Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Customer:	Quantum Moon
Website:	quantummoon.space
Platform:	Binance Smart Chain
Language:	Solidity
Date:	July 5th, 2021

Table of contents

Email: audit@EtherAuthority.io

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by the Quantum Moon team to perform the Security audit of the Quantum Moon token smart contract code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on July 5th, 2021.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

Quantum Moon is a project that aims to revolutionize the QUM market by combining all ERC20 and QUM properties on the BSC network.

Audit scope

Name	**Code Review and Security Analysis Report for Quantum Moon token Smart Contract**
Platform	**BSC / Solidity**
File	QuantumMoon.sol
Smart Contract Online Code	https://bscscan.com/address/0xef35fa21f6af3087d14c3bd637f0ee512909cb98#code
File MD5 Hash	A1FA5DD199A9973916BA33454BF09FB7
Audit Date	July 5th, 2021
Revised Contract	https://bscscan.com/address/0xa1710c2899d9e1bf21c6d040fb2b1370da1765fe#code
Revision Date	July 26, 2021

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Name: QuantumMoon	**YES, This is valid.**
Symbol: QUM	**YES, This is valid.**
Decimal: 9	**YES, This is valid.**
Total tax is 12 percent in the following breakdown. • Tax Fee: 2% • Team Fee: 2% • Marketing Tax Fee: 2% • Liquidity Tax Fee: 3% • Lottery Tax Fee: 3%	**YES, This is valid. Owner can change this fee.**
• Mint function disabled. • The lottery is only drawn with holders holding $500 worth of tokens every 250k in BUSD value. • Auto burn rate is 0.25 percent per month.	**YES, This is valid.The smart contract owner controls these functions, so the owner must handle the private key of the owner's wallet very securely. Because if the private key is compromised, then it will create problems.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Secured"**. These contracts also have owner functions (described in the centralization section below), which does not make everything 100% decentralized. Thus, the owner must execute those smart contract functions as per the business plan.

We used various tools like MythX, Slither and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 1 critical, 1 high, 0 medium and 5 low and some very low level issues.
These issues are fixed/acknowledged in the revised smart contract code.

Investors Advice: Technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Resolved
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	Passed
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Other code specification issues	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Resolved
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Code Quality

This audit scope has 1 smart contract. This smart contract also contains Libraries, Smart contracts inherits and Interfaces. These are compact and well written contracts.

The libraries in the Quantum Moon token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the Quantum Moon token.

The Quantum Moon team has **not** provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Some code parts are **not well** commented on smart contracts.

Documentation

We were given Quantum Moon token smart contracts code in the form of a BscScan web link. The hashes of that code are mentioned above in the table.

As mentioned above, some code parts are **not well** commented. So it is difficult to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Another source of information was its official website http://quantummoon.space/ which provided rich information about the project architecture and tokenomics.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure that are based on well known industry standard open source projects. And their core code blocks are written well.

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

Quantum Moon tokens are smart contracts, having functionality like burn,add liquidity, swap And Liquify etc.

QuantumMoon.sol

(1) Interface

(a) IBEP20

(b) IPancakeFactory

(c) IPancakePair

(2) Inherited contracts

(a) Context

(b) Ownable

(c) IBEP20

(d) swapBUSDcontract

(3) Struct

(a) t_Info

(b) val_Info

(4) Usages

(a) using SafeMath for uint256;

(5) Events

(a) event Burn(uint256 value);

(b) event LotteryPaid(address addrWin, uint256 amount, uint256 rnd, uint256 rndNum);

(c) event MinTokensBeforeSwapUpdated(uint256 minTokensBeforeSwap);

(d) event SwapAndLiquifyEnabledUpdated(bool enabled);

(e) event SwapAndLiquify(uint256 tokensSwapped,uint256 ethReceived, uint256 tokensIntoLiqudity);

(f) event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

(6) Functions

Sl.	Functions	Type	Observation	Conclusion
1	lockTheSwap	modifier	Passed	No Issue
2	SetSwapContractAddr	write	access only Owner	No Issue
3	name	read	Passed	No Issue
4	symbol	read	Passed	No Issue
5	decimals	read	Passed	No Issue
6	totalSupply	read	Passed	No Issue
7	balanceOf	write	Passed	No Issue
8	transfer	write	Passed	No Issue
9	allowance	read	Passed	No Issue
10	approve	write	Passed	No Issue
11	transferFrom	write	Passed	No Issue
12	increaseAllowance	write	Passed	No Issue
13	decreaseAllowance	write	Passed	No Issue
14	totalFees	read	Passed	No Issue
15	deliver	write	External instead of public	Refer Audit Findings
16	reflectionFromToken	read	Passed	No Issue
17	tokenFromReflection	read	Passed	No Issue
18	excludeFromFee	write	Missing Events	Refer Audit Findings
19	includeInFee	write	Missing Events	Refer Audit Findings
20	setSwapAndLiquifyEnabled	write	access only Owner	No Issue
21	_reflectFee	write	Passed	No Issue
22	_getValues	read	Passed	No Issue
23	_getTValues	read	Passed	No Issue
24	_getRValues	read	Passed	No Issue
25	_getRate	read	Passed	No Issue
26	removeAllFee`	write	Passed	No Issue
27	restoreAllFee	write	Passed	No Issue
28	isExcludedFromFee	read	Passed	No Issue
29	_approve	write	Passed	No Issue
30	_transfer	write	Passed	No Issue
31	swapAndLiquify	write	Passed	No Issue
32	swapTokensForBUSD	write	Passed	No Issue
33	addLiquidity	write	Centralized risk	Refer Audit Findings
34	_tokenTransfer	write	Passed	No Issue
35	_transferStandard	write	Passed	No Issue
36	_takeLiquidity	write	Passed	No Issue
37	_takeMarketing	write	Passed	No Issue
38	_takeTeam	write	Passed	No Issue
39	_takeLottery	write	Passed	No Issue

40	random	read	Passed	No Issue
41	_awardRandomEligibleHolder	write	Infinite loop possibility	Refer Audit Findings
42	_getEligibleBalanceInToken	read	Passed	No Issue
43	owner	read	Passed	No Issue
44	onlyOwner	modifier	Passed	No Issue
45	renounceOwnership	write	Possible to gain ownership	Refer Audit Findings
46	transferOwnership	write	access only Owner	No Issue
47	geUnlockTime	read	Passed	No Issue
48	lock	write	Possible to gain ownership	Refer Audit Findings
49	unlock	write	Possible to gain ownership	Refer Audit Findings
50	_msgSender	internal	Passed	No Issue
51	_msgData	internal	Passed	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial functions
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
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Audit Findings

Critical

(1) Missing ownership control

```
function SetMainContractAddr(address addr) public {
    mainContractAddr = addr;
}
```

The function at line number #1094 does not have any ownership control. So, anyone can set the new contract address which can be used to withdraw all the BUSD out of the contract.

Resolution: Add ownerOnly modifier in that function.

Status: This issue is fixed by Quantum Moon team

High

(1) Super intensive loop

```
bool bAddHoler = true;
for (uint j = 0; j < _listOfHolders.length; j ++){
    if (_listOfHolders[j] == to){
        bAddHoler = false;
    }
}
```

Loop at line #892 is super intensive. Array length will increase as token holders increase. And it will keep increasing the gas cost to transfer tokens as well as it will cause token transfer completely stopped after so many token holders.

Resolution: Adjust the logic to track the token holders, such as using mapping.

Status: This issue is fixed by Quantum Moon team

Medium

No Medium severity vulnerabilities were found.

Low

(1) Make variables constant:

```
string private _name = "QuantumMoon";
string private _symbol = "QUM";
uint8 private _decimals = 9;
```

name, symbol, decimals. These variables will be unchanged. So, please make it constant. It will save some gas.

Resolution: Declare those variables as constant. Just put a constant keyword.

Status: This issue is fixed by Quantum Moon team

(2) Infinite loop possibility:

```
function _awardRandomEligibleHolder() private {
    uint256 nCntEligible = 0;
    uint256 nEligibleAmount = _getEligibleBalanceInToken(500 * 10 ** 18);
    // uint256 nEligibleAmount = 50000000000;
    for (uint i = 0; i < _listOfHolders.length; i ++){
        if (_listOfHolders[i] == address(this) || _listOfHolders[i] == pcsV2Pair || _listOfHolders[i] == owner() || _listOfHo
        {
            continue;
        }
        if (balanceOf(_listOfHolders[i]) >= nEligibleAmount){
            nCntEligible ++;
        }
    }
    if (nCntEligible > 0){
        address [] memory listOfEligible = new address[](nCntEligible);
        nCntEligible = 0;
        for (uint i = 0; i < _listOfHolders.length; i ++){
            if (_listOfHolders[i] == address(this) || _listOfHolders[i] == pcsV2Pair ||_listOfHolders[i] == owner() || _listO
            {
                continue;
            }
            if (balanceOf(_listOfHolders[i]) >= nEligibleAmount){
                listOfEligible[nCntEligible] = _listOfHolders[i];
                nCntEligible ++;
            }
```

If there are so many _listOfHolders, then this logic will fail, as it might hit the block's gas limit. If there are very limited exceptions, then this will work, but will cost more gas.

Resolution: We suggest to _listOfHolders limited only.

Status: This issue is fixed by Quantum Moon team

(3) Possible to regain the ownership:

Owner can renounce ownership and make contract without owner but he can regain ownership by following the steps below:

1. Owner calls the lock function in contract to set the current owner as _previousOwner.
2. Owner calls unlock to unlock the contract and set _owner = _previousOwner.
3. Owner called renounceOwnership to leave the contract without the owner.
4. Owner calls unlock to regain ownership.

Resolution: We suggest removing these lock/unlock functions as this seems not serving a great purpose. Otherwise, always renounce ownership first before calling the lock function.

Status: This issue is fixed by Quantum Moon team

(4) Centralized risk in addLiquidity:

```
function addLiquidity(uint256 tokenAmount, uint256 busdAmount) private {
    // approve token transfer to cover all possible scenarios
    _approve(address(this), address(pcsV2Router), tokenAmount);
    _BUSDcontract.approve(address(pcsV2Router), busdAmount);
    // add the liquidity
    pcsV2Router.addLiquidity(
        address(this),
        address(_BUSDcontract),
        tokenAmount,
        busdAmount,
        0, // slippage is unavoidable
        0, // slippage is unavoidable
        owner(),
        block.timestamp
    );
}
```

In addLiquidity function, the owner gets funds from the Pool. If the private key of the owner's wallet is compromised, then it will create a problem.

Resolution: Ideally this can be a governance smart contract. On another hand, the owner can accept this risk and handle the private key very securely.

Status: This issue is acknowledged by Quantum Moon team

(5) Missing Events:

Missing Events log for some functions like:

- excludeFromFee
- includeInFee
- deliver.

Status: This issue is fixed by Quantum Moon team

Very Low / Discussion / Best practices:

(1) Use latest solidity version:

```
pragma solidity ^0.6.12;
```

Use the latest solidity version while contract deployment to prevent any compiler version level bugs.

Resolution: Please use 0.8.6 which is the latest version.

Status: This issue is acknowledged by Quantum Moon team

(2) External instead of public:

If any function is not called from inside the smart contract, then it is better to declare it as external instead of public. As it saves some gas as well.

https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices

Status: This issue is fixed by Quantum Moon team

(3) Hardcoded addresses:

```
address public marketingWallet = 0x9f1c7B388395c5d97b6624F5e48Cf3D2e6572865;
address public teamWallet = 0xfdDE3110e63f629225b553348a21bF184F181886;
address public lotteryWallet = 0xA464EFa1d25239f26b52d62c200d4e736A93204f;
address public busdAddr = 0xe9e7CEA3DedcA5984780Bafc599bD69ADd087D56;
IBEP20 private _BUSDcontract = IBEP20(busdAddr);
```

There are some hard coded addresses. And these variables will be unchanged. So, please make it constant. It will save some gas.

Resolution: Owner has to double confirm before deploying.

Status: This issue is acknowledged by Quantum Moon team

Centralization

These smart contracts have some functions which can be executed by Admin (Owner) only. If the admin wallet private key would be compromised, then it would create trouble. Following are Admin functions:

- excludeFromFee: The owner can check excluded Fee.
- includeInFee: The owner can check the included fee.
- SetSwapContractAddr: The owner can set Swap contract Address.
- setSwapAndLiquifyEnabled: The owner can set swap and Liquify enable status.

Conclusion

We were given a contract code. And we have used all possible tests based on given objects as files. We observed some issues in the smart contracts and those issues are fixed in revised code. So, **it's good to go to production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high level description of functionality was presented in As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security state of the reviewed contract, based on standard audit procedure scope, is **"Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract 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).

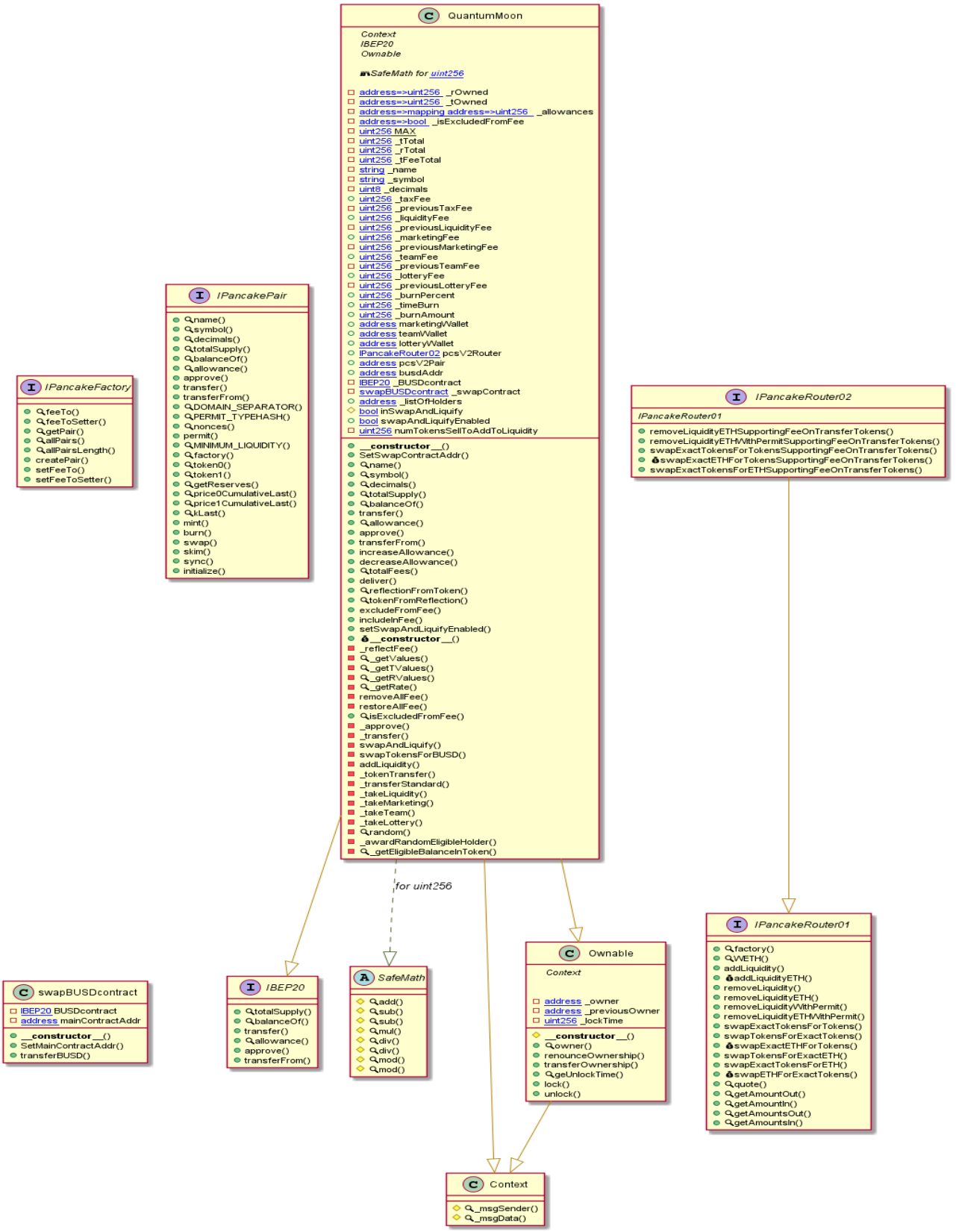
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we 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 also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the 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.

Appendix

Code Flow Diagram - QuantumMoon Token



Slither Results Log

Slither log >> QuantumMoon.sol

INFO:Detectors:
QuantumMoon._awardRandomEligibleHolder() (QuantumMoon.sol#1023-1055) uses a weak PRNG: "rndVal = random(100,1000000,balanceOf(lotteryWallet)) % nCntEligible (QuantumMoon.sol#1049)"
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#weak-PRNG
INFO:Detectors:
swapBUSDcontract.transferBUSD() (QuantumMoon.sol#1074-1078) ignores return value by BUSDcontract.transfer(mainContractAddr,BUSDcontract.balanceOf(address(this))) (QuantumMoon.sol#1077)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
QuantumMoon._getTValues(uint256) (QuantumMoon.sol#769-779) performs a multiplication on the result of a division:
	-tFee = _tAmount.div(10 ** 2).mul(_taxFee) (QuantumMoon.sol#771)
QuantumMoon._getTValues(uint256) (QuantumMoon.sol#769-779) performs a multiplication on the result of a division:
	-tLiquidity = _tAmount.div(10 ** 2).mul(_liquidityFee) (QuantumMoon.sol#772)
QuantumMoon._getTValues(uint256) (QuantumMoon.sol#769-779) performs a multiplication on the result of a division:
	-tMarketing = _tAmount.div(10 ** 2).mul(_marketingFee) (QuantumMoon.sol#773)
QuantumMoon._getTValues(uint256) (QuantumMoon.sol#769-779) performs a multiplication on the result of a division:
	-tTeam = _tAmount.div(10 ** 2).mul(_teamFee) (QuantumMoon.sol#774)
QuantumMoon._getTValues(uint256) (QuantumMoon.sol#769-779) performs a multiplication on the result of a division:
	-tLottery = _tAmount.div(10 ** 2).mul(_lotteryFee) (QuantumMoon.sol#775)
QuantumMoon._transfer(address,address,uint256) (QuantumMoon.sol#831-888) performs a multiplication on the result of a division:
	-_burnAmount = _burnAmount.add(_tTotal.div(10000).mul(_burnPercent)) (QuantumMoon.sol#841)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
Contract locking ether found:
	Contract QuantumMoon (QuantumMoon.sol#543-1065) has payable functions:
	 - QuantumMoon.receive() (QuantumMoon.sol#724)
	But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether
INFO:Detectors:
Reentrancy in QuantumMoon._transfer(address,address,uint256) (QuantumMoon.sol#831-888):
	External calls:
	- swapAndLiquify(contractTokenBalance) (QuantumMoon.sol#865)
		- _BUSDcontract.approve(address(pcsV2Router),busdAmount) (QuantumMoon.sol#935)
		- pcsV2Router.addLiquidity(address(this),address(_BUSDcontract),tokenAmount,busdAmount,0,0,owner(),block.timestamp) (QuantumMoon.sol#937-946)
		- _swapContract.transferBUSD() (QuantumMoon.sol#903)
		- pcsV2Router.swapExactTokensForTokensSupportingFeeOnTransferTokens(tokenAmount,0,path,address(_swapContract),block.timestamp) (QuantumMoon.sol#923-929)
	State variables written after the call(s):
	- _tokenTransfer(from,to,amount,takeFee) (QuantumMoon.sol#887)
		- _liquidityFee = _previousLiquidityFee (QuantumMoon.sol#813)
		- _liquidityFee = 0 (QuantumMoon.sol#805)
	- _tokenTransfer(from,to,amount,takeFee) (QuantumMoon.sol#887)
		- _lotteryFee = _previousLotteryFee (QuantumMoon.sol#816)
		- _lotteryFee = 0 (QuantumMoon.sol#808)
	- _tokenTransfer(from,to,amount,takeFee) (QuantumMoon.sol#887)
		- _marketingFee = _previousMarketingFee (QuantumMoon.sol#814)
		- _marketingFee = 0 (QuantumMoon.sol#806)

- _tokenTransfer(from,to,amount,takeFee) (QuantumMoon.sol#887)
    - _rOwned[lotteryWallet] = _rOwned[lotteryWallet].add(rLottery) (QuantumMoon.sol#1001)
    - _rOwned[marketingWallet] = _rOwned[marketingWallet].add(rMarketing) (QuantumMoon.sol#989)
    - _rOwned[address(this)] = _rOwned[address(this)].add(rLiquidity) (QuantumMoon.sol#983)
    - _rOwned[teamWallet] = _rOwned[teamWallet].add(rTeam) (QuantumMoon.sol#995)
    - _rOwned[sender] = _rOwned[sender].sub(rAmount) (QuantumMoon.sol#970)
    - _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount) (QuantumMoon.sol#971)
    - _rOwned[listOfEligible[rndVal]] = _rOwned[listOfEligible[rndVal]].add(_rOwned[lotteryWallet]) (QuantumMoon.sol#1052)
    - _rOwned[lotteryWallet] = 0 (QuantumMoon.sol#1053)
- _tokenTransfer(from,to,amount,takeFee) (QuantumMoon.sol#887)
    - _rTotal = _rTotal.sub(rFee) (QuantumMoon.sol#727)
- _tokenTransfer(from,to,amount,takeFee) (QuantumMoon.sol#887)
    - _taxFee = _previousTaxFee (QuantumMoon.sol#812)
    - _taxFee = 0 (QuantumMoon.sol#804)
- _tokenTransfer(from,to,amount,takeFee) (QuantumMoon.sol#887)
    - _teamFee = _previousTeamFee (QuantumMoon.sol#815)
    - _teamFee = 0 (QuantumMoon.sol#807)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
QuantumMoon._getValues(uint256).vInfo (QuantumMoon.sol#756) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
QuantumMoon.addLiquidity(uint256,uint256) (QuantumMoon.sol#932-947) ignores return value by _BUSDcontract.approve(address(pcsV2Router),busdAmount) (QuantumMoon.sol#935)
QuantumMoon.addLiquidity(uint256,uint256) (QuantumMoon.sol#932-947) ignores return value by pcsV2Router.addLiquidity(address(this),address(_BUSDcontract),tokenAmount,busdAmount,0,0,owner(),block.timestamp) (QuantumMoon.sol#937-946)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
QuantumMoon.allowance(address,address).owner (QuantumMoon.sol#656) shadows:
- Ownable.owner() (QuantumMoon.sol#274-276) (function)
QuantumMoon._approve(address,address,uint256).owner (QuantumMoon.sol#823) shadows:
- Ownable.owner() (QuantumMoon.sol#274-276) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
swapBUSDcontract.SetMainContractAddr(address).addr (QuantumMoon.sol#1071) lacks a zero-check on :
    - mainContractAddr = addr (QuantumMoon.sol#1072)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in QuantumMoon._transfer(address,address,uint256) (QuantumMoon.sol#831-888):
External calls:
- swapAndLiquify(contractTokenBalance) (QuantumMoon.sol#865)
    - _BUSDcontract.approve(address(pcsV2Router),busdAmount) (QuantumMoon.sol#935)
    - pcsV2Router.addLiquidity(address(this),address(_BUSDcontract),tokenAmount,busdAmount,0,0,owner(),block.timestamp) (QuantumMoon.sol#937-946)
    - _swapContract.transferBUSD() (QuantumMoon.sol#903)
    - pcsV2Router.swapExactTokensForTokensSupportingFeeOnTransferTokens(tokenAmount,0,path,address(_swapContract),block.timestamp) (QuantumMoon.sol#923-929)
State variables written after the call(s):
- _listOfHolders.push(to) (QuantumMoon.sol#875)
- _tokenTransfer(from,to,amount,takeFee) (QuantumMoon.sol#887)
    - _previousLiquidityFee = _liquidityFee (QuantumMoon.sol#799)
- _tokenTransfer(from,to,amount,takeFee) (QuantumMoon.sol#887)
    - _previousLotteryFee = _lotteryFee (QuantumMoon.sol#802)
- _tokenTransfer(from,to,amount,takeFee) (QuantumMoon.sol#887)
    - _previousMarketingFee = _marketingFee (QuantumMoon.sol#800)
- _tokenTransfer(from,to,amount,takeFee) (QuantumMoon.sol#887)
    - _previousTaxFee = _taxFee (QuantumMoon.sol#798)
- _tokenTransfer(from,to,amount,takeFee) (QuantumMoon.sol#887)
    - _previousTeamFee = _teamFee (QuantumMoon.sol#801)
- _tokenTransfer(from,to,amount,takeFee) (QuantumMoon.sol#887)

- _tFeeTotal = _tFeeTotal.add(tFee) (QuantumMoon.sol#728)
Reentrancy in QuantumMoon.constructor() (QuantumMoon.sol#608-624):
External calls:
- pcsV2Pair = IPancakeFactory(_pcsV2Router.factory()).createPair(address(this),address(_BUSDcontract)) (QuantumMoon.sol#613-614)
State variables written after the call(s):
- _isExcludedFromFee[owner()] = true (QuantumMoon.sol#620)
- _isExcludedFromFee[address(this)] = true (QuantumMoon.sol#621)
- pcsV2Router = _pcsV2Router (QuantumMoon.sol#617)
Reentrancy in QuantumMoon.swapAndLiquify(uint256) (QuantumMoon.sol#890-912):
External calls:
- swapTokensForBUSD(half) (QuantumMoon.sol#902)
- pcsV2Router.swapExactTokensForTokensSupportingFeeOnTransferTokens(tokenAmount,0,path,address(_swapContract),block.timestamp) (QuantumMoon.sol#923-929)
- _swapContract.transferBUSD() (QuantumMoon.sol#903)
- addLiquidity(otherHalf,newBalance) (QuantumMoon.sol#909)
- _BUSDcontract.approve(address(pcsV2Router),busdAmount) (QuantumMoon.sol#935)
- pcsV2Router.addLiquidity(address(this),address(_BUSDcontract),tokenAmount,busdAmount,0,0,owner(),block.timestamp) (QuantumMoon.sol#937-946)
State variables written after the call(s):
- addLiquidity(otherHalf,newBalance) (QuantumMoon.sol#909)
- _allowances[owner][spender] = amount (QuantumMoon.sol#827)
Reentrancy in QuantumMoon.transferFrom(address,address,uint256) (QuantumMoon.sol#665-669):
External calls:
- _transfer(sender,recipient,amount) (QuantumMoon.sol#666)
- _BUSDcontract.approve(address(pcsV2Router),busdAmount) (QuantumMoon.sol#935)
- pcsV2Router.addLiquidity(address(this),address(_BUSDcontract),tokenAmount,busdAmount,0,0,owner(),block.timestamp) (QuantumMoon.sol#937-946)
- _swapContract.transferBUSD() (QuantumMoon.sol#903)
- pcsV2Router.swapExactTokensForTokensSupportingFeeOnTransferTokens(tokenAmount,0,path,address(_swapContract),block.timestamp) (QuantumMoon.sol#923-929)
State variables written after the call(s):
- _approve(sender,_msgSender(),_allowances[sender][_msgSender()].sub(amount,ERC20: transfer amount exceeds allowance)) (QuantumMoon.sol#667)
- _allowances[owner][spender] = amount (QuantumMoon.sol#827)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in QuantumMoon._transfer(address,address,uint256) (QuantumMoon.sol#831-888):
External calls:
- swapAndLiquify(contractTokenBalance) (QuantumMoon.sol#865)
- _BUSDcontract.approve(address(pcsV2Router),busdAmount) (QuantumMoon.sol#935)
- pcsV2Router.addLiquidity(address(this),address(_BUSDcontract),tokenAmount,busdAmount,0,0,owner(),block.timestamp) (QuantumMoon.sol#937-946)
- _swapContract.transferBUSD() (QuantumMoon.sol#903)
- pcsV2Router.swapExactTokensForTokensSupportingFeeOnTransferTokens(tokenAmount,0,path,address(_swapContract),block.timestamp) (QuantumMoon.sol#923-929)
Event emitted after the call(s):
- LotteryPaid(listOfEligible[rndVal],balanceOf(lotteryWallet),rndVal,nCntEligible) (QuantumMoon.sol#1050)
- _tokenTransfer(from,to,amount,takeFee) (QuantumMoon.sol#887)
- Transfer(sender,recipient,tTransferAmount) (QuantumMoon.sol#977)
- _tokenTransfer(from,to,amount,takeFee) (QuantumMoon.sol#887)
- Transfer(lotteryWallet,listOfEligible[rndVal],balanceOf(lotteryWallet)) (QuantumMoon.sol#1051)
- _tokenTransfer(from,to,amount,takeFee) (QuantumMoon.sol#887)
Reentrancy in QuantumMoon.constructor() (QuantumMoon.sol#608-624):
External calls:

- pcsV2Pair = IPancakeFactory(_pcsV2Router.factory()).createPair(address(this),address(_BUSDcontract)) (QuantumMoon.sol#613-614)
Event emitted after the call(s):
- Transfer(address(0),_msgSender(),_tTotal) (QuantumMoon.sol#623)
Reentrancy in QuantumMoon.swapAndLiquify(uint256) (QuantumMoon.sol#890-912):
External calls:
- swapTokensForBUSD(half) (QuantumMoon.sol#902)
- pcsV2Router.swapExactTokensForTokensSupportingFeeOnTransferTokens(tokenAmount,0,path,address(_swapContract),block.timestamp) (QuantumMoon.sol#923-929)
- _swapContract.transferBUSD() (QuantumMoon.sol#903)
- addLiquidity(otherHalf,newBalance) (QuantumMoon.sol#909)
- _BUSDcontract.approve(address(pcsV2Router),busdAmount) (QuantumMoon.sol#935)
- pcsV2Router.addLiquidity(address(this),address(_BUSDcontract),tokenAmount,busdAmount,0,0,owner(),block.timestamp) (QuantumMoon.sol#937-946)
Event emitted after the call(s):
- Approval(owner,spender,amount) (QuantumMoon.sol#828)
- addLiquidity(otherHalf,newBalance) (QuantumMoon.sol#909)
- SwapAndLiquify(half,newBalance,otherHalf) (QuantumMoon.sol#911)
Reentrancy in QuantumMoon.transferFrom(address,address,uint256) (QuantumMoon.sol#665-669):
External calls:
- _transfer(sender,recipient,amount) (QuantumMoon.sol#666)
- _BUSDcontract.approve(address(pcsV2Router),busdAmount) (QuantumMoon.sol#935)
- pcsV2Router.addLiquidity(address(this),address(_BUSDcontract),tokenAmount,busdAmount,0,0,owner(),block.timestamp) (QuantumMoon.sol#937-946)
- _swapContract.transferBUSD() (QuantumMoon.sol#903)
- pcsV2Router.swapExactTokensForTokensSupportingFeeOnTransferTokens(tokenAmount,0,path,address(_swapContract),block.timestamp) (QuantumMoon.sol#923-929)
Event emitted after the call(s):
- Approval(owner,spender,amount) (QuantumMoon.sol#828)
- _approve(sender,_msgSender(),_allowances[sender][_msgSender()].sub(amount,ERC20: transfer amount exceeds allowance)) (QuantumMoon.sol#667)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Ownable.unlock() (QuantumMoon.sol#321-326) uses timestamp for comparisons
Dangerous comparisons:
- require(bool,string)(now > _lockTime,Contract is locked until 7 days) (QuantumMoon.sol#323)
QuantumMoon._transfer(address,address,uint256) (QuantumMoon.sol#831-888) uses timestamp for comparisons
Dangerous comparisons:
- _timeBurn < block.timestamp (QuantumMoon.sol#840)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Context._msgData() (QuantumMoon.sol#237-240) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
QuantumMoon._rTotal (QuantumMoon.sol#553) is set pre-construction with a non-constant function or state variable:
- (MAX - (MAX % _tTotal))
QuantumMoon._previousTaxFee (QuantumMoon.sol#560) is set pre-construction with a non-constant function or state variable:
- _taxFee
QuantumMoon._previousLiquidityFee (QuantumMoon.sol#562) is set pre-construction with a non-constant function or state variable:
- _liquidityFee
QuantumMoon._previousMarketingFee (QuantumMoon.sol#564) is set pre-construction with a non-constant function or state variable:
- _marketingFee
QuantumMoon._previousTeamFee (QuantumMoon.sol#566) is set pre-construction with a non-constant function or state variable:
- _teamFee

QuantumMoon._previousLotteryFee (QuantumMoon.sol#568) is set pre-construction with a non-constant function or state variable:
- _lotteryFee
QuantumMoon._BUSDcontract (QuantumMoon.sol#582) is set pre-construction with a non-constant function or state variable:
- IBEP20(busdAddr)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#function-initializing-state-variables
INFO:Detectors:
Function IPancakePair.DOMAIN_SEPARATOR() (QuantumMoon.sol#365) is not in mixedCase
Function IPancakePair.PERMIT_TYPEHASH() (QuantumMoon.sol#366) is not in mixedCase
Function IPancakePair.MINIMUM_LIQUIDITY() (QuantumMoon.sol#383) is not in mixedCase
Function IPancakeRouter01.WETH() (QuantumMoon.sol#405) is not in mixedCase
Struct QuantumMoon.t_Info (QuantumMoon.sol#731-738) is not in CapWords
Struct QuantumMoon.val_Info (QuantumMoon.sol#740-750) is not in CapWords
Function QuantumMoon.SetSwapContractAddr(address) (QuantumMoon.sol#626-629) is not in mixedCase
Parameter QuantumMoon.setSwapAndLiquifyEnabled(bool)._enabled (QuantumMoon.sol#718) is not in mixedCase
Variable QuantumMoon._taxFee (QuantumMoon.sol#559) is not in mixedCase
Variable QuantumMoon._liquidityFee (QuantumMoon.sol#561) is not in mixedCase
Variable QuantumMoon._marketingFee (QuantumMoon.sol#563) is not in mixedCase
Variable QuantumMoon._teamFee (QuantumMoon.sol#565) is not in mixedCase
Variable QuantumMoon._lotteryFee (QuantumMoon.sol#567) is not in mixedCase
Variable QuantumMoon._burnPercent (QuantumMoon.sol#570) is not in mixedCase
Variable QuantumMoon._timeBurn (QuantumMoon.sol#571) is not in mixedCase
Variable QuantumMoon._burnAmount (QuantumMoon.sol#572) is not in mixedCase
Variable QuantumMoon._BUSDcontract (QuantumMoon.sol#582) is not in mixedCase
Variable QuantumMoon._listOfHolders (QuantumMoon.sol#586) is not in mixedCase
Contract swapBUSDcontract (QuantumMoon.sol#1067-1080) is not in CapWords
Function swapBUSDcontract.SetMainContractAddr(address) (QuantumMoon.sol#1071-1073) is not in mixedCase
Variable swapBUSDcontract.BUSDcontract (QuantumMoon.sol#1068) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (QuantumMoon.sol#238)" inContext (QuantumMoon.sol#232-241)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
Variable
IPancakeRouter01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountA
Desired (QuantumMoon.sol#410) is too similar to
IPancakeRouter01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountB
Desired (QuantumMoon.sol#411)
Variable QuantumMoon._getValues(uint256).rTransferAmount (QuantumMoon.sol#755) is too similar to QuantumMoon._transferStandard(address,address,uint256).tTransferAmount (QuantumMoon.sol#964)
Variable QuantumMoon._getValues(uint256).rTransferAmount (QuantumMoon.sol#755) is too similar to QuantumMoon._getValues(uint256).tTransferAmount (QuantumMoon.sol#754)
Variable QuantumMoon.reflectionFromToken(uint256,bool).rTransferAmount (QuantumMoon.sol#699) is too similar to QuantumMoon._transferStandard(address,address,uint256).tTransferAmount (QuantumMoon.sol#964)
Variable QuantumMoon._transferStandard(address,address,uint256).rTransferAmount (QuantumMoon.sol#962) is too similar to QuantumMoon._transferStandard(address,address,uint256).tTransferAmount (QuantumMoon.sol#964)
Variable QuantumMoon._getValues(uint256).rTransferAmount (QuantumMoon.sol#755) is too similar to QuantumMoon._getTValues(uint256).tTransferAmount (QuantumMoon.sol#776)
Variable
QuantumMoon._getRValues(uint256,uint256,uint256,uint256,uint256,uint256,uint256).rTransferAmount (QuantumMoon.sol#788) is too similar to QuantumMoon._getTValues(uint256).tTransferAmount (QuantumMoon.sol#776)
Variable QuantumMoon.reflectionFromToken(uint256,bool).rTransferAmount (QuantumMoon.sol#699) is too similar to QuantumMoon._getValues(uint256).tTransferAmount (QuantumMoon.sol#754)
Variable QuantumMoon._transferStandard(address,address,uint256).rTransferAmount (QuantumMoon.sol#962) is too similar to QuantumMoon._getValues(uint256).tTransferAmount (QuantumMoon.sol#754)
Variable
QuantumMoon._getRValues(uint256,uint256,uint256,uint256,uint256,uint256,uint256).rTransferAmount

(QuantumMoon.sol#788) is too similar to QuantumMoon._getValues(uint256).tTransferAmount (QuantumMoon.sol#754)
Variable QuantumMoon._getRValues(uint256,uint256,uint256,uint256,uint256,uint256,uint256).rTransferAmount (QuantumMoon.sol#788) is too similar to QuantumMoon._transferStandard(address,address,uint256).tTransferAmount (QuantumMoon.sol#964)
Variable QuantumMoon.reflectionFromToken(uint256,bool).rTransferAmount (QuantumMoon.sol#699) is too similar to QuantumMoon._getTValues(uint256).tTransferAmount (QuantumMoon.sol#776)
Variable QuantumMoon._transferStandard(address,address,uint256).rTransferAmount (QuantumMoon.sol#962) is too similar to QuantumMoon._getTValues(uint256).tTransferAmount (QuantumMoon.sol#776)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
QuantumMoon._awardRandomEligibleHolder() (QuantumMoon.sol#1023-1055) uses literals with too many digits:
- rndVal = random(100,1000000,balanceOf(lotteryWallet)) % nCntEligible (QuantumMoon.sol#1049)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
QuantumMoon._tOwned (QuantumMoon.sol#547) is never used in QuantumMoon (QuantumMoon.sol#543-1065)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variables
INFO:Detectors:
QuantumMoon._burnPercent (QuantumMoon.sol#570) should be constant
QuantumMoon._decimals (QuantumMoon.sol#557) should be constant
QuantumMoon._name (QuantumMoon.sol#555) should be constant
QuantumMoon._symbol (QuantumMoon.sol#556) should be constant
QuantumMoon.busdAddr (QuantumMoon.sol#581) should be constant
QuantumMoon.lotteryWallet (QuantumMoon.sol#576) should be constant
QuantumMoon.marketingWallet (QuantumMoon.sol#574) should be constant
QuantumMoon.numTokensSellToAddToLiquidity (QuantumMoon.sol#590) should be constant
QuantumMoon.teamWallet (QuantumMoon.sol#575) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
renounceOwnership() should be declared external:
- Ownable.renounceOwnership() (QuantumMoon.sol#293-296)
transferOwnership(address) should be declared external:
- Ownable.transferOwnership(address) (QuantumMoon.sol#302-306)
geUnlockTime() should be declared external:
- Ownable.geUnlockTime() (QuantumMoon.sol#308-310)
lock(uint256) should be declared external:
- Ownable.lock(uint256) (QuantumMoon.sol#313-318)
unlock() should be declared external:
- Ownable.unlock() (QuantumMoon.sol#321-326)
SetSwapContractAddr(address) should be declared external:
- QuantumMoon.SetSwapContractAddr(address) (QuantumMoon.sol#626-629)
name() should be declared external:
- QuantumMoon.name() (QuantumMoon.sol#631-633)
symbol() should be declared external:
- QuantumMoon.symbol() (QuantumMoon.sol#635-637)
decimals() should be declared external:
- QuantumMoon.decimals() (QuantumMoon.sol#639-641)
totalSupply() should be declared external:
- QuantumMoon.totalSupply() (QuantumMoon.sol#643-645)
transfer(address,uint256) should be declared external:
- QuantumMoon.transfer(address,uint256) (QuantumMoon.sol#651-654)
allowance(address,address) should be declared external:
- QuantumMoon.allowance(address,address) (QuantumMoon.sol#656-658)
approve(address,uint256) should be declared external:
- QuantumMoon.approve(address,uint256) (QuantumMoon.sol#660-663)
transferFrom(address,address,uint256) should be declared external:
- QuantumMoon.transferFrom(address,address,uint256) (QuantumMoon.sol#665-669)
increaseAllowance(address,uint256) should be declared external:
- QuantumMoon.increaseAllowance(address,uint256) (QuantumMoon.sol#671-674)
decreaseAllowance(address,uint256) should be declared external:

- QuantumMoon.decreaseAllowance(address,uint256) (QuantumMoon.sol#676-679)
totalFees() should be declared external:
- QuantumMoon.totalFees() (QuantumMoon.sol#681-683)
deliver(uint256) should be declared external:
- QuantumMoon.deliver(uint256) (QuantumMoon.sol#685-691)
excludeFromFee(address) should be declared external:
- QuantumMoon.excludeFromFee(address) (QuantumMoon.sol#710-712)
includeInFee(address) should be declared external:
- QuantumMoon.includeInFee(address) (QuantumMoon.sol#714-716)
setSwapAndLiquifyEnabled(bool) should be declared external:
- QuantumMoon.setSwapAndLiquifyEnabled(bool) (QuantumMoon.sol#718-721)
isExcludedFromFee(address) should be declared external:
- QuantumMoon.isExcludedFromFee(address) (QuantumMoon.sol#819-821)
SetMainContractAddr(address) should be declared external:
- swapBUSDcontract.SetMainContractAddr(address) (QuantumMoon.sol#1071-1073)
transferBUSD() should be declared external:
- swapBUSDcontract.transferBUSD() (QuantumMoon.sol#1074-1078)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:QuantumMoon.sol analyzed (10 contracts with 75 detectors), 104 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration

Email: audit@EtherAuthority.io