QuillAudits

# Audit Report
# September, 2021

# Contents

## Scope of Audit

The scope of this audit was to analyze and document the MAST 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 analyzed 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.
SmartCheck.

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 security of smart contracts.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of 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 possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Slither, SmartCheck.

# Issue Categories

Every issue in this report has been assigned with a severity level. There are four levels of severity and each of them has been explained below.

## High severity issues

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 to be fixed before moving to a live environment.

## Medium level severity issues

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 level severity issues

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 four issues which 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 | 1 | 0 |
| Acknowledged | 0 | 0 | 0 | 0 |
| Closed | 0 | 0 | 0 | 0 |

# Introduction

During the period of **September 1, 2021 to September 3, 2021 -** QuillAudits Team performed a security audit for MASTToken smart contract.

The code for the audit was taken from the following official File: https://bscscan.com/address/0xB31FD05CAF496CEf34C945667E90dd89C20E0D09#code

# Issues Found – Code Review / Manual Testing

## High severity issues

No issues were found.

## Medium severity issues

No issues were found.

## Low level severity issues

1. Floating Pragma

   Description
   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.

   Remediation
   Lock the pragma version and also consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

   Status: Open

## Informational

No issues were found.

# Functional test

| Function Names | Testing results |
| --- | --- |
| Constructor() | Passed |
| mint() | Passed |
| _beforeTokenTransfer() | Passed |
| setBlacklist() | Passed |
| pause() | Passed |
| unpause() | Passed |

# Automated Testing

## Slither

No major or Low issues were found. There are just some Informational errors that were reported by the tool.

```
INFO:Printers:
Compiled with solc
Number of lines: 914 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 10 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 17
Number of informational issues: 17
Number of low issues: 0
Number of medium issues: 0
Number of high issues: 0

ERCs: ERC20

+----------------------+-------------+-------+---------------------+--------------+--------------+
|         Name         | # functions | ERCS  |     ERC20 info      | Complex code |   Features   |
+----------------------+-------------+-------+---------------------+--------------+--------------+
|       SafeMath       |      13     |       |                     |      No      |              |
| UpgradableGovernance |      2      |       |                     |      No      | Upgradeable  |
|     TokenDetail      |      41     | ERC20 |      Pausable       |      No      | Upgradeable  |
|                      |             |       |     ∞ Minting       |              |              |
|                      |             |       | Approve Race Cond.  |              |              |
|                      |             |       |                     |              |              |
+----------------------+-------------+-------+---------------------+--------------+--------------+
```

```
INFO:Detectors:
Context._msgData() (TokenDetail.sol#106-109) is never used and should be removed
SafeMath.div(uint256,uint256) (TokenDetail.sol#248-251) is never used and should be removed
SafeMath.div(uint256,uint256,string) (TokenDetail.sol#303-306) is never used and should be removed
SafeMath.mod(uint256,uint256) (TokenDetail.sol#265-268) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (TokenDetail.sol#323-326) is never used and should be removed
SafeMath.mul(uint256,uint256) (TokenDetail.sol#229-234) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (TokenDetail.sol#137-141) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (TokenDetail.sol#173-176) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (TokenDetail.sol#183-186) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (TokenDetail.sol#158-166) is never used and should be removed
SafeMath.trySub(uint256,uint256) (TokenDetail.sol#148-151) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version>=0.6.0<0.8.0 (TokenDetail.sol#867) is too complex
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter UpgradableProduct.upgradeImpl(address)._newImpl (TokenDetail.sol#826) is not in mixedCase
Parameter UpgradableGovernance.upgradeGovernance(address)._newGovernor (TokenDetail.sol#852) is not in mixedCase
Parameter TokenDetail.setBlacklist(address,bool)._toBlacklist (TokenDetail.sol#903) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (TokenDetail.sol#107)" inContext (TokenDetail.sol#101-110)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
TokenDetail.constructor() (TokenDetail.sol#881-884) uses literals with too many digits:
        - _mint(msg.sender,100000000000000 * 1e18) (TokenDetail.sol#883)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
```

```
INFO:Detectors:
name() should be declared external:
        - ERC20.name() (TokenDetail.sol#394-396)
symbol() should be declared external:
        - ERC20.symbol() (TokenDetail.sol#402-404)
decimals() should be declared external:
        - ERC20.decimals() (TokenDetail.sol#419-421)
totalSupply() should be declared external:
        - ERC20.totalSupply() (TokenDetail.sol#426-428)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (TokenDetail.sol#433-435)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (TokenDetail.sol#445-448)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (TokenDetail.sol#464-467)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (TokenDetail.sol#482-486)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (TokenDetail.sol#500-503)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (TokenDetail.sol#519-522)
burn(uint256) should be declared external:
        - ERC20Burnable.burn(uint256) (TokenDetail.sol#660-662)
burnFrom(address,uint256) should be declared external:
        - ERC20Burnable.burnFrom(address,uint256) (TokenDetail.sol#675-680)
upgradeImpl(address) should be declared external:
        - UpgradableProduct.upgradeImpl(address) (TokenDetail.sol#826-832)
upgradeGovernance(address) should be declared external:
        - UpgradableGovernance.upgradeGovernance(address) (TokenDetail.sol#852-858)
setBlacklist(address,bool) should be declared external:
        - TokenDetail.setBlacklist(address,bool) (TokenDetail.sol#903-905)
pause() should be declared external:
        - TokenDetail.pause() (TokenDetail.sol#907-909)
unpause() should be declared external:
        - TokenDetail.unpause() (TokenDetail.sol#911-913)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

# Disclaimer

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

# Closing Summary

Overall, in the initial audit, there is one low severity issue associated with the token. It is recommended to fix these before deployment. No instances of Integer Overflow and Underflow, Reentrancy, or other major vulnerabilities are found in the contract.

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com