SALUS SECURITY

DEC 2024

CODE SECURITY ASSESSMENT

ASTHERUS

Overview

Project Summary

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

Project Dashboard

Application Summary

Name	Astherus - Ascake
Version	v2
Type	Solidity
Dates	Dec 17 2024
Logs	Dec 09 2024; Dec 17 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	1
Total Low-Severity issues	0
Total informational issues	3
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	Token may be locked	Medium	Business Logic	Resolved
2	Redundant Code	Informational	Redundancy	Resolved
3	Missing two-step transfer ownership pattern	Informational	Business Logic	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. Token may be locked	
Severity: Medium	Category: Business Logic
Target: - src/RewardDistributionScheduler.sol	

Description

When the `addRewardsSchedule()` function is called, the `_amount` will be evenly divided into `_epochs` parts. If `_amount` cannot be evenly divided by `_epochs`, a portion of the tokens will not be recorded in the `epochs` variable, resulting in these tokens being locked within the contract.

src/RewardDistributionScheduler.sol:L125 - L128

```
function addRewardsSchedule() external override onlyRole(MANAGER) nonReentrant
whenNotPaused {
  ...
  token.safeTransferFrom(msg.sender, address(this), _amount);
  // average daily reward amount
  uint256 amountPerDay = _amount / _epochs;
  // spread rewards every day
  for (uint256 i; i < _epochs; i++) {
    // accumulation of different reward types
    epochs[startTime + i * 1 days][_rewardsType] += amountPerDay;
  }
  // emit event
  emit RewardsScheduleAdded(msg.sender, _rewardsType, _amount, _epochs, startTime);
}
```

Recommendation

Consider using the total amount minus the already recorded rewards as the `amountPerDay` value when `i == _epochs - 1`.

Status

The team has resolved this issue in commit e24306b.

2.3 Informational Findings

2. Redundant Code	
Severity: Informational	Category: Redundancy
Target: - src/Minter.sol	

Description

Unused code should be removed before deploying the contract to mainnet. We have identified the following code are not being utilized:

src/Minter.sol:L44 - L49

```
uint256 public totalVeTokenRewards;
uint256 public totalVoteRewards;
uint256 public totalDonateRewards;
```

Recommendation

Consider removing the redundant code.

Status

The team has resolved this issue in commit e24306b.

3. Missing two-step transfer ownership pattern

Severity: Informational	Category: Business logic
Target: - src/AssToken.sol	

Description

The `AssToken` contract inherits from the `OwnableUpgradeable` contract. This contract does not implement a two-step process for transferring ownership. Thus, ownership of the contract can easily be lost when making a mistake in transferring ownership.

Recommendation

Consider using the Ownable2StepUpgradeable contract from OpenZeppelin instead.

Status

The team has resolved this issue in commit e24306b.

4. Use of floating pragma

Severity: Informational	Category: Configuration
Target: - All	

Description

```
pragma solidity ^0.8.20;
```

All contracts use a floating compiler version `^0.8.20`.

Using a floating pragma `^0.8.20` 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 e24306b.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 5eb634c:

File	SHA-1 hash
src/AssToken.sol	44cc6d011dbfc8ab1c98a8af43a12baf449ea992
src/Buyback.sol	180b16174f284404a33050a1133350f4241b8315
src/Minter.sol	21739477c1bbc54a1f4519cad0f1f8200098ddb6
src/RewardDistributionScheduler.sol	d9014859ce0d6e9ef23b13a9edc5961ef3cfa8da
src/Timelock.sol	71a42a8346b50c56100a9759f2963488d344f5b4
src/UniversalProxy.sol	bdbf8e1df20b4ca5cde85bfb072a74064e9e71b7