// Security Assessment

04.25.2025 - 04.30.2025

Astherus

Aster

HALBORN

Astherus - Aster

Prepared by: HALBORN

Last Updated 06/16/2025

Date of Engagement: April 25th, 2025 - April 30th, 2025

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
6	0	0	1	1	4

TABLE OF CONTENTS

1. Introduction

`Astherus` engaged `Halborn` to conduct a security assessment on their smart contracts beginning on April 25th, 2025 and ending on April 30th, 2025. The security assessment was scoped to the smart contracts provided to Halborn. Commit hashes and further details can be found in the Scope section of this report.

The `Astherus` codebase in scope consists of a smart contract responsible for minting USDF which can then then be used for users to earn rewards through staking.

2. Assessment Summary

`Halborn` was provided 5 days for the engagement and assigned a full-time security engineer to review the security of the smart contracts in scope.

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 some improvements to reduce the likelihood and impact of risks, which were partially addressed by the `Astherus team`. The main ones were the following:

- `Implement slippage for operations dependent on dynamic variables.`
- `Apply the pause functionality in RewardDispatcher.`

3. Test Approach And Methodology

`Halborn` performed a manual review of the code. Manual testing is great to uncover flaws in logic, process, and implementation.

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

- Research into 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 led to arithmetic related vulnerabilities.

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

FILES AND REPOSITORY

(a) Repository: astherus-usdt-contract

(b) Assessed Commit ID: d0fd0bb

(c) Items in scope:

- asUSDFEarn.sol
- WithdrawVault.sol
- USDFEarn.sol
- Timelock.sol
- RewardDispatcher.sol
- TransferLimiter.sol
- USDF.sol
- asUSDF.sol
- Withdrawable.sol
- IAsERC20.sol
- IAsUSDFEarn.sol
- IUSDFEarn.sol
- IWithdrawVault.sol
- IWithdrawable.sol

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

REMEDIATION COMMIT ID:

- b3322b5

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	1	1

INFORMATIONAL
4

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
USERS MIGHT RECEIVE LESS USDF THAN EXPECTED DUE TO A RACE CONDITION	MEDIUM	RISK ACCEPTED - 06/13/2025
THE REWARD DISPATCHER CONTRACT DOES NOT UTILISE THE PAUSE FUNCTIONALITY	LOW	RISK ACCEPTED - 06/13/2025
FUNCTIONS ARE MISSING NATSPEC	INFORMATIONAL	ACKNOWLEDGED - 06/13/2025
COMMENT IN A FOREIGN LANGUAGE	INFORMATIONAL	SOLVED - 06/13/2025
REMOVING USERS FROM THE EMERGENCY WITHDRAW WHITELIST ACCEPTS AN ARRAY AS MEMORY INSTEAD OF CALLDATA	INFORMATIONAL	SOLVED - 06/13/2025
STRING REVERTS INSTEAD OF CUSTOM ERRORS	INFORMATIONAL	ACKNOWLEDGED - 06/13/2025

7. FINDINGS & TECH DETAILS

7.1 USERS MIGHT RECEIVE LESS USDF THAN EXPECTED DUE TO A RACE CONDITION

// MEDIUM

Description

The amount of USDF to mint is computed based on the below code, where `amountIn` is the provided USDT:

```
uint256 USDFAmount =  amountIn * (1e4 - commissionRate) / 1e4;
```

The commission rate serves as a type of fee, for example if it is `1e3`, then a user will receive 10% less USDF than the USDT he provided. However, a race condition might make the following happen:

1. The current commission rate is 1e3 or 10%
2. User will provide 100 USDT, expecting 90 USDF
3. At the same time, an admin decides to change the commission rate to 20%
4. Transaction from step 3 executes first and now the user only receives 80 USDF instead of 90

BVSS

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

Recommendation

Consider implementing slippage or pausing the contract when changing the commission rate.

Remediation Comment

RISK ACCEPTED: The **Astherus team** has accepted this risk as the commission rate will not be changed.

7.2 THE REWARD DISPATCHER CONTRACT DOES NOT UTILISE THE PAUSE FUNCTIONALITY

// LOW

Description

`RewardDispatcher` inherits `PausableUpgradeable` in order to take advantage of pause functionality:

```
contract RewardDispatcher is Initializable, AccessControlEnumerableUpgradeable, PausableUpgradeable, U
```

However, neither of the functions implement the `whenNotPaused` modifier in order to actually take advantage of that functionality, rendering it useless.

BVSS

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

Recommendation

Consider using the modifier in the appropriate functions.

Remediation Comment

RISK ACCEPTED: The **Astherus team** has accepted the risk of this finding.

7.3 FUNCTIONS ARE MISSING NATSPEC

// INFORMATIONAL

Description

Throughout the codebases, there are many functions with missing NatSpec. Natspec is recommended to provide in-line code documentation, resulting in the code being easier to understand and integrate with.

Currently, the only contracts with NatSpec documentation are:

- `TransferLimiter`
- `USDF` (partially)

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 adding natspec to the functions.

Remediation Comment

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

7.4 COMMENT IN A FOREIGN LANGUAGE

// INFORMATIONAL

Description

In `asUSDFEarn` , the following comment can be seen:

```
//限制最大值，USDF发行量的百分比
```

Comments in foreign languages are not recommended as they are not understandable to the majority of people and can be considered a bad practice.

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 removing the comment or writing it in English.

Remediation Comment

SOLVED: The **Astherus team** solved this finding in commit `b3322b5` by removing the comments.

Remediation Hash

https://github.com/asterdex/astherus-usdt-contract/commit/b3322b51771d037eafedfa0def16547581eecdf9

7.5 REMOVING USERS FROM THE EMERGENCY WITHDRAW WHITELIST ACCEPTS AN ARRAY AS MEMORY INSTEAD OF CALLDATA

// INFORMATIONAL

Description

Removing users from the emergency withdraw whitelist is done with the below function:

```
function removeEmergencyWithdrawWhitelist(address[] memory users) external onlyAdmin {
    Storage storage st = getStorage();
    for (uint256 i = 0; i < users.length; i++) {
        delete st.emergencyWithdrawWhitelist[users[i]];
        emit RemoveEmergencyWithdrawWhitelist(msg.sender, users[i]);
    }
}
```

However, unlike the function for adding users to the list, the function accepts the users as a memory array. This results in higher computation costs.

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 changing the array to calldata instead.

Remediation Comment

SOLVED: The **Astherus team** solved this finding in commit `b3322b5` by following the recommendation.

Remediation Hash

https://github.com/asterdex/astherus-usdt-contract/commit/b3322b51771d037eafedfa0def16547581eecdf9

7.6 STRING REVERTS INSTEAD OF CUSTOM ERRORS

// INFORMATIONAL

Description

Since Solidity 0.8.4 custom errors were introduced which significantly lower the gas expenses compared to string reverts. Currently, the codebase is using string reverts which is suboptimal.

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 using custom errors instead of string reverts.

Remediation Comment

ACKNOWLEDGED: The **Astherus team** has decided to acknowledge this finding at this time.

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.