QuillAudits

Audit Report
September, 2022

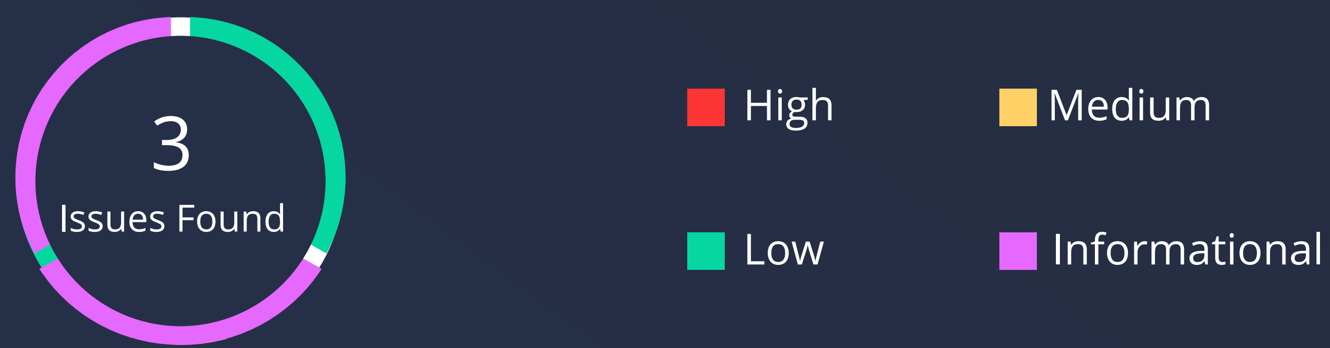
For

METARIX

Table of Content

Executive Summary

Project Name	MetarixClaim
Timeline	August 30th, 2022 to September 2nd, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyze MetarixClaim smart contract codebase for quality, security, and correctness. *Link to contract on testnet*



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	1	2
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
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- 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 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 - LPToken

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

A1. block.timestamp dependence

Description

The MetarixClaim token contract relies heavily on the block.timestamp variable in its logic. This value can be manipulated by malicious actors, miners.

Remediation

Consider alternative sources of verification for time dependence.

Status

Acknowledged

Informational Issues

A2. Variable 'name' never used after declaration

```
contract MetarixClaim is Ownable {
    using SafeERC20 for IERC20;

    string public name;
```

Description

The MetarixClaim token contract has a string variable 'name' that is never used.

Remediation

We recommend removing 'name' since it is never used in the context of the claim contract.

Status

Acknowledged

A3. No constructor utilized in contract

Description

The MetarixClaim token contract does not have a constructor. Best practice recommends the use of constructors, even if empty.

Remediation

Include a constructor in the contract with any initialization level code desired.

Status

Acknowledged

Functional Testing

- Verify number of tokens approved for spending by contract
- Revert when totalReward is zero
- Revert when users tokenValue array lengths passed are unequal
- Revert when release time passed is less than block.timestamp
- Revert when reward values don't match totalReward
- emit MetarixAdded event
- add users data successfully
- Revert if zero address is passed in
- Revert if amount passed is greater than allowance

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.

```
Reentrancy in MetarixClaim.claim(address,uint256) (contracts/MetarixCheck.sol#190-214):
        External calls:
        - ERC20Interface.safeTransfer(msg.sender,amount) (contracts/MetarixCheck.sol#211)
        Event emitted after the call(s):
        - Claimed(msg.sender,metarixAddress,amount) (contracts/MetarixCheck.sol#212)
Reentrancy in MetarixClaim.uploadUserData(address,uint256,uint256,uint256,address[],uint256[]) (contracts/MetarixCheck.sol#143-180):
        External calls:
        - ERC20Interface.safeTransferFrom(msg.sender,address(this),totalReward) (contracts/MetarixCheck.sol#177)
        Event emitted after the call(s):
        - MetarixAdded(metarixAddress,saleType,totalReward) (contracts/MetarixCheck.sol#178)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

MetarixClaim.uploadUserData(address,uint256,uint256,uint256,address[],uint256[]) (contracts/MetarixCheck.sol#143-180) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(release > block.timestamp,Invalid time) (contracts/MetarixCheck.sol#158)
        - require(bool,string)(block.timestamp < unlockTime[metarixAddress][saleType],Time Started alredy) (contracts/MetarixCheck.sol#160-163)
MetarixClaim.claim(address,uint256) (contracts/MetarixCheck.sol#190-214) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(unlockTime[metarixAddress][saleType] < block.timestamp,please wait for the unlock time) (contracts/MetarixCheck.sol#194-197)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Context._msgData() (contracts/MetarixCheck.sol#49-51) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Variable MetarixClaim.ERC20Interface (contracts/MetarixCheck.sol#118) is not in mixedCase
Modifier MetarixClaim._hasAllowance(address,uint256,address) (contracts/MetarixCheck.sol#126-137) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

MetarixClaim.name (contracts/MetarixCheck.sol#117) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (contracts/MetarixCheck.sol#76-78)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (contracts/MetarixCheck.sol#80-86)
claim(address,uint256) should be declared external:
        - MetarixClaim.claim(address,uint256) (contracts/MetarixCheck.sol#190-214)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
contracts/MetarixCheck.sol analyzed (5 contracts with 78 detectors), 11 result(s) found
```

Closing Summary

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

Some issues of Low and Informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the MetarixClaim 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 MetarixClaim 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.

600+
Audits Completed

$15B
Secured

600K
Lines of Code Audited

Follow Our Journey

Audit Report
September, 2022

For

METARIX

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com