// Private Smart Contract Security Assessment 05.21.2024 - 05.24.2024

ERC 4626

Moonwell

HALBORN

ERC 4626 - Moonwell

Prepared by: HALBORN

Last Updated 06/10/2024

Date of Engagement by: May 21st, 2024 - May 24th, 2024

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
4	0	0	1	2	1

TABLE OF CONTENTS

1. Introduction

`Moonwell` engaged Halborn to conduct a security assessment on **4626** smart contracts beginning on *05/21/2024* and ending on *05/25/2024*. The security assessment was scoped to the smart contracts provided to the Halborn team.

2. Assessment Summary

The team at Halborn dedicated 4 days for the engagement and assigned one 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 security risks that were mostly addressed by the `Moonwell team`.

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`)
- 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. (`Brownie`, `Anvil`, `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

EXPLOITABILIY 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
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

SEVERITY	SCORE VALUE RANGE
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: moonwell-contracts-v2

(b) Assessed Commit ID: 2c3d9da

(c) Items in scope:

- src/4626/Factory4626.sol
- src/4626/Factory4626Eth.sol
- src/4626/MoonwellERC4626.sol
- src/4626/MoonwellERC4626Eth.sol
- src/router/ERC4626EthRouter.sol

Out-of-Scope:

REMEDIATION COMMIT ID:

- 8d65e508d65e50
- 2589a8a2589a8a

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	2	1

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
VAULT DEPLOYMENT FAILURE FOR VOID RETURN TOKENS	Medium	SOLVED - 05/23/2024

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
MOONWELLERC4626 RETURNS MAX DEPOSIT DENOMINATED IN SHARES	Low	SOLVED - 05/23/2024
CLAIMING REWARDS DEPLETES UNDERLYING WHEN CONFIGURED AS AN EMISSION TOKEN	Low	RISK ACCEPTED
SUSCEPTIBILITY TO INFLATION GRIEFING FOR LOW PRECISION DECIMALS	Informational	ACKNOWLEDGED

7. FINDINGS & TECH DETAILS

7.1 VAULT DEPLOYMENT FAILURE FOR VOID RETURN TOKENS

// MEDIUM

Description

Using `ERC20` or `IERC20` to mediate transactions with underlying tokens will implicitly enforce strict interface conventions about those tokens - not just for input parameter definitions, but also the return types. `Factory4626` relies upon these strict interface definitions when executing both `transferFrom(address,uint256)` and `approve(address,uint256)`, and by result, expects that each call must return in a `bool` value, else `revert`:

```
require(
    ERC20(asset).transferFrom( /// @audit revert_on_void
        msg.sender,
        address(this),
        initialMintAmount
    ),
    "transferFrom failed"
);

require(
    ERC20(asset).approve(vault, initialMintAmount), /// @audit revert_on_void
    "approve failed"
)
```

In turn, this configuration leads to incompatibility with void return on transfer tokens such as USDT, as the Solidity compiler will generate code which expects non-empty return data which it can parse a boolean value from. In the case where the length of the return data is `0` (i.e., when performing a transfer with USDT), the attempt to parse will `revert`.

Although this issue could potentially be worked around by deploying a `MoonwellERC4626` vault independently of the factory, this would not be recommended, as the deployed vault loses the resistance to vault inflation attacks that the factory explicitly provides.

Proof of Concept

Attempts to deploy vaults using void-return transfer tokens will `revert` when interacted with directly via the `IERC20` interface bindings generated by the Solidity compiler:

```
/// @notice deployment fails for void return tokens
function testRevertOnVoidReturnData() public {

    address deployer = address(0xbabe);

    vm.prank(deployer);
        USDT usdt = new USDT(100 ether);

    assertEq(usdt.balanceOf(deployer), 100 ether);

    address mToken = address(0xc0ffee);

    vm.mockCall(mToken, abi.encodeWithSignature("underlying()"),
abi.encode(address(usdt)));

    vm.startPrank(deployer);
        usdt.approve(address(factory), type(uint256).max);
        vm.expectRevert();
            factory.deployMoonwellERC4626(mToken, deployer);
    vm.stopPrank();
}
```

Subsequently, the vault cannot be deployed:

```
Ran 1 test for test/integration/Moonwell4626FactoryLiveSystem.t.sol:Moonwell4626FactoryLiveSystemBaseTest
[PASS] testRevertOnVoidReturnData() (gas: 2768108)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 7.63ms (1.29ms CPU time)
```

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:M/D:N/Y:N/R:N/S:U (5.0)

Recommendation

Delegate the responsibility of reverting on failed operations to libraries that export safe ERC-20 functionality, i.e. `safeApprove` and `safeTransferFrom`.

Remediation Plan

SOLVED: The **Moonwell team** solved the issue by modifying the logic to use `safeTransferFrom` and `safeApprove`.

Remediation Hash

8d65e5064bca6748cec552c3ee096d0277284ee2

References

moonwell-fi/moonwell-contracts-v2/src/4626/Factory4626.sol#L68C9-L80C11

7.2 MOONWELLERC4626 RETURNS MAX DEPOSIT DENOMINATED IN SHARES

// LOW

Description

The `maxDeposit(address)` function of an ERC-4626 vault is expected to return the maximum number of assets which can be deposited by the specified `address`, however in `MoonwellERC4626`, it underlying returns the value of `maxMint(address)` - the maximum number of shares which can be deposited:

```
/// @notice maximum amount of underlying tokens that can be deposited
///         into the underlying protocol
function maxDeposit(address) public view override returns (uint256) {
    return maxMint(address(0));
}
```

Concerning invocations of `maxMint(address)` which do not return `type(uint256).max`, an incorrect value will be returned to the caller, which will invalidate the contractual obligations of the ERC-4626 specification.

BVSS

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

Recommendation

Scale values returned from calls to `maxMint(address(0))` back into `assets` when not unlimited:

```
/// @notice maximum amount of underlying tokens that can be deposited into the
///         underlying protocol
function maxDeposit(address) public view override returns (uint256) {
  uint256 shares = maxMint(address(0));

  // unlimited
  if (shares == type(uint256).max) return type(uint256).max;

  return convertToAssets(shares);
}
```

Remediation Plan

SOLVED: The **Moonwell team** solved the issue by reversing the inverted logic and added the required conversion from `shares` back into `assets`.

Remediation Hash

2589a8ab6d9ae714cf5d081c1d7181ba8cdde192

References

moonwell-fi/moonwell-contracts-v2/src/4626/MoonwellERC4626.sol#L173C5-L176C6

7.3 CLAIMING REWARDS DEPLETES UNDERLYING WHEN CONFIGURED AS AN EMISSION TOKEN

// LOW

Description

In the unlikely event a `MarketConfig` were to define an `emissionToken` of `mToken` (i.e. through a DAO vote or governance attack), the `MoonwellERC4626` vault may inadvertently withdraw all the underlying asset liquidity to the `rewardsRecipient` during a call to `claimRewards()`:

```
for (uint256 i = 0; i < configs.length; i++) {
    uint256 amount = ERC20(configs[i].emissionToken).balanceOf(
        address(this)
    );

    if (amount != 0) {
        /// gas opti, skip transfer and event emission if no rewards
        ERC20(configs[i].emissionToken).safeTransfer(
            rewardRecipient,
            amount
        );

        emit ClaimRewards(amount, configs[i].emissionToken);
    }
}
```

Here, the `rewardsRecipient` is able to claim the `MoonwellERC4626`'s `balanceOf` the returned `emissionTokens`. If the underlying asset of the vault was configured as an `emissionToken` for a market, the entirety of underlying vault liquidity could be withdrawn by the `rewardsRecipient`.

BVSS

AO:S/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:N/S:U (2.0)

Recommendation

If an `emissionToken` is `address(0)` or equal to `address(mToken)`, `continue` iterating.

Additionally, consider using a `try` statement to wrap limited-gas calls to `emissionTokens` if they cannot be trusted.

Remediation Plan

RISK ACCEPTED: The **Moonwell team** accepted the risk of this issue, but never intended to configure an `MToken` as an `emissionToken`.

References

moonwell-fi/moonwell-contracts-v2/src/4626/MoonwellERC4626.sol#L77C5-L111C6

7.4 SUSCEPTIBILITY TO INFLATION GRIEFING FOR LOW PRECISION DECIMALS

// INFORMATIONAL

Description

One of the unique properties of the `MoonwellERC4626` vault is its dependence upon a dedicated factory contract to initialize vault deployments in a way that is resistant to inflation attacks. This in turn liberates the vault implementation from the burden of inflation logic encapsulation.

To prevent classical inflation attacks, `Factory4626` ensures an initial quantity of vault shares are burned according to the following formula:

```
uint256 initialMintAmount = 10 ** ((ERC20(asset).decimals() * 2) / 3);
```

The resultant implication is that for tokens with zero decimals, an `initialMintAmount` of merely a single share is expected to be burned to the zero address upon new vault creation.
Burning a single share is approximate to using a `decimalOffset` of zero in a Virtual Asset Shares (VAS) implementation, which, while sufficient to negate the profitability of classical vault inflation attacks, leaves some griefing vulnerabilities remaining actionable. In these attacks, a depositor's collateral can be lost at some expense to the attacker.

BVSS

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

Recommendation

Enforce a minimum amount of shares to be burned, i.e `1_000`.

Remediation Plan

ACKNOWLEDGED: The **Moonwell team** has acknowledged the issue, and confirmed that tokens with zero decimals will never be listed on Moonwell.

References

moonwell-fi/moonwell-contracts-v2/src/4626/Factory4626.sol#L66C9-L66C79

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:
LibCompound.viewExchangeRate(MErc20) (src/4626/LibCompound.sol#21-59) uses a dangerous strict equality:
        - accrualBlockTimestampPrior == block.timestamp (src/4626/LibCompound.sol#24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Factory4626.constructor(Comptroller,address)._weth (src/4626/Factory4626.sol#36) lacks a zero-check on :
                - weth = _weth (src/4626/Factory4626.sol#38)
Factory4626Eth.constructor(Comptroller,address)._weth (src/4626/Factory4626Eth.sol#39) lacks a zero-check on :
                - weth = _weth (src/4626/Factory4626Eth.sol#41)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
MoonwellERC4626.claimRewards() (src/4626/MoonwellERC4626.sol#78-111) has external calls inside a loop: amount = ERC20(configs[i].emissionToken).balanceOf(address(this)) (src/
4626/MoonwellERC4626.sol#95-97)
MoonwellERC4626.sweepRewards(address[]) (src/4626/MoonwellERC4626.sol#117-131) has external calls inside a loop: amount = token.balanceOf(address(this)) (src/4626/MoonwellERC
4626.sol#126)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
Reentrancy in MoonwellERC4626.claimRewards() (src/4626/MoonwellERC4626.sol#78-111):
        External calls:
        - comptroller.claimReward(holders,mTokens,false,true) (src/4626/MoonwellERC4626.sol#85)
        Event emitted after the call(s):
        - ClaimRewards(amount,configs[i].emissionToken) (src/4626/MoonwellERC4626.sol#106)
Reentrancy in Factory4626.deployMoonwellERC4626(address,address) (src/4626/Factory4626.sol#44-88):
        External calls:
        - require(bool,string)(ERC20(asset).transferFrom(msg.sender,address(this),initialMintAmount),transferFrom failed) (src/4626/Factory4626.sol#68-75)
        - require(bool,string)(ERC20(asset).approve(vault,initialMintAmount),approve failed) (src/4626/Factory4626.sol#77-80)
        - require(bool,string)(MoonwellERC4626(vault).deposit(initialMintAmount,address(0)) > 0,deposit failed) (src/4626/Factory4626.sol#82-85)
        Event emitted after the call(s):
        - DeployedMoonwellERC4626(asset,mToken,rewardRecipient,vault) (src/4626/Factory4626.sol#87)
```

```
Reentrancy in Factory4626Eth.deployMoonwellERC4626Eth(address,address) (src/4626/Factory4626Eth.sol#48-92):
        External calls:
        - require(bool,string)(ERC20(weth).transferFrom(msg.sender,address(this),INITIAL_MINT_AMOUNT),transferFrom failed) (src/4626/Factory4626Eth.sol#69-76)
        - require(bool,string)(ERC20(weth).approve(vault,INITIAL_MINT_AMOUNT),approve failed) (src/4626/Factory4626Eth.sol#78-81)
        - require(bool,string)(MoonwellERC4626Eth(address(vault)).deposit(INITIAL_MINT_AMOUNT,address(0)) > 0,deposit failed) (src/4626/Factory4626Eth.sol#83-89)
        Event emitted after the call(s):
        - DeployedMoonwellERC4626(weth,mToken,rewardRecipient,vault) (src/4626/Factory4626Eth.sol#91)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
LibCompound.viewExchangeRate(MErc20) (src/4626/LibCompound.sol#21-59) uses timestamp for comparisons
        Dangerous comparisons:
        - accrualBlockTimestampPrior == block.timestamp (src/4626/LibCompound.sol#24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

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.