QuillAudits

Audit Report
April, 2022

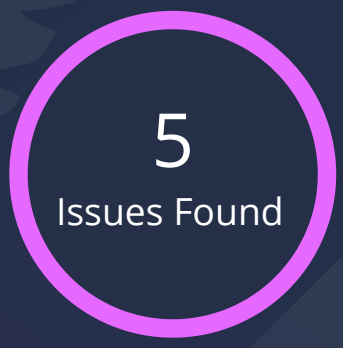
For

metria

Table of Content

Executive Summary

Project Name	Metria Network
Overview	The Metria Ecosystem - Interaction Made Seamless. An environment where DeFi, NFTs and Metaverse work in tandem across multi-chain converging at one umbrella environment. The scope of the audit was to test TokenLock.sol contract which is a ownable contract with claim functionality.
Timeline	April 26, 2022 - April 28, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyse Timelock.sol contract by Metria Network for quality, security, and correctness.
Deployed At	*https://testnet.bscscan.com/address/0x847bdbbfd8f34eb9d26299e817c6cbb67f5f7d78#code* *https://mumbai.polygonscan.com/address/0x7de0c6bcb1deab487d8e50a685ae8f326fe5ec45#code* *https://rinkeby.etherscan.io/address/0x235c11e39218c09319ead88b74472994c663cd14#readContrac*



High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	5
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	0	0

Types of Severities

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.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- 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 leve

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.

Manual Testing

A. Contract - TokenLock.sol

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

No issues were found

Informational Issues

A.1 Public functions that could be declared external in order to save gas.

Description

Whenever a function is not called internally, it is recommended to define them as external instead of public in order to save gas. For all the public functions, the input parameters are copied to memory automatically, and it costs gas. If your function is only called externally, then you should explicitly mark it as external. External function’s parameters are not copied into memory but are read from calldata directly. This small optimization in your solidity code can save you a lot of gas when the function input parameters are huge.

Here is a list of functions that could be declared external:
- Owner(#L38)
- transferOwnership(#L70)
- claimOwnership(#L78)
- claim(#L111)

Status

Acknowledged

A.2 Require statements without reason string

Description

The contract has multiple require statements. However, the reason string (error message) is missing in some instances (#L46,54,55,71,88). It is a best practice to have a human-readable revert reason, unique across the contract

Remediation

Consider adding an appropriate revert reason string whenever require check fails.

Status

Acknowledged

A.3 Missing event in claim()

Description

Whenever certain significant privileged actions are performed within the contract, it is recommended to emit an event about it. Whenever claim function is called by the owner, tokens are transferred to the beneficiary address. These are some important actions hence it is recommended to emit an event.

Remediation

Consider emitting an event whenever claim() is called.

Status

Acknowledged

A.4 Return values not being validated on transfer

Description

In claim() function the transfer function is invoked. Developers should keep in mind that all the tokens do not return revert on transfer failure, there are some tokens that return false. Whenever a function is implemented that uses transfer on different contracts, it is recommended to check the return value and perform the following actions based on the return value.

Status

Acknowledged

A.5 Missing zero address and zero amount validation in claim()

Description

```
function claim(address _tokenAddr, uint _amount) public onlyOwner {
    require(isUnlocked(), "Cannot transfer tokens while locked.");
    token(_tokenAddr).transfer(beneficiary, _amount);
}
```

Description

In claim() function _tokenAddr address is passed as a function parameter and is not validated against zero address. Adding a zero address check is necessary because, in Ethereum, a zero address is something to which if any funds or tokens are transferred, it can not be retrieved back. In this case, there won't be any loss of token but if the amount or address is zero it would be a wastage of gas. Hence, it is recommended to add a check for zero address and zero amount.

Remediation

It is recommended to check whether the provided input parameters (address and amount) are zero or not. Consider adding a require statement to mitigate the same.

Status

Acknowledged

Functional Testing

Some of the tests performed are mentioned below

- Should test all getter values.
- Should test isOwner function.
- Should test isUnlocked function.
- Should test claim function.
- Should test Ownership transfer control flow.
- Should test access controls; functions requiring privileged roles must revert if called by normal users.

Automated Tests

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 the security of the Metria Network. We performed our audit according to the procedure described above.

Some issues of informational severity were found,all the issues Metria Network team Acknowledged.

Disclaimer

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

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

500+
Audits Completed

$15B
Secured

500K
Lines of Code Audited

Follow Our Journey

Audit Report
April, 2022

For

metria

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com