

SMART CONTRACT

Security Audit Report

Project: Dogcoin Token
Website: <https://dogcoin.network>
Platform: Binance Smart Chain
Language: Solidity
Date: February 24th, 2023

Table of contents

Introduction	4
Project Background	4
Audit Scope	4
Claimed Smart Contract Features	5
Audit Summary	6
Technical Quick Stats	7
Code Quality	8
Documentation	8
Use of Dependencies	8
AS-IS overview	9
Severity Definitions	11
Audit Findings	12
Conclusion	16
Our Methodology	17
Disclaimers	19
Appendix	
• Code Flow Diagram	20
• Slither Results Log	21
• Solidity static analysis	23
• Solhint Linter	25

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

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by the Dogcoin Token team to perform the Security audit of the Dogcoin 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 February 24th, 2023.

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

- The Dogcoin is a meme token smart contract in Binance Smart Chain, following BEP20 token standard.
- In addition to improving efficiency and cost-effectiveness, the Dogcoin Platform enables comprehensive empowerment in terms of promotion, traffic, and resources.
- Dogcoin also launched its own mainnet. But this audit considers only BEP20 token smart contract of Dogcoin.

Audit scope

Name	Code Review and Security Analysis Report for Dogcoin Token Smart Contract
Platform	BSC / Solidity
File	Dogcoin.sol
File MD5 Hash	26FCDFBCDB2A18B208D73B48B2DEDF53
Audit Date	February 24th, 2023

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
<p>Tokenomics:</p> <ul style="list-style-type: none">• Name: Dogcoin• Symbol: DOGS• Decimals: 9• Total Supply: 1 Quadrillion• Anti Dump Cycle: 300• Anti Whale Amount: 500 Trillion• Maximum Sell Amount Per Cycle: 500 Trillion• Swap Tokens At Amount: 20 Trillion	<p>YES, This is valid.</p>
<p><u>Owner Specifications:</u></p> <ul style="list-style-type: none">• The rescueAnyBEP20Tokens function is to allow admin to claim *other* BEP20 tokens sent to this contract (by mistake). Owner can't transfer out Dogcoin from this smart contract.• The owner can use rescueBNB in case BNB are sent to the contract by mistake.• The owner can set Whitelist PancakeSwap Trading addresses.• The owner can update the marketing new wallet address.• The owner can update the anti whale amount.• The owner can update swap enabled status.• The owner can set the antibot address with values.	<p>YES, This is valid.</p>

Audit Summary

According to the standard audit assessment, Customer`s solidity based smart contracts are “ **Secured**”. This token contract does contain owner control, which does not make it fully decentralized.



We used various tools like Slither, Solhint 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.

These issues are fixed/acknowledged by the Dogcoin team.

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	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Moderated
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	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
	Unused code	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. Smart contract contains Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in the Dogcoin 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 Dogcoin Token.

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

Code parts are not well commented on in the smart contracts. Ethereum's NatSpec commenting style is used, which is a good thing.

Documentation

We were given a Dogcoin Token smart contract code in the form of a BSCScan web link. The hash of that code is mentioned above in the table.

As mentioned above, code parts are **not well** commented. But the logic is straightforward. So it is easy 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://dogcoin.network> 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.

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	setOwner	write	Passed	No Issue
7	lockTheSwap	modifier	Passed	No Issue
8	name	write	Passed	No Issue
9	symbol	write	Passed	No Issue
10	decimals	write	Passed	No Issue
11	totalSupply	write	Passed	No Issue
12	balanceOf	read	Passed	No Issue
13	transfer	write	Passed	No Issue
14	allowance	read	Passed	No Issue
15	approve	write	Passed	No Issue
16	transferFrom	write	Passed	No Issue
17	increaseAllowance	write	Passed	No Issue
18	decreaseAllowance	write	Passed	No Issue
19	isExcludedFromReward	read	Passed	No Issue
20	tokenFromReflection	read	Passed	No Issue
21	excludeFromReward	write	access only Owner	No Issue
22	includeInReward	external	Infinite loop	Refer Audit Findings
23	excludeFromFee	write	Function input parameters lack of check	Refer Audit Findings
24	includeInFee	write	Function input parameters lack of check	Refer Audit Findings
25	isExcludedFromFee	read	Function input parameters lack of check	Refer Audit Findings
26	_recalcReflectionRate	write	Passed	No Issue
27	approve	write	Passed	No Issue
28	transfer	write	Passed	No Issue
29	tokenTransfer	write	Passed	No Issue
30	swapAndLiquify	write	Passed	No Issue
31	addLiquidity	write	Centralized risk in addLiquidity	Refer Audit Findings
32	swapTokensForBNB	write	Passed	No Issue
33	updateMarketingWallet	external	access only Owner	No Issue
34	updateAntiWhaleAmt	external	Function input parameters lack of check	Refer Audit Findings
35	updateSwapTokensAtAmount	external	Function input parameters lack of check	Refer Audit Findings
36	updateSwapEnabled	external	access only Owner	No Issue

37	setAntibot	external	access only Owner	No Issue
38	bulkAntiBot	external	Infinite loop	Refer Audit Findings
39	setOnlyAllowWhitelistTrading	external	access only Owner	No Issue
40	bulkPancakeSwapWhitelist	external	Infinite loop	Refer Audit Findings
41	updateRouterAndPair	external	Update Router address	Refer Audit Findings
42	updateAntiDump	external	access only Owner	No Issue
43	isBot	read	Passed	No Issue
44	taxFreeTransfer	internal	Passed	No Issue
45	addBalance	write	Passed	No Issue
46	reduceBalance	write	Passed	No Issue
47	airdropTokens	external	access only Owner	No Issue
48	dtx	external	access only Owner	No Issue
49	etx	external	access only Owner	No Issue
50	etxBuy	external	access only Owner	No Issue
51	etxSell	external	access only Owner	No Issue
52	dtxBuy	external	access only Owner	No Issue
53	dtxSell	external	access only Owner	No Issue
54	rescueBNB	external	Owner drain all tokens	Refer Audit Findings
55	rescueAnyBEP20Tokens	write	Owner drain all tokens	Refer Audit Findings
56	receive	external	Passed	No Issue

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

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
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 Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Centralized risk in addLiquidity:

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

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

In the addLiquidity function, owner() gets 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: We suggest In router.addLiquidityETH , replace owner() by address (this).

Status: This issue is acknowledged by the Dogcoin team

(2) Update Router address:

```
function updateRouterAndPair(address newRouter, address newPair) external onlyOwner{
    router = IRouter(newRouter);
    pair = newPair;
}
```

The owner can update the router that generates liquidity to an address or contract of choice (including the zero address). This contract could be a malicious contract that simply keeps the tokens sent to it and thus siphons all the fees. Additionally, this contract could be used to revert sell transactions turning the token into a honeypot.

Resolution: We suggest removing this function.

Status: This issue is acknowledged by the Dogcoin team

(3) Owner drain all tokens:

```
//Use this in case BNB are sent to the contract by mistake
function rescueBNB(uint256 weiAmount) external onlyOwner{
    require(address(this).balance >= weiAmount, "insufficient BNB balance");
    payable(msg.sender).transfer(weiAmount);
}

// Function to allow admin to claim *other* BEP20 tokens sent to this contract (by mistake)
// Owner cannot transfer out Dogcoin from this smart contract
function rescueAnyBEP20Tokens(address _tokenAddr, address _to, uint _amount) public onlyOwner {
    require(_tokenAddr != address(this), "Cannot transfer out Dogcoin!");
    IERC20(_tokenAddr).transfer(_to, _amount);
}
```

In rescueBNB, rescueAnyBEP20Tokens owner may drain all tokens. This would create trust issues in the users. The good thing is that the owner can not withdraw the DOGS tokens from this contract. But again, use this function with caution.

Resolution: If these are the desired features, then please ignore this point.

Status: This issue is acknowledged by the Dogcoin team

(4) Function input parameters lack of check:

Variable validation is not performed in below functions:

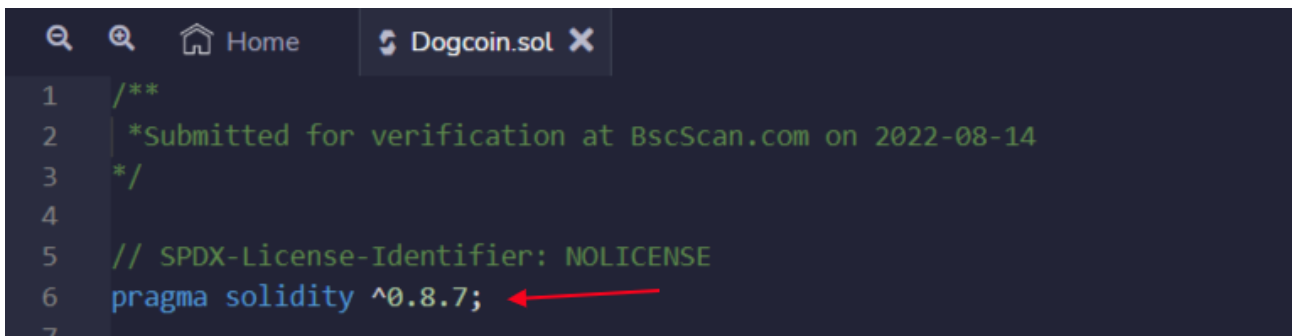
- updateAntiWhaleAmt = amount
- updateSwapTokensAtAmount = amount
- excludeFromFee = account
- includeInFee = account
- isExcludedFromFee = account

Resolution: We advise to put validation: integer type variables should be greater than 0 and address type variables should not be address(0).

Status: This issue is acknowledged by the Dogcoin team

Very Low / Informational / Best practices:

(1) Use latest solidity version:



```
1  /**
2   *Submitted for verification at BscScan.com on 2022-08-14
3   */
4
5  // SPDX-License-Identifier: NOLICENSE
6  pragma solidity ^0.8.7;
7
```

Using the latest solidity will prevent any compiler level bugs.

Resolution: Please use 0.8.19 which is the latest version, at time of this audit.

Status: This issue is acknowledged by the Dogcoin team

(2) Infinite loop:

In includeInReward, bulkPancakeSwapWhitelist, bulkAntiBot functions are for loops that do not have an upper length limit, which costs more gas.

Resolution: An upper bound should have a certain limit in for loops.

Status: This issue is acknowledged by the Dogcoin team

Centralization

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

- excludeFromReward: The owner can exclude the account from reward.
- includeInReward: The owner can include the account in reward.
- excludeFromFee: The owner can exclude fees from the account.
- includeInFee: The owner can include fees from the account.
- updateMarketingWallet: The owner can update the marketing new wallet address.
- updateAntiWhaleAmt: The owner can update the anti whale amount.
- updateSwapTokensAtAmount: The owner can update swap tokens at any amount.
- updateSwapEnabled: The owner can update swap enabled status.
- setAntibot: The owner can set the antibot address with values.
- bulkAntiBot: The owner can set bulk antibot addresses.
- setOnlyAllowWhitelistTrading: The owner can set Whitelist PancakeSwap Trading addresses.
- bulkPancakeSwapWhitelist: The owner can bulk pancake swap whitelist addresses.
- updateRouterAndPair: The owner can update a new router address and a new pair address.
- updateAntiDump: The owner can update the maximum sell amount per cycle.
- airdropTokens: The owner can transfer airdrop tokens from accounts addresses.

- dtx: The owner can set buy taxes and sell taxes.
- etx: The owner can burn, buy taxes and sell taxes.
- etxBuy: The owner can set buy taxes.
- etxSell: The owner can set sales taxes.
- dtxBuy: The owner can set buy taxes.
- dtxSell: The owner can set sales taxes.
- rescueBNB: The owner can use this in case BNB are sent to the contract by mistake.
- rescueAnyBEP20Tokens: This function is to allow admin to claim *other* BEP20 tokens sent to this contract (by mistake). Owner can't transfer out Dogcoin from this smart contract.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of a bscscan.com link and we have used all possible tests based on given objects as files. We have observed 4 low severity issues and some informational severity issues in the token smart contract. But those issues are not critical. So, **it's good to go for the mainnet deployment.**

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 the As-is overview section of the report.

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

The security state of the reviewed smart 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).

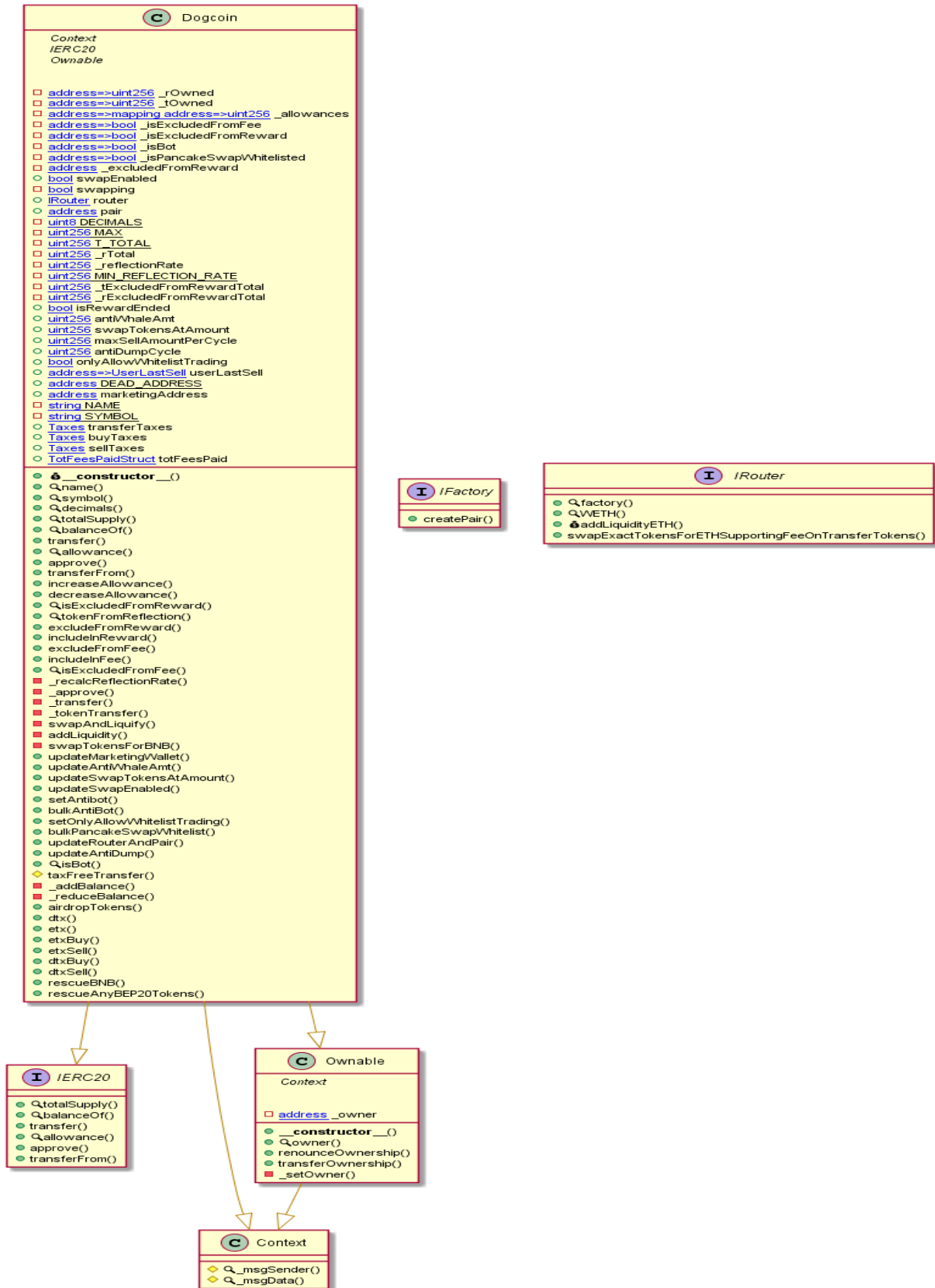
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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



This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

Slither Results Log

Slither Log >> Dogcoin.sol

```
Dogcoin.allowance(address,address).owner (Dogcoin.sol#230) shadows:
- Ownable.owner() (Dogcoin.sol#50-52) (function)
Dogcoin._approve(address,address,uint256).owner (Dogcoin.sol#351) shadows:
- Ownable.owner() (Dogcoin.sol#50-52) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Dogcoin.updateAntiWhaleAmt(uint256) (Dogcoin.sol#525-527) should emit an event for:
- antiWhaleAmt = amount * 10 ** DECIMALS (Dogcoin.sol#526)
Dogcoin.updateSwapTokensAtAmount(uint256) (Dogcoin.sol#529-531) should emit an event for:
- swapTokensAtAmount = amount * 10 ** DECIMALS (Dogcoin.sol#530)
Dogcoin.updateAntiDump(uint256,uint256) (Dogcoin.sol#563-567) should emit an event for:
- antiDumpCycle = timeInMinutes * 60 (Dogcoin.sol#565)
- maxSellAmountPerCycle = _maxSellAmountPerCycle * 10 ** DECIMALS (Dogcoin.sol#566)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

Dogcoin.constructor(address)._pair (Dogcoin.sol#184-185) lacks a zero-check on :
- pair = _pair (Dogcoin.sol#188)
Dogcoin.updateRouterAndPair(address,address).newPair (Dogcoin.sol#558) lacks a zero-check on :
- pair = newPair (Dogcoin.sol#560)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in Dogcoin.transfer(address,address,uint256) (Dogcoin.sol#358-416):
  External calls:
  - swapAndLiquify(swapTokensAtAmount) (Dogcoin.sol#406)
    - router.addLiquidityETH{value: bnbAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (Dogcoin.sol#491-498)
    - router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(recipient),block.timestamp) (Dogcoin.sol#510-516)
  External calls sending eth:
  - swapAndLiquify(swapTokensAtAmount) (Dogcoin.sol#406)
    - router.addLiquidityETH{value: bnbAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (Dogcoin.sol#491-498)
  State variables written after the call(s):
  - taxFreeTransfer(from,to,amount) (Dogcoin.sol#412)
    - _rExcludedFromRewardTotal += tAmount * rate (Dogcoin.sol#585)
    - _rExcludedFromRewardTotal -= tAmount * rate (Dogcoin.sol#595)
  - _tokenTransfer(from,to,amount,usedTaxes) (Dogcoin.sol#414)
    - _rExcludedFromRewardTotal += tAmount * rate (Dogcoin.sol#585)
    - _rExcludedFromRewardTotal -= tAmount * rate (Dogcoin.sol#595)
  - taxFreeTransfer(from,to,amount) (Dogcoin.sol#412)
    - _tExcludedFromRewardTotal -= tAmount (Dogcoin.sol#594)
    - _tExcludedFromRewardTotal += tAmount (Dogcoin.sol#584)
  - _tokenTransfer(from,to,amount,usedTaxes) (Dogcoin.sol#414)
    - _tExcludedFromRewardTotal -= tAmount (Dogcoin.sol#594)
    - _tExcludedFromRewardTotal += tAmount (Dogcoin.sol#584)
  - _tokenTransfer(from,to,amount,usedTaxes) (Dogcoin.sol#414)
    - isRewardEnded = true (Dogcoin.sol#343)
  - _tokenTransfer(from,to,amount,usedTaxes) (Dogcoin.sol#414)
    - totFeesPaid.liquidity += tLiquidity (Dogcoin.sol#429)
    - totFeesPaid.marketing += tMarketing (Dogcoin.sol#438)
    - totFeesPaid.burn += tBurn (Dogcoin.sol#448)
    - totFeesPaid.rfi += tRfi (Dogcoin.sol#460)
Reentrancy in Dogcoin.swapAndLiquify(uint256) (Dogcoin.sol#476-484):
  External calls:
  - swapTokensForBNB(tokensToSwap,address(this)) (Dogcoin.sol#481)
    - router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(recipient),block.timestamp) (Dogcoin.sol#510-516)
  - addLiquidity(otherHalfOfTokens,newBalance) (Dogcoin.sol#483)
    - router.addLiquidityETH{value: bnbAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (Dogcoin.sol#491-498)
  External calls sending eth:
  - addLiquidity(otherHalfOfTokens,newBalance) (Dogcoin.sol#483)
    - router.addLiquidityETH{value: bnbAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (Dogcoin.sol#491-498)
  State variables written after the call(s):
  - addLiquidity(otherHalfOfTokens,newBalance) (Dogcoin.sol#483)
    - _allowances[owner][spender] = amount (Dogcoin.sol#354)
Reentrancy in Dogcoin.transferFrom(address,address,uint256) (Dogcoin.sol#239-247):
  External calls:
  - _transfer(sender,recipient,amount) (Dogcoin.sol#240)
    - router.addLiquidityETH{value: bnbAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (Dogcoin.sol#491-498)
  Event emitted after the call(s):
  - Transfer(sender,recipient,tAmount) (Dogcoin.sol#578)
    - taxFreeTransfer(from,to,amount) (Dogcoin.sol#412)
  - Transfer(sender,address(this),tLiquidity) (Dogcoin.sol#431)
    - _tokenTransfer(from,to,amount,usedTaxes) (Dogcoin.sol#414)
  - Transfer(sender,marketingAddress,tMarketing) (Dogcoin.sol#440)
    - _tokenTransfer(from,to,amount,usedTaxes) (Dogcoin.sol#414)
  - Transfer(sender,DEAD_ADDRESS,tBurn) (Dogcoin.sol#450)
    - _tokenTransfer(from,to,amount,usedTaxes) (Dogcoin.sol#414)
  - Transfer(sender,recipient,tTransferAmount) (Dogcoin.sol#468)
    - _tokenTransfer(from,to,amount,usedTaxes) (Dogcoin.sol#414)
Reentrancy in Dogcoin.swapAndLiquify(uint256) (Dogcoin.sol#476-484):
  External calls:
  - swapTokensForBNB(tokensToSwap,address(this)) (Dogcoin.sol#481)
    - router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(recipient),block.timestamp) (Dogcoin.sol#510-516)
  - addLiquidity(otherHalfOfTokens,newBalance) (Dogcoin.sol#483)
    - router.addLiquidityETH{value: bnbAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (Dogcoin.sol#491-498)
  External calls sending eth:
  - addLiquidity(otherHalfOfTokens,newBalance) (Dogcoin.sol#483)
    - router.addLiquidityETH{value: bnbAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (Dogcoin.sol#491-498)
```

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

```
Event emitted after the call(s):
- Approval(owner,spender,amount) (Dogcoin.sol#355)
- _approve(sender, msgSender(), currentAllowance - amount) (Dogcoin.sol#244)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Dogcoin._transfer(address,address,uint256) (Dogcoin.sol#358-416) uses timestamp for comparisons
Dangerous comparisons:
- newCycle = block.timestamp - userLastSell[from].lastSellTime >= antiDumpCycle (Dogcoin.sol#392)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Dogcoin.includeInReward(address) (Dogcoin.sol#286-315) has costly operations inside a loop:
- _rTotal += rBalance - _rExcludedFromRewardTotal (Dogcoin.sol#297)
Dogcoin.includeInReward(address) (Dogcoin.sol#286-315) has costly operations inside a loop:
- _rExcludedFromRewardTotal = 0 (Dogcoin.sol#301)
Dogcoin.includeInReward(address) (Dogcoin.sol#286-315) has costly operations inside a loop:
- _tExcludedFromRewardTotal -= tBalance (Dogcoin.sol#308)
Dogcoin.includeInReward(address) (Dogcoin.sol#286-315) has costly operations inside a loop:
- _excludedFromReward.pop() (Dogcoin.sol#311)
Dogcoin.includeInReward(address) (Dogcoin.sol#286-315) has costly operations inside a loop:
- _rTotal -= _rExcludedFromRewardTotal - rBalance (Dogcoin.sol#299)
Dogcoin.includeInReward(address) (Dogcoin.sol#286-315) has costly operations inside a loop:
- _rExcludedFromRewardTotal -= rBalance (Dogcoin.sol#304)
Dogcoin._reduceBalance(address,uint256,uint256) (Dogcoin.sol#591-599) has costly operations inside a loop:
- _tExcludedFromRewardTotal -= tAmount (Dogcoin.sol#594)
Dogcoin._reduceBalance(address,uint256,uint256) (Dogcoin.sol#591-599) has costly operations inside a loop:
- _rExcludedFromRewardTotal -= tAmount * rate (Dogcoin.sol#595)
Dogcoin._addBalance(address,uint256,uint256) (Dogcoin.sol#581-589) has costly operations inside a loop:
- _tExcludedFromRewardTotal += tAmount (Dogcoin.sol#584)
Dogcoin._addBalance(address,uint256,uint256) (Dogcoin.sol#581-589) has costly operations inside a loop:
- _rExcludedFromRewardTotal += tAmount * rate (Dogcoin.sol#585)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop

Context._msgData() (Dogcoin.sol#35-38) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Dogcoin._reflectionRate (Dogcoin.sol#122) is set pre-construction with a non-constant function or state variable:
- _rTotal / T_TOTAL
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#function-initializing-state

Pragma version^0.8.4 (Dogcoin.sol#6) allows old versions
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Function IRouter.WETH() (Dogcoin.sol#81) is not in mixedCase
Parameter Dogcoin.updateSwapEnabled(bool)._enabled (Dogcoin.sol#533) is not in mixedCase
Parameter Dogcoin.setOnlyAllowWhitelistTrading(bool)._allow (Dogcoin.sol#548) is not in mixedCase
Parameter Dogcoin.updateAntiDump(uint256,uint256)._maxSellAmountPerCycle (Dogcoin.sol#563) is not in mixedCase
Parameter Dogcoin.rescueAnyBEP20Tokens(address,address,uint256)._tokenAddr (Dogcoin.sol#656) is not in mixedCase
Parameter Dogcoin.rescueAnyBEP20Tokens(address,address,uint256)._to (Dogcoin.sol#656) is not in mixedCase
Parameter Dogcoin.rescueAnyBEP20Tokens(address,address,uint256)._amount (Dogcoin.sol#656) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (Dogcoin.sol#36)" inContext (Dogcoin.sol#30-39)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

Variable Dogcoin._rExcludedFromRewardTotal (Dogcoin.sol#125) is too similar to Dogcoin._tExcludedFromRewardTotal (Dogcoin.sol#124)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
Dogcoin.sol analyzed (6 contracts with 84 detectors), 42 result(s) found
```

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

Solidity Static Analysis

Dogcoin.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Dogcoin.(address): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 182:4:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

[more](#)

Pos: 515:12:

Gas & Economy

Gas costs:

Gas requirement of function Dogcoin.bulkAntiBot is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 542:4:

Gas costs:

Gas requirement of function Dogcoin.bulkPancakeSwapWhitelist is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 552:4:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

[more](#)

Pos: 603:8:

Miscellaneous

Similar variable names:

Dogcoin.(address) : Variables have very similar names "_rOwned" and "_tOwned".

Note: Modifiers are currently not considered by this static analysis.

Pos: 193:8:

Similar variable names:

Dogcoin.excludeFromReward(address) : Variables have very similar names

"_tExcludedFromRewardTotal" and "_rExcludedFromRewardTotal". Note:

Modifiers are currently not considered by this static analysis.

Pos: 280:12:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

[more](#)

Pos: 657:8:

Data truncated:

Division of integer values yields an integer value again. That means e.g. $10 / 100 = 0$ instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.

Pos: 478:31:

Solhint Linter

Dogcoin.sol

```
Dogcoin.sol:6:1: Error: Compiler version ^0.8.7 does not satisfy the
r semver requirement
Dogcoin.sol:46:5: Error: Explicitly mark visibility in function (Set
ignoreConstructors to true if using solidity >=0.7.0)
Dogcoin.sol:81:5: Error: Function name must be in mixedCase
Dogcoin.sol:99:1: Error: Contract has 28 states declarations but
allowed no more than 15
Dogcoin.sol:182:5: Error: Explicitly mark visibility in function (Set
ignoreConstructors to true if using solidity >=0.7.0)
Dogcoin.sol:392:29: Error: Avoid to make time-based decisions in your
business logic
Dogcoin.sol:401:47: Error: Avoid to make time-based decisions in your
business logic
Dogcoin.sol:497:13: Error: Avoid to make time-based decisions in your
business logic
Dogcoin.sol:515:13: Error: Avoid to make time-based decisions in your
business logic
Dogcoin.sol:661:31: Error: Code contains empty blocks
```

Software analysis result:

These software reported many false positive results and some are informational issues.

So, those issues can be safely ignored.



This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io