MOON

SMART CONTRACT AUDIT REPORT

JULY 12TH 2020

PREPARED BY
CRYPTIC OCEAN

INDEX

Moon is an experimental deflationary token with a staking reward system along with referrals.

INTRODUCTION

This Audit Report highlights the overall security of Moon Smart Contract. With this report, we have tried to ensure the reliability of their smart contract by complete assessment of their system's architecture and the smart contract codebase.

Visit Website

AUDITING APPROACH AND METHODOLOGIES APPLIED

CrypticOcean team has performed thorough testing of the project starting with analysing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third party smart contracts and libraries.

Our team then performed a formal line by line inspection of the Smart Contract in order to find any potential issue like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.

In the Unit testing Phase we coded/conducted Custom unit tests written for each function in the contract to verify that each function works as expected. In Automated Testing, We tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

AUDITING APPROACH AND METHODOLOGIES APPLIED

The code was tested in collaboration of our multiple team members and this included -

- Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
- Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
- Deploying the code on testnet using multiple clients to run live tests
- Analysing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest version.
- Analyzing the security of the on-chain data.

AUDIT DETAILS

Project Name: Moon

Website/Etherscan Code : https://moon.dev/

Languages: Solidity(Smart contract), Javascript(Unit Testing)

Commit hash : d6f5e2e4d28a75aaf4b341f3d91d3488c14c9b61

SUMMARY OF MOON SMART CONTRACT

Cryptic Ocean conducted a security audit of a smart contract of Moon. Moon contract is used to create the ERC20 token which is a Moon TOKEN, Smart contract contains basic functionalities of ERC20 along with unique staking, referral and burning capability of Moon token.

All holders who do not sell or move ANY Moon V1 from that block until the snapshot before launch will get a bonus swap ratio of 1:1.2! :rocket: ie a 20% bonus reward.

AUDIT GOALS

The focus of the audit was to verify that the smart contract system is secure, resilient and working according to its specifications. The audit activities can be grouped in the following three categories:

Security: Identifying security related issues within each contract and within the system of contracts.

Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality: A full review of the contract source code.

The primary areas of focus include:

- Correctness
- Readability
- Sections of code with high complexity
- Quantity and quality of test coverage

SECURITY LEVEL REFERENCES

Every issue in this report was assigned a severity level from the following:

High severity

Will bring problems and should be fixed.

Medium severity

Could potentially bring problems and should eventually be fixed.

Low severity

Are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

High severity issues

No high Severity issue Found

Medium Severity Issues

No Medium Severity Issue.

Low severity issues

- **Ownable.initialize() and detailedERC20.initialize() access modifier can be changed to internal from public.**

- **Report of a shadow variable has been found in initialize contract**

  Many variables have been shadowed in the smart contract. Make sure, you have inherited the contract in a standard way to minimize the shadow of variables.

```
PoolManagerRole._____gap (PoolManagerRole.sol#48) shadows:
        - Initializable._____gap (Initializable.sol#61)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing
INFO:Detectors:
```

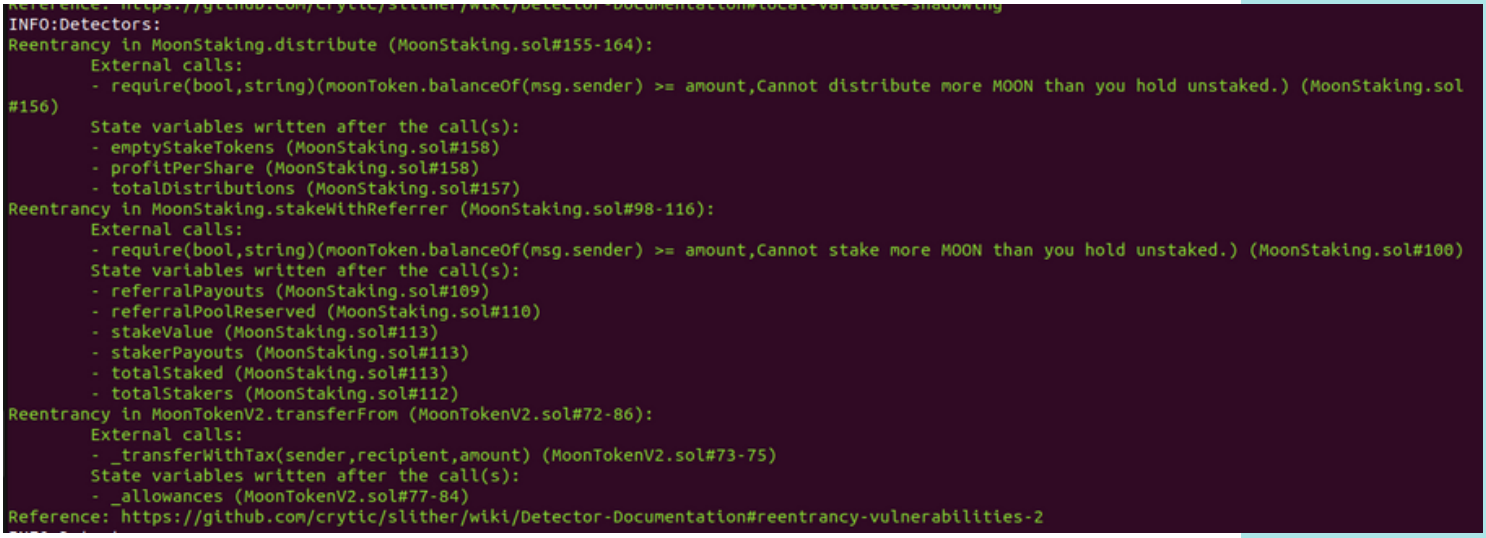
- **Ignores return value by external calls**

  Your smart contract has ignored the return values at many place. Make sure you return a function if you have implemented it.

```
INFO:Detectors:
Initializable.isConstructor (Initializable.sol#48-58) is declared view but contains assembly code
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#constant-functions-changing-the-state
INFO:Detectors:
MoonStaking.unstake (MoonStaking.sol#118-138) ignores return value by external calls "referralPool.add(referralTokens)" (MoonStaking.sol#133)
MoonStaking.withdraw (MoonStaking.sol#140-145) ignores return value by external calls "moonToken.transfer(msg.sender,amount)" (MoonStaking.sol
#143)
MoonStaking.claimReferralRewards (MoonStaking.sol#166-175) ignores return value by external calls "moonToken.transfer(msg.sender,amount)" (Moo
nStaking.sol#173)
MoonStaking.claimExcessFromReferralPool (MoonStaking.sol#177-182) ignores return value by external calls "moonToken.transfer(msg.sender,amount
)" (MoonStaking.sol#180)
MoonStaking.handleReferralDistribution (MoonStaking.sol#190-193) ignores return value by external calls "referralPool.add(amount)" (MoonStakin
g.sol#191)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
```

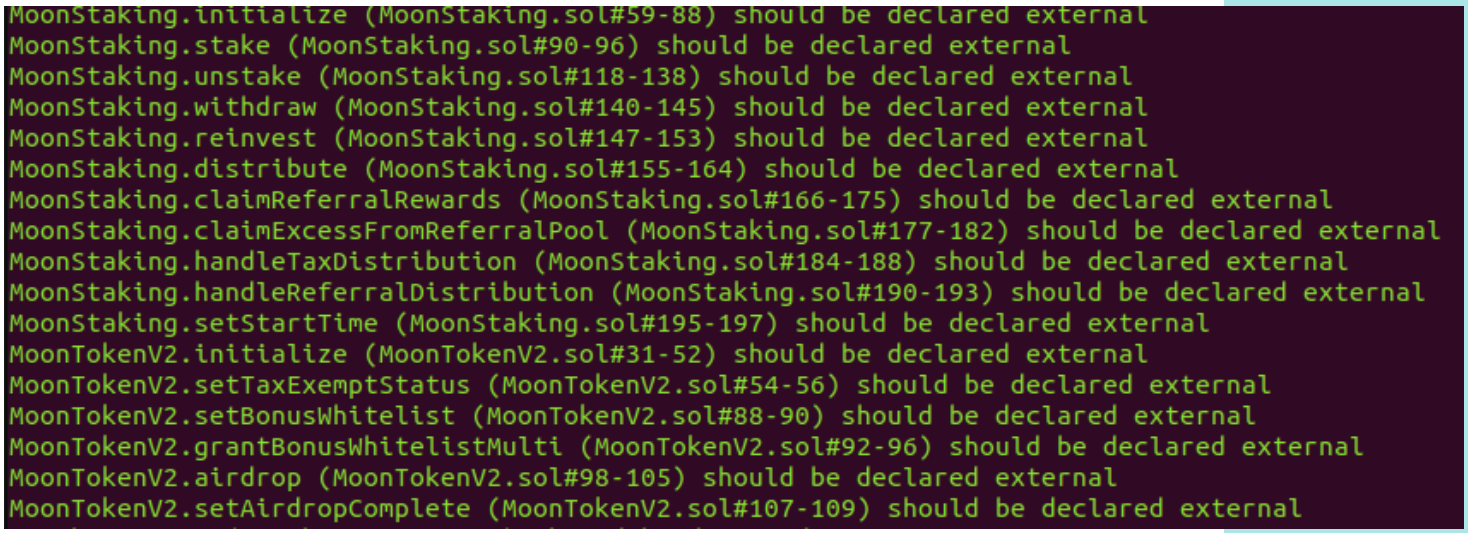
- **Reentrancy issues has been found**

  Smart contract contains some functions that arise the situation of reentrancy make sure you have all the checks to minimize the reentrancy issue in smart contract functions



- **These functions can be declared externally**

  Public functions that are never called by the contract should be declared external to save gas.



UNIT TESTING

TEST SUITE
CONTRACT: MOON V2 TOKEN CONTRACTS

Should correctly initialize constructor values of Moon Token Contract (174ms)

Should check the Total Supply of Moon TOKEN (44ms)

Should check the Maximum Total Supply of Moon TOKEN (56ms)

Should check the Name of a token of Moon TOKEN (60ms)

Should check the symbol of a token of Moon TOKEN (46ms)

Should check the decimal of a token of Moon TOKEN (52ms)

Should check the Owner of a Moon TOKEN contract (39ms)

Should check the balance of a Owner (57ms)

Should check transfered allowed or Not (46ms)

Should not be able to Mint token by Non Owner account (169ms)

Should Mint token by Owner to Account [1] (302ms)

Should check the balance of a Account [1] after minting

Should check the Total Supply of Moon TOKEN after Minting Token (40ms)

Should not be able to burn tokens of Account [1] more than account balance (91ms)

Should not be able to burn tokens of Account [1] by Non owner Account (111ms)

Should be able to burn tokens of Account [1] by owner only (136ms)

Should check the balance of a Account [1] after tokens burned by Owner (92ms)

Should check the Total Supply of Moon TOKEN after tokens are burned

Should be able to Mint tokens by Owner to Owner (362ms)

Should check the balance of a Account [0] after minting

Should check the Total Supply of Moon TOKEN after Minting Token

Owner Should be able to burn tokens using function Burn (225ms)

Should check the balance of Owner after burn

Should check the Total Supply of Moon TOKEN after Burn

Should be able to transfer tokens to accounts[2] by owner (195ms)

Should Not be able to transfer tokens to accounts[3] by accounts[2] when transfer is Not allowed (188ms)

Should correctly initialize constructor of MoonTokenV2 token Contract (187ms)

Should correctly initialize constructor of Moon Staking Contract (209ms)

Should initialize moontokenv2 (135ms)

Should check a name of a token (41ms)

Should check a symbol of a token

Should check a decimal of a token

Should check a owner of a token (39ms)

Should check is owner of a token

Should check if Air drop complete (63ms)

Should check Tax basic points

Should check referral basic points

Should check burn basic points

Should check Tax basic points of stacking contracts (39ms)

Should check referral basic points of stacking contracts

Should check burn basic points of stacking contracts

Should check dividends of investor before staking (143ms)

Should check a owner of a staking (42ms)

Should check is owner of a stacking (39ms)

Should check if a address is a pool manager (38ms)

Should check if a address is a pool manager when its not

Should check if a address is a pool manager for true

Should be able to aproove tokens to spent (40ms)

Should be able to burn own tokens (43ms)

Should be able to burn approved token

Should be able to increase the allowance (40ms)

Should be able to decrease the allowance (43ms)

Should be able to grant address whitelist (49ms)

Should be able to airdrop

Should be able to set airdrop complete (42ms)

Should be able to transfer ownership (38ms)

Should be able to check new owner of a contract (39ms)

Should be able to check is owner of a contract

Should be able to transfer tokens (38ms)

Should be able to transfer approved tokens

Should be able to set bonus whitelist

Should be able to reannounce ownership

Should be able to set tax excempt status

Should be able to check tax exempt (39ms)

Should be able to add pool manager

Should be able to check pool manage

Should be able to stake

Should be able to check stake value (39ms)

Should be able to unstake

Should be able to invest again

Should be check balance

Should be able to stake with refferal (38ms)

Should be able to check stake value (39ms)

Should be able to unstake (39ms)

Should be able to transfer ownership of contract (41ms)

Should be able to check owner of a contract

Should be able to renounce ownership (41ms)

Should be able to check referral payout (39ms)

Should be able to check stake referral 85 passing (9s)

FINAL RESULT OF TEST:

85 Passing (9s) PASSED **0 Failed**

NOTE

Unit testing is the most important part of smart contract audit that allows auditors to create scenarios to exploit vulnerabilities of smart contracts, by preparing the environment for known attacks and also to validate business logic of a smart contract. Efficiency of unit testing can be checked by coverage reports of smart contracts.

CONTRACT DESCRIPTION TABLE

Files Description Table

File Name	SHA-1 Hash
MoonTokenV2.sol	a70043f4cd73e5e0996e954995fbdc10ee743aff

Contracts Description Table

Contract	Type	Bases		
└	Function Name	Visibility	Mutability	Modifiers
MoonTokenV2	Implementation	Initializable, Ownable, ERC20Burnable, ERC20Detailed		
└	initialize	Public ❗	🔴	initializer
└	setTaxExemptStatus	Public ❗	🔴	onlyOwner
└	taxAmount	Public ❗		NO ❗
└	transfer	Public ❗	🔴	NO ❗
└	transferFrom	Public ❗	🔴	NO ❗
└	setBonusWhitelist	Public ❗	🔴	onlyOwner
└	grantBonusWhitelistMulti	Public ❗	🔴	onlyOwner
└	airdrop	Public ❗	🔴	onlyOwner
└	setAirdropComplete	Public ❗	🔴	onlyOwner
└	_airdrop	Internal 🔒	🔴	
└	_transferWithTax	Internal 🔒	🔴	

Legend

Symbol	Meaning
🔴	Function can modify state
💵	Function is payable

Files Description Table

File Name	SHA-1 Hash
MoonStaking.sol	a1f787408d703d2c24b6b926661c26e7adf39333

Contracts Description Table

Contract	Type	Bases		
└	Function Name	Visibility	Mutability	Modifiers
MoonStaking	Implementation	Initializable, PoolManagerRole, Ownable		
└	initialize	Public ❗	🔴	initializer
└	stake	Public ❗	🔴	whenStakingActive
└	stakeWithReferrer	Public ❗	🔴	whenStakingActive
└	unstake	Public ❗	🔴	whenStakingActive
└	withdraw	Public ❗	🔴	whenStakingActive
└	reinvest	Public ❗	🔴	whenStakingActive
└	distribute	Public ❗	🔴	NO ❗
└	claimReferralRewards	Public ❗	🔴	NO ❗
└	claimExcessFromReferralPool	Public ❗	🔴	onlyPoolManager
└	handleTaxDistribution	Public ❗	🔴	onlyMoonToken
└	handleReferralDistribution	Public ❗	🔴	onlyMoonToken
└	setStartTime	Public ❗	🔴	onlyOwner
└	dividendsOf	Public ❗		NO ❗
└	taxAmount	Public ❗		NO ❗
└	uintToInt	Internal 🔒		
└	_addStake	Internal 🔒	🔴	
└	_increaseProfitPerShare	Internal 🔒	🔴	

Legend

Symbol	Meaning
🔴	Function can modify state
💵	Function is payable

CODE COVERAGE FOR SOLIDITY TESTING

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	90.36	63.64	92.59	90.8	
BasicToken.sol	100	50	100	100	
DigipharmToken.sol	100	83.33	100	100	
ERC20.sol	100	100	100	100	
ERC20Basic.sol	100	100	100	100	
MintableToken.sol	100	50	100	100	
Ownable.sol	100	75	100	100	
SafeMath.sol	41.67	25	50	41.67	... 18,19,27,29
StandardToken.sol	95.24	62.5	100	95.24	87
All files	90.36	63.64	92.59	90.8	

Coverage report will let you know the efficiency of smart contract unit testing

Implementation and Recommendations :

- All the possible information based or low severity issues have been explained above. We recommend you to go through it.
- As contract deals in stacking and payout of tokens we suggest you to assure thereentrancy part again and if possible you can check for address while stacking if it's not a contract address and a wallet address.

C**omments:**

Use case of smart contract is very well designed and Implemented.Overall, the code is clearly written, and demonstrates effective use of abstraction, separation of concerns, and modularity. Moon development team demonstrated high technical capabilities, both in the design of the architecture and in the implementation.

We found some critical issue and several additional issues that require the attention of the Moon team. Given the subjective nature of some assessments, it will be up to the Moon team to decide whether any changes should be made.