C (8.2)

Recommendation:

Ensure that mToken.mint() is successful before transferring tokens.

Remediation Plan:

SOLVED: The Moonwell Finance team solved the issue by adding the return value validation.

Commit ID: c39f98bdc9dd4e448ba585923034af1d47f74dfa

4.2 (HAL-02) SILENT FAILURE DURING TOKEN REDEMPTION ON THE ROUTER CONTRACT - HIGH (8.2)

Description:

In the router contract, The redeem function aims to redeem mTokens equivalent to mTokenRedeemAmount. The call mToken.redeem(mTokenRedeemAmount); is responsible for the redemption action.

In the event of an error, the mToken.redeem() function from Compound's mToken contract does not revert, but instead returns a non-zero error code as an uint. This behavior deviates from the standard Solidity function behavior that typically reverts in case of an error.

The redeem() function in the MTOKEN contract does not check the return value of mToken.redeem(mTokenRedeemAmount);. If this redemption operation fails (returns a non-zero error code), the contract still proceeds with the remaining operations, leading to a silent failure. As a result, the contract behaves as if tokens were redeemed when they were not, creating a discrepancy between the actual and perceived balance of mtokens and eth.

Code Location:

Listing 2

```
    function redeem(uint256 mTokenRedeemAmount, address recipient) external {
        IERC20(address(mToken)).safeTransferFrom(
            msg.sender,
            address(this),
            mTokenRedeemAmount
        );

        mToken.redeem(mTokenRedeemAmount);

```

```
        weth.withdraw(weth.balanceOf(address(this)));

        (bool success, ) = payable(recipient).call{
            value: address(this).balance
        }("");
        require(success, "WETHRouter: ETH transfer failed");
    }
```

Proof Of Concept:

Step 1 : An external actor (say, an address ‘A’) calls the redeem() function with a certain mTokenRedeemAmount and recipient.

Step 2 : The function starts by transferring mTokenRedeemAmount of mTokens from ‘A’ to the contract itself. This is done via the IERC20(address(mToken)).safeTransferFrom(msg.sender, address(this), mTokenRedeemAmount); statement.

Step 3 : Next, the function attempts to redeem the mTokens that have just been transferred to the contract, using mToken.redeem(mTokenRedeemAmount);. But, for some reason, this redemption fails. In normal circumstances, this failure should cause the transaction to revert. However, due to the atypical behavior of the mToken.redeem() method (it does not revert on failure but returns a non-zero uint instead), the execution continues to the next line.

Step 4 : Now, the contract attempts to convert its entire WETH balance to ETH via weth.withdraw(weth.balanceOf(address(this)));. Since the redemption in step 3 failed, this step should not result in any additional ETH being added to the contract. However, let’s assume that the contract already had some ETH balance before the transaction began.

Step 5 : The contract then tries to transfer its entire ETH balance to the recipient specified in step 1. Despite the failed redemption, the function ends up transferring the contract’s existing ETH balance to the recipient.

BVSS:

AO:A/AC:L/AX:L/C:N/I:H/A:H/D:H/Y:H/R:P/S:C (8.2)

Recommendation:

Ensure that `mToken.redeem()` is successful before transferring tokens.

Remediation Plan:

SOLVED: The Moonwell Finance team solved the issue by adding the return value validation.

Commit ID: c39f98bdc9dd4e448ba585923034af1d47f74dfa

4.3 (HAL-03) MINT WITH PERMIT CAN BE BROKEN WHEN USING TOKENS THAT DO NOT FOLLOW THE ERC2612 STANDARD - MEDIUM (6.7)

Description:

In the mintWithPermit function, the implementation invokes the underlying token's permit() function and proceeds with the assumption that the operation was successful, without verifying the outcome. However, certain tokens may not adhere to the IERC20Permit standard. For example, the DAI Stablecoin utilizes a permit() function that deviates from the reference implementation. This lack of verification may lead to inconsistencies and unexpected behavior when interacting with non-conforming tokens.

Code Location:

Listing 3

```
function mintWithPermit(
    uint mintAmount,
    uint deadline,
    uint8 v, bytes32 r, bytes32 s
) override external returns (uint) {

    // Go submit our pre-approval signature data to the underlying token
    IERC20Permit(underlying).permit(
        msg.sender, address(this),
        mintAmount, deadline,
        v, r, s
    );
    (uint err,) = mintInternal(mintAmount);
    return err;
}
```

Proof Of Concept:

DAI Code

Listing 4

```
pragma solidity =0.5.12;

contract Dai is LibNote {

    // --- Approve by signature ---
    function permit(address holder, address spender, uint256 nonce, uint256 expiry,
                    bool allowed, uint8 v, bytes32 r, bytes32 s) external
    {
        bytes32 digest =
            keccak256(abi.encodePacked(
                "\x19\x01",
                DOMAIN_SEPARATOR,
                keccak256(abi.encode(PERMIT_TYPEHASH,
                                     holder,
                                     spender,
                                     nonce,
                                     expiry,
                                     allowed))
        ));

        require(holder != address(0), "Dai/invalid-address-0");
        require(holder == ecrecover(digest, v, r, s), "Dai/invalid-permit");
        require(expiry == 0 || now <= expiry, "Dai/permit-expired");
        require(nonce == nonces[holder]++, "Dai/invalid-nonce");
        uint wad = allowed ? uint(-1) : 0;
        allowance[holder][spender] = wad;
        emit Approval(holder, spender, wad);
    }
}
```

BVSS:

AO:A/AC:M/AX:M/C:L/I:C/A:N/D:L/Y:L/R:N/S:C (6.7)

Recommendation:

Add proper verification to the permit() function call. After calling the permit() function, ensure that the operation was successful before proceeding with the minting process.

Remediation Plan:

SOLVED: The Moonwell Finance team solved the issue by using the safePermit function.

4.4 (HAL-04) LACK OF END TIME VALIDATION LEADS TO WRONG MARKET INDEX CALCULATION ON THE NEW MARKETS - MEDIUM (6.2)

Description:

In the _addEmissionConfig function, which is responsible for creating new market emission configurations, there is no validation check for the _endTime parameter. This oversight may lead to the creation of markets with incorrect or unreasonable end times, resulting in wrong market index calculations and potentially impacting the overall functioning of the system.

Code Location:

Listing 5

```

    function _addEmissionConfig(
        MToken _mToken,
        address _owner,
        address _emissionToken,
        uint _supplyEmissionPerSec,
        uint _borrowEmissionsPerSec,
        uint _endTime
    ) external {
        requireComptrollersAdmin();

...
        MarketConfig memory config = MarketConfig({
            // Set the owner of the reward distributor config
            owner: _owner,
            // Set the emission token address
            emissionToken: _emissionToken,

            // Set the time that the emission campaign should end
 ↳ at
```

```
            endTime: _endTime,

            // Initialize the global supply
            supplyGlobalTimestamp: safe32(block.timestamp, "block timestamp exceeds 32 bits"),
            supplyGlobalIndex: initialIndexConstant,

            // Initialize the global borrow index + timestamp
            borrowGlobalTimestamp: safe32(block.timestamp, "block timestamp exceeds 32 bits"),
            borrowGlobalIndex: initialIndexConstant,

            // Set supply and reward borrow speeds
            supplyEmissionsPerSec: _supplyEmissionPerSec,
            borrowEmissionsPerSec: _borrowEmissionsPerSec
        });
...
    }
```

Proof Of Concept:

Listing 6

```
    function createDistributorWithRoundValuesAndConfig(uint tokensToMint, uint supplyEmissionsPerSecond, uint borrowEmissionsPerSecond) internal returns (MultiRewardDistributor distributor) {
        faucetToken.allocateTo(address(this), tokensToMint);
        faucetToken.approve(address(mToken), tokensToMint);

        MultiRewardDistributor distributorHarness = new MultiRewardDistributor(
            address(comptroller), address(this)
        );

        // 1 year of rewards
        uint endTime = block.timestamp - (60 * 60 * 24 * 365);

        // Add config + send emission tokens
        emissionToken.allocateTo(address(distributorHarness), 100e18);
        distributorHarness._addEmissionConfig(
```

```
            mToken,
            address(this),
            address(emissionToken),
            supplyEmissionsPerSecond,
            borrowEmissionsPerSecond,
            endTime
        );

        return distributorHarness;
    }
```

```
[PASS] testBorrowerHappyPath() (gas: 3436507)
Traces:
  [3436507] MultiRewardBorrowSideDistributorTests::testBorrowerHappyPath()
    ├─ [0] VM::warp(1678340000)
    │   └─ ← ()
    ├─ [48494] FaucetTokenWithPermit::allocateTo(MultiRewardBorrowSideDistributorTests: [0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84], 2000000000000000000)
    │   ├─ emit Transfer(from: 0x0000000000000000000000000000000000000000, to: MultiRewardBorrowSideDistributorTests: [0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84], value: 2000000000000000000)
    │   ├─ emit Transfer(from: FaucetTokenWithPermit: [0xEFc56627233b02eA95bAE7e19F648d7DcD5Bb132], to: MultiRewardBorrowSideDistributorTests: [0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84], value: 2000000000000000000)
    │   └─ ← ()
    ├─ [24673] FaucetTokenWithPermit::approve(MErc20Immutable: [0x42997aC9251E5BB0A61F4Ff790E5B991ea07Fd9B], 2000000000000000000)
    │   ├─ emit Approval(owner: MultiRewardBorrowSideDistributorTests: [0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84], spender: MErc20Immutable: [0x42997aC9251E5BB0A61F4Ff790E5B991ea07Fd9B], value: 2000000000000000000)
    │   └─ ← true
    ├─ [3040692] → new MultiRewardDistributor@0x1DD17af470f2CAa13D29c02AC190a3a1EdDc4e84
    │   ├─ [260] Comptroller::isComptroller() [staticcall]
    │   │   └─ ← true
    │   └─ ← 14715 bytes of code
    ├─ [48494] FaucetTokenWithPermit::allocateTo(MultiRewardDistributor: [0x1DD17af470f2CAa13D29c02AC190a3a1EdDc4e84], 100000000000000000000)
    │   ├─ emit Transfer(from: 0x0000000000000000000000000000000000000000, to: MultiRewardDistributor: [0x1DD17af470f2CAa13D29c02AC190a3a1EdDc4e84], value: 100000000000000000000)
    │   ├─ emit Transfer(from: FaucetTokenWithPermit: [0x0b7108E278c2E77E4e4f5c93d9E5e9A11AC837FC], to: MultiRewardDistributor: [0x1DD17af470f2CAa13D29c02AC190a3a1EdDc4e84], value: 100000000000000000000)
    │   └─ ← ()
    ├─ [189911] MultiRewardDistributor::_addEmissionConfig(MErc20Immutable: [0x42997aC9251E5BB0A61F4Ff790E5B991ea07Fd9B], MultiRewardBorrowSideDistributorTests: [0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84], FaucetTokenWithPermit: [0x0b7108E27
8c2E77E4e4f5c93d9E5e9A11AC837FC], 1000000000000000000, 500000000000000000, 1646804000)
    │   ├─ [2469] Comptroller::admin() [staticcall]
    │   │   └─ ← MultiRewardBorrowSideDistributorTests: [0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84]
    │   ├─ [4785] Comptroller::markets(MErc20Immutable: [0x42997aC9251E5BB0A61F4Ff790E5B991ea07Fd9B]) [staticcall]
    │   │   └─ ← true, 500000000000000000
    │   ├─ emit NewConfigCreated(mToken: MErc20Immutable: [0x42997aC9251E5BB0A61F4Ff790E5B991ea07Fd9B], owner: MultiRewardBorrowSideDistributorTests: [0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84], emissionToken: FaucetTokenWithPermit: [0x0b710
8E278c2E77E4e4f5c93d9E5e9A11AC837FC], supplySpeed: 1000000000000000000, borrowSpeed: 500000000000000000, endTime: 1646804000)
    │   └─ ← ()
    ├─ [4694] FaucetTokenWithPermit::allocateTo(MultiRewardDistributor: [0x1DD17af470f2CAa13D29c02AC190a3a1EdDc4e84], 1000000000000000000000000)
    │   ├─ emit Transfer(from: 0x0000000000000000000000000000000000000000, to: MultiRewardDistributor: [0x1DD17af470f2CAa13D29c02AC190a3a1EdDc4e84], value: 1000000000000000000000000)
    │   ├─ emit Transfer(from: FaucetTokenWithPermit: [0x0b7108E278c2E77E4e4f5c93d9E5e9A11AC837FC], to: MultiRewardDistributor: [0x1DD17af470f2CAa13D29c02AC190a3a1EdDc4e84], value: 1000000000000000000000000)
    │   └─ ← ()
    ├─ [24101] Comptroller::_setRewardDistributor(MultiRewardDistributor: [0x1DD17af470f2CAa13D29c02AC190a3a1EdDc4e84])
    │   ├─ emit NewRewardDistributor(oldRewardDistributor: 0x0000000000000000000000000000000000000000, newRewardDistributor: MultiRewardDistributor: [0x1DD17af470f2CAa13D29c02AC190a3a1EdDc4e84])
    │   └─ ← ()
    └─ ← ()
```

BVSS:

AO:A/AC:L/AX:L/C:M/I:M/A:L/D:H/Y:H/R:P/S:U (6.2)

Recommendation:

To address this issue, consider adding a validation check within the _addEmissionConfig function to ensure that the _endTime parameter is valid and reasonable. The check should verify that _endTime is greater than the current block timestamp.

Remediation Plan:

SOLVED: The Moonwell Finance team solved the issue by adding the _endTime validation.

4.5 (HAL-05) MISSING CHAIN ID AND RECEIVER ADDRESS VERIFICATION IN EXECUTEPROPOSAL() FUNCTION - MEDIUM (5.9)

Description:

The executeProposal() function in the current smart contract is responsible for parsing and verifying VAAs (Validators Aggregated Attestations) and then executing transactions based on these VAAs. The function does not verify the Chain ID or the receiver address (recipient of the transaction).

The absence of chain ID and receiver address verification could lead to significant security issues. Since the chain ID and recipient address are not checked, an attacker can craft a VAA to target an address on another chain, causing a cross-chain replay attack.

Code Location:

Listing 7

```
    function _executeProposal(bytes memory VAA, bool overrideDelay) private {
        // This call accepts single VAAs and headless VAAs
        (
            IWormhole.VM memory vm,
            bool valid,
            string memory reason
        ) = wormholeBridge.parseAndVerifyVM(VAA);

        require(valid, reason); /// ensure VAA parsing verification succeeded

        if (!overrideDelay) {
            require(
                queuedTransactions[vm.hash].queueTime != 0,
```

```
                "TemporalGovernor: tx not queued"
            );
            require(
                queuedTransactions[vm.hash].queueTime +
↳ proposalDelay <=
                    block.timestamp,
                "TemporalGovernor: timelock not finished"
            );
        } else if (queuedTransactions[vm.hash].queueTime == 0) {
            /// if queue time is 0 due to fast track execution,
↳ set it to current block timestamp
            queuedTransactions[vm.hash].queueTime = block.
↳ timestamp.toUint248();
        }



        require(
            !queuedTransactions[vm.hash].executed,
            "TemporalGovernor: tx already executed"
        );

        queuedTransactions[vm.hash].executed = true;

        address[] memory targets; /// contracts to call
        uint256[] memory values; /// native token amount to send
        bytes[] memory calldatas; /// calldata to send
        (, targets, values, calldatas) = abi.decode(
            vm.payload,
            (address, address[], uint256[], bytes[])
        );

        /// Interaction (s)

        _sanityCheckPayload(targets, values, calldatas);

        for (uint256 i = 0; i < targets.length; i++) {
            address target = targets[i];
            uint256 value = values[i];
            bytes memory data = calldatas[i];

            // Go make our call, and if it is not successful
↳ revert with the error bubbling up
            (bool success, bytes memory returnData) = target.call{
```

```
↳ value: value}(
                    data
                );

                /// revert on failure with error message if any
                require(success, string(returnData));

                emit ExecutedTransaction(target, value, data);
            }
        }
```

Proof Of Concept:

Step 1 : An attacker crafts a wormhole message that appears to be valid but is intended for a different chain (different chain ID) or is directed to an unintended recipient address.

Step 2 : The attacker submits this crafted payload to the _executeProposal() function in the smart contract.

Step 3 : Since there are no checks in place for the chain ID or recipient address, the function treats the VAA as valid and begins to execute the transaction(s) specified in the VAA payload.

BVSS:

AO:A/AC:L/AX:L/C:N/I:H/A:H/D:N/Y:N/R:P/S:C (5.9)

Recommendation:

Consider checking emitter Chain id, receiver on the function.

Remediation Plan:

SOLVED: The Moonwell Finance team solved the issue by adding the necessary validations.

Commit ID: c39f98bdc9dd4e448ba585923034af1d47f74dfa

4.6 (HAL-06) WRONG EVENT IS EMITTED IN THE UPDATE BORROW SPEED FUNCTION - LOW (3.4)

Description:

The _updateBorrowSpeed function in the smart contract is responsible for updating the borrow emission speed for the specified MToken and emissionToken. However, it has been identified that the wrong event is being emitted at the end of the function. The current implementation emits the NewSupplyRewardSpeed event instead of the expected NewBorrowRewardSpeed event.

This discrepancy can lead to confusion and incorrect data being captured by event listeners, potentially impacting the system's overall efficiency, accuracy, and traceability.

Code Location:

Listing 8

```
    /// @notice Update the borrow emissions for a given mtoken, emission token pair.
    function _updateBorrowSpeed(MToken _mToken, address _emissionToken, uint _newBorrowSpeed) public {
        MarketEmissionConfig storage emissionConfig = fetchConfigByEmissionToken(_mToken, _emissionToken);

        // Safety check this is the owner or the admin
        requireEmissionConfigOwnerOrAdmin(emissionConfig);

        uint currentBorrowSpeed = emissionConfig.config.borrowEmissionsPerSec;

        require(_newBorrowSpeed != currentBorrowSpeed, "Can't set new borrow emissions to be equal to current!");
        require(_newBorrowSpeed < emissionCap, "Cannot set a borrow reward speed higher than the emission cap!");
```

```

        // Make sure we update our indices before setting the new speed
        updateMarketBorrowIndexInternal(_mToken);

        // Update borrow speed
        emissionConfig.config.borrowEmissionsPerSec = _newBorrowSpeed;

        emit NewSupplyRewardSpeed(_mToken, _emissionToken, currentBorrowSpeed, _newBorrowSpeed);
    }
```

BVSS:

AO:A/AC:L/AX:L/C:M/I:M/A:L/D:N/Y:N/R:P/S:U (3.4)

Recommendation:

To address this issue, It is recommended to update the _updateBorrowSpeed function to emit the correct event, NewBorrowRewardSpeed, instead of the current NewSupplyRewardSpeed event.

Remediation Plan:

SOLVED: The Moonwell Finance team solved the issue by changing the event.

4.7 (HAL-07) EMISSIONCAP LACKS AN UPPER BOUND, LEADING TO POTENTIAL OVERFLOWS - LOW (3.4)

Description:

The emissionCap variable in the given smart contract does not have an upper bound in the _setEmissionCap function. By default, the emission cap is set to 100 * 10^18 tokens per second to avoid unbounded computation/multiplication overflows. However, the function _setEmissionCap allows changing the emissionCap value without any restriction on the upper limit, which can potentially lead to overflows and other unexpected issues in the contract's execution.

Code Location:

Listing 9

```
function _setEmissionCap(uint _newEmissionCap) external {
    requireComptrollersAdmin();

    uint oldEmissionCap = emissionCap;

    emissionCap = _newEmissionCap;

    emit NewEmissionCap(oldEmissionCap, _newEmissionCap);
}
```

BVSS:

AO:A/AC:L/AX:L/C:M/I:M/A:L/D:N/Y:N/R:P/S:U (3.4)

Recommendation:

To address this issue, consider implementing an upper bound for the emissionCap. Modify the _setEmissionCap function to include an additional require statement that checks if the new value for the emission cap is within a predefined safe range.

Remediation Plan:

RISK ACCEPTED: The Moonwell Finance team accepted the risk of this issue.

4.8 (HAL-08) UNRESTRICTED RECEIVE IN WETHROUTER ENABLES EXCESS REDEMPTIONS - LOW (3.1)

Description:

The redeem() function in the current design of the WETHRouter smart contract is designed to handle the redemption of mToken and subsequent withdrawal of WETH. However, this function does not restrict the receipt of tokens to only WETH/mToken. As a result, any native token sent directly to the WETHRouter contract will be sent to the first redeemer.

In this setup, an unintentional or malicious transfer of arbitrary tokens to the WETHRouter contract could lead to an unexpected balance increase. When the redeem() function is called, it attempts to withdraw all ETH equivalent in the contract and sends it to the recipient. If an arbitrary amount of tokens or native ETH is sent to the contract, it would inflate the balance available for withdrawal, making it retrievable by the first redeemer.

Code Location:

Listing 10

```
    function redeem(uint256 mTokenRedeemAmount, address recipient)
 ↳ external {
        IERC20(address(mToken)).safeTransferFrom(
            msg.sender,
            address(this),
            mTokenRedeemAmount
        );

        mToken.redeem(100 ether);

        weth.withdraw(weth.balanceOf(address(this)));

        (bool success, ) = payable(recipient).call{
            value: address(this).balance
```

```
        }("");
        require(success, "WETHRouter: ETH transfer failed");
    }

    receive() external payable {}
```

BVSS:

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

Recommendation:

A potential solution could be to add a mechanism that isolates the withdrawal of mToken-generated WETH from the withdrawal of other tokens that might be sent to the contract. This could be achieved by storing the contract's balance before and after the mToken redemption and only allowing the withdrawal of the difference.

Remediation Plan:

SOLVED: The Moonwell Finance team solved the issue by adding an address validation.

4.9 (HAL-09) IMPLEMENTATIONS CAN BE INITIALIZED - LOW (2.5)

Description:

The contracts are upgradable, inheriting from the Initializable contract. However, the current implementations are missing the _disableInitializers () function call in the constructors. Thus, an attacker can initialize the implementation. Usually, the initialized implementation has no direct impact on the proxy itself; however, it can be exploited in a phishing attack. In rare cases, the implementation might be mutable and may have an impact on the proxy.

BVSS:

AO:A/AC:L/AX:M/C:N/I:L/A:N/D:L/Y:L/R:N/S:U (2.5)

Recommendation:

It is recommended to call _disableInitializers within the contract's constructor to prevent the implementation from being initialized.

Remediation Plan:

SOLVED: The contracts now implement the _disableInitializers() function call in the constructors.

Commit ID: c39f98bdc9dd4e448ba585923034af1d47f74dfa

4.10 (HAL-10) HARD-CODED MTOKEN ADDRESS IN WETHUNWRAPPER CONTRACT - LOW (2.5)

Description:

The WethUnwrapper contract contains a hard-coded mToken address (0x628ff693426583D9a7FB391E54366292F509D457). This design pattern can create limitations, particularly in scenarios where cross-chain functionality or upgradability is desired.

Hard-coding an address within a contract can lead to a lack of flexibility, making it challenging to adapt to changes or to interact with different instances of a token on various chains. In a cross-chain environment, where tokens might be represented on multiple blockchains, having a fixed address can hinder the ability to seamlessly interact across different networks.

Code Location:

/src/WethUnwrapper.sol#L8

Listing 11

```
contract WethUnwrapper {
    /// @notice the mToken address
    address public constant mToken = 0x628ff693426583D9a7FB391E54366292F509D457;

    /// @notice reference to the WETH contract
    address public immutable weth;

    /// @notice construct a new WethUnwrapper
    /// @param _weth the WETH contract address
    constructor(address _weth) {
        weth = _weth;
    }
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:L/A:N/D:L/Y:L/R:N/S:U (2.5)

Recommendation:

To enhance the contract's flexibility and enable cross-chain compatibility, consider implementing a mechanism to dynamically set the mToken address. This could be achieved through a constructor, a setter function, or other means of configuration controlled by appropriate permissions. By allowing the mToken address to be set or updated, the contract can more easily adapt to different environments and interact with corresponding tokens across various chains.

Remediation Plan:

SOLVED: The Moonwell Finance team solved the issue by defining the variable as immutable.

Commit ID: 8d1848c5344c12ffb0978721efcf7d44bf250867

4.11 (HAL-11) EVENT IS MISSING INDEXED FIELDS - INFORMATIONAL (0.0)

Description:

Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields).

Code Location:

Listing 12

```
File: MultiRewardDistributorCommon.sol

61: event GlobalSupplyIndexUpdated(MToken mToken, address
↳ emissionToken, uint newSupplyIndex, uint32
↳ newSupplyGlobalTimestamp);
62: event GlobalBorrowIndexUpdated(MToken mToken, address
↳ emissionToken, uint newIndex, uint32 newTimestamp);
65: event DisbursedSupplierRewards(MToken mToken, address supplier
↳ , address emissionToken, uint totalAccrued);
66: event DisbursedBorrowerRewards(MToken mToken, address borrower
↳ , address emissionToken, uint totalAccrued);
69: event NewConfigCreated(MToken mToken, address owner, address
↳ emissionToken, uint supplySpeed, uint borrowSpeed, uint endTime);
70: event NewPauseGuardian(address oldPauseGuardian, address
↳ newPauseGuardian);
71: event NewEmissionCap(uint oldEmissionCap, uint newEmissionCap)
↳ ;
72: event NewEmissionConfigOwner(MToken mToken, address
↳ emissionToken, address currentOwner, address newOwner);
73: event NewRewardEndTime(MToken mToken, address emissionToken,
↳ uint currentEndTime, uint newEndTime);
74: event NewSupplyRewardSpeed(MToken mToken, address
↳ emissionToken, uint oldRewardSpeed, uint newRewardSpeed);
75: event NewBorrowRewardSpeed(MToken mToken, address
↳ emissionToken, uint oldRewardSpeed, uint newRewardSpeed);
76: event FundsRescued(address token, uint amount);
```

```
15 83: event InsufficientTokensToEmit(address payable user, address
↳ rewardToken, uint amount);
```

BVSS:

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

Recommendation:

Add index on the events.

Remediation Plan:

SOLVED: The Moonwell Finance team solved the issue by adding the indexed keyword on the events.

4.12 (HAL-12) FLOATING PRAGMA - INFORMATIONAL (0.0)

Description:

The project contains many instances of floating pragma. Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, either an outdated compiler version that might introduce bugs that affect the contract system negatively or a pragma version too recent which has not been extensively tested.

Code Location:

The ChainlinkCompositeOracle is affected. (^0.8.0)

BVSS:

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

Recommendation:

Consider locking the pragma version with known bugs for the compiler version by removing the caret (^) symbol. When possible, do not use floating pragma in the final live deployment. Specifying a fixed compiler version ensures that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Remediation Plan:

SOLVED: The Moonwell Finance team solved the issue by locking the pragma.

Commit ID: 17fce574c46259cb22b8b6215b8b982169eb40e7

4.13 (HAL-13) USE CUSTOM ERRORS INSTEAD OF REVERT STRINGS TO SAVE GAS - INFORMATIONAL (0.0)

Description:

Custom errors are available from solidity version 0.8.4. Custom errors save ~50 gas each time they're hit by avoiding having to allocate and store the revert string. Not defining the strings also saves deployment gas.

BVSS:

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

Recommendation:

Consider replacing all revert strings with custom errors.

Remediation Plan:

ACKNOWLEDGED: The Moonwell Finance team acknowledged this finding.

4.14 (HAL-14) INCREMENT/DECREMENT FOR LOOP VARIABLE IN AN UNCHECKED BLOCK - INFORMATIONAL (0.0)

Description:

i++ involves checked arithmetic, which is not required. This is because the value of i is always strictly less than length <= 2**256 - 1. Therefore, the theoretical maximum value of i to enter the for-loop body is 2**256 - 2. This means that the i++ in the for loop can never overflow. Regardless, the compiler performs the overflow checks.

Code Location:

Listing 13

```
File: core/Governance/TemporalGovernor.sol

60:           for (uint256 i = 0; i < _trustedSenders.length; i++) {

103:              for (uint256 i = 0; i < trustedSendersList.length; i++) {

132:              for (uint256 i = 0; i < _trustedSenders.length; i++) {

159:              for (uint256 i = 0; i < _trustedSenders.length; i++) {

380:           for (uint256 i = 0; i < targets.length; i++) {
```

Listing 14

```
File: core/MultiRewardDistributor/MultiRewardDistributor.sol

209:           for (uint256 index = 0; index < configs.length; index++) {

```

```
240:          for (uint256 index = 0; index < markets.length; index++) {

281:          for (uint256 index = 0; index < configs.length; index++) {

419:          for (uint256 index = 0; index < configs.length; index++) {

983:          for (uint256 index = 0; index < configs.length; index++) {

1009:          for (uint256 index = 0; index < configs.length; index++) {

1053:          for (uint256 index = 0; index < configs.length; index++) {

1112:          for (uint256 index = 0; index < configs.length; index++) {

1163:          for (uint256 index = 0; index < configs.length; index++) {
```

BVSS:

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

Recommendation:

Consider incrementing the for loop variable in an unchecked block.

Remediation Plan:

ACKNOWLEDGED: The Moonwell Finance team acknowledged this finding.

4.15 (HAL-15) LACK OF A DOUBLE-STEP TRANSFEROWNERSHIP PATTERN - INFORMATIONAL (0.0)

Description:

The current ownership transfer process for TemporalGovernor contract inheriting from Ownable or OwnableUpgradeable involves the current owner calling the transferOwnership() function:

Listing 15: Ownable.sol

```
function transferOwnership(address newOwner) public virtual onlyOwner {
    require(newOwner != address(0), "Ownable: new owner is the zero address");
    _setOwner(newOwner);
}
```

If the nominated EOA account is not a valid account, it is entirely possible that the owner may accidentally transfer ownership to an uncontrolled account, losing the access to all functions with the onlyOwner modifier.

BVSS:

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

Recommendation:

It is recommended to implement a two-step process where the owner nominates an account and the nominated account needs to call an acceptOwnership() function for the transfer of the ownership to fully succeed. This ensures the nominated EOA account is a valid and active account. This can be easily achieved by using OpenZeppelin's Ownable2Step contract instead of Ownable:

Listing 16: Ownable2Step.sol (Lines 52-56)

```
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.8.0) (access/Ownable2Step.sol)

pragma solidity ^0.8.0;

import "./Ownable.sol";

/**
 * @dev Contract module which provides access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership} and {acceptOwnership}.
 *
 * This module is used through inheritance. It will make available all functions
 * from parent (Ownable).
 */
abstract contract Ownable2Step is Ownable {
    address private _pendingOwner;

    event OwnershipTransferStarted(address indexed previousOwner, address indexed newOwner);

    /**
     * @dev Returns the address of the pending owner.
     */
    function pendingOwner() public view virtual returns (address) {
        return _pendingOwner;
    }

    /**
     * @dev Starts the ownership transfer of the contract to a new account. Replaces the pending transfer if there is one.
     * Can only be called by the current owner.
     */
```

```
    function transferOwnership(address newOwner) public virtual override onlyOwner {
        _pendingOwner = newOwner;
        emit OwnershipTransferStarted(owner(), newOwner);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`) and deletes any pending owner.
     * Internal function without access restriction.
     */
    function _transferOwnership(address newOwner) internal virtual override {
        delete _pendingOwner;
        super._transferOwnership(newOwner);
    }

    /**
     * @dev The new owner accepts the ownership transfer.
     */
    function acceptOwnership() external {
        address sender = _msgSender();
        require(pendingOwner() == sender, "Ownable2Step: caller is not the new owner");
        _transferOwnership(sender);
    }
}
```

Remediation Plan:

ACKNOWLEDGED: The Moonwell Finance team acknowledged this finding.

4.16 (HAL-16) CACHE ARRAY LENGTH - INFORMATIONAL (0.0)

Description:

In a for loop, the length of an array can be put in a temporary variable to save some gas. This has been done already in several other locations in the code.

In the above case, the solidity compiler will always read the length of the array during each iteration. That is,

- if it is a storage array, this is an extra sload operation (100 additional extra gas (EIP-2929) for each iteration except for the first),
- if it is a memory array, this is an extra mload operation (3 additional gas for each iteration except for the first),
- if it is a calldata array, this is an extra calldataload operation (3 additional gas for each iteration except for the first)

Code Location:

Listing 17

```
File: core/Governance/TemporalGovernor.sol

60:         for (uint256 i = 0; i < _trustedSenders.length; i++) {

103:             for (uint256 i = 0; i < trustedSendersList.length
↳ ; i++) {

132:             for (uint256 i = 0; i < _trustedSenders.length; i
↳ ++) {

159:             for (uint256 i = 0; i < _trustedSenders.length; i
↳ ++) {

380:          for (uint256 i = 0; i < targets.length; i++) {
```

Listing 18

```
File: core/MultiRewardDistributor/MultiRewardDistributor.sol

209:            for (uint256 index = 0; index < configs.length; index++) {

240:            for (uint256 index = 0; index < markets.length; index++) {

281:            for (uint256 index = 0; index < configs.length; index++) {

419:            for (uint256 index = 0; index < configs.length; index++) {

983:            for (uint256 index = 0; index < configs.length; index++) {

1009:            for (uint256 index = 0; index < configs.length; index++) {

1053:            for (uint256 index = 0; index < configs.length; index++) {

1112:            for (uint256 index = 0; index < configs.length; index++) {

1163:            for (uint256 index = 0; index < configs.length; index++) {
```

BVSS:

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

Recommendation:

In a for loop, store the length of an array in a temporary variable.

Remediation Plan:

ACKNOWLEDGED: The Moonwell Finance team acknowledged this finding.

4.17 (HAL-17) REDUNDANT SAFE CAST - INFORMATIONAL (0.0)

Description:

The TemporalGovernor contract uses the OpenZeppelin SafeCast library for type conversion operations. However, it is important to note that there is no possibility of an overflow occurring in the variable through the utilization of block.timestamp.

Code Location:

Listing 19

```
contract TemporalGovernor is ITemporalGovernor, Ownable, Pausable
↳ {
    using SafeCast for *;
}
```

BVSS:

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

Recommendation:

It is recommended to remove the unnecessary SafeCast library.

Remediation Plan:

SOLVED: The Moonwell Finance team solved the issue by removing safecast.

Commit ID: 17fce574c46259cb22b8b6215b8b982169eb40e7

4.18 (HAL-18) REVERT STRING SIZE OPTIMIZATION - INFORMATIONAL (0.0)

Description:

Shortening revert strings to fit in 32 bytes will decrease deploy time gas and will decrease runtime gas when the revert condition has been met.

Code Location:

Listing 20

```
    function setTrustedSenders(
        TrustedSender[] calldata _trustedSenders
    ) external {
        require(
            msg.sender == address(this),
            "TemporalGovernor: Only this contract can update
    ↳ trusted senders"
        );
}
```

BVSS:

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

Recommendation:

Shorten the revert strings to fit in 32 bytes. Alternatively, the code could be modified to use custom errors, introduced in Solidity 0.8.4.

Remediation Plan:

ACKNOWLEDGED: The Moonwell Finance team acknowledged this finding.

4.19 (HAL-19) NO NEED TO INITIALIZE VARIABLES WITH DEFAULT VALUES - INFORMATIONAL (0.0)

Description:

Initialization to 0 or false is not necessary, as these are the default values in Solidity.

Code Location:

Listing 21

```
File: core/Governance/TemporalGovernor.sol

60:         for (uint256 i = 0; i < _trustedSenders.length; i++) {

103:            for (uint256 i = 0; i < trustedSendersList.length; i++) {

132:            for (uint256 i = 0; i < _trustedSenders.length; i++) {

159:            for (uint256 i = 0; i < _trustedSenders.length; i++) {

380:        for (uint256 i = 0; i < targets.length; i++) {
```

Listing 22

```
File: core/MultiRewardDistributor/MultiRewardDistributor.sol

209:        for (uint256 index = 0; index < configs.length; index++) {

240:        for (uint256 index = 0; index < markets.length; index++) {

```

```
281:        for (uint256 index = 0; index < configs.length; index++) {

419:        for (uint256 index = 0; index < configs.length; index++) {

983:        for (uint256 index = 0; index < configs.length; index++) {

1009:        for (uint256 index = 0; index < configs.length; index++) {

1053:        for (uint256 index = 0; index < configs.length; index++) {

1112:        for (uint256 index = 0; index < configs.length; index++) {

1163:        for (uint256 index = 0; index < configs.length; index++) {
```

BVSS:

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

Recommendation:

Remove the initialization values of 0 or false.

Remediation Plan:

ACKNOWLEDGED: The Moonwell Finance team acknowledged this finding.

4.20 (HAL-20) RETURN VALUE NOT STORED - INFORMATIONAL (0.0)

Description:

The return value of an external call is not stored in a local or state variable.

Code Location:

Listing 23

```
    function _supportMarket(MToken mToken) external returns (uint) {
        if (msg.sender != admin) {
            return fail(Error.UNAUTHORIZED, FailureInfo.SUPPORT_MARKET_OWNER_CHECK);
        }

        if (markets[address(mToken)].isListed) {
            return fail(Error.MARKET_ALREADY_LISTED, FailureInfo.SUPPORT_MARKET_EXISTS);
        }

        mToken.isMToken(); // Sanity check to make sure its really a MToken

        Market storage newMarket = markets[address(mToken)];
        newMarket.isListed = true;
        newMarket.collateralFactorMantissa = 0;

        _addMarketInternal(address(mToken));

        emit MarketListed(mToken);

        return uint(Error.NO_ERROR);
    }
```

BVSS:

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

Recommendation:

It is recommended adding require statement for isMToken:

Listing 24

```
require(mToken.isMToken(), "Must be an MToken");
```

Remediation Plan:

SOLVED: The Moonwell Finance team solved the issue by using require statement.

4.21 (HAL-21) UNCONVENTIONAL IMPLEMENTATION OF SAFECAST - INFORMATIONAL (0.0)

Description:

The max value of the target type is usually allowed in SafeCast.

Code Location:

Listing 25

```
    function safe224(uint n, string memory errorMessage) pure internal returns (uint224) {
        require(n < 2**224, errorMessage);
        return uint224(n);
    }

    function safe32(uint n, string memory errorMessage) pure internal returns (uint32) {
        require(n < 2**32, errorMessage);
        return uint32(n);
    }
```

BVSS:

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

Recommendation:

Consider changing upper bounds.

SafeCast.sol#L32

Listing 26

```
    function toUint224(uint256 value) internal pure returns (uint224) {
        require(value <= type(uint224).max, "SafeCast: value doesn't fit in 224 bits");
        return uint224(value);
    }

    /**
     * @dev Returns the downcasted uint128 from uint256, reverting on
     * overflow (when the input is greater than largest uint128).
     *
     * Counterpart to Solidity's `uint128` operator.
     *
     * Requirements:
     *
     * - input must fit into 128 bits
     */
    function toUint128(uint256 value) internal pure returns (uint128) {
        require(value <= type(uint128).max, "SafeCast: value doesn't fit in 128 bits");
        return uint128(value);
    }
```

Remediation Plan:

ACKNOWLEDGED: The Moonwell Finance team acknowledged this finding.

4.22 (HAL-22) REQUIRE() / REVERT() STATEMENTS SHOULD HAVE DESCRIPTIVE REASON STRINGS - INFORMATIONAL (0.0)

Description:

In the current smart contract implementation, several require() and revert() statements lack descriptive reason strings. These reason strings serve as informative error messages that help developers and users understand the cause of a failed transaction or function call. Omitting these strings can result confused when diagnosing issues or debugging the smart contract, as the cause of the failure may not be immediately apparent.

Code Location:

Listing 27

```
    function _updateEndTime(MToken _mToken, address _emissionToken, uint _newEndTime) public {
        MarketEmissionConfig storage emissionConfig = fetchConfigByEmissionToken(_mToken, _emissionToken);

        // Safety check this is the owner or the admin
        requireEmissionConfigOwnerOrAdmin(emissionConfig);

        uint currentEndTime = emissionConfig.config.endTime;

        // Must be older than our existing end time AND the current block
        require(_newEndTime > currentEndTime);
        require(_newEndTime > block.timestamp);

        // Update both global indices before setting the new end time. If rewards are off this just updates the
        // global block timestamp to the current second
        updateMarketBorrowIndexInternal(_mToken);
        updateMarketSupplyIndexInternal(_mToken);

        emissionConfig.config.endTime = _newEndTime;
```

```
        emit NewRewardEndTime(_mToken, _emissionToken,
↳ currentEndTime, _newEndTime);
    }
```

BVSS:

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

Recommendation:

Consider defining descriptive strings.

Remediation Plan:

SOLVED: The Moonwell Finance team solved the issue by adding descriptive strings.

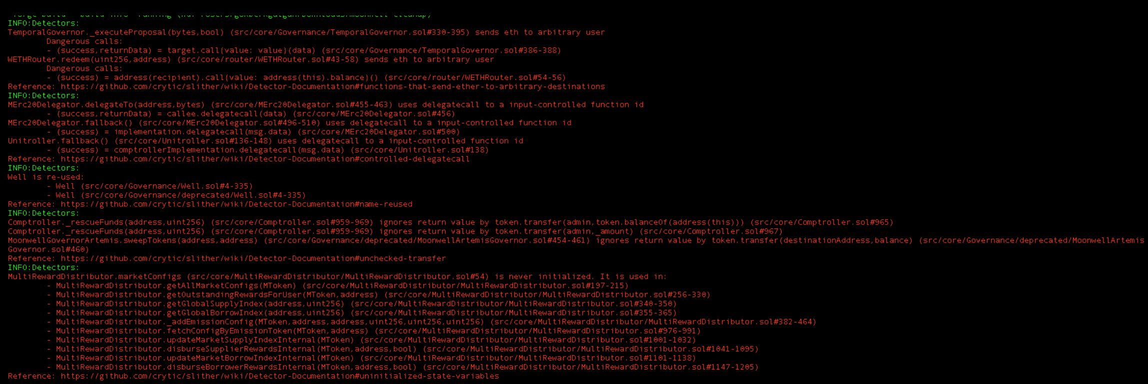
AUTOMATED TESTING

5.1 STATIC ANALYSIS REPORT

Description:

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.

Results:



- No major issues found by Slither.

5.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the smart contracts and sent the compiled results to the analyzers in order to locate any vulnerabilities.

Results:

- No major issues were found by MythX.

THANK YOU FOR CHOOSING

HALBORN