// Security Assessment

12.11.2025 - 12.11.2025

Mamo Contracts

Moonwell

HALBORN

Mamo Contracts - Moonwell

Prepared by: HALBORN

Last Updated 12/18/2025

Date of Engagement: December 11th, 2025 - December 11th, 2025

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
4	0	0	0	2	2

TABLE OF CONTENTS

1. INTRODUCTION

`Moonwell` engaged `Halborn` to perform a security assessment of their smart contracts starting and ending on December 11th, 2025. The assessment scope was limited to the smart contracts provided to Halborn. Commit hashes and additional details are available in the Scope section of this report.

The `ERC20MoonwellMorphoStrategy` contract implements an automated yield-splitting strategy, allocating funds between the Moonwell mToken market and MetaMorpho vaults. It supports non-custodial deposits, proportional rebalancing, slippage-checked swaps, and Merkle-based reward claims, while enforcing configurable parameters such as allocation splits, slippage limits, and compound fees.

2. ASSESSMENT SUMMARY

`Halborn` was allocated 1 day for this engagement and assigned 1 full-time security engineers to conduct a comprehensive review of the smart contracts within scope. The engineers are experts in blockchain and smart contract security, with advanced skills in penetration testing and smart contract exploitation, as well as extensive knowledge of multiple blockchain protocols.

The objectives of this assessment are to:

- Identify potential security vulnerabilities within the smart contracts.
- Verify that the smart contract functionality operates as intended.

In summary, `Halborn` identified several areas for improvement to reduce the likelihood and impact of security risks, which were mostly addressed by the `Moonwell team`. The main recommendations were:

- `Replace > with >= in withdraw().`
- `Either fix the comment or replace onlyBackend with onlyOwner.`

3. TEST APPROACH AND METHODOLODY

`Halborn` conducted a combination of manual code review and automated security testing to balance efficiency, timeliness, practicality, and accuracy within the scope of this assessment. While manual testing is crucial for identifying flaws in logic, processes, and implementation, automated testing enhances coverage of smart contracts and quickly detects deviations from established security best practices.

The following phases and associated tools were employed throughout the term of the assessment:

- Research into the platform's architecture, purpose and use.
- Manual code review and walkthrough of smart contracts to identify any logical issues.
- Comprehensive assessment of the safety and usage of critical Solidity variables and functions within scope that could lead to arithmetic-related vulnerabilities.
- Local testing using custom scripts (`Foundry`).
- Fork testing against main networks (`Foundry`).
- Static security analysis of scoped contracts, and imported functions (`Slither`).

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
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 (C:N) Low (C:L) Medium (C:M) High (C:H) Critical (C: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

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
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

SEVERITY	SCORE VALUE RANGE
Low	2 - 4.4
Informational	0 - 1.9

5. SCOPE

REPOSITORY

(a) Repository: mamo-contracts

(b) Assessed Commit ID: 49d0617

(c) Items in scope:

- src/ERC20MoonwellMorphoStrategy.sol

Out-of-Scope: External dependencies and economic attacks.

REMEDIATION COMMIT ID:

- 5a3265a

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	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
INCORRECT INEQUALITY CHECK PREVENTS USERS FROM WITHDRAWING FULL BALANCE	LOW	SOLVED - 12/12/2025
ACCESS CONTROL DISCREPANCY BETWEEN COMMENT AND IMPLEMENTATION	LOW	SOLVED - 12/12/2025
HARDCODED WETH ADDRESS	INFORMATIONAL	ACKNOWLEDGED
STRATEGY TYPE INCONSISTENCY BETWEEN CODE AND DESCRIPTION	INFORMATIONAL	SOLVED - 12/12/2025

7. FINDINGS & TECH DETAILS

7.1 INCORRECT INEQUALITY CHECK PREVENTS USERS FROM WITHDRAWING FULL BALANCE

// LOW

Description

The `withdraw()` function in `ERC20MoonwellMorphoStrategy` uses an incorrect strict inequality (`>`) instead of a non-strict inequality (`>=`) when validating withdrawal amounts. This off-by-one error prevents users from withdrawing their entire balance, they can withdraw at most balance - 1 wei or they have to use `withdrawAll()`.

```
221 require(_getTotalBalance() > amount,
222     "Withdrawal amount exceeds available balance in strategy"
223 );
```

BVSS

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

Recommendation

Replace `>` with `>=` in `withdraw()`.

Remediation Comment

SOLVED: The **Moonwell team** solved the issue by replacing `>` with `>=` in `withdraw()`.

Remediation Hash

https://github.com/moonwell-fi/mamo-contracts/commit/5a3265ae0ef3ca5f258bc5d8c0d78547c5ece303

7.2 ACCESS CONTROL DISCREPANCY BETWEEN COMMENT AND IMPLEMENTATION

// LOW

Description

The function `setFeeRecipient()` is documented as **"Only callable by the strategy owner"**, but the code uses the `onlyBackend` modifier, allowing the backend role, not the owner to change the `feeRecipient`. Either the comment should be updated to reflect the intended design, or the modifier should be corrected to `onlyOwner` if the comment is accurate.

BVSS

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

Recommendation

Either fix the comment or replace `onlyBackend` with `onlyOwner`.

Remediation Comment

SOLVED: The **Moonwell team** solved the issue by changing the comment in `setFeeRecipient()`.

Remediation Hash

https://github.com/moonwell-fi/mamo-contracts/commit/5a3265ae0ef3ca5f258bc5d8c0d78547c5ece303

7.3 HARDCODED WETH ADDRESS

// INFORMATIONAL

Description

The `ERC20MoonwellMorphoStrategy` contract hardcodes the `WETH` address (`0x4200…0006`), which is network-specific. This prevents the contract from being safely deployed on other chains where WETH may have a different address. Instead, the WETH address should be passed through the constructor or initializer to avoid cross-chain deployment issues.

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (1.7)

Recommendation

Remove the hardcoded address and inject the WETH address via initializer parameters to ensure cross-chain compatibility.

Remediation Comment

ACKNOWLEDGED: The **Moonwell team** acknowledged this finding stating that "**mamo contracts are only deployed on Base and will only be deployed on Base".**

7.4 STRATEGY TYPE INCONSISTENCY BETWEEN CODE AND DESCRIPTION

// INFORMATIONAL

Description

The `description()` function states that the proposal will whitelist the implementation for strategy type 1, but the contract logic clearly uses `STRATEGY_TYPE_ID = 3`. This mismatch between documentation and code can cause operator confusion, misconfigured deployments, or incorrect assumptions during audits. The comment must be corrected to ensure clarity and correctness.

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (1.7)

Recommendation

Update the description string to reference **strategy type 3.**

Remediation Comment

SOLVED: The **Moonwell team** solved the issue by replacing the token type 1 to token type 3 in `description()`.

Remediation Hash

https://github.com/moonwell-fi/mamo-contracts/commit/5a3265ae0ef3ca5f258bc5d8c0d78547c5ece303
