QuillAudits

Audit Report
July, 2021

RAGE FAN

Contents

Overview

Rage.Fan is a platform offering Fantasy Sports and Quizzing games that require skill, knowledge, and interaction.

Scope of Audit

The scope of this audit was to analyze Stake.sol smart contract's codebase for quality, security, and correctness.

Contract: Stake.sol
Commit: e55c3e0f28887705cf5029355ebc36ef2dd2b1d1
Fixed In: 7bd9eb1d2df723b61fb177dbdaee5aa607e126ac

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC20 transfer() does not return boolean
- ERC20 approve() race
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis
In this step we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.
SmartCheck.

Static Analysis
Static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step a series of automated tools are used to test security of smart contracts.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of automated analysis were manually verified.

Gas Consumption
In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Mythril, Slither, SmartCheck, Surya, Solhint.

Issue Categories

Every issue in this report has been assigned with a severity level. There are four levels of severity and each of them has been explained below.

High severity issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality and we recommend these issues to be fixed before moving to a live environment.

Medium level severity issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems and they should still be fixed.

Low level severity issues

Low level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity four issues which indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	1	2
Closed	0	0	0	1

Issues Found – Code Review / Manual Testing

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low level severity issues

1. Multiple Pragma Directives have been used. Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Informational

1. [FIXED] **Missing Zero Address Validation**

   [#L946-970]**constructor()**: Missing Zero Address check for **_controller** address

   [#L1028-1030]function **setRescuer()**: Missing Zero Address check for **_rescuer** address

2. [#L812, 1062, 1085, 1176] block.timestamp has been used for comparisons. Avoid using block.timestamp as it can be manipulated by minors.

3. ERC20 approve() race

The standard ERC20 implementation contains a widely-known racing condition in its approve function, wherein a spender is able to witness the token owner broadcast a transaction altering their approval and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender is able to spend their entire approval amount twice.

Reference:

- https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0 mooh4DYKjA_jp-RLM/edit

- https://eips.ethereum.org/EIPS/eip-20

Gas Optimization

Public functions that are never called by the contract should be declared external to save gas.

```
addNfts(uint256,uint256,address,uint256,uint256) should be declared external:
        - Stake.addNfts(uint256,uint256,address,uint256,uint256) (stake.sol#1149-1164)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

Automated Testing

Slither

Slither didn’t detect any high severity issues.

Mythril

Mythril didn’t detect any high severity issues.

Smartcheck

Smartcheck didn’t detect any high severity issues.

Solhint

```
rage/stake.sol
   21:2  error  Line length must be no more than 120 but current length is 132  max-line-length
  333:2  error  Line length must be no more than 120 but current length is 160  max-line-length
  358:2  error  Line length must be no more than 120 but current length is 156  max-line-length

✖ 3 problems (3 errors, 0 warnings)
```

Disclaimer

The audit does not give any warranties on the security of the code. One audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of the code. Besides a security audit, please don’t consider this report as investment advice.

Closing Summary

Some issues of low severity have been reported during the audit. No high severity issues have been found. Some suggestions have also been made to improve the code quality and gas optimization.

RAGE FAN

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com