// Smart Contract Security Assessment 11.14.2024 - 11.15.2024

Contracts V2

Moonwell

HALBORN

Contracts V2 - Moonwell

Prepared by: HALBORN

Last Updated 11/20/2024

Date of Engagement by: November 14th, 2024 - November 15th, 2024

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

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

TABLE OF CONTENTS

1. Introduction

`MoonWell` engaged Halborn to conduct a security assessment on their smart contracts revisions beginning on November 14th, 2024 and ending on November 15th, 2024 . The security assessment was scoped to the smart contracts provided to the Halborn team.

2. Assessment Summary

The team at Halborn was provided 2 days for the engagement and assigned a full-time security engineer to evaluate 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 improvements to reduce the likelihood and impact of risks, which were addressed and acknowledged by the `MoonWell team`:

- `Add a transfer event and emit it after successful transfers.`
- `Add proper validation for rate limit parameters.`

3. Test Approach And Methodology

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance code coverage and 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,draw.io`)
- 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. (`Hardhat`,`Foundry`)

4. RISK METHODOLOGY

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

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

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

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

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

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

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

ATTACK COMPLEXITY (AX):

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

METRICS:

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33

EXPLOITABILIY 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

FILES AND REPOSITORY

(a) Repository: moonwell-contracts-v2

(b) Assessed Commit ID: 5cacb9e

(c) Items in scope:

- IRateLimitedAllowance.sol
- CypherAutoLoad.sol
- ERC4626RateLimitedAllowance.sol
- RateLimitedAllowance.sol

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

REMEDIATION COMMIT ID:

- https://github.com/moonwell-fi/moonwell-contracts-v2/pull/421/commits/b0ae4cf34eceb95f6131202b1939ae3df480e1e4

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	3

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
MISSING TRANSFER EVENT EMISSION IN TRANSFERFROM FUNCTION	INFORMATIONAL	SOLVED - 11/15/2024
MISSING INPUT VALIDATION FOR RATE LIMIT PARAMETERS	INFORMATIONAL	SOLVED - 11/15/2024

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
DANGEROUS ONE-STEP OWNER TRANSFER	INFORMATIONAL	ACKNOWLEDGED - 11/15/2024

7. FINDINGS & TECH DETAILS

7.1 MISSING TRANSFER EVENT EMISSION IN TRANSFERFROM FUNCTION

// INFORMATIONAL

Description

The `transferFrom` function in the **RateLimitedAllowance** contract does not emit any events to track token transfers. While the `depleteBuffer` function emits a `BufferUsed` event, there is no specific event for the token transfer itself:

```
71  function transferFrom(
72    address from,
73    address to,
74    uint256 amount,
75    address token
76  ) external whenNotPaused {
77    require(msg.sender == spender, "Caller is not the authorized spender"
78    RateLimit storage limit = limitedAllowance[from][token];
79
80    limit.depleteBuffer(amount);
81
82    _transfer(from, to, amount, token);
83  }
```

Score

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

Recommendation

Add a transfer event and emit it after successful transfers:

```
event TokenTransferred(
    address indexed token,
    address indexed from,
    address indexed to,
    uint256 amount
);

function transferFrom(
    address from,
    address to,
    uint256 amount,
```

```
    address token
) external whenNotPaused nonReentrant {
    // ... existing code
    _transfer(from, to, amount, token);
    emit TokenTransferred(token, from, to, amount);
}
```

Remediation

SOLVED: Event emission has been added to the code.

Remediation Hash

https://github.com/moonwell-fi/moonwell-contracts-v2/pull/421/commits/b0ae4cf34eceb95f6131202b1939ae3df480e1e4

7.2 MISSING INPUT VALIDATION FOR RATE LIMIT PARAMETERS

// INFORMATIONAL

Description

The `approve` function in the **RateLimitedAllowance** contract accepts `rateLimitPerSecond` and `bufferCap` parameters without validating their values:

```
52  function approve(
53    address token,
54    uint128 rateLimitPerSecond,
55    uint128 bufferCap
56  ) external {
57    RateLimit storage limit = limitedAllowance[msg.sender][token];
58
59    limit.setBufferCap(bufferCap);
60    limit.bufferStored = bufferCap;
61    limit.setRateLimitPerSecond(rateLimitPerSecond);
62
63    emit Approved(token, msg.sender, rateLimitPerSecond, bufferCap);
64  }
```

Score

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

Recommendation

It is recommended to add proper validation for rate limit parameters:

```
function approve(
    address token,
    uint128 rateLimitPerSecond,
    uint128 bufferCap
) external {
    require(rateLimitPerSecond > 0, "Rate limit must be positive");
    require(bufferCap >= rateLimitPerSecond, "Buffer cap must be >= rate li
    require(bufferCap <= type(uint128).max, "Buffer cap too high");
// ... rest of code
}
```

Remediation

SOLVED: `require(rateLimitPerSecond > 0,...)` has been added to the code.

Remediation Hash

https://github.com/moonwell-fi/moonwell-contracts-v2/pull/421/commits/b0ae4cf34eceb95f6131202b1939ae3df480e1e4

7.3 DANGEROUS ONE-STEP OWNER TRANSFER

// INFORMATIONAL

Description

The **RateLimitedAllowance** contract inherits from OpenZeppelin's **Ownable**, which implements a direct, single-step ownership transfer pattern. The transfer is executed immediately upon calling `transferOwnership()` without requiring confirmation from the new owner.

In **RateLimitedAllowance.sol**, ownership is transferred in the constructor:

```
43 constructor(address owner, address _spender) Ownable() {
44     spender = _spender;
45     _transferOwnership(owner);
46 }
```

This pattern creates risks in the case the transfer is made to an incorrect address, resulting in permanent loss of ownership.

Score

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

Recommendation

It is recommended to use `Ownable2Step`.

Remediation

ACKNOWLEDGED : The **Moonwell team** 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:
Pragma version0.8.19 (src/cypher/CypherAutoLoad.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.8.18.
Pragma version0.8.19 (src/cypher/IRateLimitedAllowance.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.8.18.
solc-0.8.19 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter CypherAutoLoad.setBeneficiary(address)._beneficiary (src/cypher/CypherAutoLoad.sol#43) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Pragma version0.8.19 (src/cypher/IRateLimitedAllowance.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.8.18.
solc-0.8.19 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Pragma version0.8.19 (src/cypher/ERC4626RateLimitedAllowance.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.8.18.
Pragma version0.8.19 (src/cypher/RateLimitedAllowance.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.8.18.
solc-0.8.19 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter RateLimitedAllowance.setSpender(address)._spender (src/cypher/RateLimitedAllowance.sol#42) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Pragma version0.8.19 (src/cypher/RateLimitedAllowance.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.8.18.
solc-0.8.19 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter RateLimitedAllowance.setSpender(address)._spender (src/cypher/RateLimitedAllowance.sol#42) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
RateLimitedAllowance (src/cypher/RateLimitedAllowance.sol#11-171) does not implement functions:
        - RateLimitedAllowance._transfer(address,address,uint256,address) (src/cypher/RateLimitedAllowance.sol#165-170)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions
INFO:Slither:src/cypher/ analyzed (32 contracts with 79 detectors), 14 result(s) found
```

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

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