DOGECEO Token

AUDIT REPORT

Version 1.0.0

Serial No. 20230321000120016

Presented by Fairyproof

March 21, 2023

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the DOGE CEO Token Issuance project.

Audit Start Time:

March 20, 2023

Audit End Time:

March 21, 2023

Audited Source File's Address:

https://bscscan.com/address/0xb3e1d840bb97d9fc0012a6372a7c05bcc3d71077#code

The goal of this audit is to review DOGE CEO's solidity implementation for its Token Issuance function, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

We make observations on specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

This audit only applies to the specified code, software or any materials supplied by the DOGE CEO team for specified versions. Whenever the code, software, materials, settings, environment etc is changed, the comments of this audit will no longer apply.

— Disclaimer

Note that as of the date of publishing, the contents of this report reflect the current understanding of known security patterns and state of the art regarding system security. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. If the audited source files are smart contract files, risks or issues introduced by using data feeds from offchain sources are not extended by this review either.

Given the size of the project, the findings detailed here are not to be considered exhaustive, and further testing and audit is recommended after the issues covered are fixed.

To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

— Methodology

The above files' code was studied in detail in order to acquire a clear impression of how the its specifications were implemented. The codebase was then subject to deep analysis and scrutiny, resulting in a series of observations. The problems and their potential solutions are discussed in this document and, whenever possible, we identify common sources for such problems and comment on them as well.

The Fairyproof auditing process follows a routine series of steps:

1. Code Review, Including:

- Project Diagnosis

Understanding the size, scope and functionality of your project's source code based on the specifications, sources, and instructions provided to Fairyproof.

- Manual Code Review

Reading your source code line-by-line to identify potential vulnerabilities.

- Specification Comparison

Determining whether your project's code successfully and efficiently accomplishes or executes its functions according to the specifications, sources, and instructions provided to Fairyproof.

2. Testing and Automated Analysis, Including:

- Test Coverage Analysis

Determining whether the test cases cover your code and how much of your code is exercised or executed when test cases are run.

- Symbolic Execution

Analyzing a program to determine the specific input that causes different parts of a program to execute its functions.

3. Best Practices Review

Reviewing the source code to improve maintainability, security, and control based on the latest established industry and academic practices, recommendations, and research.

— Structure of the document

This report contains a list of issues and comments on all the above source files. Each issue is assigned a severity level based on the potential impact of the issue and recommendations to fix it, if applicable. For ease of navigation, an index by topic and another by severity are both provided at the beginning of the report.

— Documentation

For this audit, we used the following source(s) of truth about how the token issuance function should work:

Website:https://doge.ceo/

Source Code: https://bscscan.com/address/0xb3e1d840bb97d9fc0012a6372a7c05bcc3d71077#code

These were considered the specification, and when discrepancies arose with the actual code behavior, we consulted with the DOGE CEO team or reported an issue.

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2023032100012016	Fairyproof Security Team	Mar 20, 2023 - Mar 21, 2023	Passed

Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, no issues were uncovered.

02. About Fairyproof

Fairyproof is a leading technology firm in the blockchain industry, providing consulting and security audits for organizations. Fairyproof has developed industry security standards for designing and deploying blockchain applications.

03. Introduction to DOGE CEO

Doge CEO a Meme token in the BSC ecosystem, $DOGECEO is comunity-driven cannot control by anyone.

The above description is quoted from relevant documents of DOGE CEO.

04. Major functions of audited code

The audited code mainly implements a token issuance function. Here are the details:

- Blockchain: BNB Chain
- Token Standard: BEP-20
- Token Address: 0xb3e1D840bB97d9fC0012A6372a7c05BCC3D71077
- Token Name: Doge CEO
- Token Symbol: DOGECEO
- Decimals: 9
- Current Supply: 420,000,000,000,000,000,
- Max Supply: 420,000,000,000,000,000,
- Taxable: Yes

Note:

DOGECEO is a token with a reflection mechanism. In every token transfer, 10% of the total mount will be charged as tax. Half of the tax is distributed to all token holders, and the remaining half is converted to BNBs and sent to `marketWallet`。

- The addresses of DOGECEO included token pairs are excluded from the addresses that receive taxes.
- The access control of `owner` has been revoked.

05. Coverage of issues

The issues that the Fairyproof team covered when conducting the audit include but are not limited to the following ones:

- Access Control
- Admin Rights
- Arithmetic Precision
- Code Improvement
- Contract Upgrade/Migration
- Delete Trap
- Design Vulnerability
- DoS Attack
- EOA Call Trap
- Fake Deposit
- Function Visibility

- Gas Consumption
- Implementation Vulnerability
- Inappropriate Callback Function
- Injection Attack
- Integer Overflow/Underflow
- IsContract Trap
- Miner's Advantage
- Misc
- Price Manipulation
- Proxy selector clashing
- Pseudo Random Number
- Re-entrancy Attack
- Replay Attack
- Rollback Attack
- Shadow Variable
- Slot Conflict
- Token Issuance
- Tx.origin Authentication
- Uninitialized Storage Pointer

06. Severity level reference

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

Critical severity issues need to be fixed as soon as possible.

High severity issues will probably bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

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

Informational is not an issue or risk but a suggestion for code improvement.

07. Major areas that need attention

Based on the provided source code the Fairyproof team focused on the possible issues and risks related to the following functions or areas.

- Function Implementation

We checked whether or not the functions were correctly implemented.
We didn't find issues or risks in these functions or areas at the time of writing.

- Access Control

We checked each of the functions that could modify a state, especially those functions that could only be accessed by owner or administrator
We didn't find issues or risks in these functions or areas at the time of writing.

- Token Issuance & Transfer

We examined token issuance and transfers for situations that could harm the interests of holders.
We didn't find issues or risks in these functions or areas at the time of writing.

- State Update

We checked some key state variables which should only be set at initialization.
We didn't find issues or risks in these functions or areas at the time of writing.

- Asset Security

We checked whether or not all the functions that transfer assets were safely handled.
We didn't find issues or risks in these functions or areas at the time of writing.

- Miscellaneous

We checked the code for optimization and robustness.
We didn't find issues or risks in these functions or areas at the time of writing.

08. issues by severity

- N/A

09. Issue descriptions

- N/A

10. Recommendations to enhance the overall security

We list some recommendations in this section. They are not mandatory but will enhance the overall security of the system if they are adopted.

- N/A

11. Appendices

11.1 Unit Test

1. Dodoceo.t.js

```
const {
    time,
    loadFixture,
  } = require("@nomicfoundation/hardhat-network-helpers");
const { anyValue } = require("@nomicfoundation/hardhat-chai-matchers/withArgs");
const { expect } = require("chai");
const { ethers } = require("hardhat");

describe("DODOCEO", function () {
    const DEAD_ADDRESS = "0x000000000000000000000000000000000000dEaD";
    const MARKET_ADDRESS = "0xaa313121bd678d01880dad8Aa68E9B4fa8848DFD";
```

```
    async function deployUniswapAndDodoCeo() {
        const [owner, Alice,Bob] = await ethers.getSigners();
        const WETH9 = await ethers.getContractFactory("WETH9");
        const weth = await WETH9.deploy();
        // deploy uniswap
        const UniswapV2Factory = await ethers.getContractFactory("UniswapV2Factory");
        const factory = await UniswapV2Factory.deploy(owner.address);
        const UniswapV2Router02 = await ethers.getContractFactory("UniswapV2Router02");
        const router = await UniswapV2Router02.deploy(factory.address, weth.address);
        const DOGECEO = await ethers.getContractFactory("DOGECEO");
        const instance = await DOGECEO.deploy(router.address);
        // addLiquid
        let balance = await instance.balanceOf(owner.address);
        let amount = balance.div(2);
        await instance.approve(router.address, ethers.constants.MaxUint256);
        await router.addLiquidityETH(instance.address,amount,1,1,owner.address,9876543210,{
            value:ethers.utils.parseEther("100")
        });
        const pair = await factory.getPair(instance.address,weth.address);
        let supply = ethers.utils.parseUnits("420",15);
        let unit = ethers.utils.parseUnits("1.0",9);
        supply = supply.mul(unit);
        return {router,pair,owner,Alice,Bob,instance,supply}
    }

    describe("Deployment", function () {
        it("Owner should be the deployer", async () => {
            const {owner,instance} = await loadFixture(deployUniswapAndDodoCeo);
            expect(await instance.owner()).to.equal(owner.address);
        });

        it("MetaData check", async () => {
            const {instance} = await loadFixture(deployUniswapAndDodoCeo);
            expect(await instance.name()).to.equal("Doge CEO");
            expect(await instance.symbol()).to.equal("DOGECEO");
            expect(await instance.decimals()).to.equal(9);
        });

        it("ExcludeFee and exclude should be set", async () => {
            const {owner,pair,instance} = await loadFixture(deployUniswapAndDodoCeo);
            expect(await instance.isExcludedFromReward(pair)).to.true;
            expect(await instance.isExcludedFromReward(DEAD_ADDRESS)).to.true;
            expect(await instance.isExcludedFromFee(instance.address)).to.true;
            expect(await instance.isExcludedFromFee(owner.address)).to.true;
            expect(await instance.isExcludedFromFee(MARKET_ADDRESS)).to.true;
            expect(await instance.isExcludedFromFee(DEAD_ADDRESS)).to.true;
        });

        it("Should set the right initial supply", async () => {
            const {owner,pair,instance,supply} = await
loadFixture(deployUniswapAndDodoCeo);
            expect(await instance.totalSupply()).to.equal(supply);
            expect(await instance.balanceOf(owner.address)).to.equal(supply.div(2));
```

```
            expect(await instance.balanceOf(pair)).to.equal(supply.div(2));
        });
    });

    describe("Renounce Owner test", function () {
        it("Should be failed without owner", async () => {
            const {Alice,instance} = await loadFixture(deployUniswapAndDodoCeo);
            await expect(instance.connect(Alice).renounceOwnership()).to.revertedWith(
                "Ownable: caller is not the owner"
            );
        });
        it("Should emit the event", async () => {
            const {owner,instance} = await loadFixture(deployUniswapAndDodoCeo);
            await expect(instance.renounceOwnership()).to.emit(
                instance,"OwnershipTransferred"
            ).withArgs(owner.address,ethers.constants.AddressZero);
        });
    });

    describe("Interface of onlyOwner", function () {
        it("IncludeAndExcludeReward test", async () => {
            const {Alice,Bob,instance} = await loadFixture(deployUniswapAndDodoCeo);
            expect(await instance.isExcludedFromReward(Alice.address)).to.false;
            await instance.excludeFromReward(Alice.address);
            expect(await instance.isExcludedFromReward(Alice.address)).to.true;
            await
expect(instance.excludeFromReward(Alice.address)).to.revertedWith("Account is already
excluded");
            await instance.excludeFromReward(Bob.address);
            await instance.includeInReward(Alice.address);
            expect(await instance.isExcludedFromReward(Alice.address)).to.false;
            await expect(instance.includeInReward(Alice.address)).to.revertedWith("Account
is not excluded");
        });

        it("IncludeAndExcludeFee test", async () => {
            let {Alice,instance} = await loadFixture(deployUniswapAndDodoCeo);
            expect(await instance.isExcludedFromFee(Alice.address)).to.false;
            await instance.excludeFromFee(Alice.address);
            expect(await instance.isExcludedFromFee(Alice.address)).to.true;
            // again
            await instance.excludeFromFee(Alice.address);
            expect(await instance.isExcludedFromFee(Alice.address)).to.true;

            await instance.includeInFee(Alice.address);
            expect(await instance.isExcludedFromFee(Alice.address)).to.false;
            await instance.includeInFee(Alice.address);
            expect(await instance.isExcludedFromFee(Alice.address)).to.false;

            ({Alice,Bob,instance} = await loadFixture(deployUniswapAndDodoCeo));
            let accounts = [Alice.address,Bob.address];
            await instance.bulkExcludeFee(accounts,true);
            expect(await instance.isExcludedFromFee(Alice.address)).to.true;
```

```
            expect(await instance.isExcludedFromFee(Bob.address)).to.true;

            await instance.bulkExcludeFee(accounts,false);
            expect(await instance.isExcludedFromFee(Alice.address)).to.false;
            expect(await instance.isExcludedFromFee(Bob.address)).to.false;
        });
    });

    describe("Approve", function() {
        it("Approve should emit event", async () => {
            const {Alice,Bob,instance} = await loadFixture(deployUniswapAndDodoCeo);
            await expect(instance.connect(Alice).approve(Bob.address,100000)).to.emit(
                instance,"Approval"
            ).withArgs(Alice.address,Bob.address,100000);
            expect(await instance.allowance(Alice.address,Bob.address)).to.equal(100000);

            await
expect(instance.connect(Alice).increaseAllowance(Bob.address,200000)).to.emit(
                instance,"Approval"
            ).withArgs(Alice.address,Bob.address,300000);
            expect(await instance.allowance(Alice.address,Bob.address)).to.equal(300000);

            await
expect(instance.connect(Alice).decreaseAllowance(Bob.address,120000)).to.emit(
                instance,"Approval"
            ).withArgs(Alice.address,Bob.address,180000);
            expect(await instance.allowance(Alice.address,Bob.address)).to.equal(180000);

        });
    });

    describe("Reflection", function() {
        it("reflectionFromToken and tokenFromReflection ", async () => {
            const {supply,instance} = await loadFixture(deployUniswapAndDodoCeo);
            const max = ethers.constants.MaxUint256;
            const rTotal = max.sub(max.mod(supply));
            const rate = rTotal.div(supply);
            for(let i=0;i<10;i++) {
                let tAmount = parseInt(Math.random() * 10000 + 10000);
                tAmount = ethers.BigNumber.from("" + tAmount);
                let rAmount = rate.mul(tAmount);
                //
                let fee = tAmount.mul(5).div(100).mul(rate).mul(2);
                let rTransferAmount = rAmount.sub(fee);
                let v1 = await instance.reflectionFromToken(tAmount,false);
                let v2 = await instance.reflectionFromToken(tAmount,true);
                expect(v1).to.equal(rAmount);
                expect(v2).to.equal(rTransferAmount);

                let v3 = await instance.tokenFromReflection(rAmount);
                expect(v3).to.equal(tAmount);
            }
        });
```

```
    });

    describe("Transfer", function () {
        it("Transfer from address of excludeFee", async () => {
            const {Alice,Bob,instance} = await loadFixture(deployUniswapAndDodoCeo);
            expect(await instance.balanceOf(Bob.address)).to.equal(0);
            expect(await instance.balanceOf(Alice.address)).to.equal(0);
            let value = ethers.utils.parseUnits("10000",9);
            await instance.transfer(Alice.address,value);
            expect(await instance.balanceOf(Alice.address)).to.equal(value);

            await instance.excludeFromFee(Alice.address);
            await expect(instance.connect(Alice).transfer(Bob.address,value)).to.emit(
                instance,"Transfer"
            ).withArgs(Alice.address,Bob.address,value);
            expect(await instance.balanceOf(Bob.address)).to.equal(value);
        });

        it("Transfer without reward", async () => {
            // prepare
            const {Alice,Bob,instance,supply} = await loadFixture(deployUniswapAndDodoCeo);
            let value = ethers.utils.parseUnits("10000",9);
            await instance.transfer(Alice.address,value);
            await expect(instance.connect(Alice).transfer(Bob.address,value)).to.emit(
                instance,"Transfer"
            ).withArgs(Alice.address,Bob.address,value.mul(9).div(10));
            expect(await instance.balanceOf(Bob.address)).to.equal(value.mul(9).div(10));
            expect(await
instance.balanceOf(instance.address)).to.equal(value.mul(5).div(100));
            let {rfi,marketing} = await instance.totFeesPaid();
            expect(rfi).equal(value.mul(5).div(100))
            expect(marketing).equal(value.mul(5).div(100))
        });

        it("Transfer with reward", async () => {
            const {owner,Alice,Bob,instance,supply} = await
loadFixture(deployUniswapAndDodoCeo);
            let balance = await instance.balanceOf(owner.address);
            await instance.transfer(Alice.address,balance);
            await instance.connect(Alice).transfer(Bob.address,balance);
            let bob_balance = await instance.balanceOf(Bob.address);
            expect(bob_balance.gt(balance.mul(9).div(10)));
        });

        it("Transfer with swap", async () => {
            const {Alice,router,Bob,instance,supply,pair} = await
loadFixture(deployUniswapAndDodoCeo);
            let swapTokensAtAmount = await instance.swapTokensAtAmount();
            await instance.transfer(Alice.address,supply.div(2));
            await instance.connect(Alice).transfer(Bob.address,supply.div(2));
            let bob_balance = await instance.balanceOf(Bob.address);
            let balance = await instance.balanceOf(instance.address);
            expect(balance).gt(swapTokensAtAmount);
```

```
            let balance_before = await instance.balanceOf(pair);
            const UniswapV2Pair = await ethers.getContractFactory("UniswapV2Pair");
            const pair_contract = UniswapV2Pair.attach(pair);
            let token0 = await pair_contract.token0();
            let is_token0 = token0 === instance.address;
            // emit swap
            let amount0In = is_token0 ? balance : 0;
            let amount1In = is_token0 ? 0 : balance;
            let amount0Out = is_token0 ? 0 : anyValue;
            let amount1Out = is_token0 ? anyValue : 0;
            await expect(instance.connect(Bob).transfer(Alice.address,1)).to.emit(
                pair_contract,"Swap"

).withArgs(router.address,amount0In,amount1In,amount0Out,amount1Out,router.address);
            expect(await instance.balanceOf(pair)).equal(balance.add(balance_before));
            expect(await await instance.balanceOf(Bob.address)).equal(bob_balance.sub(1));
            expect(await instance.totalSupply()).equal(supply);
        });
    });

    describe("TransferFrom", function() {
        it("TransferFrom should decrease approval", async () => {
            const {Alice,owner,Bob,instance,supply} = await
loadFixture(deployUniswapAndDodoCeo);
            let value = ethers.utils.parseUnits("10000",9);
            await instance.approve(Alice.address,value);
            await
expect(instance.connect(Alice).transferFrom(owner.address,Bob.address,value.div(10))).emit(
                instance,"Approval"
            ).withArgs(owner.address,Alice.address,value.mul(9).div(10));
            expect(await
instance.allowance(owner.address,Alice.address)).equal(value.mul(9).div(10));
            expect(await
instance.balanceOf(owner.address)).equal(supply.div(2).sub(value.div(10)));
            expect(await instance.balanceOf(Bob.address)).equal(value.div(10));
        });
    });
});
```

11.2 External Functions Check Points

1. DODOCEO.sol_output.md

File: contracts/DODOCEO.sol

(Empty fields in the table represent things that are not required or relevant)

contract: DOGECEO is Context, IBEP20, Ownable

Index	Function	Visibility	StateMutability	Permission Check	IsUserInterface	Unit Test	Notes
1	name()	public	pure			Passed	
2	symbol()	public	pure			Passed	
3	decimals()	public	pure			Passed	
4	totalSupply()	public	view			Passed	
5	balanceOf(address)	public	view			Passed	
6	allowance(address,address)	public	view			Passed	
7	approve(address,uint256)	public			Yes	Passed	
8	transferFrom(address,address,uint256)	public			Yes	Passed	
9	increaseAllowance(address,uint256)	public			Yes	Passed	
10	decreaseAllowance(address,uint256)	public			Yes	Passed	
11	transfer(address,uint256)	public			Yes	Passed	
12	isExcludedFromReward(address)	public	view			Passed	
13	reflectionFromToken(uint256,bool)	public	view			Passed	
14	tokenFromReflection(uint256)	public	view			Passed	
15	excludeFromReward(address)	public		onlyOwner		Passed	
16	includeInReward(address)	external		onlyOwner		Passed	
17	excludeFromFee(address)	public		onlyOwner		Passed	
18	includeInFee(address)	public		onlyOwner		Passed	
19	isExcludedFromFee(address)	public	view			Passed	
20	bulkExcludeFee(address[],bool)	external		onlyOwner		Passed	
21	receive()	external	payable			Passed	
22	owner()	public	view			Passed	
23	renounceOwnership()	public		onlyOwner		Passed	

FAIRYPROOF

https://medium.com/@FairyproofT

https://twitter.com/FairyproofT

https://www.linkedin.com/company/fairyproof-tech

https://t.me/Fairyproof_tech

Reddit: https://www.reddit.com/user/FairyproofTech