// Security Assessment

08.18.2025 - 08.20.2025

Transfer and Earn

Moonwell

HALBORN

Transfer and Earn - Moonwell

Prepared by: HALBORN

Last Updated 08/27/2025

Date of Engagement: August 18th, 2025 - August 20th, 2025

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
4	0	0	0	1	3

TABLE OF CONTENTS

1. Introduction

`Moonwell` engaged `Halborn` to conduct a security assessment on their smart contracts beginning on August 18th, 2025 and ending on August 19th, 2025. The security assessment was scoped to the smart contracts provided to the `Halborn` team. Commit hashes and further details can be found in the Scope section of this report.

2. Assessment Summary

The team at `Halborn` assigned one full-time security engineer to assess the security of the smart contracts. The security engineers are blockchain and smart-contract security experts 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 improvements to reduce the likelihood and impact of risks, which were partially addressed by the `Moonwell team`. The main ones were the following:

- `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.`
- `Replace all revert strings with custom errors.`
- `Add a zero-address check in the constructor, consistent with the setFeeCollector function.`
- `Consider disallowing the owner address to be set to address(0).`

3. Test Approach And Methodology

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

- Research into the architecture, purpose, and use of the platform.
- Smart contract manual code review and walkthrough to identify any logic issue.
- Thorough assessment of safety and usage of critical Solidity variables and functions in scope that could lead to arithmetic related vulnerabilities.
- Manual testing by custom scripts.
- Graphing out functionality and contract logic/connectivity/functions (`solgraph`).
- Static Analysis of security for scoped contract, and imported functions. (`Slither`).
- Local or public testnet deployment (`Foundry`, `Remix IDE`).

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

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
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

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
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
Informational	0 - 1.9

5. SCOPE

REPOSITORY

(a) Repository: mamo-contracts

(b) Assessed Commit ID: 86bd51e

(c) Items in scope:

- src/TransferAndEarn.sol

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

REMEDIATION COMMIT ID:

- https://github.com/moonwell-fi/mamo-contracts/pull/42/commits/5c55e5d1462d12ccba675b9c1f5b2c95672c673d

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
MISSING ZERO-ADDRESS VALIDATION FOR FEECOLLECTOR	LOW	SOLVED - 08/22/2025
USE OF OWNABLE LIBRARY WITH SINGLE-STEP OWNERSHIP TRANSFER	INFORMATIONAL	ACKNOWLEDGED - 08/22/2025

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
USE OF CUSTOM ERRORS INSTEAD OF REVERT STRINGS	INFORMATIONAL	ACKNOWLEDGED - 08/22/2025
OWNER CAN RENOUNCE OWNERSHIP	INFORMATIONAL	ACKNOWLEDGED - 08/22/2025

7. FINDINGS & TECH DETAILS

7.1 MISSING ZERO-ADDRESS VALIDATION FOR FEECOLLECTOR

// LOW

Description

In the constructor, `feeCollector` is set without checking for `address(0)`.

```
21 constructor(address _feeCollector, address _owner) Ownable(_owner) {
22     feeCollector = _feeCollector;
23 }
```

If the contract is deployed with a zero address, subsequent calls to `earn` will attempt to `safeTransfer` ERC20 tokens to `address(0)`.

BVSS

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

Recommendation

It is recommended to add a zero-address check in the constructor, consistent with the `setFeeCollector` function.

Remediation Comment

SOLVED : The **Moonwell team** solved the issue. Validation was implemented.

Remediation Hash

https://github.com/moonwell-fi/mamo-contracts/pull/42/commits/5c55e5d1462d12ccba675b9c1f5b2c95672c673d

7.2 USE OF OWNABLE LIBRARY WITH SINGLE-STEP OWNERSHIP TRANSFER

// INFORMATIONAL

Description

The ownership of the contract can be lost as the **TransferAndEarn** inherited from the **Ownable** contract and their ownership can be transferred in a single-step process. The address the ownership is changed to should be verified to be active or willing to act as the owner.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:M/I:N/D:N/Y:N (1.0)

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 achieved by using OpenZeppelin's **Ownable2Step** contract instead of the **Ownable**.

Remediation Comment

ACKNOWLEDGED : The **Moonwell team** acknowledged the issue.

7.3 USE OF CUSTOM ERRORS INSTEAD OF REVERT STRINGS

// INFORMATIONAL

Description

Throughout the entire codebases, `require` statements are used. In Solidity development, replacing hard-coded revert message strings with the `Error()` syntax is an optimization strategy that can significantly reduce gas costs. Hardcoded strings, stored on the blockchain, increase the size and cost of deploying and executing contracts. The `Error()` syntax allows for the definition of reusable, parameterized custom errors, leading to a more efficient use of storage and reduced gas consumption.

This approach not only optimizes gas usage during deployment and interaction with the contract but also enhances code maintainability and readability by providing clearer, context-specific error information.

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 replace all revert strings with custom errors. Since the protocol uses Solidity `0.8.28`, custom errors can also be embedded directly within `require` statements, enabling gas-efficient error handling without requiring explicit `revert` calls.

Remediation Comment

ACKNOWLEDGED : The **Moonwell team** acknowledged the issue.

7.4 OWNER CAN RENOUNCE OWNERSHIP

// INFORMATIONAL

Description

The **TransferAndEarn** contract indirectly inherits from **Ownable**. In this contract contract, the `renounceOwnership` function is used to renounce the `owner` permission.

```
76 | function renounceOwnership() public virtual onlyOwner {
77 |     _transferOwnership(address(0));
78 | }
```

Renouncing ownership before transferring would result in the contract having no owner, eliminating the ability to call privileged functions.

BVSS

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

Recommendation

Consider disallowing the owner address to be set to `address(0)`.

Remediation Comment

ACKNOWLEDGED : The **Moonwell team** acknowledged the issue.
