DictatorDOGE Smart Contract Final Audit Report

Scope of Audit

The scope of this audit was to analyze and document the DictatorDOGE Token smart contract codebase for quality, security, and correctness.

Check Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis
In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis
A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption
In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis, Theo.

Issue Categories

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues
A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues
The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues
Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational Issues
These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of security issues per severity.

TYPE	HIGH	MEDIUM	LOW	INFORMATIONAL
Open	0	0	0	0
Acknowledged	0	3	2	10
Closed	1	1	0	0

Introduction

During the period of **July 26, 2021 to July 28, 2021** - QuillAudits Team performed a security audit for **DictatorDOGE** smart contracts.

The code for the audit was taken from following the official link:

V	Date	Link	Note
1	July	https://bscscan.com/address/0x011b8f10aab6d71f90dd5b71b2fe0fa76a1bad89#code	Version 1
2	July	https://bscscan.com/address/0x840c53b309b683e673a765019e57b9e851e2c65f#code	Version 2

A. Contract - DictatorDOGE

Issues Found – Code Review / Manual Testing

High Severity Issues

A.1 Anomalous behavior with special fee transactions

Line	Code/Function

754-789	`// Buy`
	`if(from == uniswapV2Pair){`
	`            removeAllFee();`
	`            _taxFee = _buyTaxFee;`
	`        _liquidityFee = _buyLiquidityFee;`
	`}`
	`// Sell`
	`if(to == uniswapV2Pair){`
	`            removeAll`
	`……`
	`……`
	`…….`
	
	`                        if(from == uniswapV2Pair){`
	`                        _taxFee = _addressFees[to]._buyTaxFee;`
	`                                    _liquidityFee = _addressFees[to]._buyLiquidityFee;`
	`                        }`
	`            }`
	`}`

Description

Case 1:

A transfer of tokens to addressFee enabled Address will remove all the fees, and all transactions after this can be done without any taxFee and liquidityFee. This can be exploited by any user to bypass the fees by just sending a small amount to addressFee enabled Address, before doing the actual transfer to other addresses.

Case 2:
After a user with special fee enabled does a transaction, any following transactions will also have the same _taxFee and _liquidityFee which this user is charged. This happens because the removeAllFee() function removes the fees and stores it in variables to restore later. But it is never restored after the transaction. The transactions, after a special fee transaction, are still charged the same fee.

Remediation:

Restore the fee after the transaction with the special fee is complete.

Status: Closed
In version 2, restoreFee() function is called after transfer is complete.

Medium Severity Issues

A.2 Costly loops leading to DOS attack

Line	Code/Function

1029-1045	for (uint256 j = 0; j < _sellHistories.length; j ++) {
	if (_sellHistories[j].time >= maxStartTimeForHistories) {
	_sellHistories[i].time = _sellHistories[j].time;
	_sellHistories[i].bnbAmount = _sellHistories[j].bnbAmount;

```
                    i = i + 1;
          }
}

uint256 removedCnt = _sellHistories.length - i;

for (uint256 j = 0; j < removedCnt; j ++) {

          _sellHistories.pop();
}
```

Description

The _sellHistories[] array is used in a for loop in the _removeOldSellHistories() function. Elements are inserted into this array, whenever there is a transfer of tokens to the UniswapV2Pair address. This can be done by any user multiple times to increase the array size.

Whenever _removeOldSellHistories() function is called, the array is traversed, and some elements are updated. If the size of the array is very large, the transaction's gas usage will exceed the block gas limit, and the transaction will fail.

Remediation:

We recommend having a check on the size of _sellHistories[] array. The logic in the _removeOldSellHistories() function should be changed to prevent such a scenario.

Status: Closed

The team has set _isAutoBuyBack to false, which means the _removeOldSellHistories() function is never called.
Note: This setting can be changed again by the owner at any point of time in the future.
Comments from Auditee: "We have disabled _isAutoBuyBack to avoid the _sellHistories array size issue."

A.3 BuyBack feature can be exploited

Line	Code/Function

691-697	`if (to == uniswapV2Pair && balanceOf(uniswapV2Pair) > 0) {`
	`    SellHistories memory sellHistory;`
	`    sellHistory.time = block.timestamp;`
	`    sellHistory.bnbAmount = _getSellBnBAmount(amount);`
	
	`    _sellHistories.push(sellHistory);`
	`}`

Description

A sell transaction or the swap from DictatorDOGE to ETH will have to == uniswapV2Pair . But any user can exploit this by sending normal transactions to uniswapV2Pair address. This will fill the _sellHistories[] array with incorrect entries.

Any user can send in very small amounts of tokens to fill up the _sellHistories[] array, for _buyBackTimeInterval minutes. And when the _isAutoBuyBack == true , _bBSLimitMax will be the average of transactions in the last _buyBackTimeInterval minutes. And due to this, the tokens bought back with the buyBackTokens() function will be less. This can lead to token price manipulation.

Remediation:

We recommend changing the logic used to determine a token sale.
In the _bBSLimitMax calculation, taking the average of the last few transactions can be used by a malicious user to manipulate the token price.

Status: Acknowledged by the Auditee

Comments from Auditee: “We understand the risks of this possible exploitation and are monitoring specifically for it.”

A.4 External calls to untrusted contracts

Line	Code/Function

572-575	`function transfer(address recipient, uint256 amount) public override returns (bool) {`
	`_transfer(_msgSender(), recipient, amount);`
	`return true;`
	`}`

586-590	`function transferFrom(address sender, address recipient, uint256 amount) public override returns (bool) {`
	`_transfer(sender, recipient, amount);`
	`_approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance"));`
	`return true;`
	`}`

Description

Calling DictatorDOGE.transfer() and DictatorDOGE.transferFrom() might trigger function uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens() and uniswapV2Router.swapExactETHForTokensSupportingFeeOnTransferTokens() , which is implemented by a third party at uniswapV2Router. Also the Router address can be changed by the owner by using the function changeRouterVersion().

The scope of the audit would treat the third-party implementation at uniswapV2Router as a black box and assume its functional correctness. However, third parties may be compromised in the real world that leads to assets lost or stolen.

Remediation:

We understand that the DictatorDOGE contract requires interaction with the PancakeSwap router for token swaps or adding liquidity to the pool. We recommend the team to monitor the status of these interactions.

Status: **Acknowledged by the Auditee**
Comments from Auditee: "We will monitor the PCS router for possible exploits."

A.5 Centralization Risks

Description:

The role owner has the authority to
- update settings (transaction fees and addresses)
- manage the list containing contracts excluding from reward, fee, or max transaction limitation.
- withdraw ether from the contract at any point of time.
- swap contract BNB balance to tokens

Remediation:

We advise the client to handle the governance account carefully to avoid any potential hack. We also advise the client to consider the following solutions:

1. with reasonable latency for community awareness on privileged operations;
2. Multisig with community-voted 3rd-party independent co-signers;
3. DAO or Governance module increasing transparency and community involvement;

Status: **Acknowledged by the Auditee**
Comments from Auditee: "The contract needs to have flexibility in updating taxes, fees and addresses to support our future dApps and exchange listings. We will implement community governance via EverOwn when it is released."

Low Severity Issues

A.6 Missing Range Check for Input Variable

Description:

The owner can set the following state variables arbitrary large or small causing potential risks in fees and anti whale :

- _buyBackMaxTimeForHistories
- _buyBackDivisor
- _buyBackTimeInterval
- _intervalMinutesForSwap
- _taxFee
- _buyTaxFee
- _buyLiquidityFee
- _sellTaxFee
- _sellLiquidityFee
- _liquidityFee
- buyBackSellLimit
- _maxTxAmount
- marketingDivisor
- minimumTokensBeforeSwap

Remediation:

We recommend setting ranges and check the above input variables.

Status: **Acknowledged by the Auditee**

A.7 Missing zero address validation

Line	Code
1112-1114	function setMarketingAddress(address _marketingAddress) external onlyOwner {

	`marketingAddress = payable(_marketingAddress);`
	`}`

Description

When updating the marketing address, it should be checked for zero address. Otherwise, tokens/ETH sent to the zero address may be burnt forever.

Remediation

Use a require statement to check for zero address when updating the marketing address.

Status: **Acknowledged by the Auditee**

Informational Issues

A.8 Presence of unused code

Line	Code
499	bool public _isEnabledBuyBackAndBurn = true;
501	event RewardLiquidityProviders(uint256 tokenAmount);

505-509	event SwapAndLiquify(
	        uint256 tokensSwapped,
	        uint256 ethReceived,
	        uint256 tokensIntoLiqudity
	);

850-863	function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
	      // Approve token transfer to cover all possible scenarios
	      _approve(address(this), address(uniswapV2Router), tokenAmount);
	
	      // Add the liquidity
	      uniswapV2Router.addLiquidityETH{value: ethAmount}(
	         address(this),
	         tokenAmount,
	         0, // Slippage is unavoidable
	         0, // Slippage is unavoidable
	         owner(),
	         block.timestamp
	         );
	}

Description:

The program contains code that is not essential for execution, i.e., makes no state changes and has no side effects that alter data or control flow, such that removal of the code would have no impact on functionality or correctness.

Remediation:

We recommend removing the unused code.

Status: **Acknowledged by the Auditee**

A.9 Missing Events for Significant Transactions

Description:

The missing event makes it difficult to track off-chain liquidity fee changes. An event should be emitted for significant transactions calling the following functions:

1. setMarketingAddress
2. setNumTokensSellToAddToBuyBack
3. setMarketingDivisor
4. setMaxTxAmount
5. setBuyBackSellLimit
6. setLiquidityFeePercent
7. setSellFee
8. setBuyFee
9. setTaxFeePercent
10. SetSwapMinutes
11. SetBuyBackRangeRate
12. SetBuyBackTimeInterval
13. SetBuyBackDivisor
14. SetBuyBackMaxTimeForHistories
15. includeInFee
16. excludeFromFee

Remediation:

We recommend emitting an event to log the update of the variables.

Status: **Acknowledged by the Auditee**

A.10 Unlocked pragma

```
pragma solidity ^0.8.4;
```

Description

Contracts should be deployed using the same compiler version/flags with which they have been tested. Locking the pragma (for e.g., by not using ^ in *pragma solidity 0.8.0)* ensures that contracts do not accidentally get deployed using an older compiler version with unfixed bugs.

Remediation

Lock the pragma version.

Status: Acknowledged by the Auditee

A.11 State variables that could be declared constant

```
_decimals
_tTotal
_isEnabledBuyBackAndBurn
_name
deadAddress
_symbol
```

Description:

The above constant state variables should be declared constant to save gas.

Remediation:

Add the constant attributes to state variables that never change.

Status: Acknowledged by the Auditee

A.12 Variable Typos

Line	Code
508	tokensIntoLiqudity

Description

There are typos in the above variables.

Remediation

We recommend correcting and changing tokensIntoLiqudity to tokensIntoLiquidity.

Status: Acknowledged by the Auditee

A.13 Conformance to Solidity naming conventions

Description

In the contract, many function names were found to be starting with capital letters. Functions other than constructors should use mixedCase. Examples: getBalance, transfer, verifyOwner, addMember, changeOwner

Remediation

Follow the Solidity naming convention.

Status: Acknowledged by the Auditee

A.14 State Variable Default Visibility

Line	Code
495	bool inSwapAndLiquify;

Description

The Visibility of the inSwapAndLiquify variable is not defined. Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

The default is internal for state variables, but it should be made explicit.

Remediation

We recommend adding the visibility for the state variable of inSwapAndLiquify.

Variables can be specified as being public, internal or private. Explicitly define visibility for all state variables.

Status: **Acknowledged by the Auditee**

A.15 Public function that could be declared external

Description

The following public functions that are never called by the contract should be declared external to save gas:

- deliver()
- reflectionFromToken()
- totalFees()
- GetBuyBackTimeInterval()
- GetSwapMinutes()
- setBuyBackEnabled()
- setAutoBuyBackEnabled()

- transferForeignToken()
- changeRouterVersion(address)
- theDictator()

Remediation

Use the external attribute for functions never called from the contract.

Status: Acknowledged by the Auditee

A.16 ERC20 transfer() does not return boolean

Line	Code

1168-1172	function transferForeignToken(address _token, address _to) public onlyOwner returns(bool _sent){
	    require(_token != address(this), "Can't let you take all native token");
	    uint256 _contractBalance = IERC20(_token).balanceOf(address(this));
	    _sent = IERC20(_token).transfer(_to, _contractBalance);
	}

Description

As many tokens do not follow the ERC20 standard faithfully, they may not return a bool variable in this function's execution, meaning that simply expecting it can cause incompatibility with these types of tokens.

Remediation

Use SafeERC20 provided by the OpenZeppelin library for IERC20. And use the safeTransfer() function for token transfers. The OpenZeppelin implementation optionally checks for a return value which makes it compatible with all ERC20 token implementations.

Status: Acknowledged by the Auditee

A.17 Avoid using .transfer() to transfer Ether

Line	Code

1145-1147	function transferToAddressETH(address payable recipient, uint256 amount) private {
	    recipient.transfer(amount);
	}

1174-1177	function Sweep() external onlyOwner {
	    uint256 balance = address(this).balance;
	    payable(owner()).transfer(balance);
	}

Description

Although transfer() and send() have been recommended as a security best-practice to prevent reentrancy attacks because they only forward 2300 gas, the gas repricing of opcodes may break deployed contracts.

For reference, read more.

Remediation

Use .call{ value: ... }("") instead, without hardcoded gas limits along with checks-effects-interactions pattern or reentrancy guards for reentrancy protection.

Status: **Acknowledged by the Auditee**

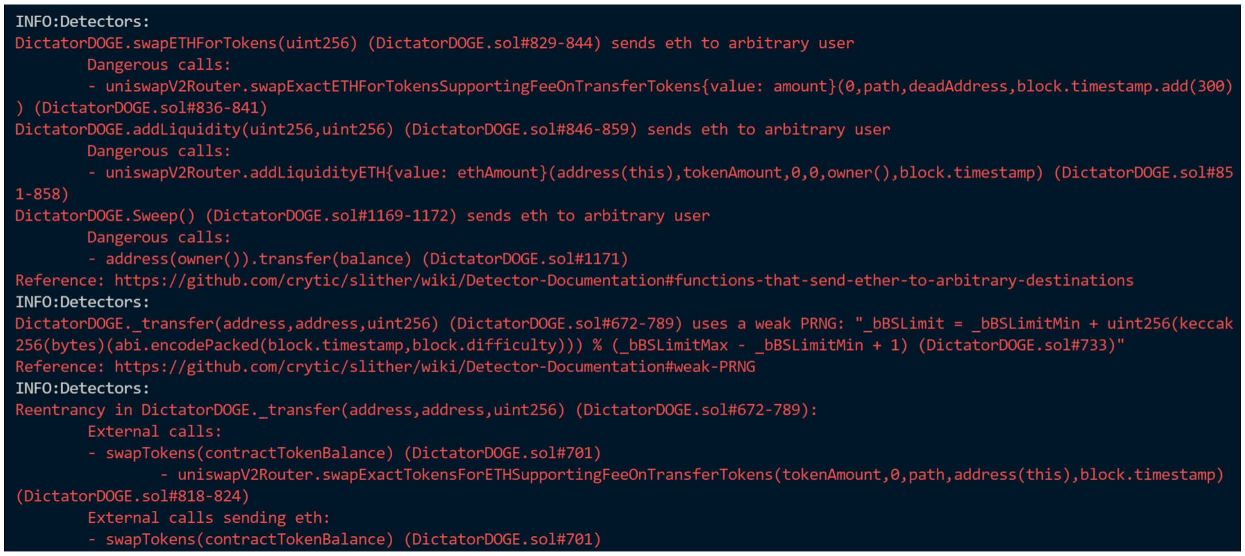
Functional Tests

Function Names	Testing results
transfer()	Passed
transferFrom()	Passed
approve()	Passed
increaseAllowance()	Passed
decreaseAllowance()	Passed
deliver()	Passed
reflectionFromToken()	Passed
tokenFromReflection()	Passed
excludeFromReward()	Passed
includeFromReward()	Passed
excludeFromFee()	Passed
includeInFee()	Passed
setSwapAndLiquifyEnabled()	Passed
setBuyBackEnabled()	Passed
setAutoBuyBackEnabled()	Passed
changeRouterVersion()	Passed
transferForeignToken()	Passed
SetBuyBackMaxTimeForHistories()	Passed
SetBuyBackDivisor()	Passed
SetBuyBackTimeInterval()	Passed

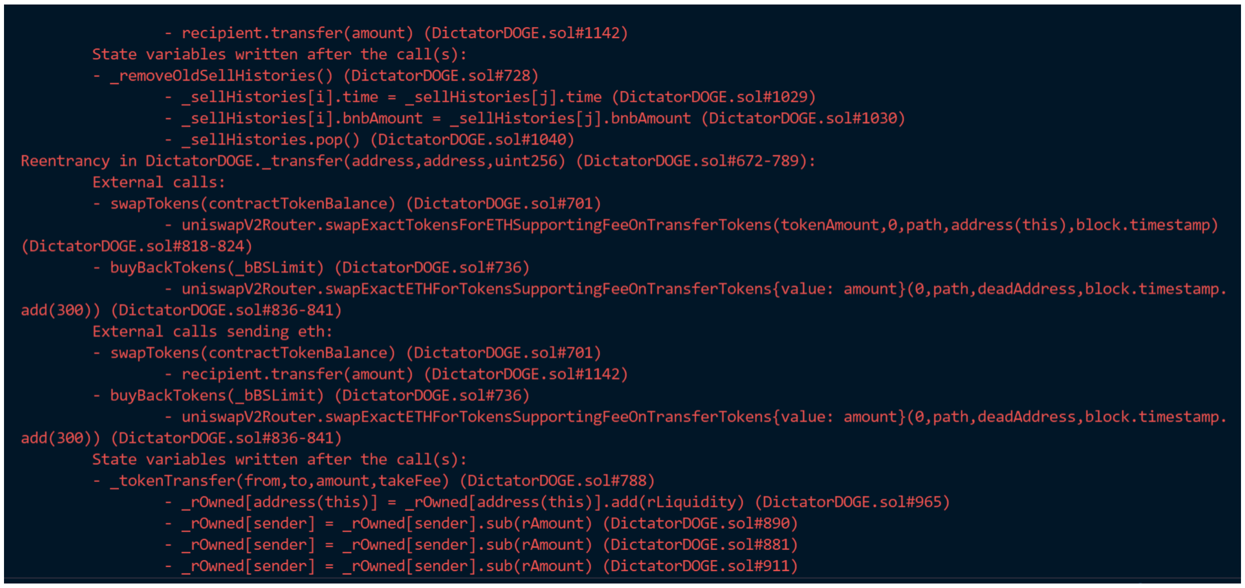
SetBuyBackRangeRate()	Passed
SetSwapMinutes()	Passed
setTaxFeePercent()	Passed
setBuyFee()	Passed
setSellFee()	Passed
setLiquidityFeePercent()	Passed
setBuyBackSellLimit()	Passed
setMaxTxAmount()	Passed
setMarketingDivisor()	Passed
setNumTokensSellToAddToBuyBack()	Passed
setMarketingAddress()	Passed
prepareForPreSale()	Passed
afterPreSale()	Passed
Sweep()	Passed
setAddressFee()	Passed
setBuyAddressFee()	Passed
setSellAddressFee()	Passed

Automated Tests

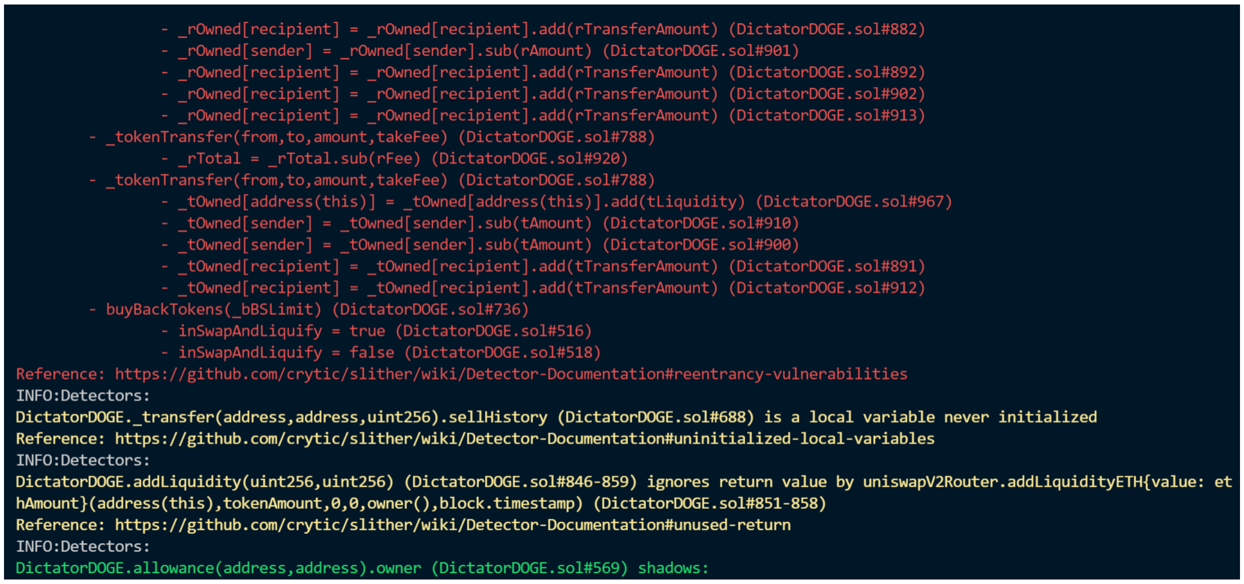
Slither:



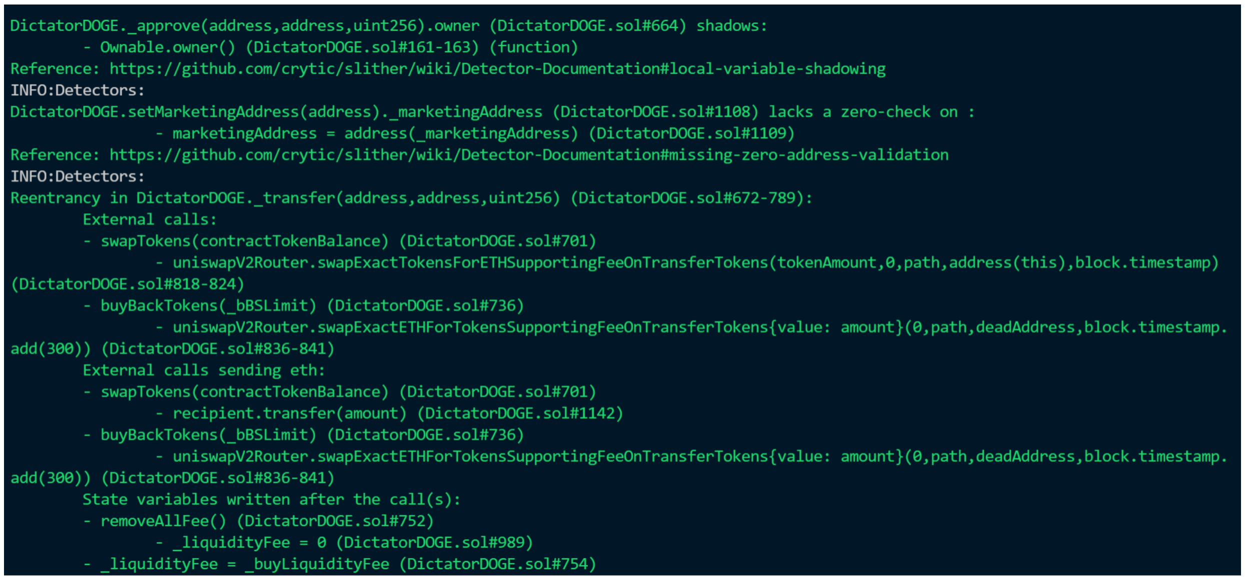
```
INFO:Detectors:
DictatorDOGE.swapETHForTokens(uint256) (DictatorDOGE.sol#829-844) sends eth to arbitrary user
        Dangerous calls:
        - uniswapV2Router.swapExactETHForTokensSupportingFeeOnTransferTokens{value: amount}(0,path,deadAddress,block.timestamp.add(300)
) (DictatorDOGE.sol#836-841)
DictatorDOGE.addLiquidity(uint256,uint256) (DictatorDOGE.sol#846-859) sends eth to arbitrary user
        Dangerous calls:
        - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (DictatorDOGE.sol#85
1-858)
DictatorDOGE.Sweep() (DictatorDOGE.sol#1169-1172) sends eth to arbitrary user
        Dangerous calls:
        - address(owner()).transfer(balance) (DictatorDOGE.sol#1171)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
DictatorDOGE._transfer(address,address,uint256) (DictatorDOGE.sol#672-789) uses a weak PRNG: "_bBSLimit = _bBSLimitMin + uint256(keccak
256(bytes)(abi.encodePacked(block.timestamp,block.difficulty))) % (_bBSLimitMax - _bBSLimitMin + 1) (DictatorDOGE.sol#733)"
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#weak-PRNG
INFO:Detectors:
Reentrancy in DictatorDOGE._transfer(address,address,uint256) (DictatorDOGE.sol#672-789):
        External calls:
        - swapTokens(contractTokenBalance) (DictatorDOGE.sol#701)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)
(DictatorDOGE.sol#818-824)
        External calls sending eth:
        - swapTokens(contractTokenBalance) (DictatorDOGE.sol#701)
```



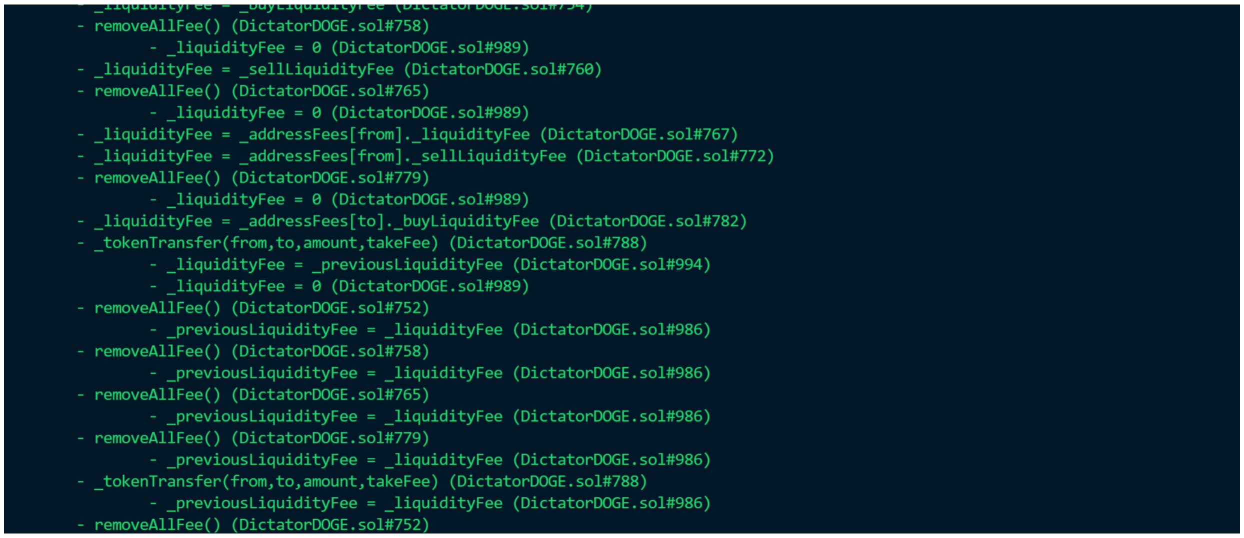
```
                - recipient.transfer(amount) (DictatorDOGE.sol#1142)
        State variables written after the call(s):
        - _removeOldSellHistories() (DictatorDOGE.sol#728)
                - _sellHistories[i].time = _sellHistories[j].time (DictatorDOGE.sol#1029)
                - _sellHistories[i].bnbAmount = _sellHistories[j].bnbAmount (DictatorDOGE.sol#1030)
                - _sellHistories.pop() (DictatorDOGE.sol#1040)
Reentrancy in DictatorDOGE._transfer(address,address,uint256) (DictatorDOGE.sol#672-789):
        External calls:
        - swapTokens(contractTokenBalance) (DictatorDOGE.sol#701)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)
(DictatorDOGE.sol#818-824)
        - buyBackTokens(_bBSLimit) (DictatorDOGE.sol#736)
                - uniswapV2Router.swapExactETHForTokensSupportingFeeOnTransferTokens{value: amount}(0,path,deadAddress,block.timestamp.
add(300)) (DictatorDOGE.sol#836-841)
        External calls sending eth:
        - swapTokens(contractTokenBalance) (DictatorDOGE.sol#701)
                - recipient.transfer(amount) (DictatorDOGE.sol#1142)
        - buyBackTokens(_bBSLimit) (DictatorDOGE.sol#736)
                - uniswapV2Router.swapExactETHForTokensSupportingFeeOnTransferTokens{value: amount}(0,path,deadAddress,block.timestamp.
add(300)) (DictatorDOGE.sol#836-841)
        State variables written after the call(s):
        - _tokenTransfer(from,to,amount,takeFee) (DictatorDOGE.sol#788)
                - _rOwned[address(this)] = _rOwned[address(this)].add(rLiquidity) (DictatorDOGE.sol#965)
                - _rOwned[sender] = _rOwned[sender].sub(rAmount) (DictatorDOGE.sol#890)
                - _rOwned[sender] = _rOwned[sender].sub(rAmount) (DictatorDOGE.sol#881)
                - _rOwned[sender] = _rOwned[sender].sub(rAmount) (DictatorDOGE.sol#911)
```



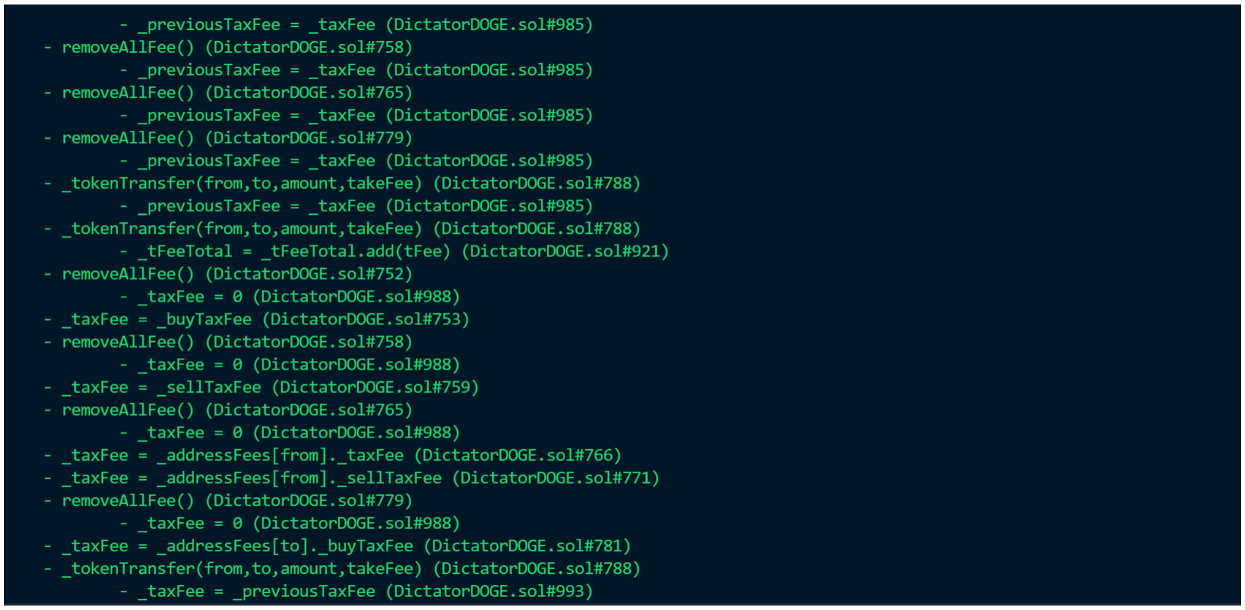
```
                - _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount) (DictatorDOGE.sol#882)
                - _rOwned[sender] = _rOwned[sender].sub(rAmount) (DictatorDOGE.sol#901)
                - _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount) (DictatorDOGE.sol#892)
                - _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount) (DictatorDOGE.sol#902)
                - _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount) (DictatorDOGE.sol#913)
        - _tokenTransfer(from,to,amount,takeFee) (DictatorDOGE.sol#788)
                - _rTotal = _rTotal.sub(rFee) (DictatorDOGE.sol#920)
        - _tokenTransfer(from,to,amount,takeFee) (DictatorDOGE.sol#788)
                - _tOwned[address(this)] = _tOwned[address(this)].add(tLiquidity) (DictatorDOGE.sol#967)
                - _tOwned[sender] = _tOwned[sender].sub(tAmount) (DictatorDOGE.sol#910)
                - _tOwned[sender] = _tOwned[sender].sub(tAmount) (DictatorDOGE.sol#900)
                - _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount) (DictatorDOGE.sol#891)
                - _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount) (DictatorDOGE.sol#912)
        - buyBackTokens(_bBSLimit) (DictatorDOGE.sol#736)
                - inSwapAndLiquify = true (DictatorDOGE.sol#516)
                - inSwapAndLiquify = false (DictatorDOGE.sol#518)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
DictatorDOGE._transfer(address,address,uint256).sellHistory (DictatorDOGE.sol#688) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
DictatorDOGE.addLiquidity(uint256,uint256) (DictatorDOGE.sol#846-859) ignores return value by uniswapV2Router.addLiquidityETH{value: et
hAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (DictatorDOGE.sol#851-858)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
DictatorDOGE.allowance(address,address).owner (DictatorDOGE.sol#569) shadows:
```



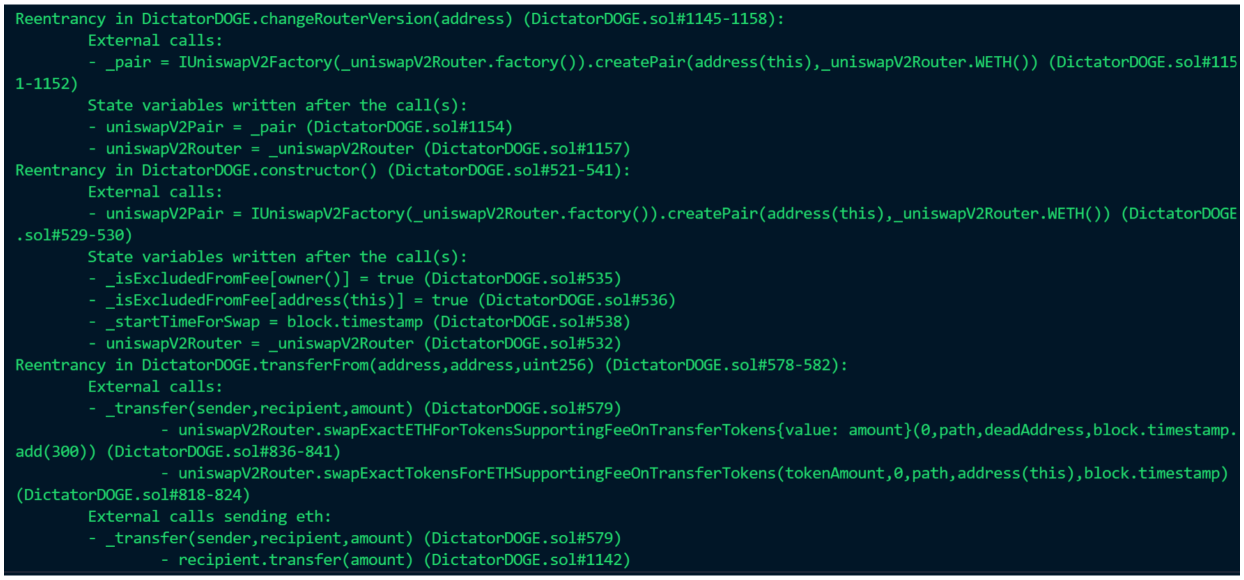
```
DictatorDOGE._approve(address,address,uint256).owner (DictatorDOGE.sol#664) shadows:
        - Ownable.owner() (DictatorDOGE.sol#161-163) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
DictatorDOGE.setMarketingAddress(address)._marketingAddress (DictatorDOGE.sol#1108) lacks a zero-check on :
                - marketingAddress = address(_marketingAddress) (DictatorDOGE.sol#1109)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in DictatorDOGE._transfer(address,address,uint256) (DictatorDOGE.sol#672-789):
        External calls:
        - swapTokens(contractTokenBalance) (DictatorDOGE.sol#701)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)
(DictatorDOGE.sol#818-824)
        - buyBackTokens(_bBSLimit) (DictatorDOGE.sol#736)
                - uniswapV2Router.swapExactETHForTokensSupportingFeeOnTransferTokens{value: amount}(0,path,deadAddress,block.timestamp.
add(300)) (DictatorDOGE.sol#836-841)
        External calls sending eth:
        - swapTokens(contractTokenBalance) (DictatorDOGE.sol#701)
                - recipient.transfer(amount) (DictatorDOGE.sol#1142)
        - buyBackTokens(_bBSLimit) (DictatorDOGE.sol#736)
                - uniswapV2Router.swapExactETHForTokensSupportingFeeOnTransferTokens{value: amount}(0,path,deadAddress,block.timestamp.
add(300)) (DictatorDOGE.sol#836-841)
        State variables written after the call(s):
        - removeAllFee() (DictatorDOGE.sol#752)
                - _liquidityFee = 0 (DictatorDOGE.sol#989)
        - _liquidityFee = _buyLiquidityFee (DictatorDOGE.sol#754)
```



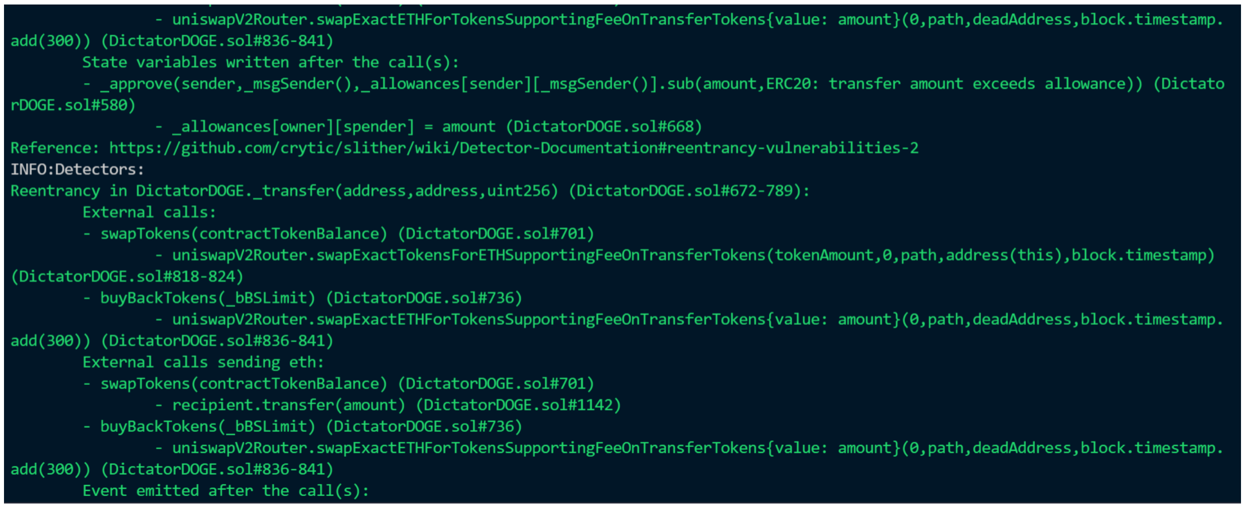
```
        - _liquidityFee = _buyLiquidityFee (DictatorDOGE.sol#754)
        - removeAllFee() (DictatorDOGE.sol#758)
                - _liquidityFee = 0 (DictatorDOGE.sol#989)
        - _liquidityFee = _sellLiquidityFee (DictatorDOGE.sol#760)
        - removeAllFee() (DictatorDOGE.sol#765)
                - _liquidityFee = 0 (DictatorDOGE.sol#989)
        - _liquidityFee = _addressFees[from]._liquidityFee (DictatorDOGE.sol#767)
        - _liquidityFee = _addressFees[from]._sellLiquidityFee (DictatorDOGE.sol#772)
        - removeAllFee() (DictatorDOGE.sol#779)
                - _liquidityFee = 0 (DictatorDOGE.sol#989)
        - _liquidityFee = _addressFees[to]._buyLiquidityFee (DictatorDOGE.sol#782)
        - _tokenTransfer(from,to,amount,takeFee) (DictatorDOGE.sol#788)
                - _liquidityFee = _previousLiquidityFee (DictatorDOGE.sol#994)
                - _liquidityFee = 0 (DictatorDOGE.sol#989)
        - removeAllFee() (DictatorDOGE.sol#752)
                - _previousLiquidityFee = _liquidityFee (DictatorDOGE.sol#986)
        - removeAllFee() (DictatorDOGE.sol#758)
                - _previousLiquidityFee = _liquidityFee (DictatorDOGE.sol#986)
        - removeAllFee() (DictatorDOGE.sol#765)
                - _previousLiquidityFee = _liquidityFee (DictatorDOGE.sol#986)
        - removeAllFee() (DictatorDOGE.sol#779)
                - _previousLiquidityFee = _liquidityFee (DictatorDOGE.sol#986)
        - _tokenTransfer(from,to,amount,takeFee) (DictatorDOGE.sol#788)
                - _previousLiquidityFee = _liquidityFee (DictatorDOGE.sol#986)
        - removeAllFee() (DictatorDOGE.sol#752)
```



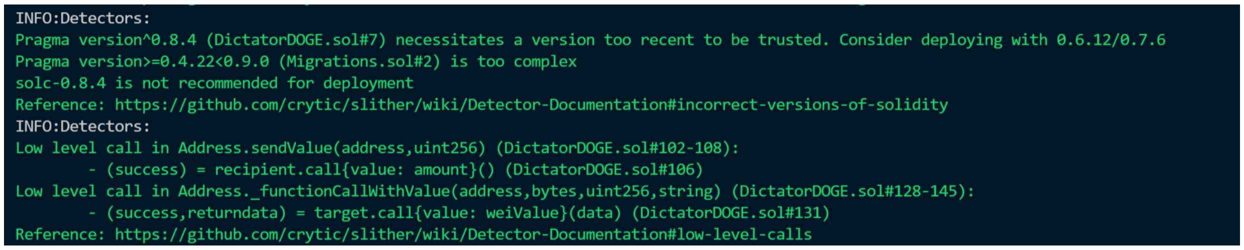
```
                - _previousTaxFee = _taxFee (DictatorDOGE.sol#985)
        - removeAllFee() (DictatorDOGE.sol#758)
                - _previousTaxFee = _taxFee (DictatorDOGE.sol#985)
        - removeAllFee() (DictatorDOGE.sol#765)
                - _previousTaxFee = _taxFee (DictatorDOGE.sol#985)
        - removeAllFee() (DictatorDOGE.sol#779)
                - _previousTaxFee = _taxFee (DictatorDOGE.sol#985)
        - _tokenTransfer(from,to,amount,takeFee) (DictatorDOGE.sol#788)
                - _previousTaxFee = _taxFee (DictatorDOGE.sol#985)
        - _tokenTransfer(from,to,amount,takeFee) (DictatorDOGE.sol#788)
                - _tFeeTotal = _tFeeTotal.add(tFee) (DictatorDOGE.sol#921)
        - removeAllFee() (DictatorDOGE.sol#752)
                - _taxFee = 0 (DictatorDOGE.sol#988)
        - _taxFee = _buyTaxFee (DictatorDOGE.sol#753)
        - removeAllFee() (DictatorDOGE.sol#758)
                - _taxFee = 0 (DictatorDOGE.sol#988)
        - _taxFee = _sellTaxFee (DictatorDOGE.sol#759)
        - removeAllFee() (DictatorDOGE.sol#765)
                - _taxFee = 0 (DictatorDOGE.sol#988)
        - _taxFee = _addressFees[from]._taxFee (DictatorDOGE.sol#766)
        - _taxFee = _addressFees[from]._sellTaxFee (DictatorDOGE.sol#771)
        - removeAllFee() (DictatorDOGE.sol#779)
                - _taxFee = 0 (DictatorDOGE.sol#988)
        - _taxFee = _addressFees[to]._buyTaxFee (DictatorDOGE.sol#781)
        - _tokenTransfer(from,to,amount,takeFee) (DictatorDOGE.sol#788)
                - _taxFee = _previousTaxFee (DictatorDOGE.sol#993)
```



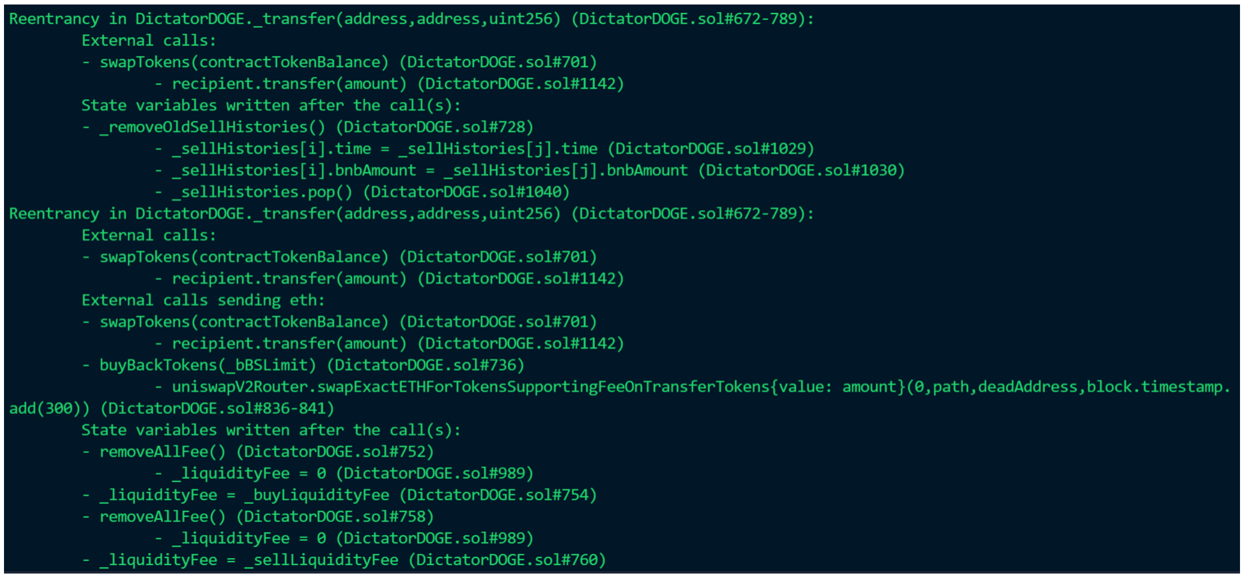
```
Reentrancy in DictatorDOGE.changeRouterVersion(address) (DictatorDOGE.sol#1145-1158):
        External calls:
        - _pair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2Router.WETH()) (DictatorDOGE.sol#115
1-1152)
        State variables written after the call(s):
        - uniswapV2Pair = _pair (DictatorDOGE.sol#1154)
        - uniswapV2Router = _uniswapV2Router (DictatorDOGE.sol#1157)
Reentrancy in DictatorDOGE.constructor() (DictatorDOGE.sol#521-541):
        External calls:
        - uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2Router.WETH()) (DictatorDOGE
.sol#529-530)
        State variables written after the call(s):
        - _isExcludedFromFee[owner()] = true (DictatorDOGE.sol#535)
        - _isExcludedFromFee[address(this)] = true (DictatorDOGE.sol#536)
        - _startTimeForSwap = block.timestamp (DictatorDOGE.sol#538)
        - uniswapV2Router = _uniswapV2Router (DictatorDOGE.sol#532)
Reentrancy in DictatorDOGE.transferFrom(address,address,uint256) (DictatorDOGE.sol#578-582):
        External calls:
        - _transfer(sender,recipient,amount) (DictatorDOGE.sol#579)
                - uniswapV2Router.swapExactETHForTokensSupportingFeeOnTransferTokens{value: amount}(0,path,deadAddress,block.timestamp.
add(300)) (DictatorDOGE.sol#836-841)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)
(DictatorDOGE.sol#818-824)
        External calls sending eth:
        - _transfer(sender,recipient,amount) (DictatorDOGE.sol#579)
                - recipient.transfer(amount) (DictatorDOGE.sol#1142)
```



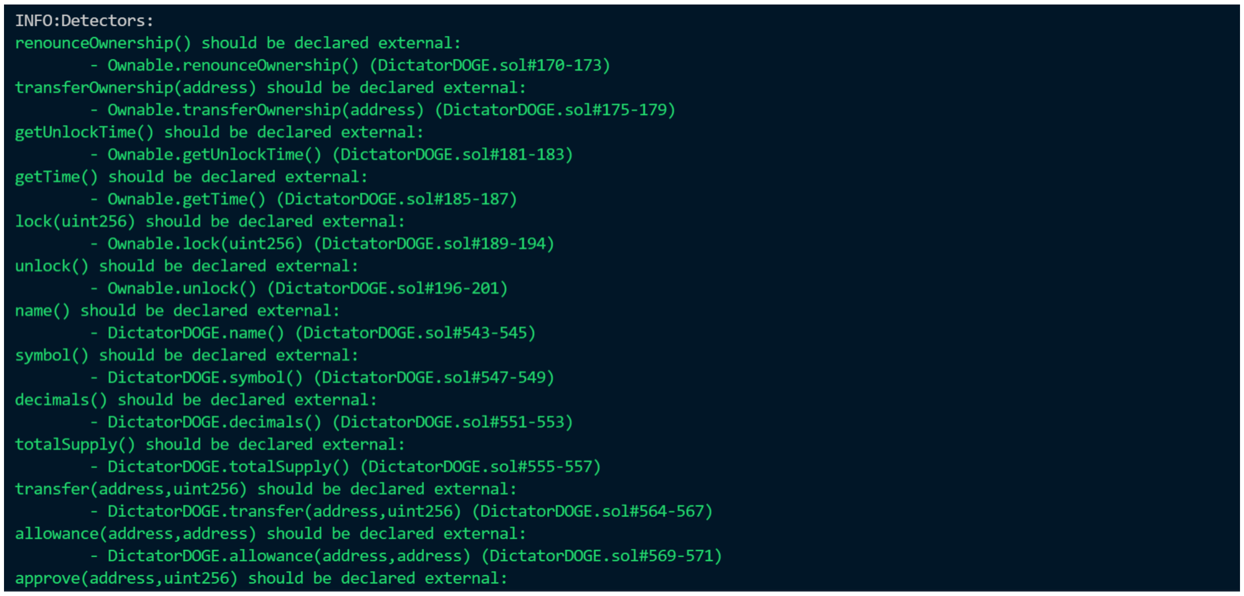
```
                - uniswapV2Router.swapExactETHForTokensSupportingFeeOnTransferTokens{value: amount}(0,path,deadAddress,block.timestamp.
add(300)) (DictatorDOGE.sol#836-841)
        State variables written after the call(s):
        - _approve(sender,_msgSender(),_allowances[sender][_msgSender()].sub(amount,ERC20: transfer amount exceeds allowance)) (Dictato
rDOGE.sol#580)
                - _allowances[owner][spender] = amount (DictatorDOGE.sol#668)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in DictatorDOGE._transfer(address,address,uint256) (DictatorDOGE.sol#672-789):
        External calls:
        - swapTokens(contractTokenBalance) (DictatorDOGE.sol#701)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)
(DictatorDOGE.sol#818-824)
        - buyBackTokens(_bBSLimit) (DictatorDOGE.sol#736)
                - uniswapV2Router.swapExactETHForTokensSupportingFeeOnTransferTokens{value: amount}(0,path,deadAddress,block.timestamp.
add(300)) (DictatorDOGE.sol#836-841)
        External calls sending eth:
        - swapTokens(contractTokenBalance) (DictatorDOGE.sol#701)
                - recipient.transfer(amount) (DictatorDOGE.sol#1142)
        - buyBackTokens(_bBSLimit) (DictatorDOGE.sol#736)
                - uniswapV2Router.swapExactETHForTokensSupportingFeeOnTransferTokens{value: amount}(0,path,deadAddress,block.timestamp.
add(300)) (DictatorDOGE.sol#836-841)
        Event emitted after the call(s):
```



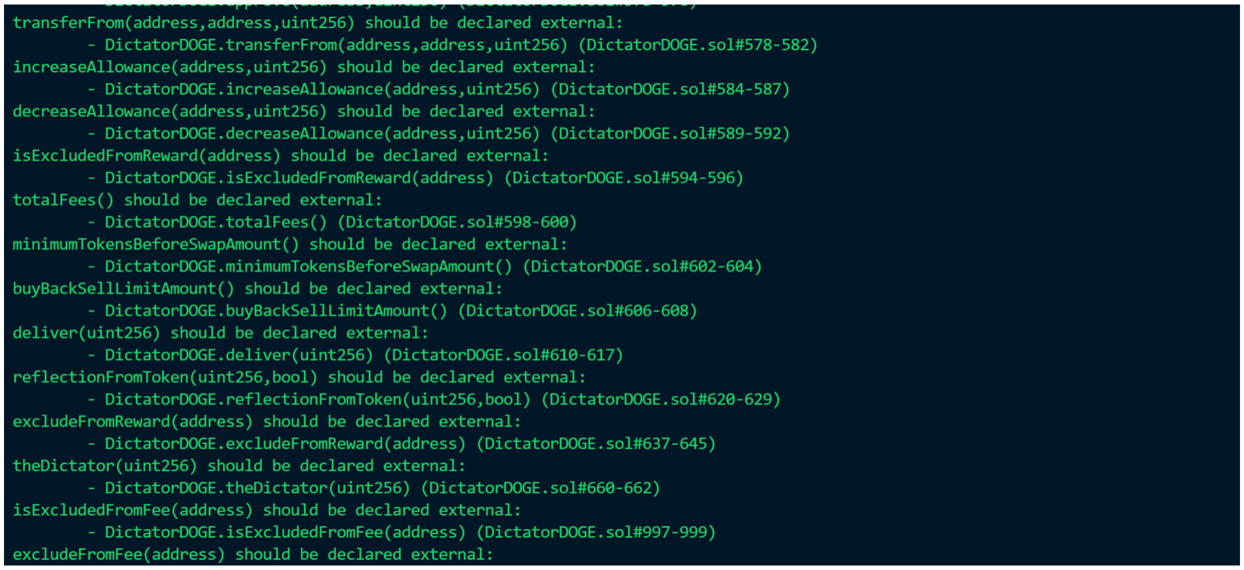
```
INFO:Detectors:
Pragma version^0.8.4 (DictatorDOGE.sol#7) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version>=0.4.22<0.9.0 (Migrations.sol#2) is too complex
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (DictatorDOGE.sol#102-108):
        - (success) = recipient.call{value: amount}() (DictatorDOGE.sol#106)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (DictatorDOGE.sol#128-145):
        - (success,returndata) = target.call{value: weiValue}(data) (DictatorDOGE.sol#131)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```



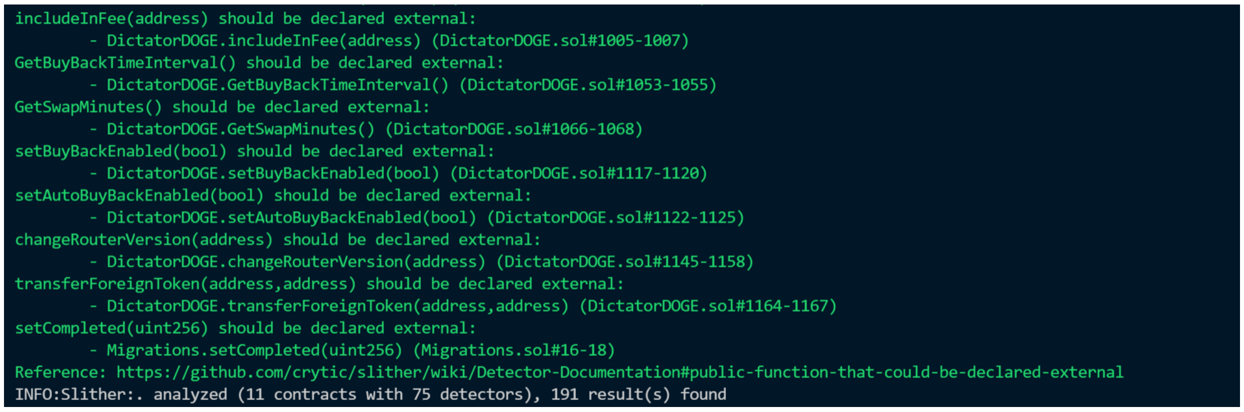
```
Reentrancy in DictatorDOGE._transfer(address,address,uint256) (DictatorDOGE.sol#672-789):
        External calls:
        - swapTokens(contractTokenBalance) (DictatorDOGE.sol#701)
                - recipient.transfer(amount) (DictatorDOGE.sol#1142)
        State variables written after the call(s):
        - _removeOldSellHistories() (DictatorDOGE.sol#728)
                - _sellHistories[i].time = _sellHistories[j].time (DictatorDOGE.sol#1029)
                - _sellHistories[i].bnbAmount = _sellHistories[j].bnbAmount (DictatorDOGE.sol#1030)
                - _sellHistories.pop() (DictatorDOGE.sol#1040)
Reentrancy in DictatorDOGE._transfer(address,address,uint256) (DictatorDOGE.sol#672-789):
        External calls:
        - swapTokens(contractTokenBalance) (DictatorDOGE.sol#701)
                - recipient.transfer(amount) (DictatorDOGE.sol#1142)
        External calls sending eth:
        - swapTokens(contractTokenBalance) (DictatorDOGE.sol#701)
                - recipient.transfer(amount) (DictatorDOGE.sol#1142)
        - buyBackTokens(_bBSLimit) (DictatorDOGE.sol#736)
                - uniswapV2Router.swapExactETHForTokensSupportingFeeOnTransferTokens{value: amount}(0,path,deadAddress,block.timestamp.
add(300)) (DictatorDOGE.sol#836-841)
        State variables written after the call(s):
        - removeAllFee() (DictatorDOGE.sol#752)
                - _liquidityFee = 0 (DictatorDOGE.sol#989)
        - _liquidityFee = _buyLiquidityFee (DictatorDOGE.sol#754)
        - removeAllFee() (DictatorDOGE.sol#758)
                - _liquidityFee = 0 (DictatorDOGE.sol#989)
        - _liquidityFee = _sellLiquidityFee (DictatorDOGE.sol#760)
```



```
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (DictatorDOGE.sol#170-173)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (DictatorDOGE.sol#175-179)
getUnlockTime() should be declared external:
        - Ownable.getUnlockTime() (DictatorDOGE.sol#181-183)
getTime() should be declared external:
        - Ownable.getTime() (DictatorDOGE.sol#185-187)
lock(uint256) should be declared external:
        - Ownable.lock(uint256) (DictatorDOGE.sol#189-194)
unlock() should be declared external:
        - Ownable.unlock() (DictatorDOGE.sol#196-201)
name() should be declared external:
        - DictatorDOGE.name() (DictatorDOGE.sol#543-545)
symbol() should be declared external:
        - DictatorDOGE.symbol() (DictatorDOGE.sol#547-549)
decimals() should be declared external:
        - DictatorDOGE.decimals() (DictatorDOGE.sol#551-553)
totalSupply() should be declared external:
        - DictatorDOGE.totalSupply() (DictatorDOGE.sol#555-557)
transfer(address,uint256) should be declared external:
        - DictatorDOGE.transfer(address,uint256) (DictatorDOGE.sol#564-567)
allowance(address,address) should be declared external:
        - DictatorDOGE.allowance(address,address) (DictatorDOGE.sol#569-571)
approve(address,uint256) should be declared external:
```



```
transferFrom(address,address,uint256) should be declared external:
        - DictatorDOGE.transferFrom(address,address,uint256) (DictatorDOGE.sol#578-582)
increaseAllowance(address,uint256) should be declared external:
        - DictatorDOGE.increaseAllowance(address,uint256) (DictatorDOGE.sol#584-587)
decreaseAllowance(address,uint256) should be declared external:
        - DictatorDOGE.decreaseAllowance(address,uint256) (DictatorDOGE.sol#589-592)
isExcludedFromReward(address) should be declared external:
        - DictatorDOGE.isExcludedFromReward(address) (DictatorDOGE.sol#594-596)
totalFees() should be declared external:
        - DictatorDOGE.totalFees() (DictatorDOGE.sol#598-600)
minimumTokensBeforeSwapAmount() should be declared external:
        - DictatorDOGE.minimumTokensBeforeSwapAmount() (DictatorDOGE.sol#602-604)
buyBackSellLimitAmount() should be declared external:
        - DictatorDOGE.buyBackSellLimitAmount() (DictatorDOGE.sol#606-608)
deliver(uint256) should be declared external:
        - DictatorDOGE.deliver(uint256) (DictatorDOGE.sol#610-617)
reflectionFromToken(uint256,bool) should be declared external:
        - DictatorDOGE.reflectionFromToken(uint256,bool) (DictatorDOGE.sol#620-629)
excludeFromReward(address) should be declared external:
        - DictatorDOGE.excludeFromReward(address) (DictatorDOGE.sol#637-645)
theDictator(uint256) should be declared external:
        - DictatorDOGE.theDictator(uint256) (DictatorDOGE.sol#660-662)
isExcludedFromFee(address) should be declared external:
        - DictatorDOGE.isExcludedFromFee(address) (DictatorDOGE.sol#997-999)
excludeFromFee(address) should be declared external:
```



```
includeInFee(address) should be declared external:
        - DictatorDOGE.includeInFee(address) (DictatorDOGE.sol#1005-1007)
GetBuyBackTimeInterval() should be declared external:
        - DictatorDOGE.GetBuyBackTimeInterval() (DictatorDOGE.sol#1053-1055)
GetSwapMinutes() should be declared external:
        - DictatorDOGE.GetSwapMinutes() (DictatorDOGE.sol#1066-1068)
setBuyBackEnabled(bool) should be declared external:
        - DictatorDOGE.setBuyBackEnabled(bool) (DictatorDOGE.sol#1117-1120)
setAutoBuyBackEnabled(bool) should be declared external:
        - DictatorDOGE.setAutoBuyBackEnabled(bool) (DictatorDOGE.sol#1122-1125)
changeRouterVersion(address) should be declared external:
        - DictatorDOGE.changeRouterVersion(address) (DictatorDOGE.sol#1145-1158)
transferForeignToken(address,address) should be declared external:
        - DictatorDOGE.transferForeignToken(address,address) (DictatorDOGE.sol#1164-1167)
setCompleted(uint256) should be declared external:
        - Migrations.setCompleted(uint256) (Migrations.sol#16-18)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:. analyzed (11 contracts with 75 detectors), 191 result(s) found
```

Results:

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines.

No instances of Integer Overflow and Underflow vulnerabilities or Back-Door Entry were found in the contract, but relying on other contracts might cause Reentrancy Vulnerability.

Numerous issues were discovered during the initial audit. In the end, the major issues were fixed or rest were acknowledged by the Auditee.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the **DictatorDOGE platform**. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the DictatorDOGE Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.