Audit report of QuestDM

Prepared By:- Kishan Patel
Prepared On:- 01/03/2022

Prepared for: QuestDM

Table of contents

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	QuestDM
Platform	Ethereum
Order Started Date	28/02/2022
Order Completed Date	01/03/2022

4. List of attacks checked

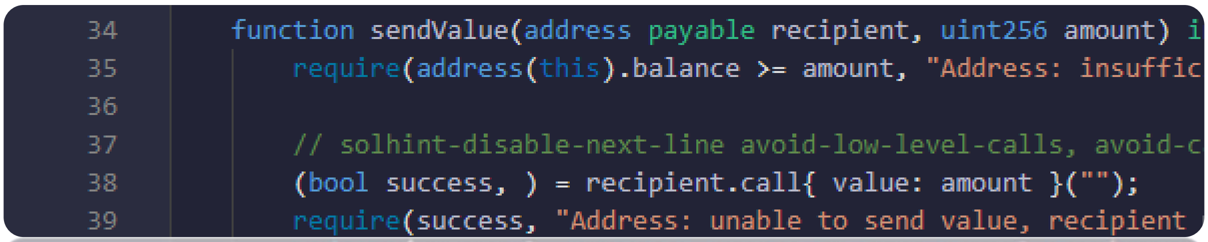
- Over and under flows
- Short address attack
- Visibility & Delegate call
- Reentrancy / TheDAO hack
- Forcing Ethereum to a contract
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
-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

5. Severity Definitions

Risk	Level Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens 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

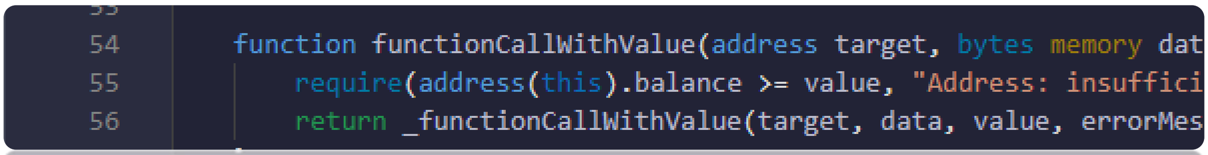
6. Good things in code

- **Good required condition in functions:-**
    - Here you are checking that balance of contract should be bigger than or equal to amount, and transfer is successfully done or not.



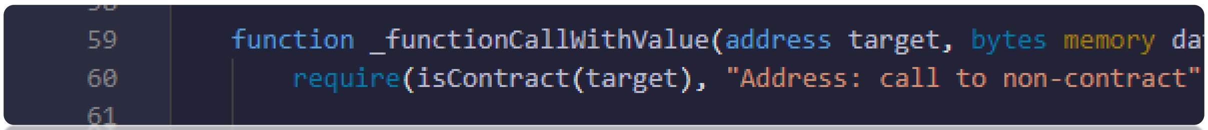
```
function sendValue(address payable recipient, uint256 amount) i
    require(address(this).balance >= amount, "Address: insuffic

    // solhint-disable-next-line avoid-low-level-calls, avoid-c
    (bool success, ) = recipient.call{ value: amount }("");
    require(success, "Address: unable to send value, recipient
```

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



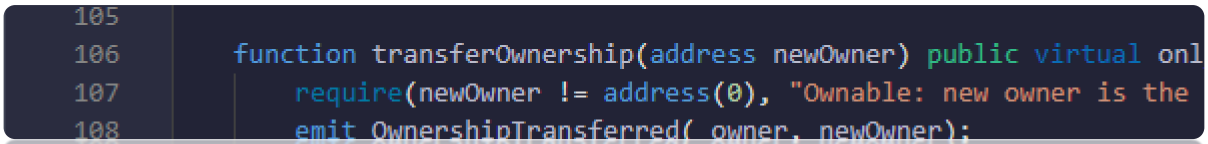
```
function functionCallWithValue(address target, bytes memory dat
    require(address(this).balance >= value, "Address: insuffici
    return _functionCallWithValue(target, data, value, errorMes
```

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



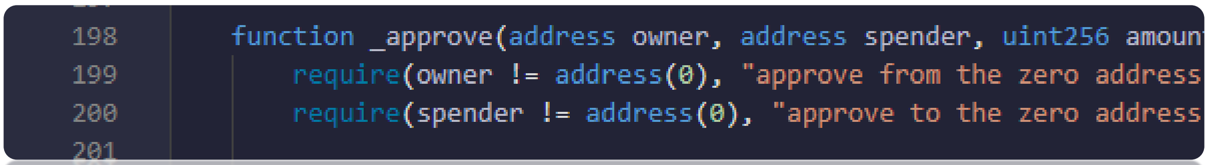
```
function _functionCallWithValue(address target, bytes memory da
    require(isContract(target), "Address: call to non-contract"
```

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



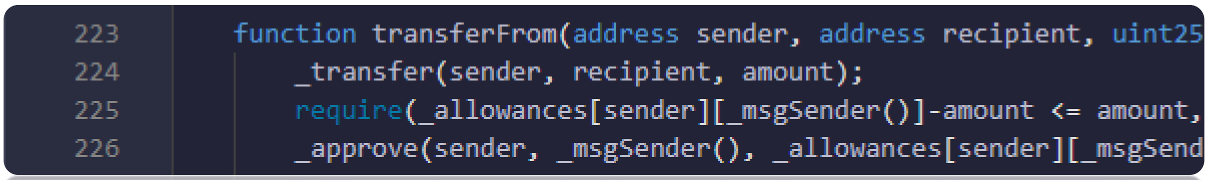
```
function transferOwnership(address newOwner) public virtual onl
    require(newOwner != address(0), "Ownable: new owner is the
    emit OwnershipTransferred(_owner, newOwner);
```

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



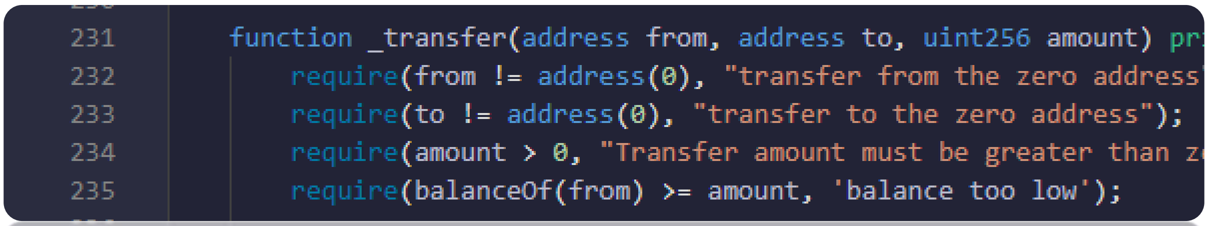
```
function _approve(address owner, address spender, uint256 amoun
    require(owner != address(0), "approve from the zero address
    require(spender != address(0), "approve to the zero address
```

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



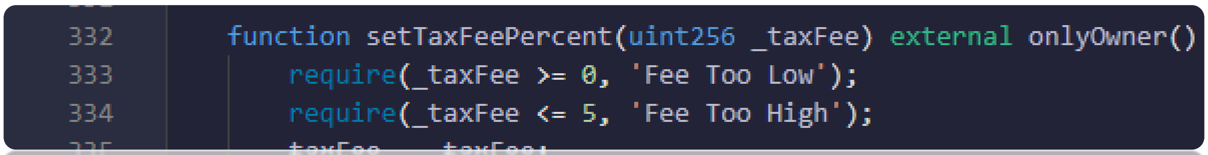
```
function transferFrom(address sender, address recipient, uint25
    _transfer(sender, recipient, amount);
    require(_allowances[sender][_msgSender()]-amount <= amount,
    _approve(sender, _msgSender(), _allowances[sender][_msgSend
```

- 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.

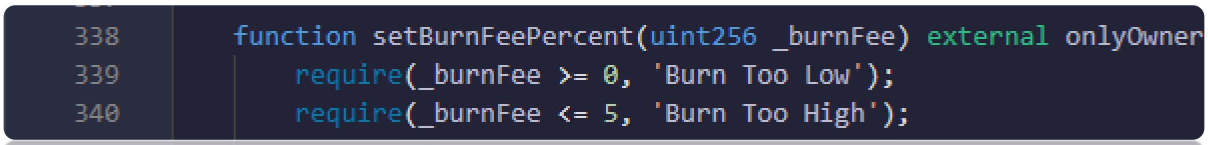


```
function _transfer(address from, address to, uint256 amount) pr
    require(from != address(0), "transfer from the zero address
    require(to != address(0), "transfer to the zero address");
    require(amount > 0, "Transfer amount must be greater than z
    require(balanceOf(from) >= amount, 'balance too low');
```

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

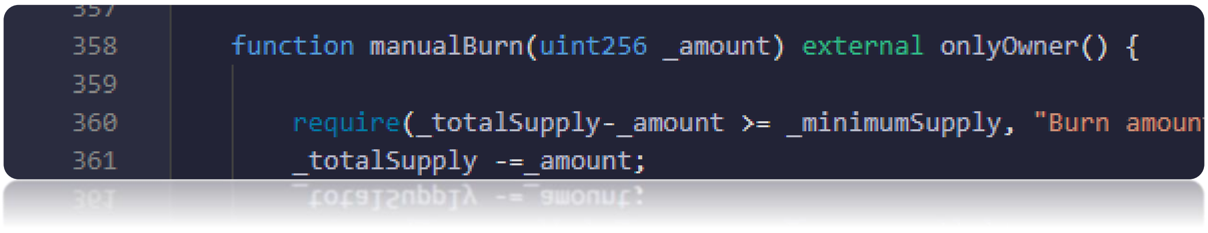


```
function setTaxFeePercent(uint256 _taxFee) external onlyOwner()
    require(_taxFee >= 0, 'Fee Too Low');
    require(_taxFee <= 5, 'Fee Too High');
```



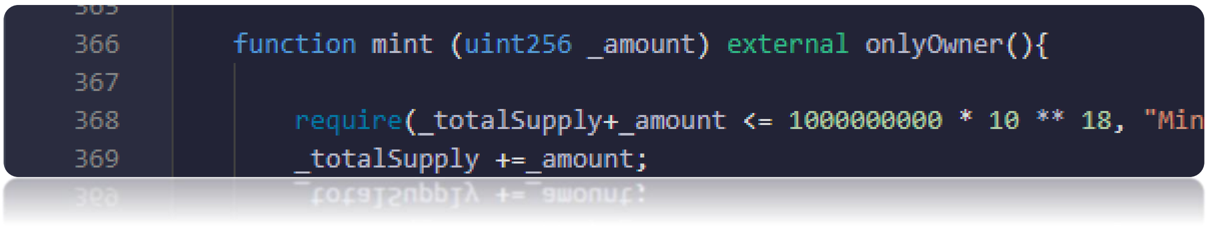
```
function setBurnFeePercent(uint256 _burnFee) external onlyOwner
    require(_burnFee >= 0, 'Burn Too Low');
    require(_burnFee <= 5, 'Burn Too High');
```

- 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.



```
    function manualBurn(uint256 _amount) external onlyOwner() {

        require(_totalSupply-_amount >= _minimumSupply, "Burn amoun
        _totalSupply -=_amount;
```

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



```
    function mint (uint256 _amount) external onlyOwner(){

        require(_totalSupply+_amount <= 1000000000 * 10 ** 18, "Min
        _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:-

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

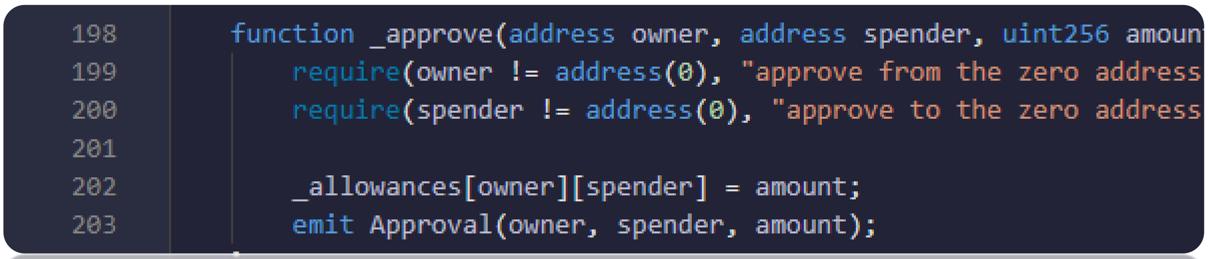
=> Solidity source files indicate the versions of the compiler they can be compiled with. Pragma solidity ^0.8.10; // bad: compiles 0.8.10 and above pragma solidity 0.8.10; //good: compiles 0.8.10 only

=> 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:-

=> 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.

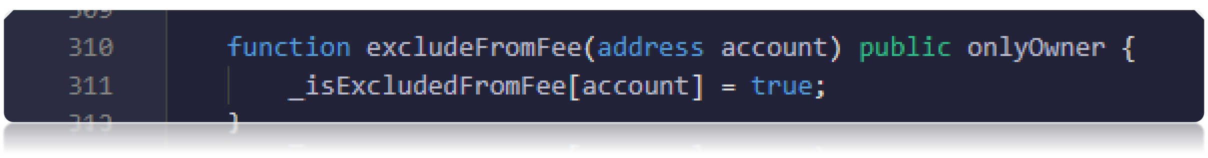
Function: - approve



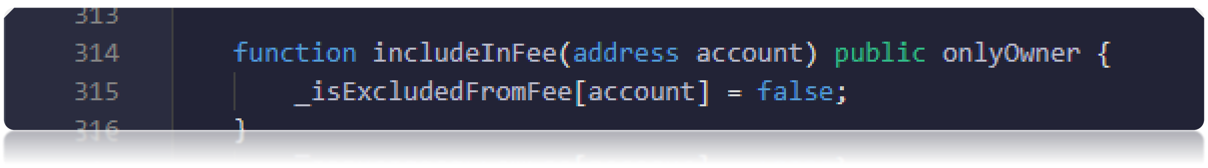
```
    function _approve(address owner, address spender, uint256 amoun
        require(owner != address(0), "approve from the zero address
        require(spender != address(0), "approve to the zero address

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
```

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

Function: - excludeFromFee, includeInFee



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



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

- Here in excludeFromFee function you can check that account address is included for fee.
- Here in excludeFromFee 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**

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.