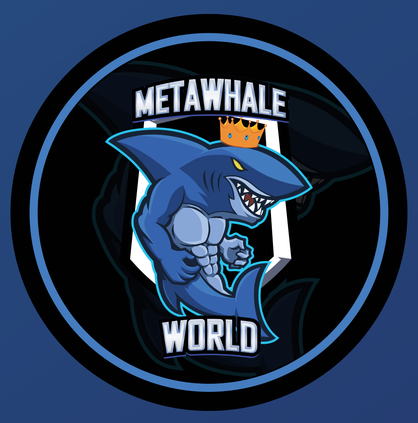
QuillAudits

Audit Report
March, 2022

For



Contents

Overview

MetaWhale

A blockchain-based virtual Gaming and NFT Metaverse. A virtual gaming platform allows players to build and monetize their gaming experiences. It is working on the Binance Blockchain using MTW’s utility Token.

Scope of Audit

The scope of this audit was to analyze MetaWhale smart contract’s for quality, security, and correctness.

MetaWhale Contract:
Contract Address 0xd3ac199e6e6a1668ed6566b6f6dcdf7641868731 | BscScan

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
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC20 transfer() does not return boolean
- ERC20 approve() race
- 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.
- 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
Mythril, Slither, Surya, Solhint.

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	3
Closed	0	0	0	0

Functional Testing Results

Some of the tests performed are mentioned below:

- ✓ Should be able to transfer tokens.
- ✓ Should be able to approve tokens.
- ✓ Should be able to spend allowed tokens.
- ✓ Should be able to increase and decrease allowance.
- ✓ Only the owner can mint tokens to the owner address.
- ✓ Reverts if spender exceeds allowance while transferring allowed tokens.
- ✓ Reverts if the transfer amount exceeds the current balance.
- ✓ Reverts to zero address transfers.
- ✗ Should be able to burn tokens.

Issues Found – Code Review / Manual Testing

High severity issues

No issues found

Medium severity issues

No issues found

Low severity issues

No issues found

Informational issues

1. **Old version of Solidity**

   This contract is using solidity version 0.5.16, solc frequently releases new compiler versions. Using an old version prevents access to new Solidity security checks.

   Recommendation
   Use the latest compiler version in order to avoid bugs introduced in older versions.

   Status: Acknowledged

2. **Dead-code/Unused functions**

   [#L492] _burnFrom internal function is unused in contract.[#L460] _burn function is getting called in _burnFrom but there's no function which calls this function. Hence both the functions are unused.

```
460   function _burn(address account, uint256 amount) internal {
461     require(account != address(0), "BEP20: burn from the zero address");
462
463     _balances[account] = _balances[account].sub(amount, "BEP20: burn amount exceeds balance");
464     _totalSupply = _totalSupply.sub(amount);
465     emit Transfer(account, address(0), amount);
466   }

492   function _burnFrom(address account, uint256 amount) internal {
493     _burn(account, amount);
494     _approve(account, _msgSender(), _allowances[account][_msgSender()].sub(amount, "BEP20: burn amount exceeds allowance"));
495   }
496 }
```

Recommendation
Consider removing unused functions.

Status: Acknowledged

3. Token Decimals

Contract is using 8 decimals for tokens.

Here 1 token would be 1*(10**8) = 100000000 Wei.

It may happen that any other smart contract uses/accepts this token for some reason.

That smart contract calculates the token amount sent by the user assuming its 18 decimal token, which can result in unwanted outcomes.

Eg: care needs to be taken in this type of scenario.
1. User sends 1 token (“1*(10**8)” in this case) to a smart contract.
2. The smart contract which accepts this token checks the token amount sent by User which was 1*(10**8) = 100000000
3. While calculating amount sent by the User, smart contract uses 18 decimals and expects 1 token sent to be 1*(10**18) = 1000000000000000000

In this case this condition will fail since token amount sent by User is 100000000 i.e 1*(10**8) and not 1*(10**18)

```
287   constructor() public {
288     _name = "MetaWhale";
289     _symbol = "MTW";
290     _decimals = 8;
291     _totalSupply = 227000000000000000;
292     _balances[msg.sender] = _totalSupply;
293
294     emit Transfer(address(0), msg.sender, _totalSupply);
295   }
296
```

Recommendation
1. In this case tokens decimals are only used for representation purposes,but we recommend reviewing business logic.
2. In cases like sending,approving or integrating token contract’s with other functionality, care needs to be taken according to logic.

Closing Summary

No Major Issues Found During the Audit,only Some Informational issues were discovered, which are Acknowledged by the Metawhale Team.

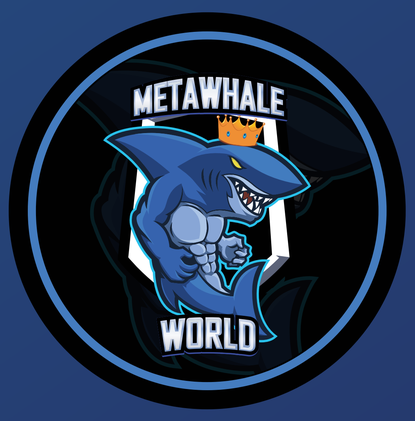
Disclaimer

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

Audit Report
March, 2022

For



QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com