QuillAudits

Audit Report
May, 2022

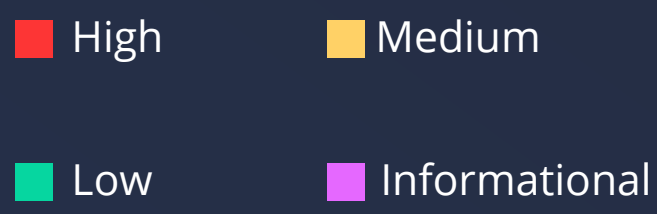
For

METARIX

Table of Content

Executive Summary

Project Name	Metarix
Overview	Metarix is a globally operational Metaverse platform to revolutionize the virtual world and provide an interesting and satisfying experience to a huge number of people.
Timeline	May 2nd, 2022 to May 6th, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyze Metarix codebase for quality, security, and correctness.
Sourcecode	*https://github.com/Metarix-Network/Smart-Contracts/blob/master/CrowdFunding/CrowdSale.sol*
Commit	2a0637dd12dc74deba11818c88ae4c2a2adff2a3
Fixed in	*https://github.com/Metarix-Network/Smart-Contracts/commit/9e51f665135443e5ed2fcb5b9b8c94463f7973fa*
Commit	9e51f665135443e5ed2fcb5b9b8c94463f7973fa



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	**2**	**2**

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 Analysis

A. Contract - Metarix

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

A.1 Missing address verification [#74]

```
69  // CONSTRUCTOR
70  constructor(uint _maxCap, uint256 _saleStartTime, uint256 _saleEndTime, address payable _projectOwner) public {
71    maxCap = _maxCap;
72    saleStartTime = _saleStartTime;
73    saleEndTime = _saleEndTime;
74    projectOwner = _projectOwner;
75  }
```

Description

Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, the contract's functionality may become inaccessible or tokens may be burned in perpetuity.

Remediation

It’s recommended to undertake further validation of address by checking the address passed is not address(0).

Status

Fixed

A.2 Missing Decimal [#71]

```
69  // CONSTRUCTOR
70  constructor(uint _maxCap, uint256 _saleStartTime, uint256 _saleEndTime, address payable _projectOwner) public {
71    maxCap = _maxCap;
72    saleStartTime = _saleStartTime;
73    saleEndTime = _saleEndTime;
74    projectOwner = _projectOwner;
75  }
```

Description

The MaxCap,can be mistakenly initialize to any value with out decimals and Token are in decimal, any digit can be inputed as the maxCap. This will cause the Token to get to max e.g Maxcap = 100 , pay ether value = 20 Ether which is 20*10**18. This checks that totalBnbRecieve is less than or equal to MaxCap ,20*10**18 <= 100, this will always fail.

```
require(totalBnbReceived + msg.value <= maxCap, "buyTokens: purchase would exceed max cap");
```

Remediation

We recommend the MaxCap value should be in 18 decimals or it should be is MaxCap *10**18, with this, the `value of MaxCap will be in decimals.

Status

Fixed

Informational Issues

A.3 Missing Zero Check [#L70]

```
69  // CONSTRUCTOR
70  constructor(uint _maxCap, uint256 _saleStartTime, uint256 _saleEndTime, address payable _projectOwner) public {
71    maxCap = _maxCap;
72    saleStartTime = _saleStartTime;
73    saleEndTime = _saleEndTime;
74    projectOwner = _projectOwner;
75  }
```

Description

Contracts lack zero address checks, hence are prone to be initialized with zero addresses.

Recommendation

Consider adding zero address checks in order to avoid risks of incorrect contract initializations.

Status

Fixed

A.4 Incorrect variable Name [#L66]

Description

Variable name totalparticipant is not accurately telling the total participant but keep track of a successful Buy Transaction

Recommendation

Consider Changing the variable name from totalparticipant to totalBuy

Status

Fixed

Functional Testing

Some of the tests performed are mentioned below

- Should be able call all getters
- Should be able to Buy by paying ETH
- Should be able to initialize MaxCap, SaleStartTime, SaleEndTime,projectOwner at Deployment
- Should be able to transferOwnership
- Should revert if user try to Buy before SaleStartTime
- Should revert if User try to Buy after SaleEndTime
- Should revert if TotalBnBReceive +value is greater than MaxCap
- Should be able to transferFrom
- Should revert if transfer amount exceeds balance

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 Metarix. 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. In The End ,Metarix Team Fixed all issues.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Metarix 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 Metarix 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
May, 2022

For

METARIX

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com