SALUS SECURITY

MAR 2024

CODE SECURITY ASSESSMENT

AETHER GAMES

Overview

Project Summary

- Name: Aether Games - AEGStaking
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/Aether-Forge/contracts_v2
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Aether Games - AEGStaking
Version	v3
Type	Solidity
Dates	Mar 25 2024
Logs	Mar 20 2024; Mar 22 2024; Mar 25 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	1
Total Low-Severity issues	2
Total informational issues	4
Total	7

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	Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom()	Low	Risky External Calls	Resolved
3	Missing events for functions that change critical state	Low	Logging	Resolved
4	Use of floating pragma and vulnerable dependency	Informational	Configuration	Resolved
5	Missing zero address checks	Informational	Data Validation	Resolved
6	Implementation contract could be initialized by everyone	Informational	Business Logic	Acknowledged
7	Missing two-step transfer ownership pattern	Informational	Business logic	Acknowledged

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/AEGStaking_2.sol	

Description

There is a privileged owner role in the AEGStaking_2 contract. The owner can:

1. Calling the createPool function, which can create a new pool with specified parameters.
2. Toggle isPaused to prevent the contract from being used.
3. Unlimited set end time. If the endTime is set to a value lower than the current block timestamp, it will cause the stake functionality of the pool to no longer work properly.
4. Withdraw all reward tokens by withdrawTokens function. Even though the function restricts the withdrawal to tokens outside of totalStaked, based on the contract logic, rewardToken is not considered part of totalStaked. Therefore, the owner is entitled to withdraw rewardToken.

If the owner's private key is compromised, an attacker could exploit the owner’s privileged functions to disrupt the project, which could damage the team’s reputation.

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 removing the withdrawTokens function.

2. Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom()

Severity: Low	Category: Risky external calls
Target: - contracts/AEGStaking_2.sol	

Description

AEGStaking_2.sol:L129, L156, L277

```
aegToken.transferFrom(msg.sender, address(this), amount);
aegToken.transfer(msg.sender, totalAmount + totalReward);
IERC20(tokenAddress).transfer(owner, amount);
```

Tokens not compliant with the ERC20 specification could return false from the transfer function call to indicate the transfer fails, while the calling contract would not notice the failure if the return value is not checked. Checking the return value is a requirement, as written in the EIP-20 specification:

```
Callers MUST handle false from returns (bool success). Callers MUST NOT assume that
false is never returned!
```

Recommendation

Consider using the SafeERC20 library implementation from OpenZeppelin and call safeTransfer or safeTransferFrom when transferring ERC20 tokens.

Status

This issue has been resolved by the team with commit cd3db94.

3. Missing events for functions that change critical state	
Severity: Low	Category: Logging
Target: - contracts/AEGStaking_2.sol	

Description

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes that allow users to evaluate them. Missing events do not promote transparency and if such changes immediately affect users' perception of fairness or trustworthiness, they could exit the protocol causing a reduction in protocol users.

In the AEGStaking_2, events are lacking in the createPool(), togglePause(), pauseAllPools(), setEndTime() and withdrawTokens() functions.

Recommendation

It is recommended to emit events for critical state changes.

Status

This issue has been resolved by the team with commit a524458.

2.3 Informational Findings

4. Use of floating pragma and vulnerable dependency

Severity: Informational	Category: Configuration
Target: - contracts/AEGStaking_2.sol	

Description

AEGStaking_2.sol:L2

```
pragma solidity ^0.8.19;
```

The AEGStaking_2 uses a floating compiler version 0.8.19.

Using a floating pragma 0.8.19 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.

The project is using a vulnerable version of the OpenZeppelin dependency.

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.

Consider using the latest version of OpenZeppelin.

Status

This issue has been resolved by the team with commit a524458.

5. Missing zero address checks

Severity: Informational	Category: Data Validation
Target: - contracts/AEGStaking_2.sol	

Description

It is considered a security best practice to verify addresses against the zero address during initialization or setting. However, this precautionary step is absent for address variables `aegToken` and `nft`.

AEGStaking_2.sol:L59-L60

```
function initialize(IERC20 _aegToken, IERC721 _nft) external initializer {
        aegToken = _aegToken;
        nft = _nft;
        owner = msg.sender;
        __ReentrancyGuard_init();
    }
```

Recommendation

Consider adding zero address checks for address variables `aegToken` and `nft`.

Status

This issue has been resolved by the team with commit a524458.

6. Implementation contract could be initialized by everyone

Severity: Informational	Category: Business Logic
Target: - contracts/AEGStaking_2.sol	

Description

According to OpenZeppelin, the implementation contract should not be left uninitialized.

An uninitialized implementation contract can be taken over by an attacker, which may impact the proxy. There is nothing preventing the attacker from calling the initialize function in AEGStaking_2's implementation contract.

Recommendation

To prevent the implementation contract from being used, consider invoking the _disableInitializers function in the constructor of the AEGStaking_2 contract to automatically lock it when it is deployed.

Status

This issue has been acknowledged by the team.

7. Missing two-step transfer ownership pattern

Severity: Informational	Category: Business logic
Target: - contracts/AEGStaking_2.sol	

Description

AEGStaking_2.sol:L216-L218

```
    function transferOwnership(address _owner) external onlyOwner {
        owner = _owner;
    }
```

The AEGStaking_2 contract inherits from the Ownable 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 Ownable2Step contract from OpenZeppelin instead.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit c14fdfa:

File	SHA-1 hash
contracts/AEGStaking_2.sol	525c9ca0e8b2a2e435e59062036134422dc0b251