QuillAudits

Audit Report
March, 2022

For

REDLINE BLOCKCHAIN

Contents

Scope of the Audit

The scope of this audit was to analyze and document the RedLine Token smart contract codebase for quality, security, and correctness.

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
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

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 analysed 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.

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 the security of smart contracts.

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

Gas Consumption
In this step, we have checked the behaviour 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
Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis, Theo.

Issue Categories

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

Risk-level	Description
High	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 be fixed before moving to a live environment.
Medium	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	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 issues that 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	0	0
Acknowledged	1	0	1	3
Closed	0	0	0	0

Introduction

During the period of March 2, 2022 to March 7, 2022 - QuillAudits Team performed a security audit for the RedLine smart contract.

The code for the RedLine contract was obtained from:
https://github.com/RLC2021/RLC/tree/71d30d0bdf606509047d11e08abbf61dda80d6e9

The contract was deployed and tested on the BNB testnet and you can find it here:
RedLine: 0xe7d9692367c3ba963413496145c962ca8f53050c

Issues Found – Code Review / Manual Testing

A. Contract - RLC_Contract.sol

High severity issues

1. Loss of tokens forever

In the transfer and transferFrom functions there is no check for a zero address. If a user transfers his/her tokens to a zero address they will be lost forever and there is no way of retrieving them.

Suggestion: Please add a check in both the functions to check whether the to address entered is a zero address or not, something like this => require(to != address(0), “Invalid address”);

Status: Acknowledged
Client’s Comment: Our products are in charge of controlling and managing it. This is done so that our token is not handled if it is used outside of the Our ecosystem. If this is utilized within our ecosystem, it is maintained through our wallet.
We want people to download our wallet and use it,.people can't send our tokens to metamask etc but that's where we don't control and we clearly state if it's lost its lost, hence people have to download and use our wallet

Medium severity issues

No issues were found.

Low severity issues

2. Gas savings

All the functions inside the RLC contract(except the constructor) can be declared as external. We do this because external functions consume lesser gas as compared to public functions. You can read more about it here. The view functions can be skipped as they do not use any gas.

Status: Acknowledged

Informational issues

3. Total Supply

The standard is to show the total tokens in circulation when it comes to totalSupply.
Even then, if you decide to keep it this way(as shown below) please use safeSub to keep uniformity throughout the code.
Although, after putting in a check for the zero address as suggested above we can remove balances[address(0)] altogether on line 60.

```
59      function totalSupply() public view returns (uint) {
60          return _totalSupply  - balances[address(0)];
61      }
```

Status: Acknowledged

4. Indentation

Please follow the solidity style guide for better code readability. You can read more about it here.

```
27      function safeSub(uint a, uint b) public pure returns (uint c) {
28          require(b <= a); c = a - b; } function safeMul(uint a, uint b) public
            pure returns (uint c) { c = a * b; require(a == 0 || c / a == b); }
            function safeDiv(uint a, uint b) public pure returns (uint c) { require
            (b > 0);
29          c = a / b;
30      }
```

Status: Acknowledged

5. Require messages

Consider adding error messages inside the require statements which will make debugging failed transactions much easier. You can read more about error handling here.

Status: Acknowledged

Functional Tests

Function Names	Testing results
transfer	Passed
transferFrom	Passed
approve	Passed

Functionality Tests Performed

- Users should be able to transfer tokens not more than their balance. PASS
- approve. PASS
- Users should be able to transferFrom tokens not more than their approval and also not more than the owner's(token owner) balance. PASS

Automated Tests

Mythril

Mythril is a security analysis tool for EVM bytecode. It detects security vulnerabilities in smart contracts built for Ethereum, Hedera, Quorum, Vechain, Roostock, Tron and other EVM-compatible blockchains. It uses symbolic execution, SMT solving and taint analysis to detect a variety of security vulnerabilities. Mythril raised the following concerns:

```
The analysis was completed successfully. No issues were detected.
```

Slither

Slither is a Solidity static analysis framework written in Python 3. It runs a suite of vulnerability detectors, prints visual information about contract details, and provides an API to easily write custom analyses. Slither enables developers to find vulnerabilities, enhance their code comprehension, and quickly prototype custom analyses.

```
Pragma version^0.5.0 (Redline.sol#5) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Variable RLC._totalSupply (Redline.sol#39) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

RLC.constructor() (Redline.sol#49-57) uses literals with too many digits:
        - _totalSupply = 3000000000 * 10 ** 18 (Redline.sol#53)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

totalSupply() should be declared external:
        - ERC20Interface.totalSupply() (Redline.sol#8)
        - RLC.totalSupply() (Redline.sol#59-61)
balanceOf(address) should be declared external:
        - ERC20Interface.balanceOf(address) (Redline.sol#9)
        - RLC.balanceOf(address) (Redline.sol#63-65)
```

```
allowance(address,address) should be declared external:
        - ERC20Interface.allowance(address,address) (Redline.sol#10)
        - RLC.allowance(address,address) (Redline.sol#67-69)
transfer(address,uint256) should be declared external:
        - ERC20Interface.transfer(address,uint256) (Redline.sol#11)
        - RLC.transfer(address,uint256) (Redline.sol#77-82)
approve(address,uint256) should be declared external:
        - ERC20Interface.approve(address,uint256) (Redline.sol#12)
        - RLC.approve(address,uint256) (Redline.sol#71-75)
transferFrom(address,address,uint256) should be declared external:
        - ERC20Interface.transferFrom(address,address,uint256) (Redline.sol#13)
        - RLC.transferFrom(address,address,uint256) (Redline.sol#84-90)
safeMul(uint256,uint256) should be declared external:
        - SafeMath.safeMul(uint256,uint256) (Redline.sol#28)
safeDiv(uint256,uint256) should be declared external:
        - SafeMath.safeDiv(uint256,uint256) (Redline.sol#28-30)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
Redline.sol analyzed (3 contracts with 77 detectors), 11 result(s) found
```

Results

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered various possible vulnerabilities listed above for the security of the RLC contract. We performed our audit according to the procedure outlined above.

The audit showed one high and one low issue ,which the Auditee has Acknowledged.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the RedlineBlockchain platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the RedlineBlockchain Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
March, 2022

For

REDLINE BLOCKCHAIN

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com