



Kishan Patel

Audit report of **QuestDM**



Prepared By: Kishan Patel

Prepared On: 03/01/2022

Prepared for: QuestDM

Table of contents

1. Disclaimer
2. Introduction
3. Project information
4. List of attacks checked
5. Severity Definitions
6. Good things in code
7. Critical vulnerabilities in code
8. Medium vulnerabilities in code
9. Low vulnerabilities in code
10. Summary



THIS AUDIT REPORT WILL CONTAIN CONFIDENTIAL INFORMATION ABOUT THE SMART CONTRACT AND INTELLECTUAL PROPERTY OF THE CUSTOMER AS WELL AS INFORMATION ABOUT POTENTIAL VULNERABILITIES OF THEIR EXPLOITATION.

THE INFORMATION FROM THIS AUDIT REPORT CAN BE USED INTERNALLY BY THE CUSTOMER OR IT CAN BE DISCLOSED PUBLICLY AFTER ALL VULNERABILITIES ARE FIXED - UPON THE DECISION OF THE CUSTOMER.



1. Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions). Because the total numbers of test cases are unlimited, the audit makes no statements or warranties on the security of the code.

It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.



2. Introduction

Kishan Patel (Consultant) was contacted by QuestDM (Customer) to conduct a Smart Contracts Code Review and Security Analysis. This report presents the findings of the security assessment of Customer`s smart contracts and its code review conducted between 28/02/2022 – 01/03/2022.

The project has 1 file. It contains approx 380 lines of Solidity code. All the functions and state variables are well commented on using the natspec documentation, but that does not create any vulnerability.

3. Project information

Token Name	QuestDM
Token Symbol	QSTD
Platform	Binance BSC
Order Started Date	28/02/2022
Order Completed Date	01/03/2022



4. List of attacks checked

□ Over and under flows	<u>TEST PASSED</u>
□ Short address attack	<u>TEST PASSED</u>
□ Visibility & Delegate call	<u>TEST PASSED</u>
□ Reentrancy / TheDAO hack	<u>TEST PASSED</u>
□ Forcing Ethereum to a contract	<u>TEST PASSED</u>
□ Timestamp Dependence	<u>TEST PASSED</u>
□ Gas Limit and Loops	<u>TEST PASSED</u>
□ DoS with (Unexpected) Throw	<u>TEST PASSED</u>
□ DoS with Block Gas Limit	<u>TEST PASSED</u>
□ Transaction-Ordering Dependence	<u>TEST PASSED</u>
□ Byte array vulnerabilities	<u>TEST PASSED</u>
□ Style guide violation	<u>TEST PASSED</u>
□ Transfer forwards all gas	<u>TEST PASSED</u>
□ ERC20 API violation	<u>TEST PASSED</u>
□ Malicious libraries	<u>TEST PASSED</u>
□ Compiler version not fixed	<u>TEST PASSED</u>
□ Unchecked external call - Unchecked math	<u>TEST PASSED</u>
□ Unsafe type inference	<u>TEST PASSED</u>



5. Severity Definitions

Risk	Level Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution



6. Good things in code

o Good required condition in functions:

- o Here you are checking that balance of contract should be bigger than or equal to amount, and transfer is successfully done or not.

```
34     function sendValue(address payable recipient, uint256 amount) i
35         require(address(this).balance >= amount, "Address: insuffic
36
37         // solhint-disable-next-line avoid-low-level-calls, avoid-c
38         (bool success, ) = recipient.call{ value: amount }("");
39         require(success, "Address: unable to send value, recipient
```

- o Here you are checking that balance of contract should be bigger than 0.

```
54     function functionCallWithValue(address target, bytes memory dat
55         require(address(this).balance >= value, "Address: insuffici
56         return _functionCallWithValue(target, data, value, errorMes
```

- o Here you are checking that target address is contract address or not.

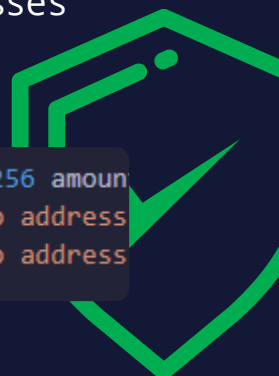
```
59     function _functionCallWithValue(address target, bytes memory da
60         require(isContract(target), "Address: call to non-contract"
61
```

- o Here you are checking that newOwner address is valid and proper.

```
105
106     function transferOwnership(address newOwner) public virtual onl
107         require(newOwner != address(0), "Ownable: new owner is the
108         emit OwnershipTransferred( owner, newOwner);
```

- o Here you are checking that owner and spender addresses are valid and proper.

```
198     function _approve(address owner, address spender, uint256 amoun
199         require(owner != address(0), "approve from the zero address
200         require(spender != address(0), "approve to the zero address
201
```



- o Here you are checking that msg.sender has sufficient allowance from sender address to transfer amount of token.

```
223     function transferFrom(address sender, address recipient, uint256 amount) public {
224         _transfer(sender, recipient, amount);
225         require(_allowances[sender][_msgSender()] - amount >= 0, "Insufficient allowance");
226         _approve(sender, _msgSender(), _allowances[sender][_msgSender()] - amount, recipient);
    }
```

- o Here you are checking that to and from addresses are valid and proper, amount value is bigger than 0 and balance of from address is bigger or equal to amount value.

```
231     function _transfer(address from, address to, uint256 amount) private {
232         require(from != address(0), "transfer from the zero address");
233         require(to != address(0), "transfer to the zero address");
234         require(amount > 0, "Transfer amount must be greater than 0");
235         require(balanceOf(from) >= amount, 'balance too low');
    }
```

- o Here you are checking that _taxFee and _burnFee should be bigger than or equal to 0 and smaller than or equal to 5.

```
332     function setTaxFeePercent(uint256 _taxFee) external onlyOwner() {
333         require(_taxFee >= 0, 'Fee Too Low');
334         require(_taxFee <= 5, 'Fee Too High');
335         taxFee = _taxFee;
    }
```

```
338     function setBurnFeePercent(uint256 _burnFee) external onlyOwner() {
339         require(_burnFee >= 0, 'Burn Too Low');
340         require(_burnFee <= 5, 'Burn Too High');
    }
```



ISSUE FIXED

- o Here you are checking that totalSupply - amount should be bigger or equal to minimumSupply. I can suggest first you can tokens from msg.sender balance and then from totalSupply.

```
357
358     function manualBurn(uint256 _amount) external onlyOwner() {
359
360         require(_totalSupply-_amount >= _minimumSupply, "Burn amount
361         _totalSupply -=_amount;
```

- o Here you are checking that totalSupply + amount should not exceed maxSupply.

```
365
366     function mint (uint256 _amount) external onlyOwner(){
367
368         require(_totalSupply+_amount <= 1000000000 * 10 ** 18, "Min
369         _totalSupply +=_amount;
```

7. Critical vulnerabilities in code

- No Critical vulnerabilities found

8. Medium vulnerabilities in code

- No Medium vulnerabilities found



9. Low vulnerabilities in code

9.1. Compiler version is not fixed:- ISSUE FIXED

=> In this file you have put "pragma solidity ^0.8.10;" which is not a good way to define compiler version.

=> Solidity source files indicate the versions of the compiler they can be compiled with. Pragma solidity ^0.8.10; // bad: compiles 0.8.10 and above
pragma solidity 0.8.10; //good: compiles 0.8.10 only

=> If you put(>=) symbol then you are able to get compiler version 0.8.10 and above. But if you don't use(^/>=) symbol then you are able to use only 0.8.10 version. And if there are some changes come in the compiler and you use the old version then some issues may come at deploy time.

9.2. Suggestions to add code validations:- ISSUE FIXED

=> You have implemented required validation in contract.

=> There are some place where you can improve validation and security of your code.

=> These are all just suggestion it is not bug.

Function: - approve

```
199     require(owner != address(0), "approve from the zero address");
200     require(spender != address(0), "approve to the zero address");
201
202     _allowances[owner][spender] = amount;
203     emit Approval(owner, spender, amount);
```

- Here you can check you have more allowance than balance.



Function: - excludeFromFee, includeInFee

```
310     function excludeFromFee(address account) public onlyOwner {  
311         _isExcludedFromFee[account] = true;  
312     }
```

```
313  
314     function includeInFee(address account) public onlyOwner {  
315         _isExcludedFromFee[account] = false;  
316     }
```

- o Here in excludeFromFee function you can check that account address is included for fee.
- o Here in includeInFee function you can check that account address is not already included for fee.



10. Summary

- Number of problems in the smart contract as per severity level

Critical	Medium	Low
0	0	4

4 ISSUES FIXED

According to the assessment, the smart contract code is well secured. The code is written with all validation and all security is implemented. Code is performing well and there is no way to steal funds from this contract.

- Good Point: Code performance and quality are good. Address validation and value validation is done properly
- Suggestions: Please use the static version of solidity, try to check return response of transfer method and try to add suggested code validation.

