Code Security Assessment

For Match_game

07

November

2022

Overview

Project Summary

- Name: Match_game
- Version: 1.0
- Platform:
- Language: solidity
- Codebase:
- Audit Range: (Match_game)Context.sol; ERC20.sol; IERC20.sol; IERC20Metadata.sol; MatchERC20.sol

Project Dashboard

Application Summary

Name	Match_game
Version	v1.0
Type	Solidity
Dates	07/11
Logs	07/11

Vulnerability Summary

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

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' reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to 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 lowmpact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.

Content

Introduction

1.1 About SALUS

Salus Security is an all-rounded blockchain security company. With rich experiences in traditional and blockchain security, we are born to solve some of the most complex security issues in the industry and make security services accessible for all. Our smart contract auditing service is equipped with an automated tool and expert services. Every project needs an invincible shield to achieve long-term success; with complete coverage from traditional to blockchain, Salus Security is what you need. We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

Objective was to evaluate the repository for security-related issues, code quality, and adherence to specification 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

Findings

2.1 Key Findings

Severity	Title	Category	Status
Informational	Floating compiler version	Coding Practice	Unresolved
Informational	Latest OpenZepplin version	Coding Practice	Unresolved

2.2 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 of 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. As one audit-based assessment cannot be considered

Informational

Floating compiler version

- **Target Contract: All files**
- **Category: Coding Practice**
- **File(s) affected: All files**

Description

It is recommended to use the up-to-date stable compiler version

Original code:

Pragma solidity ^0.8.0;

Recommended version:

pragma solidity 0.8.17

Latest OpenZepplin version

- **Target Contract: ERC20.sol**
- **Category: Coding Practice**
- **File(s) affected: ERC20.sol**

Description

It is recommended to use the up-to-date Open Zeppelin version

Original code:

OpenZeppelin Contracts (last updated v4.5.0)

Recommended version:

OpenZeppelin Contracts (last updated v4.7.0)

SALUS