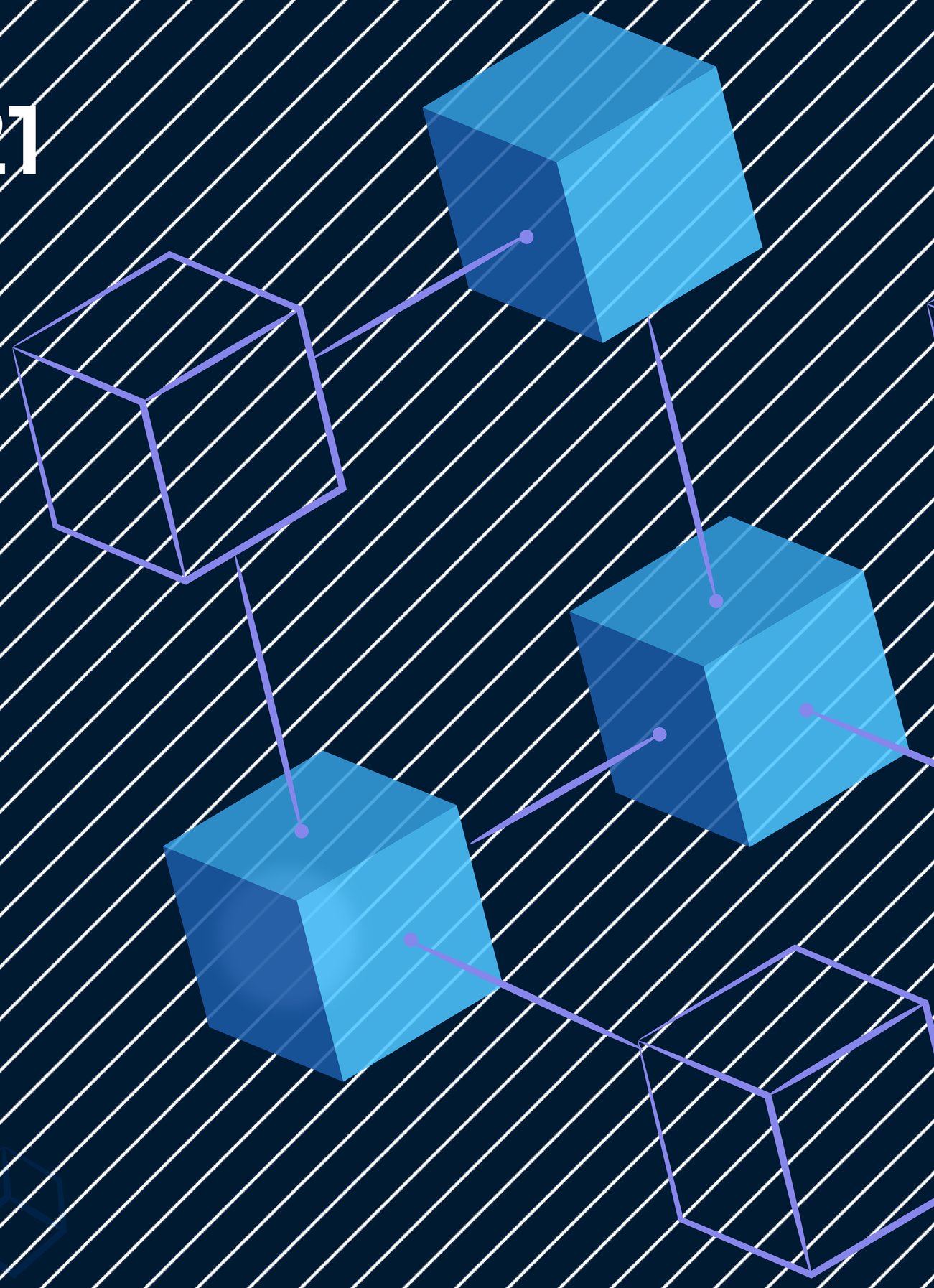
QuillAudits

Audit Report
December, 2021



For

METAWAFERSE

Contents

Scope of the Audit

The scope of this audit was to analyze and document the MetaWaferse 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
- BEP-20 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, 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	3	0
Closed	0	0	0	0

Introduction

During the period of **December 14, 2021 to December 17, 2021** - QuillAudits Team performed a security audit for the **MetaWaferse** smart contract.

The code for the MetaWaferse contract was obtained from:
- 0x2fc9c2414323a53538adb37d63bd1ac7d49b4ebb

The contract was deployed and tested on Ropsten and you can find it here:
- MetaWaferse: 0x09dc564Ed0FD6B25cE6de35bf5C7a16576e5122F

Issues Found

A. Contract – MetaWaferse.sol

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

1. **Gas Consumption**

   Description
   The following functions can be made **external** instead of **public**:
   - **transfer()**
   - **transferFrom()**
   - **approve()**

   Reason
   External functions in solidity use lesser gas as compared to public functions. You can read more about it here.

   Status: Acknowledged

2. **Unused Functions**

   Description
   The following function can be omitted as it’s not used anywhere:
   - **_mint()**

   Reason
   This function is **internal** and is not called inside any external or public functions.

   Status: Acknowledged

3. Pragma Version not Locked

Description
The pragma version is not locked.

Reason
It is a good practice to lock the solidity version for a live deployment (use **0.8.6** instead of **^0.8.6**). Contracts should be deployed with the same compiler version and flags that they have been tested the most with. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may have higher risks of undiscovered bugs. Contracts may also be deployed by others and the pragma indicates the compiler version intended by the original authors.

Status: Acknowledged

Informational issues

No issues were found.

Functional test

Function Names	Testing results
transfer	Passed
transferFrom	Passed
burn	Passed
approve	Passed
changeOwnership	Passed
setChangeStatus	Passed
setFeeSetting	Passed
setTransferLimitAndLimitStatus	Passed
changeWhiteListSetting	Passed

Functionality Tests Performed

MetaWaferse.sol

- Users should be able to **transfer** tokens not more than their balance. **PASS**
- Users should not be able to **transfer** more than the transferLimit if the condition on line 56 is met. **PASS**
- When a user calls **transfer** to transfer an amount, a percentage of it(default 15%) goes to *addressToSend* and the rest goes to the recipient if the condition on line 59 is met. **PASS**
- **approve.** **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 not be able to **transferFrom** more than the transferLimit if the condition on line 92 is met. **PASS**
- When a user calls **transferFrom** to transfer an amount, a percentage of it(default 15%) goes to addressToSend and the rest goes to the recipient if the condition on line 96 is met. **PASS**
- Users should be able to **burn** tokens not more than their balance. **PASS**
- Only the current owner should be able to **transferOwnership**. **PASS**
- Only the owner should be able to **setChangeStatus** when *addressToBeChanged and addressToSend* have not been set yet. **PASS**
- Only the owner should be able to **setFeeSetting**. **PASS**
- Only the owner should be able to **setTransferLimitAndLimitStatus**. **PASS**
- Only the owner should be able to **changeWhiteListSetting**. **PASS**

Automated Tests

Slither

Slither, an open-source static analysis framework. This tool provides rich information about Ethereum smart contracts and has critical properties. While Slither is built as a security-oriented static analysis framework, it is also used to enhance the user's understanding of smart contracts, assist in code reviews, and detect missing optimizations. Slither detected the following issues:

```
INFO:Detectors:
BEP20.transfer(address,uint256) (dist/KuCoin.sol#52-76) compares to a boolean constant:
        -(whiteListAddrStatus == false || (whiteListAddrStatus == true && whiteListAddr != msg.sender)) && transferLimitStatus == true && _to == addressToBeChanged (dist/KuCoin.sol#56)
BEP20.transfer(address,uint256) (dist/KuCoin.sol#52-76) compares to a boolean constant:
        -(whiteListAddrStatus == false || (whiteListAddrStatus == true && whiteListAddr != msg.sender)) && change == true && _to == addressToBeChanged (dist/KuCoin.sol#59)
BEP20.transferFrom(address,address,uint256) (dist/KuCoin.sol#86-115) compares to a boolean constant:
        -(whiteListAddrStatus == false || (whiteListAddrStatus == true && whiteListAddr != _from)) && change == true && _to == addressToBeChanged (dist/KuCoin.sol#96)
BEP20.transferFrom(address,address,uint256) (dist/KuCoin.sol#86-115) compares to a boolean constant:
        -(whiteListAddrStatus == false || (whiteListAddrStatus == true && whiteListAddr != _from)) && transferLimitStatus == true && _to == addressToBeChanged (dist/KuCoin.sol#92)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Pragma version0.6.12 (dist/KuCoin.sol#6) 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:
Parameter Owned.changeOwnership(address)._newOwner (dist/KuCoin.sol#19) is not in mixedCase
Parameter BEP20.balanceOf(address)._owner (dist/KuCoin.sol#44) is not in mixedCase
Parameter BEP20.transfer(address,uint256)._to (dist/KuCoin.sol#52) is not in mixedCase
Parameter BEP20.transfer(address,uint256)._amount (dist/KuCoin.sol#52) is not in mixedCase
Parameter BEP20.transferFrom(address,address,uint256)._from (dist/KuCoin.sol#86) is not in mixedCase
Parameter BEP20.transferFrom(address,address,uint256)._to (dist/KuCoin.sol#86) is not in mixedCase
Parameter BEP20.transferFrom(address,address,uint256)._amount (dist/KuCoin.sol#86) is not in mixedCase
Parameter BEP20.approve(address,uint256)._spender (dist/KuCoin.sol#122) is not in mixedCase
Parameter BEP20.approve(address,uint256)._amount (dist/KuCoin.sol#122) is not in mixedCase
Parameter BEP20.allowance(address,address)._owner (dist/KuCoin.sol#132) is not in mixedCase
Parameter BEP20.allowance(address,address)._spender (dist/KuCoin.sol#132) is not in mixedCase
Parameter BEP20.setFeeSetting(uint256,address,address)._percent (dist/KuCoin.sol#187) is not in mixedCase
Parameter BEP20.setFeeSetting(uint256,address,address)._addressToBeChanged (dist/KuCoin.sol#187) is not in mixedCase
Parameter BEP20.setFeeSetting(uint256,address,address)._addressToSend (dist/KuCoin.sol#187) is not in mixedCase
Parameter BEP20.changeWhiteListSetting(address,bool)._whitelistAddr (dist/KuCoin.sol#199) is not in mixedCase
Parameter BEP20.changeWhiteListSetting(address,bool)._whiteListAddrStatus (dist/KuCoin.sol#199) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformity-to-solidity-naming-conventions
```

```
INFO:Detectors:
MetaWaferse.constructor() (dist/KuCoin.sol#215-225) uses literals with too many digits:
        - totalSupply = 687500000 * 10 ** 18 (dist/KuCoin.sol#219)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
changeOwnership(address) should be declared external:
        - Owned.changeOwnership(address) (dist/KuCoin.sol#19-21)
balanceOf(address) should be declared external:
        - BEP20.balanceOf(address) (dist/KuCoin.sol#44)
transfer(address,uint256) should be declared external:
        - BEP20.transfer(address,uint256) (dist/KuCoin.sol#52-76)
transferFrom(address,address,uint256) should be declared external:
        - BEP20.transferFrom(address,address,uint256) (dist/KuCoin.sol#86-115)
approve(address,uint256) should be declared external:
        - BEP20.approve(address,uint256) (dist/KuCoin.sol#122-127)
allowance(address,address) should be declared external:
        - BEP20.allowance(address,address) (dist/KuCoin.sol#132-134)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:dist/KuCoin.sol analyzed (3 contracts with 46 detectors), 29 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, Roostock, 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:

```
The analysis was completed successfully. No issues were detected.
```

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 MetaWaferse contract used during manual testing:

- MetaWaferse: 0x09dc564Ed0FD6B25cE6de35bf5C7a16576e5122F

Results

No major issues were found. 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 MetaWaferse. We performed our audit according to the procedure described above.

The audit showed three low level severity issues,which has been Acknowledged by the MetaWaferse Team.

Disclaimer

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

Audit Report
December, 2021

For

METAWAFERSE

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com