Audit report of DOGELABS

Prepared By: - Kishan Patel **Prepared for: DOGECOINLABLS**

Prepared On: - 31/07/2022

Connect with auditor: - https://t.me/SolidityContractAuditor

Table of contents

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 DOGECOINLABLS.(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 31/07/2022 – 01/08/2022.

The project has 1 file. It contains approx 300 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	DOGECOINLABS
Token Symbol	DOGELABS
Platform	Binance Smart Contract
Order Started Date	31/07/2022
Order Completed Date	02/08/2022

4. List of attacks checked

- Over and under flows
- Short address attack
- Visibility & Delegate call
- Reentrancy / TheDAO hack
- Forcing BSC 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 newAddress is not same as uniswapV2Router address.

```
1807    function updateUniswapV2Router(address newAddress) public onlyO
1808        require(newAddress != address(uniswapV2Router), "The router
1809        emit UpdateUniswapV2Router(newAddress, address(uniswapV2Rou
```

  - Here you are checking that account address has not same value as excluded in isExcludedFromFees mapping.

```
1813    function excludeFromFees(address account, bool excluded) public
1814        require(_isExcludedFromFees[account] != excluded, "Account
```

  - Here you are checking that pair address is not same as uniswapV2Pair address.

```
1825    function setAutomatedMarketMakerPair(address pair, bool value)
1826        require(pair != uniswapV2Pair, "The PancakeSwap pair cannot
```

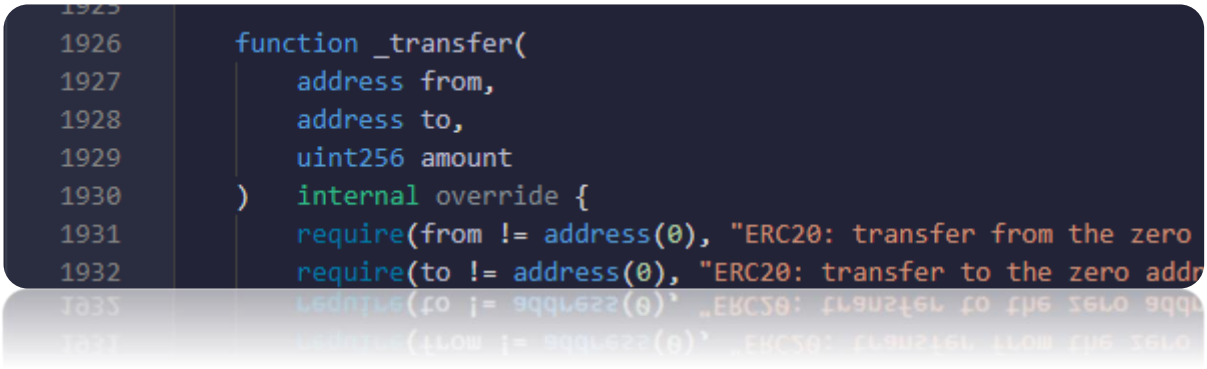
  - Here you are checking that current pair address value in automatedMarketMakerPairs mapping is not same as value.

```
1831    function _setAutomatedMarketMakerPair(address pair, bool value)
1832        require(automatedMarketMakerPairs[pair] != value, "Automatec
1833        automatedMarketMakerPairs[pair] = value;
```

  - Here you are checking that newValue is between 20000 and 50000. newValue is not same as existing (gasForProcessing) value.

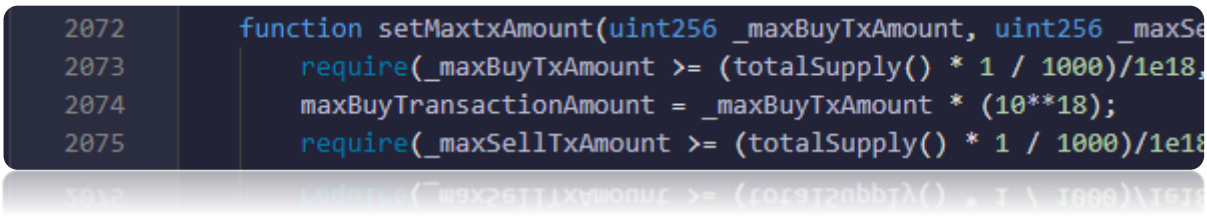
```
1842    function updateGasForProcessing(uint256 newValue) public onlyOw
1843        require(newValue >= 200000 && newValue <= 500000, "gasForPr
1844        require(newValue != gasForProcessing, "Cannot update gasFor
```

- Here you are checking that from and to address is valid and proper, amount is bigger or equal to maxBuyTransactionAmount or maxSellTransactionAmount.



```
1926    function _transfer(
1927        address from,
1928        address to,
1929        uint256 amount
1930    )   internal override {
1931        require(from != address(0), "ERC20: transfer from the zero
1932        require(to != address(0), "ERC20: transfer to the zero addr
```

- Here you are checking that _maxBuyTxAmount and _maxSellTxAmount is bigger or equal to 0.1% of totalSupply.



```
2072    function setMaxtxAmount(uint256 _maxBuyTxAmount, uint256 _maxSe
2073        require(_maxBuyTxAmount >= (totalSupply() * 1 / 1000)/1e18,
2074        maxBuyTransactionAmount = _maxBuyTxAmount * (10**18);
2075        require(_maxSellTxAmount >= (totalSupply() * 1 / 1000)/1e18
```

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. 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: - updateClaimWait**

```
1849    function updateClaimWait(uint256 claimWait) external onlyOwner
1850        dividendTracker.updateClaimWait(claimWait);
```

  - Here in updateClaimWait method you can check that claimWait value is bigger than 0.

- **Function: - setSwapAndLiquiifyEnabled**

```
1921    function setSwapAndLiquifyEnabled(bool _enabled) public onlyOwn
1922        swapAndLiquifyEnabled = _enabled;
1923        emit SwapAndLiquifyEnabledUpdated(_enabled);
```

  - Here in setSwapAndLiquifyEnabled method you can check that _enabled value is not same as swapAndLiquifyEnabled value.

- **Function: - setSwapTokensAtAmount**

```
2079    function setSwapTokensAtAmount(uint256 newAmount) public onlyOw
2080        swapTokensAtAmount = newAmount;
2081    }
```

  - Here in setSwapTokensAtAmount method you can check that newAmount value is bigger than 0.

10. Summary

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

Critical	Medium	Low
0	**0**	**3**

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 try to implement suggested code validations.