// Security Assessment

03.06.2025 - 03.06.2025

Curve Multirewards

Moonwell

HALBORN

Curve Multirewards - Moonwell

Prepared by: HALBORN

Last Updated 03/17/2025

Date of Engagement by: March 6th, 2025 - March 6th, 2025

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	0	0	0	0	1

TABLE OF CONTENTS

1. Introduction

`Moonwell` engaged `Halborn` to conduct a security assessment on their smart contracts beginning on March 6th, 2025 and ending on March 7th. 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 three 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 one improvement to reduce the likelihood and impact of risks, which was acknowledged by the `Moonwell team`.

- Update the Solidity compiler version to a current one - 0.8.x or equivalent.

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:

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

EXPLOITABILITY 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

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
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
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: moonwell-contracts-v2

(b) Assessed Commit ID: 99700a5

(c) Items in scope:

- crv-rewards/MultiRewards.sol

Out-of-Scope: Third-party dependencies and economic attacks.

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	0	0

INFORMATIONAL
1

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
OUTDATED SOLIDITY COMPILER VERSION	INFORMATIONAL	ACKNOWLEDGED - 03/13/2025

7. FINDINGS & TECH DETAILS

7.1 OUTDATED SOLIDITY COMPILER VERSION

// INFORMATIONAL

Description

The `MultiRewards` smart contract is written under an older Solidity compiler version (pragma solidity 0.5.17). There are some known issues with this compiler version, as indicated by Slither results.

Newer compiler releases, particularly 0.8.x, include additional features, optimizations and built-in safety checks, such safe arithmetic.

Remaining on an older compiler version can potentially miss out on these enhancements and any relevant bug fixes introduced in newer releases. It means the code will not benefit from the latest language-level security improvements and best practices.

BVSS

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

Recommendation

It is recommended to update the Solidity compiler version to the current 0.8.x. Adjust the test scenarios if needed to ensure the contract's functionality remains correct under new compiler settings.

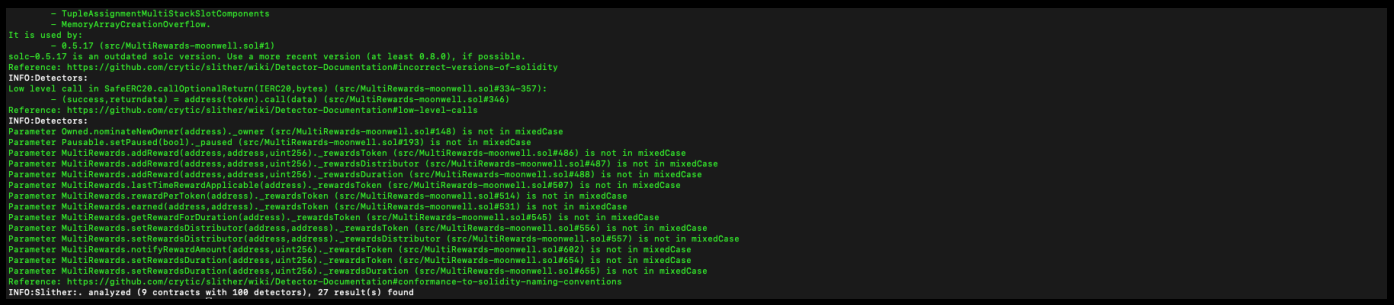
Remediation Comment

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

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.

```
INFO:Detectors:
Reentrancy in MultiRewards.exit() (src/MultiRewards-moonwell.sol#594-597):
        External calls:
        - withdraw(_balances[msg.sender]) (src/MultiRewards-moonwell.sol#595)
                - (success,returndata) = address(token).call(data) (src/MultiRewards-moonwell.sol#346)
                - stakingToken.safeTransfer(msg.sender,amount) (src/MultiRewards-moonwell.sol#578)
        - getReward() (src/MultiRewards-moonwell.sol#596)
                - (success,returndata) = address(token).call(data) (src/MultiRewards-moonwell.sol#346)
                - IERC20(_rewardsToken).safeTransfer(msg.sender,reward) (src/MultiRewards-moonwell.sol#588)
        State variables written after the call(s):
        - getReward() (src/MultiRewards-moonwell.sol#596)
                - _guardCounter += 1 (src/MultiRewards-moonwell.sol#240)
        ReentrancyGuard._guardCounter (src/MultiRewards-moonwell.sol#224) can be used in cross function reentrancies:
        - ReentrancyGuard.nonReentrant() (src/MultiRewards-moonwell.sol#239-247)
        - getReward() (src/MultiRewards-moonwell.sol#596)
                - rewardData[token].rewardPerTokenStored = rewardPerToken(token) (src/MultiRewards-moonwell.sol#675)
                - rewardData[token].lastUpdateTime = lastTimeRewardApplicable(token) (src/MultiRewards-moonwell.sol#676)
        MultiRewards.rewardData (src/MultiRewards-moonwell.sol#468) can be used in cross function reentrancies:
        - MultiRewards.addReward(address,address,uint256) (src/MultiRewards-moonwell.sol#485-494)
        - MultiRewards.getRewardForDuration(address) (src/MultiRewards-moonwell.sol#544-551)
        - MultiRewards.lastTimeRewardApplicable(address) (src/MultiRewards-moonwell.sol#506-511)
        - MultiRewards.notifyRewardAmount(address,uint256) (src/MultiRewards-moonwell.sol#601-635)
        - MultiRewards.rewardData (src/MultiRewards-moonwell.sol#468)
        - MultiRewards.rewardPerToken(address) (src/MultiRewards-moonwell.sol#513-527)
        - MultiRewards.setRewardsDistributor(address,address) (src/MultiRewards-moonwell.sol#555-560)
        - MultiRewards.setRewardsDuration(address,uint256) (src/MultiRewards-moonwell.sol#653-668)
        - MultiRewards.updateReward(address) (src/MultiRewards-moonwell.sol#672-684)
        - getReward() (src/MultiRewards-moonwell.sol#596)
                - rewards[msg.sender][_rewardsToken] = 0 (src/MultiRewards-moonwell.sol#587)
                - rewards[account][token] = earned(account,token) (src/MultiRewards-moonwell.sol#678)
        MultiRewards.rewards (src/MultiRewards-moonwell.sol#474) can be used in cross function reentrancies:
        - MultiRewards.earned(address,address) (src/MultiRewards-moonwell.sol#529-542)
        - MultiRewards.getReward() (src/MultiRewards-moonwell.sol#582-592)
        - MultiRewards.rewards (src/MultiRewards-moonwell.sol#474)
        - MultiRewards.updateReward(address) (src/MultiRewards-moonwell.sol#672-684)
        - getReward() (src/MultiRewards-moonwell.sol#596)
                - userRewardPerTokenPaid[account][token] = rewardData[token].rewardPerTokenStored (src/MultiRewards-moonwell.sol#679-680)
        MultiRewards.userRewardPerTokenPaid (src/MultiRewards-moonwell.sol#472-473) can be used in cross function reentrancies:
        - MultiRewards.earned(address,address) (src/MultiRewards-moonwell.sol#529-542)
        - MultiRewards.updateReward(address) (src/MultiRewards-moonwell.sol#672-684)
        - MultiRewards.userRewardPerTokenPaid (src/MultiRewards-moonwell.sol#472-473)
Reentrancy in MultiRewards.getReward() (src/MultiRewards-moonwell.sol#582-592):
        External calls:
        - IERC20(_rewardsToken).safeTransfer(msg.sender,reward) (src/MultiRewards-moonwell.sol#588)
        State variables written after the call(s):
        - rewards[msg.sender][_rewardsToken] = 0 (src/MultiRewards-moonwell.sol#587)
        MultiRewards.rewards (src/MultiRewards-moonwell.sol#474) can be used in cross function reentrancies:
        - MultiRewards.earned(address,address) (src/MultiRewards-moonwell.sol#529-542)
        - MultiRewards.getReward() (src/MultiRewards-moonwell.sol#582-592)
        - MultiRewards.rewards (src/MultiRewards-moonwell.sol#474)
        - MultiRewards.updateReward(address) (src/MultiRewards-moonwell.sol#672-684)
Reentrancy in MultiRewards.notifyRewardAmount(address,uint256) (src/MultiRewards-moonwell.sol#601-635):
        External calls:
        - IERC20(_rewardsToken).safeTransferFrom(msg.sender,address(this),reward) (src/MultiRewards-moonwell.sol#608-612)
        State variables written after the call(s):
        - rewardData[_rewardsToken].rewardRate = reward.div(rewardData[_rewardsToken].rewardsDuration) (src/MultiRewards-moonwell.sol#615-617)
        MultiRewards.rewardData (src/MultiRewards-moonwell.sol#468) can be used in cross function reentrancies:
        - MultiRewards.addReward(address,address,uint256) (src/MultiRewards-moonwell.sol#485-494)
        - MultiRewards.getRewardForDuration(address) (src/MultiRewards-moonwell.sol#544-551)
        - MultiRewards.lastTimeRewardApplicable(address) (src/MultiRewards-moonwell.sol#506-511)
        - MultiRewards.notifyRewardAmount(address,uint256) (src/MultiRewards-moonwell.sol#601-635)
        - MultiRewards.rewardData (src/MultiRewards-moonwell.sol#468)
        - MultiRewards.rewardPerToken(address) (src/MultiRewards-moonwell.sol#513-527)
        - MultiRewards.setRewardsDistributor(address,address) (src/MultiRewards-moonwell.sol#555-560)
        - MultiRewards.setRewardsDuration(address,uint256) (src/MultiRewards-moonwell.sol#653-668)
        - MultiRewards.updateReward(address) (src/MultiRewards-moonwell.sol#672-684)
        - rewardData[_rewardsToken].rewardRate = reward.add(leftover).div(rewardData[_rewardsToken].rewardsDuration) (src/MultiRewards-moonwell.sol#625-627)
        MultiRewards.rewardData (src/MultiRewards-moonwell.sol#468) can be used in cross function reentrancies:
        - MultiRewards.addReward(address,address,uint256) (src/MultiRewards-moonwell.sol#485-494)
        - MultiRewards.getRewardForDuration(address) (src/MultiRewards-moonwell.sol#544-551)
```



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.