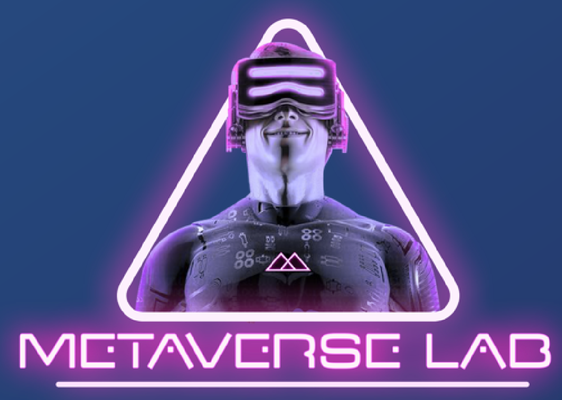
QuillAudits

Audit Report
January, 2022

For



Contents

Scope of the Audit

The scope of this audit was to analyze and document the Metaverse 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
- 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 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 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, 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	3	4
Closed	0	0	0	0

Introduction

During the period of **January 12, 2021 to January 17, 2021** - QuillAudits Team performed a security audit for Metaverse token smart contracts.

The code for the audit was taken from following the official link:
Codebase: 0x57E2A2FF2622cacEC775cf8D4C2848Aa9B31528c

V	Date	Contract address	Network
1	Jan 12	0x57E2A2FF2622cacEC775cf8D4C2848Aa9B31528c	Binance

Issues Found – Code Review / Manual Testing

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

1. **Missing zero address validation**

   Missing zero address check for to address in the following methods:
   - changeFeeAddress()
   - changeWhiteListSetting()

   Recommendation
   Add a ‘require’ to check to address != address(0)

   Status: Acknowledged

2. **Missing validation for percentage**

   Missing check for value of percent less than 100.
   - changeFeePercentage()

   Recommendation
   Add a ‘require’ to check to check percent less than 100.

   Status: Acknowledged

3. **Violation of BEP20 compliance standard**

   There are certain guidelines proposed by binance-chain for the development of BEP20 tokens that includes the token's status, specification, methods, implementation, etc.
   However, this contract violates the guidelines by not adding the “getOwner” method that Returns the bep20 token owner which is necessary for binding with bep2 token.
   Tokens which don’t implement this method will never flow across the Binance Chain and Binance Smart Chain.

5.1.1.6 getOwner

```
function getOwner() external view returns (address);
```

- Returns the bep20 token owner which is necessary for binding with bep2 token.
- **NOTE** - This is an extended method of EIP20. Tokens which don't implement this method will never flow across the Binance Chain and Binance Smart Chain.

Recommendation

Use BEP20 token guidelines when developing a smart contract for a binance smart chain token.

References
BEPs/BEP20.md at master - Tokens on Binance Smart Chain

Status: Acknowledged

Informational issues

4. Missing comments and description

Comments and Description of the methods and the variables are missing, it's hard to read and understand the purpose of the variables and the methods in context of the whole picture

Recommendation
Consider adding NatSpec format comments for the comments and state variables

Status: Acknowledged

5. Centalization risk

Some functions that use the “onlyOwner” pose a centralization risk to the contracts the “onlyOwner” modifier may hinder with the working of smart contract in case of loss of the public key. Effected functions are:
- changeFeeAddress
- changeWhiteListSetting
- setFeeStatusAndAddress
- changeFeePercent

```
// Set change status
function setFeeStatusAndAddress(bool val,address _feeApplyAddress) external onlyOwner {
    require(applyFeeStatus != val, "Already in this state");
    require(burnPerAddress != address(0) && developmentPerAddress != address(0) && holderPer
    applyFeeStatus = val;
    feeApplyAddress = _feeApplyAddress;
}

// Set percent amount
function changeFeePercent(uint _burnPercent, uint _developmentPercent, uint _holderPercent)
    burnPercent = _burnPercent;
    developmentPercent = _developmentPercent;
    holderPercent = _holderPercent;
}

 // Set percent amount address
function changeFeeAddress(address _burnPerAddress, address _developmentPerAddress, address _
    burnPerAddress = _burnPerAddress;
    developmentPerAddress = _developmentPerAddress;
    holderPerAddress = _holderPerAddress;
}
```

Recommendation

Instead of using a signal owned account and providing ownership to the signer. We recommend to use the multisig account and provide ownership to the same multisig address and use that account for the owner rights.

Status: Acknowledged

6. Comparison with boolean constants

Function “transfer” and “transferFrom” compare with a boolean constant, whereas Boolean constants can be used directly and do not need to be compared to true or false.

```
if((whiteListAddrStatus == false || (whiteListAddrStatus == true && whiteListAddr != _from)
    require(_amount<=transferLimit,"Transfer Limit Exceeds");
}

if((whiteListAddrStatus == false || (whiteListAddrStatus == true && whiteListAddr != _from)
```

Recommendation

Boolean constants can be used directly and do not need to be compared to true or false.

Status: Acknowledged

7. Floating Pragma

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively. Here, we can see that the pragma is unlocked, which is not a good coding practice.

```
// SPDX-License-Identifier: none
pragma solidity ^0.8.6;

contract Owned {

    /// Modifier for owner only function call
    modifier onlyOwner() {
        require(msg.sender == owner, "Only owner");
        _;
    }
```

Recommendation

The pragma versions used in the contract are not locked. Consider using the latest versions among 0.8.11 for deploying the contracts and libraries as it does not compile for any other version and can be confusing for a developer. Solidity source files indicate the versions of the compiler they can be compiled with.

Status: Acknowledged

Functional Tests

- should be able to transfer tokens PASS
- should able to mint tokens PASS
- should be able to burn tokens PASS
- should be able to transfer ownership PASS
- should be able to approve PASS
- should be able to return allowance of the approved address PASS
- should revert if burn amount exceeds balance PASS
- Should revert if the sender’s balance is less than the amount PASS
- Should revert if the recipient’s balance and transfer amount’s sum is less than the recipient’s balance PASS
- Should revert if the caller don’t have the allowance for making a call to transferFrom function PASS
- Should revert if applyFeeStatus is already set to true for an address. PASS
- should revert if the sender or recipient address is zero PASS
- should revert if the transfer amount is greater than transfer limit PASS

Binance Testnet Test Contract

Metaverse Token: 0xefcDc0fc735a3cBb3a1b9F7D75da65507Af06498

tansferLimitStatus

- Value is false - No transfer check PASS
- Set value to true PASS
- Not able to Set feeApplyStatus PASS
- Set Change Address PASS
- Able to set feeApplyStatus PASS
- Change fee percent PASS

// transfer

- Transfer with change fee percent set PASS

whiteListAddress

- set whiteListAddress - changeWhiteListSetting PASS

// transfer From

- No approval for other address PASS
- Set approval PASS
- transferFrom PASS

Burn

- Amount greater than the user balance PASS
- Amount less than user balance PASS

Automated Tests

Mythril

```
Owned.changeOwnership(address) (metaverse.sol#23-25) should emit an event for:
        - owner = _newOwner (metaverse.sol#24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control

BEP20.setFeeStatusAndAddress(bool,address)._feeApplyAddress (metaverse.sol#219) lacks a zero-check on :
                - feeApplyAddress = _feeApplyAddress (metaverse.sol#223)
BEP20.changeFeeAddress(address,address,address)._burnPerAddress (metaverse.sol#234) lacks a zero-check on :
                - burnPerAddress = _burnPerAddress (metaverse.sol#235)
BEP20.changeFeeAddress(address,address,address)._developmentPerAddress (metaverse.sol#234) lacks a zero-check on :
                - developmentPerAddress = _developmentPerAddress (metaverse.sol#236)
BEP20.changeFeeAddress(address,address,address)._holderPerAddress (metaverse.sol#234) lacks a zero-check on :
                - holderPerAddress = _holderPerAddress (metaverse.sol#237)
BEP20.changeWhiteListSetting(address,bool)._whitelistAddr (metaverse.sol#246) lacks a zero-check on :
                - whiteListAddr = _whitelistAddr (metaverse.sol#247)
Owned.changeOwnership(address)._newOwner (metaverse.sol#23) lacks a zero-check on :
                - owner = _newOwner (metaverse.sol#24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

BEP20.transfer(address,uint256) (metaverse.sol#61-100) compares to a boolean constant:
        -(whiteListAddrStatus == false || (whiteListAddrStatus == true && whiteListAddr != msg.sender)) && applyFeeStatus == true && _to == feeApplyAddress (m
etaverse.sol#68)
BEP20.transfer(address,uint256) (metaverse.sol#61-100) compares to a boolean constant:
        -(whiteListAddrStatus == false || (whiteListAddrStatus == true && whiteListAddr != msg.sender)) && transferLimitStatus == true && _to == feeApplyAddre
ss (metaverse.sol#85)
BEP20.transferFrom(address,address,uint256) (metaverse.sol#110-154) compares to a boolean constant:
        -(whiteListAddrStatus == false || (whiteListAddrStatus == true && whiteListAddr != _from)) && applyFeeStatus == true && _to == feeApplyAddress (metave
rse.sol#120)
BEP20.transferFrom(address,address,uint256) (metaverse.sol#110-154) compares to a boolean constant:
        -(whiteListAddrStatus == false || (whiteListAddrStatus == true && whiteListAddr != _from)) && transferLimitStatus == true && _to == feeApplyAddress (m
etaverse.sol#116)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality

BEP20._mint(address,uint256) (metaverse.sol#184-192) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.6 (metaverse.sol#10) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Parameter Owned.changeOwnership(address)._newOwner (metaverse.sol#23) is not in mixedCase
Parameter BEP20.balanceOf(address)._owner (metaverse.sol#53) is not in mixedCase
Parameter BEP20.transfer(address,uint256)._to (metaverse.sol#61) is not in mixedCase
Parameter BEP20.transfer(address,uint256)._amount (metaverse.sol#61) is not in mixedCase
Parameter BEP20.transferFrom(address,address,uint256)._from (metaverse.sol#110) is not in mixedCase
Parameter BEP20.transferFrom(address,address,uint256)._to (metaverse.sol#110) is not in mixedCase
Parameter BEP20.transferFrom(address,address,uint256)._amount (metaverse.sol#110) is not in mixedCase
Parameter BEP20.approve(address,uint256)._spender (metaverse.sol#161) is not in mixedCase
Parameter BEP20.approve(address,uint256)._amount (metaverse.sol#161) is not in mixedCase
Parameter BEP20.allowance(address,address)._owner (metaverse.sol#171) is not in mixedCase
Parameter BEP20.allowance(address,address)._spender (metaverse.sol#171) is not in mixedCase
Parameter BEP20.setFeeStatusAndAddress(bool,address)._feeApplyAddress (metaverse.sol#219) is not in mixedCase
Parameter BEP20.changeFeePercent(uint256,uint256,uint256)._burnPercent (metaverse.sol#227) is not in mixedCase
Parameter BEP20.changeFeePercent(uint256,uint256,uint256)._developmentPercent (metaverse.sol#227) is not in mixedCase
Parameter BEP20.changeFeePercent(uint256,uint256,uint256)._holderPercent (metaverse.sol#227) is not in mixedCase
Parameter BEP20.changeFeeAddress(address,address,address)._burnPerAddress (metaverse.sol#234) is not in mixedCase
Parameter BEP20.changeFeeAddress(address,address,address)._developmentPerAddress (metaverse.sol#234) is not in mixedCase
Parameter BEP20.changeFeeAddress(address,address,address)._holderPerAddress (metaverse.sol#234) is not in mixedCase
Parameter BEP20.changeWhiteListSetting(address,bool)._whitelistAddr (metaverse.sol#246) is not in mixedCase
Parameter BEP20.changeWhiteListSetting(address,bool)._whiteListAddrStatus (metaverse.sol#246) is not in mixedCase
Contract METAVERSE_LAB (metaverse.sol#254-287) is not in CapWords
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

METAVERSE_LAB.constructor() (metaverse.sol#262-273) uses literals with too many digits:
        - totalSupply = 20000000 * 10 ** 18 (metaverse.sol#266)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

changeOwnership(address) should be declared external:
        - Owned.changeOwnership(address) (metaverse.sol#23-25)
balanceOf(address) should be declared external:
        - BEP20.balanceOf(address) (metaverse.sol#53)
approve(address,uint256) should be declared external:
        - BEP20.approve(address,uint256) (metaverse.sol#101-106)
allowance(address,address) should be declared external:
        - BEP20.allowance(address,address) (metaverse.sol#171-173)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

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, documented, and adhere to guidelines. Several issues of Low severity and information issues have been reported which has been Acknowledged by the Auditee.

Disclaimer

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

Audit Report
January, 2022

For

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com