QuillAudits

Audit Report, June, 2024

For

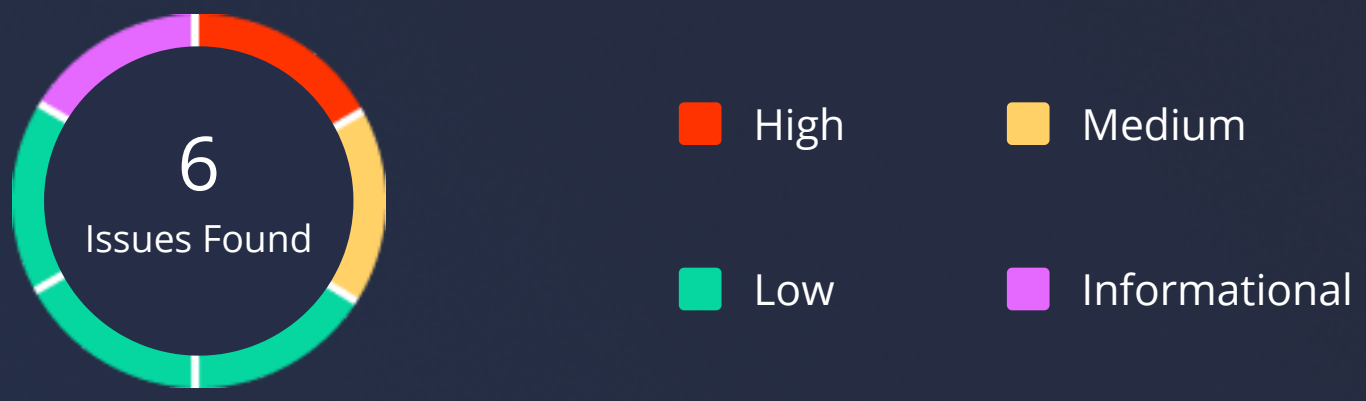
Table of Content

Table of Content

Executive Summary

Project Name	MDC Token
Project URL	https://muslimdigitalcoin.io
Overview	The MDC Token contract is built on Ethereum and adheres to the ERC20 standard. It includes features such as token vesting, managing multiple wallet roles, and distributing initial token supplies. The contract is designed to handle the complexities of token distribution and vesting for various stakeholders including founders, advisors, and initiatives like the Learn Islam program.
Audit Scope	The scope of this audit was to analyze the MDC codebase for quality, security, and correctness.
Contracts in Scope	https://github.com/MuslimDigitalCoin/smart-contract
Branch	Main
Contracts	mdcToken.sol
Commit Hash	fc2e717c6f8c8e9a028bf736d937daa6c666c791
Language	Solidity
Blockchain	Ethereum
Method	Manual Review, Automated tools,Functional testing
Review 1	31st May 2024 - 3rd June 2024
Updated Code Received	18th June 2024
Review 2	19th June 2024 - 22nd June 2024
Fixed In	e6b4ebab6f1d090bb330e814741916ec58431258

Number of Issues per Severity



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	1	1	3	1

Checked Vulnerabilities

- Access Management
- Arbitrary write to storage
- Centralization of control
- Ether theft
- Improper or missing events
- Logical issues and flaws
- Arithmetic Correctness
- Race conditions/front running
- SWC Registry
- 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
- ERC's conformance
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Multiple Sends
- Using suicide
- Using delegatecall
- Upgradeable safety
- Using throw

Checked Vulnerabilities

- Using inline assembly
- Style guide violation
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments, match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC 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.

Static Analysis

A 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 analyzed, 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 behavior 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,Solhint, Mythril, Slither, Solidity static analysis.

Types of Severity

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.

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 be fixed before moving to a live environment.

Medium 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 Severity Issues

Low-level severity issues can cause minor impact and 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 four 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.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

A. Contract - MDCToken.sol

High Severity Issues

A.1 Tokens Will Be Locked Indefinitely Due to Mismatched Arrays

Line	Function - Constructor
119	lockTokens(mdcTeamAndAdvisorsWallet, TEAM_AND_ADVISORS_UNLOCK_PERCENTAGES, _unlockFrequencyFoundersAndMdcTeam, 80000000000000000000000000); lockTokens(foundersAndEarlyInvestorsWallet, FOUNDERS_AND_EARLY_INVESTORS_UNLOCK_PERCENTAGES, _unlockFrequencyFoundersAndMdcTeam, 42500000000000000000000000); lockTokens(learnIslamWallet, LEARN_ISLAM_UNLOCK_PERCENTAGES, _unlockFrequencyForIslamFund, 12500000000000000000000000);

Description

The constructor accepts arrays _unlockFrequencyFoundersAndMdcTeam and _unlockFrequencyForIslamFund as parameters to specify unlock frequencies. These arrays are used in the lockTokens function along with TEAM_AND_ADVISORS_UNLOCK_PERCENTAGES to define token unlock schedules. However, there is no check to ensure that the length of _unlockFrequencyFoundersAndMdcTeam matches the length of TEAM_AND_ADVISORS_UNLOCK_PERCENTAGES. This mismatch can lead to a situation where tokens remain locked indefinitely if the arrays are not of equal length.

Remediation

Add a check in the constructor to ensure that the lengths of the _unlockFrequencyFoundersAndMdcTeam, _unlockFrequencyForIslamFund and TEAM_AND_ADVISORS_UNLOCK_PERCENTAGES and LEARN_ISLAM_UNLOCK_PERCENTAGES arrays match. This will prevent the contract from being deployed with invalid configurations that could lead to locked tokens.

Status

Resolved

Medium Severity Issues

A.2 Unlock Intervals Must Be Greater Than the Current block.timestamp

Line	Function - Constructor
119	lockTokens(mdcTeamAndAdvisorsWallet, TEAM_AND_ADVISORS_UNLOCK_PERCENTAGES, _unlockFrequencyFoundersAndMdcTeam, 80000000000000000000000000);

Description

When setting up token locks in the lockTokens function, the provided unlock intervals must be checked to ensure that they are all greater than the current block.timestamp. Without this validation, it is possible to set intervals that are in the past, which would cause tokens to become immediately available or never unlocked as intended.

Remediation

Add a validation step to check that all provided unlock intervals are greater than the current block.timestamp before locking tokens. This ensures that the vesting schedule is set correctly and prevents immediate or never-to-be-unlocked tokens.

Status

Resolved

Low Severity Issues

A.3 Inefficient Token Handling by Transferring and Locking Tokens

Line	Function - Constructor
119	require(initialTransfers(supply), “Failed At MDC Tokens Transfer”); // @audit it’s not a good idea to transfer and then lock the tokens

```
lockTokens(
    mdcTeamAndAdvisorsWallet,
    TEAM_AND_ADVISORS_UNLOCK_PERCENTAGES,
    _unlockFrequencyFoundersAndMdcTeam,
    8000000000000000000000000
);
```

Description

In the current implementation, tokens are initially transferred and then locked, which can lead to inefficiencies and potential security risks. It’s better to lock the tokens directly without the need for an initial transfer and then transferring them back to the contract. This simplifies the process and reduces the risk of errors or losing tokens.

Remediation

Refactor the code to lock the tokens directly without an initial transfer. This ensures that the tokens are securely locked from the start and reduces the complexity of the token management process.

Status

Resolved

A.4 Missing Setter Function for Zakat Wallet

Line	Function - Constructor

119	constructor(
	address _zakatWallet){
	zakatWallet = _newZakatWallet;
	}

Description

The contract does not provide a setter function for the zakatWallet. This wallet is used for collecting fees, and without a way to update the address, it could cause issues if the initial address becomes compromised or needs to be changed for any reason.

Remediation

Add a setter function to allow updating the zakatWallet address. Ensure that only authorized personnel (e.g., the contract owner) can call this function to prevent unauthorized changes.

Status

Resolved

A.5 Missing Zero Address Checks

Line	Function - Constructor

```
119 constructor(
        address _exchangesAndDeflationaryIncentivesWallet,
        address _islamicFundWallet,
        address _mdcTeamAndAdvisorsWallet,
        address _foundersAndEarlyInvestorsWallet,
        address _marketingWallet,
        address _ambassadorsWallet,
        address _learnIslamWallet,
        address _zakatWallet,
        uint256[] memory _unlockFrequencyFoundersAndMdcTeam,
        uint256[] memory _unlockFrequencyForIslamFund
        ) ERC20(“Muslim Digital Coin”, “MDC”) {
        islamicFundWallet = _islamicFundWallet;
        mdcTeamAndAdvisorsWallet = _mdcTeamAndAdvisorsWallet;
        foundersAndEarlyInvestorsWallet = _foundersAndEarlyInvestorsWallet;
        marketingWallet = _marketingWallet;
        ambassadorsWallet = _ambassadorsWallet;
        learnIslamWallet = _learnIslamWallet;
        zakatWallet = _zakatWallet;
        exchangesAndDeflationaryIncentivesWallet =
_exchangesAndDeflationaryIncentivesWallet;}
```

Description

The constructor is missing checks to ensure that the provided wallet addresses are not zero addresses. This could lead to potential issues where the contract's critical functionality depends on these addresses being valid, non-zero addresses.

Remediation

To address this issue and enhance the security of the contract, you should implement zero-address validation checks in critical functions. Here's a recommended remediation approach:

1. Require Non-Zero Address: Add a validation check at the beginning of each critical function to ensure that the input addresses are non-zero. If an input address is zero, the function should revert with an error message.
2. Input Validation: Validate the parameters provided to the functions, especially those involving asset transfers or administrative actions, to prevent potential vulnerabilities or loss of assets.

Status

Resolved

Informational Issues

A.6: Consider using custom errors

Description

Custom errors reduce the contract size and can provide easier integration with a protocol. Consider using those instead of require statements with string error.

Status

Resolved

Functional Tests

Some of the tests performed are mentioned below:

- Should be able to lock tokens
- Should be able to Mint and Burn tokens (from the owner's account as well as from any other account)
- Should be able to transfer tokens
- Should be able to unlock tokens
- Should revert if transfer amount exceeds balance
- Should revert if tokens are not unlocked
- Should be able to deduct the fee from the unlock amount and transfer it to the fee recipient

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. 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 the MDC Token. We performed our audit according to the procedure described above.

Some issues of Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in MDC Token smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of MDC Token smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the MDC Token to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
June, 2024

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com