



QuillAudits

Audit Report November, 2021

For



Metastarter

Contents

Scope of Audit	01
Check Vulnerabilities	01
Techniques and Methods	02
Issue Categories	03
Number of security issues per severity.	03
Introduction	04
Issues Found – Code Review / Manual Testing	05
High Severity Issues	05
Medium Severity Issues	05
Low Severity Issues	05
Informational Issues	05
1. Multiple pragma statements	05
Functional Test Cases	07
Automated Tests	08
Results	09
Closing Summary	10

Scope of the Audit

The scope of this audit was to analyze and document the Metastarter 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
- BEP20 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 BEP-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	0	0	0	0
Closed	0	0	0	1

Introduction

During the period of November 13, 2021 to November 17, 2021 - QuillAudits Team performed a security audit for Metastarter smart contracts.

The code for the audit was taken from following the official link:
<https://github.com/Metastarter/Metastarter/blob/main/Metastarter.sol>

Ver	Date	Commit hash	Files
1	9 November	361749a8c19778fd1a816 bf07e0896336918cc4a	1. Metastarter.sol
2	17 November	9781631aab17b3143965 b1507e979206c6b475b3	1. Metastarter.sol

Issues Found

A. Contract – Metastarter

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

No issues were found.

Informational issues

1. Multiple pragma statements

Line	Code
1	pragma solidity >=0.6.0 <0.8.0;
23	pragma solidity >=0.6.0 <0.8.0;
100	pragma solidity >=0.6.0 <0.8.0;
313	pragma solidity >=0.6.0 <0.8.0;
617	pragma solidity >=0.6.0 <0.8.0;
657	pragma solidity >=0.6.2 <0.8.0;
845	pragma solidity >=0.6.0 <0.8.0;
914	pragma solidity 0.7.6;

Description

There are multiple pragma statements in the code. Only the compiler version 0.7.6 will work with the code, but keeping only one pragma statement helps in maintaining readability of the code.

Remediation

Keep a single pragma statement.

Status: Fixed

In version 2, the team fixed the issue with the recommended changes.



Functional Test Cases

- Should be able to transfer/transferFrom

PASS

- Burn and burnFrom should decrease totalSupply

PASS

- burnFrom and transferFrom should take allowance into consideration

PASS

- totalSupply should equal the specified amount

PASS

- Increase and decrease allowance should update allowance

PASS



Automated Tests

Slither

Address.isContract(address) (Metastarter.sol#680-689) uses assembly
 - INLINE ASM (Metastarter.sol#687)
 Address._verifyCallResult(bool,bytes,string) (Metastarter.sol#825-842) uses assembly
 - INLINE ASM (Metastarter.sol#834-837)
 Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage>

Different versions of Solidity is used:
 - Version used: ['0.7.6', '>=0.4.22<0.9.0', '>=0.6.0<0.8.0', '>=0.6.2<0.8.0']
 - >=0.6.0<0.8.0 (Metastarter.sol#1)
 - >=0.6.0<0.8.0 (Metastarter.sol#24)
 - >=0.6.0<0.8.0 (Metastarter.sol#100)
 - >=0.6.0<0.8.0 (Metastarter.sol#313)
 - >=0.6.0<0.8.0 (Metastarter.sol#617)
 - >=0.6.2<0.8.0 (Metastarter.sol#657)
 - >=0.6.0<0.8.0 (Metastarter.sol#845)
 - 0.7.6 (Metastarter.sol#914)
 - >=0.4.22<0.9.0 (Migrations.sol#2)
 Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used>

Address._verifyCallResult(bool,bytes,string) (Metastarter.sol#825-842) is never used and should be removed
 Address.functionCall(address,bytes) (Metastarter.sol#733-735) is never used and should be removed
 Address.functionCall(address,bytes,string) (Metastarter.sol#743-745) is never used and should be removed
 Address.functionCallWithValue(address,bytes,uint256) (Metastarter.sol#758-760) is never used and should be removed
 Address.functionCallWithValue(address,bytes,uint256,string) (Metastarter.sol#768-775) is never used and should be removed
 Address.functionDelegateCall(address,bytes) (Metastarter.sol#807-809) is never used and should be removed
 Address.functionDelegateCall(address,bytes,string) (Metastarter.sol#817-823) is never used and should be removed
 Address.functionStaticCall(address,bytes) (Metastarter.sol#783-785) is never used and should be removed

Address.functionStaticCall(address,bytes,string) (Metastarter.sol#793-799) is never used and should be removed
 Address.isContract(address) (Metastarter.sol#680-689) is never used and should be removed
 Address.sendValue(address,uint256) (Metastarter.sol#707-713) is never used and should be removed
 BEP20._setupDecimals(uint8) (Metastarter.sol#596-598) is never used and should be removed
 Context._msgData() (Metastarter.sol#18-21) is never used and should be removed
 SafeMath.div(uint256,uint256) (Metastarter.sol#232-235) is never used and should be removed
 SafeMath.div(uint256,uint256,string) (Metastarter.sol#287-290) is never used and should be removed
 SafeMath.mod(uint256,uint256) (Metastarter.sol#249-252) is never used and should be removed
 SafeMath.mod(uint256,uint256,string) (Metastarter.sol#307-310) is never used and should be removed
 SafeMath.mul(uint256,uint256) (Metastarter.sol#213-218) is never used and should be removed
 SafeMath.tryAdd(uint256,uint256) (Metastarter.sol#121-125) is never used and should be removed
 SafeMath.tryDiv(uint256,uint256) (Metastarter.sol#157-160) is never used and should be removed
 SafeMath.tryMod(uint256,uint256) (Metastarter.sol#167-170) is never used and should be removed
 SafeMath.tryMul(uint256,uint256) (Metastarter.sol#142-150) is never used and should be removed
 SafeMath.trySub(uint256,uint256) (Metastarter.sol#132-135) is never used and should be removed
 Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code>

Pragma version>=0.6.0<0.8.0 (Metastarter.sol#1) is too complex
 Pragma version>=0.6.0<0.8.0 (Metastarter.sol#24) is too complex
 Pragma version>=0.6.0<0.8.0 (Metastarter.sol#100) is too complex
 Pragma version>=0.6.0<0.8.0 (Metastarter.sol#313) is too complex
 Pragma version>=0.6.0<0.8.0 (Metastarter.sol#617) is too complex
 Pragma version>=0.6.2<0.8.0 (Metastarter.sol#657) is too complex
 Pragma version>=0.6.0<0.8.0 (Metastarter.sol#845) is too complex
 Pragma version>=0.4.22<0.9.0 (Migrations.sol#2) is too complex
 Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity>


```

Low level call in Address.sendValue(address,uint256) (Metastarter.sol#707-713):
  - (success) = recipient.call{value: amount}() (Metastarter.sol#711)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (Metastarter.sol#768-775):
  - (success, returndata) = target.call{value: value}(data) (Metastarter.sol#773)
Low level call in Address.functionStaticCall(address,bytes,string) (Metastarter.sol#793-799):
  - (success, returndata) = target.staticcall(data) (Metastarter.sol#797)
Low level call in Address.functionDelegateCall(address,bytes,string) (Metastarter.sol#817-823):
  - (success, returndata) = target.delegatecall(data) (Metastarter.sol#821)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Variable Migrations.last_completed_migration (Migrations.sol#6) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (Metastarter.sol#19)" inContext (Metastarter.sol#13-22)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

name() should be declared external:
  - BEP20.name() (Metastarter.sol#373-375)
symbol() should be declared external:
  - BEP20.symbol() (Metastarter.sol#381-383)
decimals() should be declared external:
  - BEP20.decimals() (Metastarter.sol#398-400)
totalSupply() should be declared external:
  - BEP20.totalSupply() (Metastarter.sol#405-407)
balanceOf(address) should be declared external:
  - BEP20.balanceOf(address) (Metastarter.sol#412-414)
transfer(address,uint256) should be declared external:
  - BEP20.transfer(address,uint256) (Metastarter.sol#424-427)
approve(address,uint256) should be declared external:
  - BEP20.approve(address,uint256) (Metastarter.sol#443-446)
transferFrom(address,address,uint256) should be declared external:
  - BEP20.transferFrom(address,address,uint256) (Metastarter.sol#461-465)
increaseAllowance(address,uint256) should be declared external:
  - BEP20.increaseAllowance(address,uint256) (Metastarter.sol#479-482)
decreaseAllowance(address,uint256) should be declared external:
  - BEP20.decreaseAllowance(address,uint256) (Metastarter.sol#498-501)
burn(uint256) should be declared external:
  - BEP20Burnable.burn(uint256) (Metastarter.sol#634-636)
burnFrom(address,uint256) should be declared external:
  - BEP20Burnable.burnFrom(address,uint256) (Metastarter.sol#649-654)
renounceOwnership() should be declared external:
  - Ownable.renounceOwnership() (Metastarter.sol#895-898)
transferOwnership(address) should be declared external:
  - Ownable.transferOwnership(address) (Metastarter.sol#904-908)
setCompleted(uint256) should be declared external:
  - Migrations.setCompleted(uint256) (Migrations.sol#16-18)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
. analyzed (9 contracts with 75 detectors), 55 result(s) found

```

Results

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

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines.

No instances of Integer Overflow and Underflow vulnerabilities or Back-Door Entry were found in the contract, but relying on other contracts might cause Reentrancy Vulnerability.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the **Metastarter** 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 **Metastarter** 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



Metastarter



QuillAudits

📍 Canada, India, Singapore, United Kingdom

🌐 audits.quillhash.com

✉ audits@quillhash.com