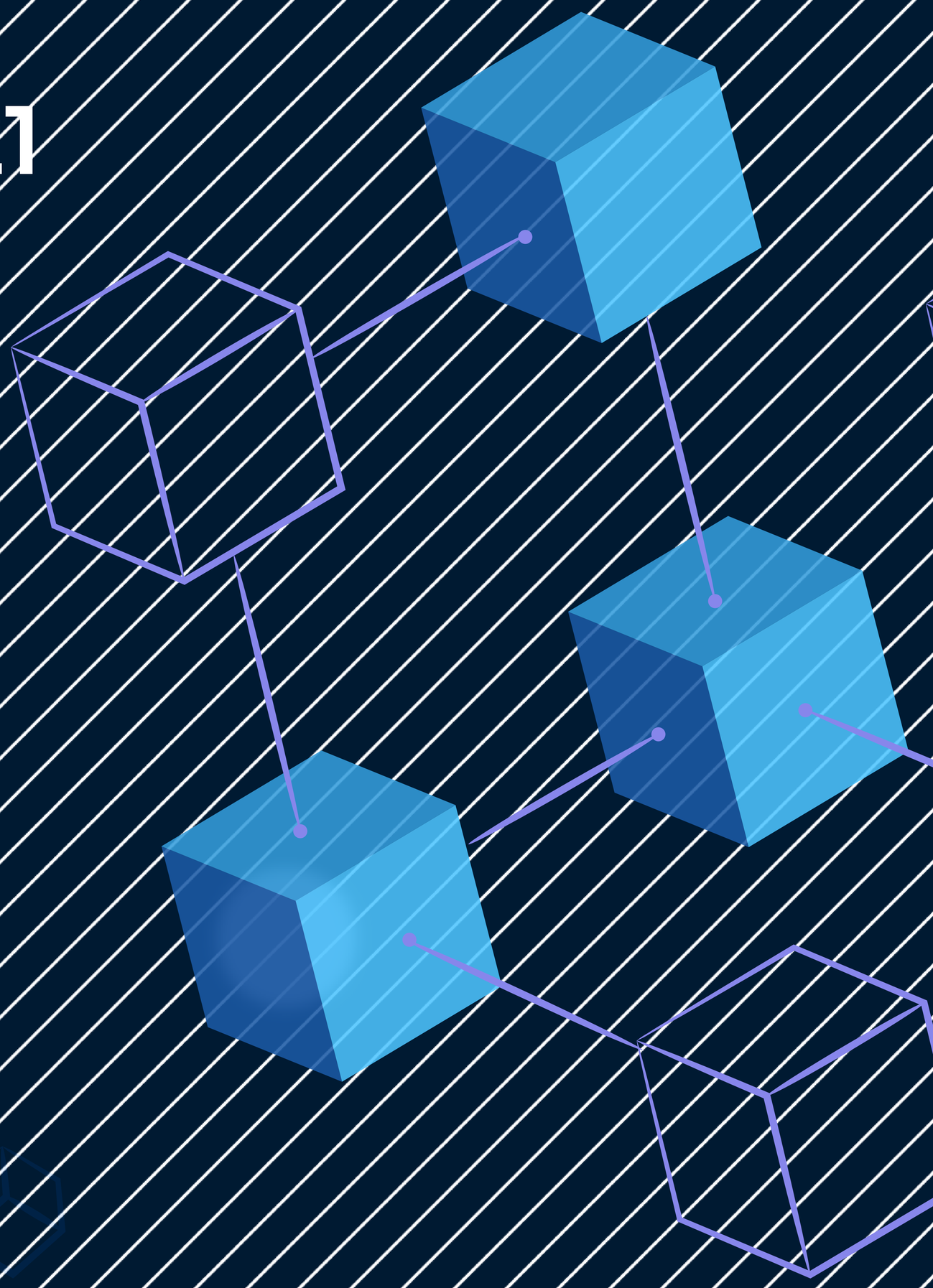
QuillAudits

Audit Report
November, 2021



For

Merkle
Network

Contents

Scope of the Audit

The scope of this audit was to analyze and document the Merkle Network 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, Solhint, Mythril, Slither.

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	0	0	0	0
Closed	0	0	0	0

Introduction

During the period of **October 26, 2021 to October 28 , 2021** - QuillAudits Team performed a security audit for **Merkle Network** smart contract.

The code for the MerkleNetworkERC20 contract was obtained from:
- 0x000000000ca5171087c18fb271ca844a2370fc0a

The contract was deployed and tested on Ropsten and you can find it here:
- MerkleNetworkERC20: 0xFF7b3761cc789CB8F314C1f554b8513572517EBF

Issues Found

A. Contract – MerkleNetworkERC20.sol

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

No issues were found.

Informational issues

No issues were found.

Functional test

Function Names	Testing results
transfer	Passed
transferFrom	Passed
rescueTokens	Passed
burn	Passed
approve	Passed
increaseAllowance	Passed
decreaseAllowance	Passed
renounceOwnership	Passed
transferOwnership	Passed

Functionality Tests Performed

MerkleNetworkERC20

- Users should be able to transfer tokens not more than their balance.

  PASS
- approve.

  PASS
- Users should be able to decreaseAllowance not more than the current approval.

  PASS
- increaseApproval.

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

  PASS
- Users should be able to burn tokens not more than their balance.

  PASS
- Only the contract owner should be able to call the rescueTokens function and withdraw not more than the contract’s balance of the token being rescued. Also, the token address and recipient address shouldn't be the same.

  PASS
- Only the current owner should be able to transferOwnership.

  PASS
- Only the owner should be able to renounceOwnership.

  PASS

Automated Tests

Slither

```
INFO:Detectors:
ERC20.constructor(string,string).name (dist/Merkle.sol#422) shadows:
        - ERC20.name() (dist/Merkle.sol#430-432) (function)
ERC20.constructor(string,string).symbol (dist/Merkle.sol#422) shadows:
        - ERC20.symbol() (dist/Merkle.sol#438-440) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Verify.splitSignature(bytes) (dist/Merkle.sol#330-351) uses assembly
        - INLINE ASM (dist/Merkle.sol#341-348)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Pragma version0.6.12 (dist/Merkle.sol#5) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
solc-0.6.12 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
MerkleNetworkERC20.constructor() (dist/Merkle.sol#679-688) uses literals with too many digits:
        - supply_ = 60000000 * 10 ** uint256(decimals_) (dist/Merkle.sol#684)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
```

```
INFO:Detectors:
owner() should be declared external:
        - Ownable.owner() (dist/Merkle.sol#292-294)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (dist/Merkle.sol#311-314)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (dist/Merkle.sol#320-324)
name() should be declared external:
        - ERC20.name() (dist/Merkle.sol#430-432)
symbol() should be declared external:
        - ERC20.symbol() (dist/Merkle.sol#438-440)
decimals() should be declared external:
        - ERC20.decimals() (dist/Merkle.sol#455-457)
totalSupply() should be declared external:
        - ERC20.totalSupply() (dist/Merkle.sol#462-464)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (dist/Merkle.sol#469-471)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (dist/Merkle.sol#481-484)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address) (dist/Merkle.sol#489-491)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (dist/Merkle.sol#500-503)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (dist/Merkle.sol#517-521)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (dist/Merkle.sol#535-538)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (dist/Merkle.sol#554-557)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:dist/Merkle.sol analyzed (7 contracts with 46 detectors), 20 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Mythril

Mythril is a security analysis tool for EVM bytecode. It detects security vulnerabilities in smart contracts built for Ethereum, Hedera, Quorum, Vechain, Rootstock, 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:

```
==== External Call To User-Supplied Address ====
SWC ID: 107
Severity: Low
Contract: MerkleNetworkERC20
Function name: rescueTokens(address,address,uint256)
PC address: 3632
Estimated Gas Usage: 2977 - 38059
A call to a user-supplied address is executed.
An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behavior. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.
--------------------
In file: dist/Merkle.sol:696

ERC20(token).transfer(recipient, amount)

--------------------
Initial State:

Account: [CREATOR], balance: 0x10005, nonce:0, storage:{}
Account: [ATTACKER], balance: 0x0, nonce:0, storage:{}
Account: [SOMEGUY], balance: 0x0, nonce:0, storage:{}

Transaction Sequence:

Caller: [CREATOR], calldata: , value: 0x0
Caller: [CREATOR], function: transferFrom(address,address,uint256), txdata:
0x23b872dd0000000000000000000000080002010000010000100020040004000000040000000000000000000000000000000020800200004408000010808808002010200000000000000000000000000000000000000000000000000000000000000000, value: 0x0
Caller: [CREATOR], function: rescueTokens(address,address,uint256), txdata:
0xcea9d26f020202020202020202020202deadbeefdeadbeefdeadbeefdeadbeefdeadbeef020202020202020202020202020101010101010101010101010101010101010102020202020202020202020202020202020202020202020202020202, value: 0x0
```

Contract Library:

Contract-library contains the most complete, high-level decompiled representation of all Ethereum smart contracts, with security analysis applied to these in real-time.

We performed analysis using the contract Library on the Ropsten address of the Merkle Network contract used during manual testing:
Merkle Network: 0xFF7b3761cc789CB8F314C1f554b8513572517EBF

No issues were found. There was one issue isolated by Mythril regarding the **rescueTokens()** function which was ignored in the manual audit under the assumption that the owner is a smart user(because this function can only be called by the owner).

Closing Summary

In this report, we have considered the security of Merkle Network. We performed our audit according to the procedure described above.

The audit showed no issues.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the **Merkle Network** 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 **Merkle Network** Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
November, 2021

For

Merkle Network

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com

QuillAudits