SALUS SECURITY

SEP 2024

CODE SECURITY ASSESSMENT

ASTHERUS

Overview

Project Summary

- Name: Astherus - earn
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
  - https://github.com/astherus-contract/astherus-earn-contract
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Astherus - earn
Version	v2
Type	Solidity
Dates	Sep 12 2024
Logs	Sep 10 2024; Sep 12 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	1
Total Low-Severity issues	1
Total informational issues	2
Total	4

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Centralization risk	Medium	Centralization	Mitigated
2	Missing fee on transfer token support	Low	Business Logic	Resolved
3	Redundant Code	Informational	Redundancy	Resolved
4	Use of floating pragma	Informational	Configuration	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Centralization risk	
Severity: Medium	Category: Centralization
Target: - contracts/oft/AssXXX.sol - contracts/Earn.sol - contracts/Timelock.sol - contracts/WithdrawVault.sol	

Description

All critical contracts are utilizing the Access Control module by OpenZeppelin, giving the `owner` the `ADMIN_ROLE` and the power to modify any other roles. This includes roles that manage vital functions such as `mint()` and `burn()` and the setting of key variables.

If the `owner's` private key is compromised, an attacker can directly manipulate the contract and cause significant impact. If the privileged accounts are plain EOA accounts, this can be worrisome and pose a risk to the other users.

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

The team has mitigated the issue by properly allocating permissions.

2. Missing fee on transfer token support	
Severity: Low	Category: Business Logic
Target: - contracts/Earn.sol	

Description

There are ERC20 tokens that charge fee for every transfer() or transferFrom().

The deposit and withdrawal operations in `Earn` contract don't account for tokens with fees on transfers. This can lead to discrepancies and potential loss of user funds, as the received or sent amounts may differ from the intended amounts.

Recommendation

Consider adjusting the functions to compare the contract's token balance before and after transfers to handle tokens with fees accurately. This ensures correct balance tracking and calculations.

Status

The team has resolved this issue in commit f4b622a

2.3 Informational Findings

3. Redundant Code	
Severity: Informational	Category: Redundancy
Target: - contracts/oft/TransferLimiter.sol	

Description

Redundant code should be removed before deploying the contract to mainnet. We have identified the following code as unnecessary:

contracts/oft/TransferLimiter.sol:L64-L66

```
require(limit.maxDailyTransferAmount > limit.singleTransferUpperLimit,
"maxDailyTransferAmount must be greater than singleTransferUpperLimit");
require(limit.dailyTransferAmountPerAddress > limit.singleTransferUpperLimit,
"dailyTransferAmountPerAddress must be greater than singleTransferUpperLimit");
require(limit.maxDailyTransferAmount > limit.dailyTransferAmountPerAddress,
"maxDailyTransferAmount must be greater than dailyTransferAmountPerAddress");
```

In the second and third require statements in the above code, it is required that `maxDailyTransferAmount` is greater than `dailyTransferAmountPerAddress`, and `dailyTransferAmountPerAddress` is greater than `singleTransferUpperLimit`. Therefore, the first require statement is redundant code.

Recommendation

Consider removing the redundant code.

Status

The team has resolved this issue in commit 7fdb462.

4. Use of floating pragma	
Severity: Informational	Category: Configuration
Target: - contracts/oft/AssXXX.sol - contracts/oft/TransferLimiter.sol	

Description

```
pragma solidity ^0.8.25;
```

The `TransferLimiter` contract and `AssXXX` use a floating compiler version `^0.8.25`.

Using a floating pragma `^0.8.25` statement is discouraged, as code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. Also use the latest Solidity version to get all the compiler features, bug fixes and optimizations.

Recommendation

It is recommended to use a locked Solidity version throughout the project. It is also recommended to use the most stable and up-to-date version.

Status

The team has resolved this issue in commit 668c0e8.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit edeb7de:

File	SHA-1 hash
contracts/Earn.sol	d302f9ce4c0780f41de84a7d35addc8d62e9cd62
contracts/Timelock.sol	e5a372c76545ef06f626b9b343e292dc58f62ebf
contracts/WithdrawVault.sol	3927ee14dd3ccee1ca4f8eb289df5b5df85638e0
contracts/oft/AssXXX.sol	5c6d798accad0b8a5d9e500296680bc857ffaf90
contracts/oft/TransferLimiter.sol	1a2c96d26c18b6a8196c6a42306be560c29b47ea