Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Customer:	DreamDoge
Website:	dreamdoge.finance
Platform:	Binance Smart Chain
Language:	Solidity
Date:	July 12th, 2021

Table of contents

Introduction

EtherAuthority was contracted by the DreamDoge team to perform the Security audit of the DreamDoge 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 12th, 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

Dogecoin (DOGE) is an open-source cryptocurrency stemming from a fork of the Litecoin codebase.

Audit scope

Name	**Code Review and Security Analysis Report for DreamDoge token Smart Contract**
Platform	**BSC / Solidity**
File	DreamDoge.sol
File MD5 Hash	515D54AF07201AB5FF74F00572ACDB7D
File SHA256 Hash	F6E5AB90F98E82188FB464895D7EE5A03B103D3497 408660F4F4B12CB8EDB5EF
Audit Date	July 12th, 2021

Claimed Smart Contract Features

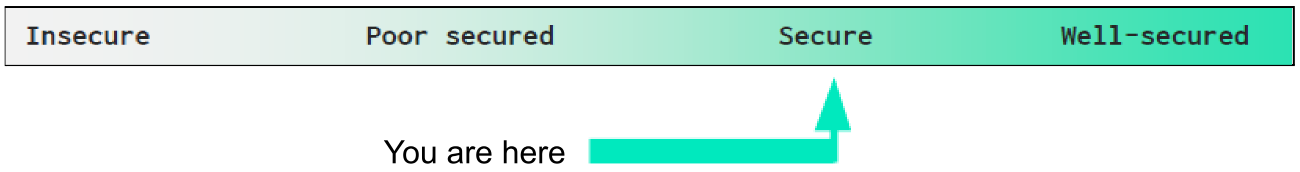
Claimed Feature Detail	Our Observation
Name: Dream Doge	**YES, This is valid.**
Symbol: DDoge	**YES, This is valid.**
Decimal: 9	**YES, This is valid.**
● swapAndLiquify: The lockTheSwap owner can access swapAndLiquify function. ● reward: The lockTheReward owner can access the reward function.	**YES, 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.**
● Handling fee: 6% for buy and 12% for sell in the first 14 days plus 14% additional handling fee, the additional handling fee decreases by 1% per day until it is 0. The additional handling fee goes to the bonus pool. ● After 14 days, the buy and sell fees are 12%. 6% consists of 1% liquidity pool, 1% marketing wallet, 2% rewards pool, and 2% of the 100 billion rewards.	**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.**
● Anti-BOT : All players who bought within a certain period of time in PCs online are blacklisted and can trade normally after being excluded from blacklist.	**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**

● Prevention of plunge: the first 14 days selling commission of 12% + 14% additional selling commission, additional selling commission decreases 1% per day (deducted from the reward pool).	**compromised, then it will create problems.**
● Small account welfare: under 10 billion position trading is not subject to any mechanism ● Reward Pool: The amount of tokens bought within every 2 hours will be accumulated, and the address that buys the most and has no selling record within these two hours will be rewarded with 100 billion tokens. ● Buy benefits: if a buy volume exceeds 30% of the current position, you can reset the last sell record and randomly generate the maximum sell volume again based on the current position. ● 100 Billion Benefit: Buy more than 100 billion at one time, you will enjoy 2% (including yourself) of the transaction fee for each subsequent transaction until the next 100 billion buyers appear. If you sell while enjoying the transaction fee, you will no longer enjoy the 2% commission (no one will enjoy it and it will be credited to the bonus pool) ● Sell limit (within 50% of position and	**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.**

less than 1 trillion): each sale will update the lock time (time randomly generated within 60 minutes), if more than two consecutive sales, the lock time is longer (randomly generated + second lock time and so on until under 10 billion), the maximum sale volume again reduced (limit the second sale 50% of the volume and so on until below 10 billion)	

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 0 critical, 0 high, 0 medium and 4 low and some very low level issues.

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	Moderated
	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	Moderated
	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

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 DreamDoge 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 DreamDoge token.

The DreamDoge 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 DreamDoge token smart contracts code in the form of a hash code MD5 and SHA256 format. 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 https://dreamdoge.finance 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

DreamDoge.sol

(1) Interface

(a) IERC20

(b) IUniswapV2Factory

(c) IUniswapV2Pair

(d) IUniswapV2Router01

(e) IUniswapV2Router02

(2) Inherited contracts

(a) Context

(b) Ownable

(c) IERC20

(3) Struct

(a) TaxRate

(b) TaxAmount

(c) Threshold

(d) Range

(4) Usages

(a) using SafeMath for uint256;

(b) using Address for address;

(5) Events

(a) event SwapAndLiquifyEnabledUpdated(bool enabled);

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

(c) event LuckyWinner(uint256 timestamp,uint256 round,uint256 totalReward,address luckyWinner);

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

(6) Functions

Sl.	Functions	Type	Observation	Conclusion
1	lockTheSwap	modifier	Passed	No Issue
2	lockTheReward	modifier	Passed	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	read	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	excludeFromFee	write	access only Owner	No Issue
15	includeInFee	write	access only Owner	No Issue
16	excludeBlacklist	write	access only Owner	No Issue
17	includeBlacklist	write	access only Owner	No Issue
18	excludeWhitelist	write	access only Owner	No Issue
19	includeWhitelist	write	access only Owner	No Issue
20	setSwapAndLiquifyEnabled	write	access only Owner	No Issue
21	setRewardEnabled	write	access only Owner	No Issue
22	setMaxLiquidity	write	access only Owner	No Issue
23	setTradeReward	write	access only Owner	No Issue
24	setMinBalance	write	access only Owner	No Issue
25	setTimespam	write	access only Owner	No Issue
26	setUpdateRatio	write	access only Owner	No Issue
27	setSalesRatio	write	access only Owner	No Issue
28	setRewardCycle	write	access only Owner	No Issue
29	setBigBuyerReward	write	access only Owner	No Issue

30	setBlacklistThreshold	write	access only Owner	No Issue
31	setDeadline	write	access only Owner	No Issue
32	extraFee	read	Passed	No Issue
33	getTimeAndMaxSellTxAmount	read	Passed	No Issue
34	isExcludedFromFee	read	Passed	No Issue
35	isInBlacklist	read	Passed	No Issue
36	isInWhiteList	read	Passed	No Issue
37	buyerLength	read	Passed	No Issue
38	_approve	write	Passed	No Issue
39	_transfer	write	Passed	No Issue
40	random	read	Passed	No Issue
41	potentialWinner	read	Passed	No Issue
42	reward	write	Passed	No Issue
43	update	write	Passed	No Issue
44	punish	write	Passed	No Issue
45	_getTValues	write	Reading state functions declared without pure or view	Refer Audit Findings
46	calcuteFees	write	Reading state functions declared without pure or view	Refer Audit Findings
47	calcuteTaxAmount	write	Reading state functions declared without pure or view	Refer Audit Findings
48	_transferStandard	write	Passed	No Issue
49	swapAndLiquify	write	Passed	No Issue
50	swapTokensForEth	write	Passed	No Issue
51	addLiquidity	write	Centralized risk in addLiquidity	Refer Audit Findings
52	owner	read	Passed	No Issue
53	onlyOwner	modifier	Passed	No Issue
54	renounceOwnership	write	Possible to gain ownership	Refer Audit Findings
55	transferOwnership	write	access only Owner	No Issue
56	geUnlockTime	read	Passed	No Issue
57	lock	write	Possible to gain ownership	Refer Audit Findings
58	unlock	write	Possible to gain ownership	Refer Audit Findings
59	_msgSender	internal	Passed	No Issue
60	_msgData	internal	Passed	No Issue
61	receive	external	Typing mistake in contract	Refer Audit Findings

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

No critical severity vulnerabilities were found

High

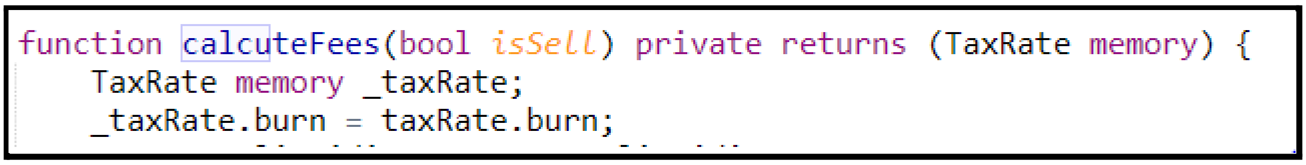
No high severity vulnerabilities were found.

Medium

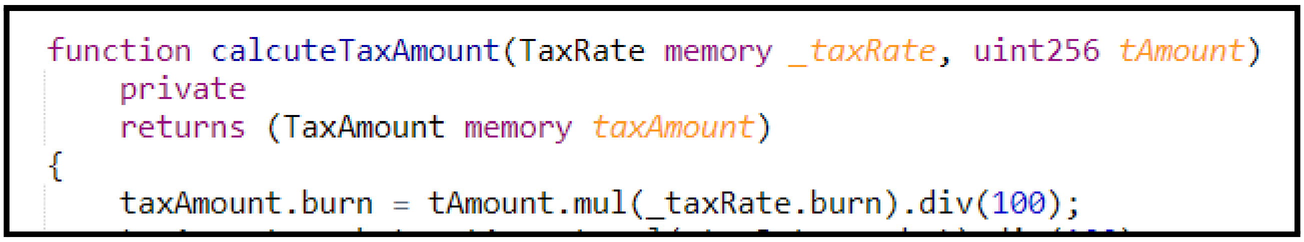
No Medium severity vulnerabilities were found.

Low

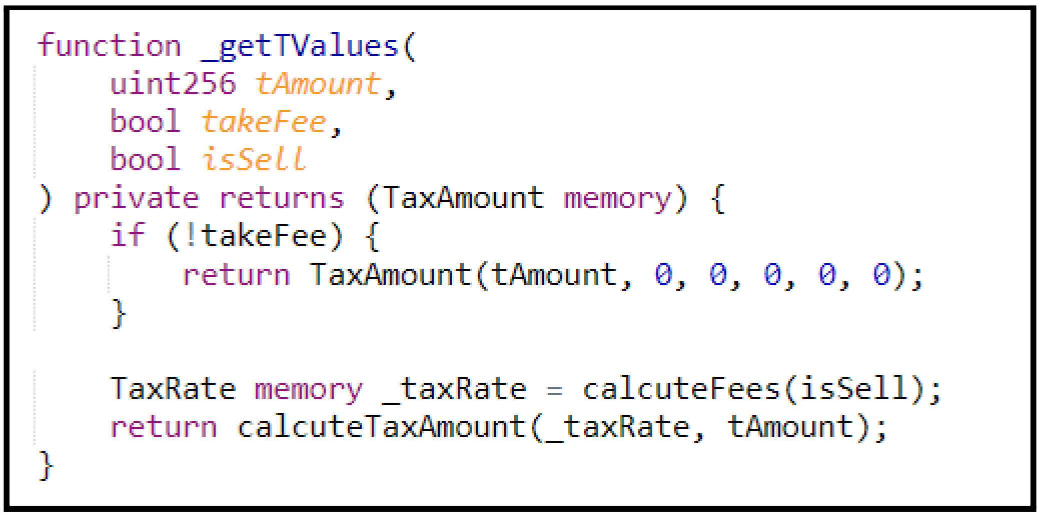
(1) Reading state functions declared without pure or view:



```
function calcuteFees(bool isSell) private returns (TaxRate memory) {
    TaxRate memory _taxRate;
    _taxRate.burn = taxRate.burn;
```



```
function calcuteTaxAmount(TaxRate memory _taxRate, uint256 tAmount)
    private
    returns (TaxAmount memory taxAmount)
{
    taxAmount.burn = tAmount.mul(_taxRate.burn).div(100);
```



```
function _getTValues(
    uint256 tAmount,
    bool takeFee,
    bool isSell
) private returns (TaxAmount memory) {
    if (!takeFee) {
        return TaxAmount(tAmount, 0, 0, 0, 0, 0);
    }

    TaxRate memory _taxRate = calcuteFees(isSell);
    return calcuteTaxAmount(_taxRate, tAmount);
}
```

Add pure or view in functions which return only values and no state changes.

Resolution: Line no 1497 calculate Fees add view in this function,

Line no 1516 calcutTa Amount add pure in this function

Line no 1484 _getTValues add view in this function.

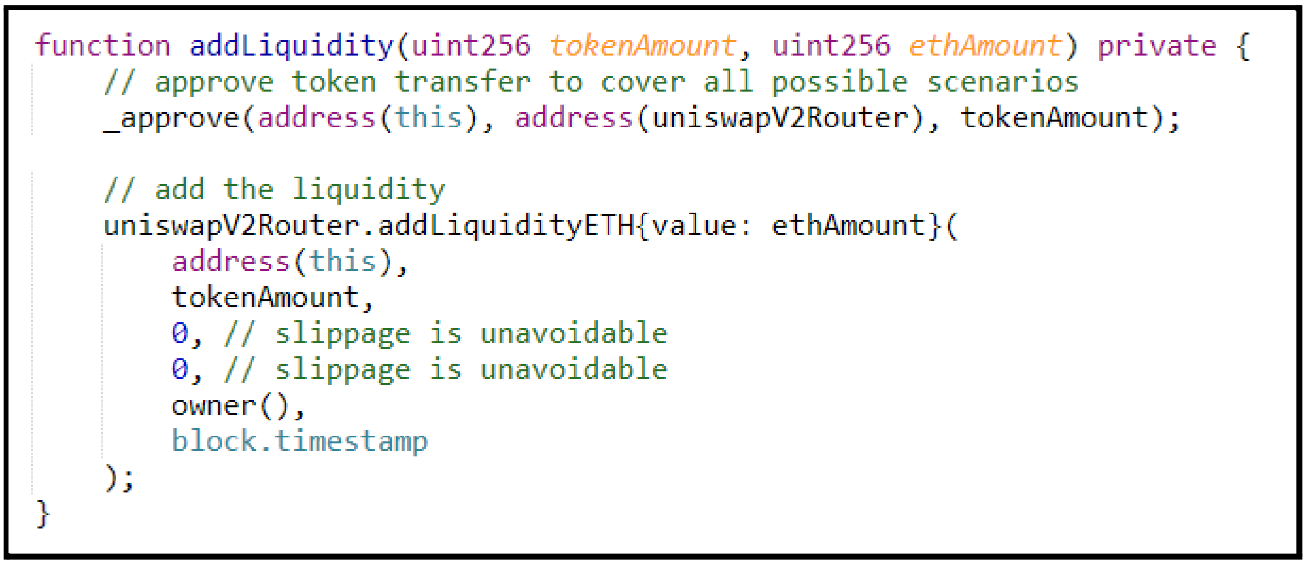
(2) Possible to gain ownership:

Possible to gain ownership after renouncing the contract 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.

(3) Centralized risk in addLiquidity:



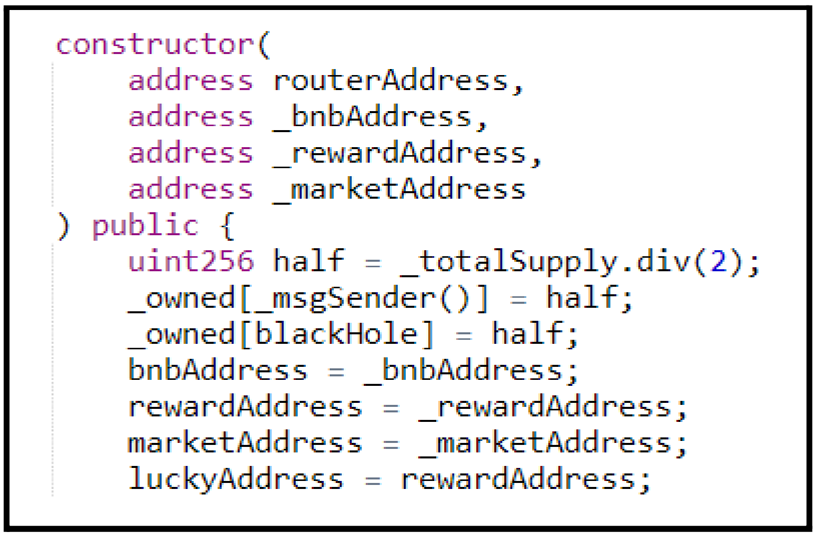
```
function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
    // approve token transfer to cover all possible scenarios
    _approve(address(this), address(uniswapV2Router), tokenAmount);

    // add the liquidity
    uniswapV2Router.addLiquidityETH{value: ethAmount}(
        address(this),
        tokenAmount,
        0, // slippage is unavoidable
        0, // slippage is unavoidable
        owner(),
        block.timestamp
    );
}
```

addLiquidityETH function has to be addressed as owner() to get Tokens from the Pool. At some time the owner will accumulate significant tokens. If the _owner is an EOA(Externally Owned Account), mishandling of its private key can have devastating consequences to the project.

Resolution: addLiquidityETH function call to be replaced by address(this),and to restrict the management of the tokens.This will protect the tokens From being stolen if the _owner account is compromised.

(4) Missing zero address validation:



```
constructor(
    address routerAddress,
    address _bnbAddress,
    address _rewardAddress,
    address _marketAddress
) public {
    uint256 half = _totalSupply.div(2);
    _owned[_msgSender()] = half;
    _owned[blackHole] = half;
    bnbAddress = _bnbAddress;
    rewardAddress = _rewardAddress;
    marketAddress = _marketAddress;
    luckyAddress = rewardAddress;
```

Detecting missing zero address validation to prevent contract losses, in constructor check the address is not zero _bnbAddress, _rewardAddress and _marketAddress.

Resolution: Check that the address is not zero.

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.

(2) State variables that could be declared constant:

```
uint256 private _totalSupply = 1000 * 10**12 * 10**9;
string private _name = "Dream Doge";
string private _symbol = "DDoge";
uint8 private _decimals = 9;
```

Constant state variables should be declared constant to save gas.

Resolution: Add the constant attributes to state variables that never change.

(3) Typing mistake in contract:

```
function calcuteFees(bool isSell) private view returns (TaxRate memory) {
    TaxRate memory _taxRate;
```

```
function calcuteTaxAmount(TaxRate memory _taxRate, uint256 tAmount)
    private
    returns (TaxAmount memory taxAmount)
{
```

```
//to recieve ETH from uniswapV2Router when swaping
receive() external payable {}
```

There are many typing mistakes in code and comments.

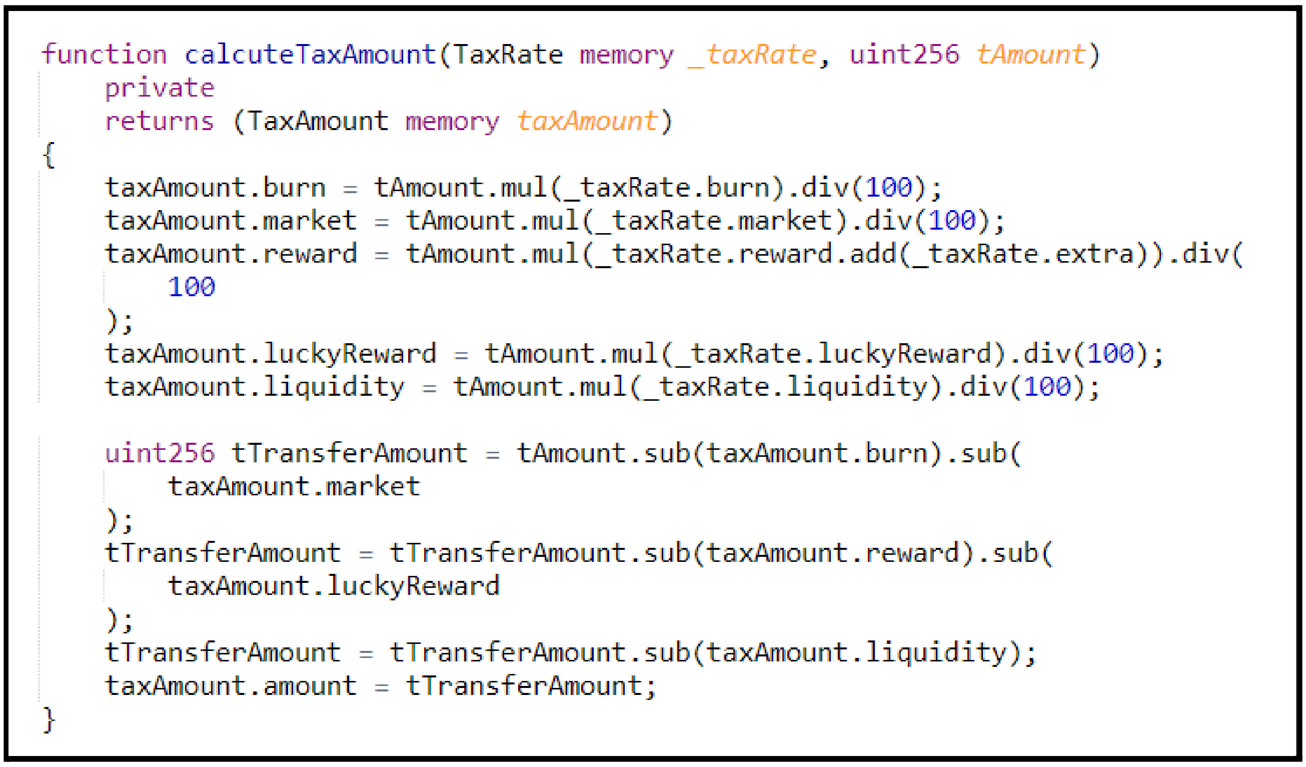
Resolution: Line no 1276 //to receive ETH from uniswapV2Router when swapping

Correct spellings of receive and swapping

Line no 1479 calcuteFees function name change to calculateFees

Line no 1516 calcuteTaxAmount function name change to calculateTaxAmount

(4) Ignored return value:



```
function calcuteTaxAmount(TaxRate memory _taxRate, uint256 tAmount)
    private
    returns (TaxAmount memory taxAmount)
{
    taxAmount.burn = tAmount.mul(_taxRate.burn).div(100);
    taxAmount.market = tAmount.mul(_taxRate.market).div(100);
    taxAmount.reward = tAmount.mul(_taxRate.reward.add(_taxRate.extra)).div(
        100
    );
    taxAmount.luckyReward = tAmount.mul(_taxRate.luckyReward).div(100);
    taxAmount.liquidity = tAmount.mul(_taxRate.liquidity).div(100);

    uint256 tTransferAmount = tAmount.sub(taxAmount.burn).sub(
        taxAmount.market
    );
    tTransferAmount = tTransferAmount.sub(taxAmount.reward).sub(
        taxAmount.luckyReward
    );
    tTransferAmount = tTransferAmount.sub(taxAmount.liquidity);
    taxAmount.amount = tTransferAmount;
}
```

calcuteTaxAmount is not a void-returning function. Ignoring the return value might cause some unexpected exception, especially if the callee function doesn't revert automatically when falling.

Resolution: We recommend checking the output of calcuteTaxAmount before continuing processing.

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.
- excludeBlacklist: The Owner can check excluded blacklist.
- includeBlacklist: The Owner can check the included blacklist.
- excludeWhitelist: The Owner can check excluded whitelist.
- includeWhitelist: The Owner can check the included whitelist.
- setRewardEnabled: The Owner can set reward enabled stats.
- setSwapAndLiquifyEnabled: The Owner can set swap and Liquify enable status.
- setMaxLiquidity: The Owner can set maxLiquidity.
- setTradeReward: The Owner can set tradeRewards.
- setMinBalance: The Owner can set Minimum Balance.
- setTimespam: The Owner can set a timestamp.
- setUpdateRatio: The Owner can set the update ratio.
- setSalesRatio: The Owner can set the sales ratio.
- setRewardCycle: The Owner can set a reward cycle.
- setBigBuyerReward: The Owner can set a big buyer reward.
- setBlacklistThreshold: The Owner can set a blacklistThreshold.
- setDeadline: The Owner can set Deadline.

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 are fixed/acknowledged in the smart contracts. **So it is good to go for the 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).

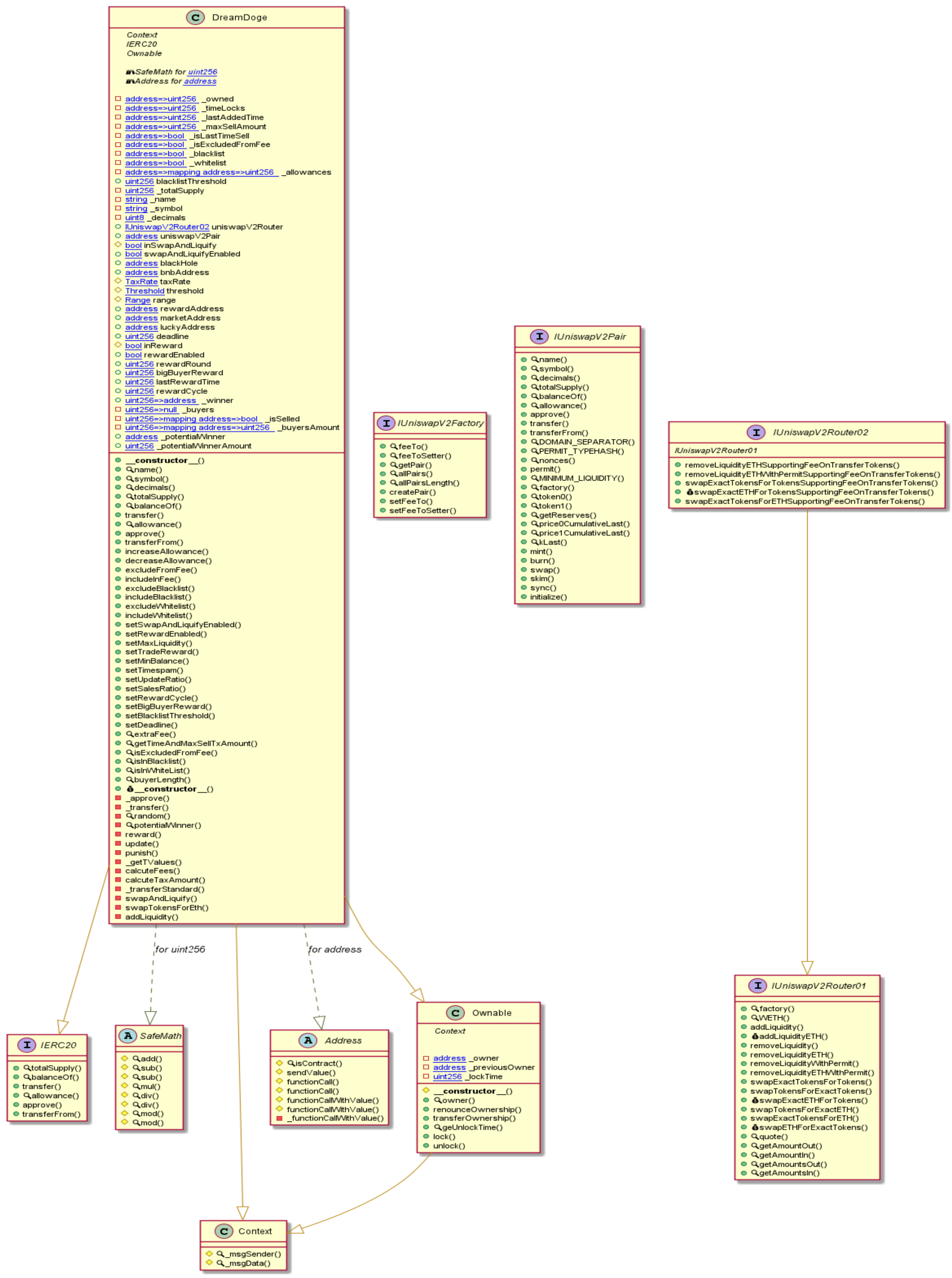
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we 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 - DreamDogeToken



Slither Results Log

Slither log >> DreamDoge.sol

```
INFO:Detectors:
DreamDoge.addLiquidity(uint256,uint256) (DreamDoge.sol#1583-1596) sends eth to arbitrary user
	Dangerous calls:
	- uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (DreamDoge.sol#1588-1595)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinatio
ns
INFO:Detectors:
DreamDoge.random(uint256) (DreamDoge.sol#1402-1407) uses a weak PRNG:
"uint256(keccak256(bytes)(abi.encodePacked(now,block.difficulty,msg.sender))) % _length
(DreamDoge.sol#1403-1406)"
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#weak-PRNG
INFO:Detectors:
Reentrancy in DreamDoge._transfer(address,address,uint256) (DreamDoge.sol#1275-1400):
	External calls:
	- swapAndLiquify(contractTokenBalance) (DreamDoge.sol#1326)
		- uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (DreamDoge.sol#1588-1595)
		-
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(t
his),block.timestamp) (DreamDoge.sol#1574-1580)
	External calls sending eth:
	- swapAndLiquify(contractTokenBalance) (DreamDoge.sol#1326)
		- uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (DreamDoge.sol#1588-1595)
	State variables written after the call(s):
	- _blacklist[to] = true (DreamDoge.sol#1381)
	- update(to,amount.add(balanceTo)) (DreamDoge.sol#1366)
		- _maxSellAmount[account] = threshold.maxTxAmount (DreamDoge.sol#1449-1451)
		- _maxSellAmount[account] = maxSellAmount (DreamDoge.sol#1449-1451)
	- update(to,amount) (DreamDoge.sol#1377)
		- _maxSellAmount[account] = threshold.maxTxAmount (DreamDoge.sol#1449-1451)
		- _maxSellAmount[account] = maxSellAmount (DreamDoge.sol#1449-1451)
	- update(from,balanceOf(from)) (DreamDoge.sol#1388)
		- _maxSellAmount[account] = threshold.maxTxAmount (DreamDoge.sol#1449-1451)
		- _maxSellAmount[account] = maxSellAmount (DreamDoge.sol#1449-1451)
	- punish(from) (DreamDoge.sol#1390)
		- _maxSellAmount[account] = _maxSellAmount[account].div(2) (DreamDoge.sol#1458)
	- reward() (DreamDoge.sol#1337)
		- _owned[rewardAddress] = _owned[rewardAddress].sub(totalReward) (DreamDoge.sol#1439)
		- _owned[luckyWinner] = _owned[luckyWinner].add(totalReward) (DreamDoge.sol#1440)
	- _transferStandard(from,to,amount,takeFee,isSell) (DreamDoge.sol#1357)
		- _owned[sender] = _owned[sender].sub(tAmount) (DreamDoge.sol#1523)
		- _owned[recipient] = _owned[recipient].add(taxAmount.amount) (DreamDoge.sol#1524)
		- _owned[marketAddress] = _owned[marketAddress].add(taxAmount.market)
(DreamDoge.sol#1527)
		- _owned[address(this)] = _owned[address(this)].add(taxAmount.liquidity)
(DreamDoge.sol#1528-1530)
		- _owned[luckyAddress] = _owned[luckyAddress].add(taxAmount.luckyReward)
(DreamDoge.sol#1531-1533)
		- _owned[rewardAddress] = _owned[rewardAddress].add(taxAmount.reward)
(DreamDoge.sol#1534)
		- _owned[blackHole] = _owned[blackHole].add(taxAmount.burn) (DreamDoge.sol#1536)
	- update(to,amount.add(balanceTo)) (DreamDoge.sol#1366)
```

```
                - _timeLocks[account] = addedTime.add(block.timestamp) (DreamDoge.sol#1446)
        - update(to,amount) (DreamDoge.sol#1377)
                - _timeLocks[account] = addedTime.add(block.timestamp) (DreamDoge.sol#1446)
        - update(from,balanceOf(from)) (DreamDoge.sol#1388)
                - _timeLocks[account] = addedTime.add(block.timestamp) (DreamDoge.sol#1446)
        - punish(from) (DreamDoge.sol#1390)
                - _timeLocks[account] = _lastAddedTime[account].add(block.timestamp) (DreamDoge.sol#1457)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
DreamDoge.calcuteFees(bool)._taxRate (DreamDoge.sol#1475) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
DreamDoge.addLiquidity(uint256,uint256) (DreamDoge.sol#1583-1596) ignores return value by
uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (DreamDoge.sol#1588-1595)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
DreamDoge.allowance(address,address).owner (DreamDoge.sol#1086) shadows:
        - Ownable.owner() (DreamDoge.sol#488-490) (function)
DreamDoge._approve(address,address,uint256).owner (DreamDoge.sol#1264) shadows:
        - Ownable.owner() (DreamDoge.sol#488-490) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
DreamDoge.constructor(address,address,address,address)._bnbAddress (DreamDoge.sol#1022) lacks a
zero-check on :
                - bnbAddress = _bnbAddress (DreamDoge.sol#1029)
DreamDoge.constructor(address,address,address,address)._rewardAddress (DreamDoge.sol#1023) lacks a
zero-check on :
                - rewardAddress = _rewardAddress (DreamDoge.sol#1030)
DreamDoge.constructor(address,address,address,address)._marketAddress (DreamDoge.sol#1024) lacks a
zero-check on :
                - marketAddress = _marketAddress (DreamDoge.sol#1031)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in DreamDoge._transfer(address,address,uint256) (DreamDoge.sol#1275-1400):
        External calls:
        - swapAndLiquify(contractTokenBalance) (DreamDoge.sol#1326)
                - uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (DreamDoge.sol#1588-1595)
                -
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(t
his),block.timestamp) (DreamDoge.sol#1574-1580)
        External calls sending eth:
        - swapAndLiquify(contractTokenBalance) (DreamDoge.sol#1326)
                - uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (DreamDoge.sol#1588-1595)
        State variables written after the call(s):
        - _buyers[currentRound].push(to) (DreamDoge.sol#1369)
        - _buyersAmount[currentRound][to] = _buyersAmount[currentRound][to] + amount
(DreamDoge.sol#1372-1374)
        - _isLastTimeSell[to] = false (DreamDoge.sol#1367)
        - _isLastTimeSell[from] = true (DreamDoge.sol#1394)
        - _isSelled[currentRound][from] = true (DreamDoge.sol#1393)
        - update(to,amount.add(balanceTo)) (DreamDoge.sol#1366)
                - _lastAddedTime[account] = addedTime (DreamDoge.sol#1447)
        - update(to,amount) (DreamDoge.sol#1377)
                - _lastAddedTime[account] = addedTime (DreamDoge.sol#1447)
        - update(from,balanceOf(from)) (DreamDoge.sol#1388)
                - _lastAddedTime[account] = addedTime (DreamDoge.sol#1447)
        - punish(from) (DreamDoge.sol#1390)
                - _lastAddedTime[account] = addedTime.add(_lastAddedTime[account]) (DreamDoge.sol#1456)
        - (_potentialWinner,_potentialWinnerAmount) = potentialWinner(currentRound)
(DreamDoge.sol#1397-1399)
        - (_potentialWinner,_potentialWinnerAmount) = potentialWinner(currentRound)
(DreamDoge.sol#1397-1399)
```

```
        - reward() (DreamDoge.sol#1337)
                - _winner[lastRound] = luckyWinner (DreamDoge.sol#1438)
        - reward() (DreamDoge.sol#1337)
                - inReward = true (DreamDoge.sol#1013)
                - inReward = false (DreamDoge.sol#1015)
        - lastRewardTime = currentTimestamp (DreamDoge.sol#1336)
        - luckyAddress = to (DreamDoge.sol#1348)
        - luckyAddress = rewardAddress (DreamDoge.sol#1352)
        - reward() (DreamDoge.sol#1337)
                - rewardRound = lastRound + 1 (DreamDoge.sol#1431)
Reentrancy in DreamDoge.constructor(address,address,address,address) (DreamDoge.sol#1020-1055):
        External calls:
        - uniswapV2Pair =
IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),bnbAddress)
(DreamDoge.sol#1036-1037)
        State variables written after the call(s):
        - _isExcludedFromFee[owner()] = true (DreamDoge.sol#1043)
        - _isExcludedFromFee[address(this)] = true (DreamDoge.sol#1044)
        - _isExcludedFromFee[rewardAddress] = true (DreamDoge.sol#1045)
        - _isExcludedFromFee[marketAddress] = true (DreamDoge.sol#1046)
        - _whitelist[owner()] = true (DreamDoge.sol#1048)
        - _whitelist[address(this)] = true (DreamDoge.sol#1049)
        - _whitelist[rewardAddress] = true (DreamDoge.sol#1050)
        - _whitelist[marketAddress] = true (DreamDoge.sol#1051)
        - uniswapV2Router = _uniswapV2Router (DreamDoge.sol#1040)
Reentrancy in DreamDoge.swapAndLiquify(uint256) (DreamDoge.sol#1542-1563):
        External calls:
        - swapTokensForEth(half) (DreamDoge.sol#1554)
                -
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(t
his),block.timestamp) (DreamDoge.sol#1574-1580)
        - addLiquidity(otherHalf,newBalance) (DreamDoge.sol#1560)
                - uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (DreamDoge.sol#1588-1595)
        External calls sending eth:
        - addLiquidity(otherHalf,newBalance) (DreamDoge.sol#1560)
                - uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (DreamDoge.sol#1588-1595)
        State variables written after the call(s):
        - addLiquidity(otherHalf,newBalance) (DreamDoge.sol#1560)
                - _allowances[owner][spender] = amount (DreamDoge.sol#1271)
Reentrancy in DreamDoge.transferFrom(address,address,uint256) (DreamDoge.sol#1104-1119):
        External calls:
        - _transfer(sender,recipient,amount) (DreamDoge.sol#1109)
                - uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (DreamDoge.sol#1588-1595)
                -
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(t
his),block.timestamp) (DreamDoge.sol#1574-1580)
        External calls sending eth:
        - _transfer(sender,recipient,amount) (DreamDoge.sol#1109)
                - uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (DreamDoge.sol#1588-1595)
        State variables written after the call(s):
        - _approve(sender,_msgSender(),_allowances[sender][_msgSender()].sub(amount,ERC20: transfer
amount exceeds allowance)) (DreamDoge.sol#1110-1117)
                - _allowances[owner][spender] = amount (DreamDoge.sol#1271)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in DreamDoge._transfer(address,address,uint256) (DreamDoge.sol#1275-1400):
        External calls:
        - swapAndLiquify(contractTokenBalance) (DreamDoge.sol#1326)
                - uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (DreamDoge.sol#1588-1595)
```

```
		-
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(t
his),block.timestamp) (DreamDoge.sol#1574-1580)
	External calls sending eth:
	- swapAndLiquify(contractTokenBalance) (DreamDoge.sol#1326)
		- uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (DreamDoge.sol#1588-1595)
	Event emitted after the call(s):
	- LuckyWinner(block.timestamp,lastRound,totalReward,luckyWinner) (DreamDoge.sol#1441)
		- reward() (DreamDoge.sol#1337)
	- Transfer(sender,recipient,taxAmount.amount) (DreamDoge.sol#1539)
		- _transferStandard(from,to,amount,takeFee,isSell) (DreamDoge.sol#1357)
Reentrancy in DreamDoge.constructor(address,address,address,address) (DreamDoge.sol#1020-1055):
	External calls:
	- uniswapV2Pair =
IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),bnbAddress)
(DreamDoge.sol#1036-1037)
	Event emitted after the call(s):
	- Transfer(address(0),_msgSender(),half) (DreamDoge.sol#1053)
	- Transfer(address(0),blackHole,half) (DreamDoge.sol#1054)
Reentrancy in DreamDoge.swapAndLiquify(uint256) (DreamDoge.sol#1542-1563):
	External calls:
	- swapTokensForEth(half) (DreamDoge.sol#1554)
		-
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(t
his),block.timestamp) (DreamDoge.sol#1574-1580)
	- addLiquidity(otherHalf,newBalance) (DreamDoge.sol#1560)
		- uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (DreamDoge.sol#1588-1595)
	External calls sending eth:
	- addLiquidity(otherHalf,newBalance) (DreamDoge.sol#1560)
		- uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (DreamDoge.sol#1588-1595)
	Event emitted after the call(s):
	- Approval(owner,spender,amount) (DreamDoge.sol#1272)
		- addLiquidity(otherHalf,newBalance) (DreamDoge.sol#1560)
	- SwapAndLiquify(half,newBalance,otherHalf) (DreamDoge.sol#1562)
Reentrancy in DreamDoge.transferFrom(address,address,uint256) (DreamDoge.sol#1104-1119):
	External calls:
	- _transfer(sender,recipient,amount) (DreamDoge.sol#1109)
		- uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (DreamDoge.sol#1588-1595)
		-
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(t
his),block.timestamp) (DreamDoge.sol#1574-1580)
	External calls sending eth:
	- _transfer(sender,recipient,amount) (DreamDoge.sol#1109)
		- uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (DreamDoge.sol#1588-1595)
	Event emitted after the call(s):
	- Approval(owner,spender,amount) (DreamDoge.sol#1272)
		- _approve(sender,_msgSender(),_allowances[sender][_msgSender()].sub(amount,ERC20:
transfer amount exceeds allowance)) (DreamDoge.sol#1110-1117)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Ownable.unlock() (DreamDoge.sol#538-546) uses timestamp for comparisons
	Dangerous comparisons:
	- require(bool,string)(now > _lockTime,Contract is locked until 7 days) (DreamDoge.sol#543)
DreamDoge.extraFee() (DreamDoge.sol#1229-1234) uses timestamp for comparisons
	Dangerous comparisons:
	- deadline >= block.timestamp (DreamDoge.sol#1230-1233)
DreamDoge._transfer(address,address,uint256) (DreamDoge.sol#1275-1400) uses timestamp for
comparisons
	Dangerous comparisons:
        - require(bool,string)(block.timestamp >= _timeLocks[from],Untradeable in this time period)
(DreamDoge.sol#1297-1300)
        - require(bool,string)(amount <= _maxSellAmount[from],Transfer amount exceeds the maxTxAmount.)
(DreamDoge.sol#1303-1306)
        - (currentTimestamp - lastRewardTime) > rewardCycle && ! inReward && rewardEnabled
(DreamDoge.sol#1332-1334)
        - currentTimestamp <= blacklistThreshold (DreamDoge.sol#1380)
DreamDoge.update(address,uint256) (DreamDoge.sol#1444-1452) uses timestamp for comparisons
        Dangerous comparisons:
        - maxSellAmount > threshold.maxTxAmount (DreamDoge.sol#1449-1451)
DreamDoge.calcuteFees(bool) (DreamDoge.sol#1474-1491) uses timestamp for comparisons
        Dangerous comparisons:
        - deadline >= block.timestamp (DreamDoge.sol#1484)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (DreamDoge.sol#294-308) uses assembly
        - INLINE ASM (DreamDoge.sol#304-306)
Address._functionCallWithValue(address,bytes,uint256,string) (DreamDoge.sol#423-451) uses assembly
        - INLINE ASM (DreamDoge.sol#443-446)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Address._functionCallWithValue(address,bytes,uint256,string) (DreamDoge.sol#423-451) is never used and
should be removed
Address.functionCall(address,bytes) (DreamDoge.sol#358-363) is never used and should be removed
Address.functionCall(address,bytes,string) (DreamDoge.sol#371-377) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (DreamDoge.sol#390-402) is never used and should
be removed
Address.functionCallWithValue(address,bytes,uint256,string) (DreamDoge.sol#410-421) is never used and
should be removed
Address.isContract(address) (DreamDoge.sol#294-308) is never used and should be removed
Address.sendValue(address,uint256) (DreamDoge.sol#326-338) is never used and should be removed
Context._msgData() (DreamDoge.sol#267-270) is never used and should be removed
SafeMath.mod(uint256,uint256) (DreamDoge.sol#236-238) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (DreamDoge.sol#252-259) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (DreamDoge.sol#326-338):
        - (success) = recipient.call{value: amount}() (DreamDoge.sol#333)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (DreamDoge.sol#423-451):
        - (success,returndata) = target.call{value: weiValue}(data) (DreamDoge.sol#432-434)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Function IUniswapV2Pair.DOMAIN_SEPARATOR() (DreamDoge.sol#616) is not in mixedCase
Function IUniswapV2Pair.PERMIT_TYPEHASH() (DreamDoge.sol#618) is not in mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (DreamDoge.sol#649) is not in mixedCase
Function IUniswapV2Router01.WETH() (DreamDoge.sol#697) is not in mixedCase
Parameter DreamDoge.setSwapAndLiquifyEnabled(bool)._enabled (DreamDoge.sol#1175) is not in
mixedCase
Parameter DreamDoge.setRewardEnabled(bool)._enabled (DreamDoge.sol#1180) is not in mixedCase
Parameter DreamDoge.setMaxLiquidity(uint256)._maxLiquidity (DreamDoge.sol#1184) is not in mixedCase
Parameter DreamDoge.setTradeReward(uint256)._tradeReward (DreamDoge.sol#1188) is not in mixedCase
Parameter DreamDoge.setMinBalance(uint256)._minBalance (DreamDoge.sol#1192) is not in mixedCase
Parameter DreamDoge.setTimespam(uint256)._timespam (DreamDoge.sol#1196) is not in mixedCase
Parameter DreamDoge.setUpdateRatio(uint256)._updateRatio (DreamDoge.sol#1200) is not in mixedCase
Parameter DreamDoge.setSalesRatio(uint256)._salesRatio (DreamDoge.sol#1204) is not in mixedCase
Parameter DreamDoge.setRewardCycle(uint256)._rewardCycle (DreamDoge.sol#1208) is not in mixedCase
Parameter DreamDoge.setBigBuyerReward(uint256)._bigBuyerReward (DreamDoge.sol#1212) is not in
mixedCase
Parameter DreamDoge.setBlacklistThreshold(uint256)._blacklistThreshold (DreamDoge.sol#1217) is not in
mixedCase
Parameter DreamDoge.setDeadline(uint256)._deadline (DreamDoge.sol#1224) is not in mixedCase
Parameter DreamDoge.random(uint256)._length (DreamDoge.sol#1402) is not in mixedCase
Parameter DreamDoge.calcuteTaxAmount(DreamDoge.TaxRate,uint256)._taxRate (DreamDoge.sol#1493)
is not in mixedCase
Variable DreamDoge._winner (DreamDoge.sol#986) is not in mixedCase
Variable DreamDoge._potentialWinner (DreamDoge.sol#990) is not in mixedCase
Variable DreamDoge._potentialWinnerAmount (DreamDoge.sol#991) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (DreamDoge.sol#268)" inContext (DreamDoge.sol#262-271)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
Variable
IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amoun
tADesired (DreamDoge.sol#702) is too similar to
IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amoun
tBDesired (DreamDoge.sol#703)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
DreamDoge.slitherConstructorVariables() (DreamDoge.sol#902-1598) uses literals with too many digits:
        - blackHole = 0x000000000000000000000000000000000000dEaD (DreamDoge.sol#959)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
DreamDoge._decimals (DreamDoge.sol#951) should be constant
DreamDoge._name (DreamDoge.sol#949) should be constant
DreamDoge._symbol (DreamDoge.sol#950) should be constant
DreamDoge._totalSupply (DreamDoge.sol#948) should be constant
DreamDoge.blackHole (DreamDoge.sol#959) 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() (DreamDoge.sol#507-510)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (DreamDoge.sol#516-523)
geUnlockTime() should be declared external:
        - Ownable.geUnlockTime() (DreamDoge.sol#525-527)
lock(uint256) should be declared external:
        - Ownable.lock(uint256) (DreamDoge.sol#530-535)
unlock() should be declared external:
        - Ownable.unlock() (DreamDoge.sol#538-546)
name() should be declared external:
        - DreamDoge.name() (DreamDoge.sol#1057-1059)
symbol() should be declared external:
        - DreamDoge.symbol() (DreamDoge.sol#1061-1063)
decimals() should be declared external:
        - DreamDoge.decimals() (DreamDoge.sol#1065-1067)
totalSupply() should be declared external:
        - DreamDoge.totalSupply() (DreamDoge.sol#1069-1071)
transfer(address,uint256) should be declared external:
        - DreamDoge.transfer(address,uint256) (DreamDoge.sol#1077-1084)
allowance(address,address) should be declared external:
        - DreamDoge.allowance(address,address) (DreamDoge.sol#1086-1093)
approve(address,uint256) should be declared external:
        - DreamDoge.approve(address,uint256) (DreamDoge.sol#1095-1102)
transferFrom(address,address,uint256) should be declared external:
        - DreamDoge.transferFrom(address,address,uint256) (DreamDoge.sol#1104-1119)
increaseAllowance(address,uint256) should be declared external:
        - DreamDoge.increaseAllowance(address,uint256) (DreamDoge.sol#1121-1132)
decreaseAllowance(address,uint256) should be declared external:
        - DreamDoge.decreaseAllowance(address,uint256) (DreamDoge.sol#1134-1148)
excludeFromFee(address) should be declared external:
        - DreamDoge.excludeFromFee(address) (DreamDoge.sol#1151-1153)
includeInFee(address) should be declared external:
        - DreamDoge.includeInFee(address) (DreamDoge.sol#1155-1157)
excludeBlacklist(address) should be declared external:
        - DreamDoge.excludeBlacklist(address) (DreamDoge.sol#1159-1161)
includeBlacklist(address) should be declared external:
        - DreamDoge.includeBlacklist(address) (DreamDoge.sol#1163-1165)
```

```
excludeWhitelist(address) should be declared external:
	- DreamDoge.excludeWhitelist(address) (DreamDoge.sol#1167-1169)
includeWhitelist(address) should be declared external:
	- DreamDoge.includeWhitelist(address) (DreamDoge.sol#1171-1173)
setSwapAndLiquifyEnabled(bool) should be declared external:
	- DreamDoge.setSwapAndLiquifyEnabled(bool) (DreamDoge.sol#1175-1178)
setRewardEnabled(bool) should be declared external:
	- DreamDoge.setRewardEnabled(bool) (DreamDoge.sol#1180-1182)
setMaxLiquidity(uint256) should be declared external:
	- DreamDoge.setMaxLiquidity(uint256) (DreamDoge.sol#1184-1186)
setTradeReward(uint256) should be declared external:
	- DreamDoge.setTradeReward(uint256) (DreamDoge.sol#1188-1190)
setMinBalance(uint256) should be declared external:
	- DreamDoge.setMinBalance(uint256) (DreamDoge.sol#1192-1194)
setTimespam(uint256) should be declared external:
	- DreamDoge.setTimespam(uint256) (DreamDoge.sol#1196-1198)
setUpdateRatio(uint256) should be declared external:
	- DreamDoge.setUpdateRatio(uint256) (DreamDoge.sol#1200-1202)
setSalesRatio(uint256) should be declared external:
	- DreamDoge.setSalesRatio(uint256) (DreamDoge.sol#1204-1206)
setRewardCycle(uint256) should be declared external:
	- DreamDoge.setRewardCycle(uint256) (DreamDoge.sol#1208-1210)
setBigBuyerReward(uint256) should be declared external:
	- DreamDoge.setBigBuyerReward(uint256) (DreamDoge.sol#1212-1214)
setBlacklistThreshold(uint256) should be declared external:
	- DreamDoge.setBlacklistThreshold(uint256) (DreamDoge.sol#1217-1222)
setDeadline(uint256) should be declared external:
	- DreamDoge.setDeadline(uint256) (DreamDoge.sol#1224-1226)
getTimeAndMaxSellTxAmount(address) should be declared external:
	- DreamDoge.getTimeAndMaxSellTxAmount(address) (DreamDoge.sol#1236-1242)
isExcludedFromFee(address) should be declared external:
	- DreamDoge.isExcludedFromFee(address) (DreamDoge.sol#1244-1246)
isInBlacklist(address) should be declared external:
	- DreamDoge.isInBlacklist(address) (DreamDoge.sol#1248-1250)
isInWhiteList(address) should be declared external:
	- DreamDoge.isInWhiteList(address) (DreamDoge.sol#1252-1254)
buyerLength(uint256) should be declared external:
	- DreamDoge.buyerLength(uint256) (DreamDoge.sol#1256-1258)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:DreamDoge.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
```