Roolend Audit Report

Version 1.0.0

Serial No. 2021061400022020

Presented by Fairyproof

June 14, 2021

灵踪安全
FAIRYPROOF

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Roolend project, at the request of the Roolend team.

Project Token's Name:

ROO

Project Token's Onchain Address:

N/A

Audited Code's Github Repository:

https://github.com/roolend-finance/contracts

Audited Code's Github Commit Number:

3c4b5ce99308d7d25b2abd3e4a83b92715e7336d

Audited Contract Files' Onchain Address:

N/A

Audited Contract Files:

The contract files audited include all the files with the extension "sol" as follows:

```
contracts/
├── Comptroller
|   ├── Comptroller.sol
|   ├── ComptrollerInterface.sol
|   ├── ComptrollerStorage.sol
|   └── Unitroller.sol
├── Controller
|   ├── Context.sol
|   ├── Controller.sol
|   └── Ownable.sol
├── InterextRateModel
|   ├── InterestRateModel.sol
|   └── WhitePaperInterestRateModel.sol
├── Math
|   ├── CarefulMath.sol
|   ├── Exponential.sol
|   └── SafeMath.sol
├── PriceOracle
|   ├── DexPrice.sol
|   ├── IDexPair.sol
|   ├── PriceOracle.sol
|   ├── PriceOracleProxy.sol
|   └── SimplePriceOracle.sol
├── ROO
```

```
|   ├── OIP20
|   |   ├── OIP20.sol
|   |   ├── OIP20Burnable.sol
|   |   └── OIP20Detailed.sol
|   ├── IOIP20.sol
|   └── ROO.sol
├── Tokens
|   ├── ERC20
|   |   ├── ERC20.sol
|   |   ├── ERC20Burnable.sol
|   |   ├── ERC20Detailed.sol
|   |   └── IERC20.sol
|   ├── ERC20NonStandardInterface.sol
|   ├── Maximillion.sol
|   ├── NativeAddress.sol
|   ├── RErc20.sol
|   ├── RErc20Delegate.sol
|   ├── RErc20Delegator.sol
|   ├── RErc20Immutable.sol
|   ├── RNativeToken.sol
|   ├── RToken.sol
|   ├── RTokenInterfaces.sol
|   └── Reservoir.sol
├── ErrorReporter.sol
└── Migrations.sol
```

Note: functions such as `receiveApproval` defined in the `OIP20.sol` contract file, which are imported from third-party contracts, were not covered by this audit.

The goal of this audit is to review Roolend's solidity implementation for its token issurance, lending, price retrieval functions, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

We make observations on specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

— Disclaimer

Note that as of the date of publishing, the contents of this report reflect the current understanding of known security patterns and state of the art regarding smart contract security. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. Risks or issues introduced by using data feeds from offchain sources are not extended by this review either.

Given the size of the project, the findings detailed here are not to be considered exhaustive, and further testing and audit is recommended after the issues covered are fixed.

To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

— Methodology

The above files' code was studied in detail in order to acquire a clear impression of how the its specifications were implemented. The codebase was then subject to deep analysis and scrutiny, resulting in a series of observations. The problems and their potential solutions are discussed in this document and, whenever possible, we identify common sources for such problems and comment on them as well.

The Fairyproof auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Fairyproof to make sure we understand the size, scope, and functionality of the project's smart contracts.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Fairyproof describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run the test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

— Structure of the document

This report contains a list of issues and comments on all the above contract files. Each issue is assigned a severity level based on the potential impact of the issue and recommendations to fix it, if applicable. For ease of navigation, an index by topic and another by severity are both provided at the beginning of the report.

— Documentation

For this audit, we used the following sources of truth about how the Roolend system should work:

https://roolend.finance/#/markets

whitepaper

These were considered the specification, and when discrepancies arose with the actual code behavior, we consulted with the Roolend team or reported an issue.

— Comments from Auditee

No vulnerabilities with critical or high-severity were found in the above contract files.

One vulnerability with medium-severity and one vulnerability with low-severity were found in the above contract files.

02. About Fairyproof

Fairyproof is a leading technology firm in the blockchain industry, providing consulting and security audits for organizations. Fairyproof has developed industry security standards for designing and deploying smart contract systems.

03. Introduction to Roolend

Roolend is a decentralized lending application.

04. Major functions of audited code

The audited code implements a token issurance function, decentralized lending and a price retrieval mechanism.

Attention:

1 this application uses off-chain oracles as price feeds. Risks or issues introduced by using off-chain oracles were not covered by this audit.

2 The DexPrice.sol was initially used for retrieving a token's price but later was replaced by RlPriceOracle.sol. Usage of DexPrice.sol should be avoided.

3 The `receiveApproval` function is implemented in a third-party contract and issues or risks introduced by this function was not covered by this audit.

05. Key points in audit

During the audit Fairyproof worked closely with the Roolend team and reviewed possible vulnerabilities in its token issurance, decentralized lending and price retrieval mechanism.

06. Coverage of issues

The issues that the Fairyproof team covered when conducting the audit include but are not limited to the following ones:

- Re-entrancy Attack
- DDos Attack
- Integer Overflow
- Function Visibility
- Logic Vulnerability
- Uninitialized Storage Pointer
- Arithmetic Precision
- Tx.origin
- Shadow Variable
- Design Vulnerability
- Token Issurance
- Asset Security
- Access Control

07. Severity level reference

Every issue in this report was assigned a severity level from the following:

Critical severity issues need to be fixed as soon as possible.

High severity issues will probably bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

08. Major areas that need attention

Based on the provided contract files the Fairyproof team focused on the possible issues and risks related to the following functions or areas.

- Token Issurance

We checked whether or not the contract files could mint tokens at will.

We didn't find issues or risks in these functions or areas at the time of writing.

- Access Control

We checked each of the functions that could modify a state, especially those functions that could only be accessed by "owner".

We found one issue, for more details please refer to "11. Issue descriptions".

- Asset Security

We checked whether or not all the deposited assets were safely hanlded.

We didn't find issues or risks in these functions or areas at the time of writing.

- Price Retrieval and Calculation

We checked whether or not the price retrieval mechanism and the price calculation algorithms had vulnerabilities that could be exploited to manipulate asset prices.

We didn't find issues or risks in these functions or areas at the time of writing.

- Contract Migration/Upgrade

We checked whether or not the contract files introduced issues or risks associated with contract migration/upgrade.

We found one issue, for more details please refer to "11. Issue descriptions".

- Miscellaneous

We didn't find issues or risks in other functions or areas at the time of writing.

09. List of issues by severity

A. Critical

- N/A

B. High

- N/A

C. Medium

- RErc20Delegator.sol

Contract Migration/Upgrade

D. Low

- setUnderlyingPrice

Inappropriate Access Control in Testnet Contracts

10. List of issues by contract file

- RErc20Delegator.sol

Contract Migration/Upgrade: Medium

- setUnderlyingPrice

Inappropriate Access Control in Testnet Contracts: Low

11. Issue descriptions

- Contract Migration/Upgrade: Medium

Source and Description:

The `RErc20Delegator.sol` contract files support contract migration/upgrade.

Recommendation:

Consider doing contract migration/upgrade with extreme caution and conducting an audit prior to contract migration/upgrade if it is needed.

Update: the Roolend team will do contract migration/upgrade with extreme caution and do an audit prior to contract migration/upgrade if such a migration/upgrade is needed.

- Inappropriate Access Control in Testnet Contracts: Low

Source and Description:

The `setUnderlyingPrice` and `setDirectPrice` functions in the `SimplePriceOracle.sol` contract file don't have access control, therefore any address can call them to manipulate token prices. However these functons are derived from the Compound's source code and are only used in testnet contracts.

Recommendation:

Consider adding access control for these two functions or add special comments or notes on them.

Update: Acknowledged by the Roolend team. These functions will not be used in mainnet deployment.

12. Recommendations to enhance the overall security

We list some recommendations in this section. They are not mandatory but will enhance the overall security of the system if they are adopted.

- Allowing Assets With High Liquidity to Be Collateral

Decentralized lending applications may suffer this attack: users can raise the price of an asset with low liquidity by price manipulation and then use it as collateral to borrow assets with high liquidity.

Consider only allowing assets with high liquidity to be collateral.