FAIRYPROOF

Dogecoin Token

AUDIT REPORT

Version 1.0.0

Serial No. 2023082800022029

Presented by Fairyproof

August 28, 2023

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Dogecoin token issuance project.

Audit Start Time:

August 23, 2023

Audit End Time:

August 23, 2023

Audited Source File's Address:

https://bscscan.com/token/0xba2ae424d960c26247dd6c32edc70b295c744c43#code

The goal of this audit is to review Dogecoin'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 Dogecoin 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://dogecoin.com/

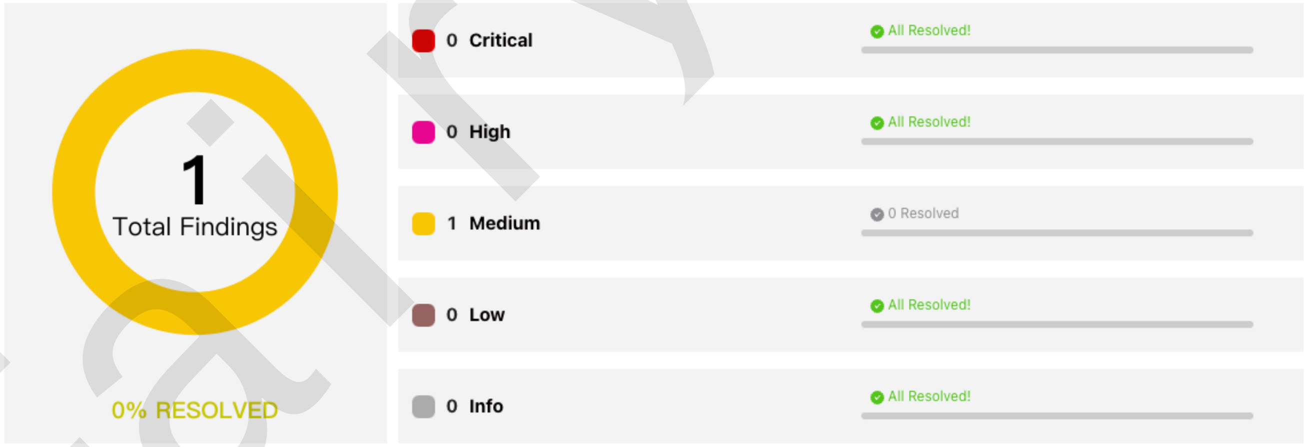
Whitepaper:https://github.com/dogecoin/dogecoin/blob/master/README.md

Source Code: https://bscscan.com/token/0xba2ae424d960c26247dd6c32edc70b295c744c43#code

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

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2023082800022029	Fairyproof Security Team	Aug 23, 2023 - Aug 23, 2023	Medium Risk



Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, one issue of medium-severity was uncovered. The Dogecoin team acknowledged the issue.

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 Dogecoin

Dogecoin (DOGE) is based on the popular "doge" Internet meme and features a Shiba Inu on its logo. Dogecoin is a community-driven cryptocurrency that was inspired by a Shiba Inu meme. The Dogecoin Core software allows anyone to operate a node in the Dogecoin blockchain networks and uses the Scrypt hashing method for Proof of Work. It is adapted from Bitcoin Core and other cryptocurrencies.

The above description is quoted from relevant documents of Dogecoin.

04. Major functions of audited code

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

- Blockchain: BNB Smart Chain
- Token Standard: BEP-20
- Token Address: 0xba2ae424d960c26247dd6c32edc70b295c744c43
- Token Name: Dogecoin
- Token Symbol: DOGE
- Decimals: 8
- Current Supply: 142,999,999,999,999,998
- Max Supply: No Cap
- Burnable: Yes
- Mintable: Yes

Note:

Thiscontract is upgradeable. The access control to upgrade is owned by an EOA address (0xD2f93484f2D319194cBa95C5171B18C1d8cfD6C4).

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 found one issue, for more details please refer to [FP-1] in "09. Issue description".

- 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. List of issues by severity

Index	Title	Issue/Risk	Severity	Status
FP-1	Unlimited Token Issuance	Token Issuance	Medium	Acknowledged

09. Issue descriptions

[FP-1] Unlimited Token Issuance

Token Issuance Medium Acknowledged

Issue/Risk: Token Issuance

Description:

In the current contract, tokens can be issued additionally and there is no cap on issuance,which may cause losses to token holders in certain scenarios.

Recommendation:

Consider setting a cap on token issuance.

Update/Status:

The Dogecoin team prefers to keep it now and will improve the code in the future.

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.

- Consider managing the owner's access control with great care and transfering it to a multi-sig wallet or DAO when necessary.

11. Appendices

11.1 Unit Test

1. Doge.t.js

```
const { expect } = require("chai");
const { ethers } = require("hardhat");

describe("Binance Peg Dogecoin Test", function () {
  let owner, admin, addr1;
  const totalSupply = ethers.parseEther("100000")
  const AddressZero = "0x0000000000000000000000000000000000000000"

  async function deployToken() {
    [owner, admin, addr1] = await ethers.getSigners();
    const BEP20TokenImplementation = await
ethers.getContractFactory("BEP20TokenImplementation");
    const data = BEP20TokenImplementation.interface
      .encodeFunctionData("initialize", ["Dogecoin", "DOGE", 8, totalSupply, true,
owner.address])
    const tokenInstance = await BEP20TokenImplementation.deploy();

    const BEP20UpgradeableProxy = await ethers.getContractFactory("BEP20UpgradeableProxy");
    const proxy = await BEP20UpgradeableProxy.deploy(await tokenInstance.getAddress(),
admin.address, data);
    const instance = BEP20TokenImplementation.attach(await proxy.getAddress());
    return { instance };
  }

  describe("Deployment test", function () {
    it("Should set the correct metadata", async function () {
      const { instance } = await deployToken();

      expect(await instance.totalSupply()).equal(totalSupply);
      expect(await instance.balanceOf(owner.address)).equal(totalSupply);
      expect(await instance.name()).equal("Dogecoin");
      expect(await instance.symbol()).equal("DOGE");
      expect(await instance.decimals()).equal(8);
    });
  });


  describe("Ownership test", function () {
    it("Should transfer ownership correctly", async function () {
      const { instance } = await deployToken();

      expect(await instance.getOwner()).to.equal(owner.address);
      await expect(instance.transferOwnership(addr1.address))
        .be.emit(instance, "OwnershipTransferred").withArgs(owner.address, addr1.address);
      await expect(instance.renounceOwnership()).to.revertedWith("Ownable: caller is not
the owner");
      await instance.connect(addr1).renounceOwnership();
    });

    it("Should lose ownership if the owner renounces ownership", async function () {
      const { instance } = await deployToken();

      await expect(instance.renounceOwnership())
        .be.emit(instance, "OwnershipTransferred").withArgs(owner.address, AddressZero);
      await expect(instance.renounceOwnership()).to.revertedWith("Ownable: caller is not
the owner");
      expect(await instance.getOwner()).to.equal(AddressZero);

    });
  });


  describe("Transactions test", function () {
    it("Should transfer tokens between accounts", async function () {
      const { instance } = await deployToken();
      const transferAmount = 5000;

      await expect(instance.transfer(addr1.address, transferAmount))
        .be.emit(instance, "Transfer").withArgs(owner.address, addr1.address,
transferAmount);
      expect(await instance.balanceOf(addr1.address)).to.equal(transferAmount);
    });

    it("Should be failed if sender doesn’t have enough tokens", async function () {
      const { instance } = await deployToken();
      const initialOwnerBalance = await instance.balanceOf(owner.address);
      await expect(instance.connect(addr1).transfer(owner.address,
1)).to.revertedWith("BEP20: transfer amount exceeds balance");
      expect(await instance.balanceOf(owner.address)).to.equal(initialOwnerBalance);
    });

    it("Should be failed if sender transfer to zero address", async function () {
      const { instance } = await deployToken();
      const transferAmount = 5000;
      await expect(instance.transfer(AddressZero, transferAmount)).to.revertedWith("BEP20:
transfer to the zero address");
      await instance.approve(owner.address, transferAmount);
      await expect(instance.transferFrom(owner.address, AddressZero,
transferAmount)).to.revertedWith("BEP20: transfer to the zero address");
    });

    it("Should be successful if sender transfer to himself", async function () {
      const { instance } = await deployToken();
      const transferAmount = 5000;

      await expect(instance.transfer(owner.address, transferAmount))
        .be.emit(instance, "Transfer").withArgs(owner.address, owner.address,
transferAmount);
      await instance.approve(owner.address, transferAmount);
      await expect(instance.transferFrom(owner.address, owner.address, transferAmount))
        .be.emit(instance, "Transfer").withArgs(owner.address, owner.address,
transferAmount);
      expect(await instance.balanceOf(owner.address)).to.equal(totalSupply);
    });

    it("Should be successful if sender transfer zero amount", async function () {
      const { instance } = await deployToken();

      await expect(instance.transfer(addr1.address, 0))
        .be.emit(instance, "Transfer").withArgs(owner.address, addr1.address, 0);
      await expect(instance.transferFrom(owner.address, addr1.address, 0))
        .be.emit(instance, "Transfer").withArgs(owner.address, addr1.address, 0);
      expect(await instance.balanceOf(owner.address)).to.equal(totalSupply);
    });

    it("TransferFrom should need enough allowance", async function () {
      const { instance } = await deployToken();
      const transferAmount = 5000;

      await expect(instance.transferFrom(owner.address, addr1.address,
transferAmount)).to.revertedWith("BEP20: transfer amount exceeds allowance")
      await instance.approve(owner.address, transferAmount);
      await expect(instance.transferFrom(owner.address, addr1.address, transferAmount))
        .be.emit(instance, "Transfer").withArgs(owner.address, addr1.address,
transferAmount);
      expect(await instance.balanceOf(addr1.address)).to.equal(transferAmount);

      await instance.connect(addr1).approve(owner.address, transferAmount);
      await instance.transferFrom(addr1.address, owner.address, transferAmount)
      expect(await instance.balanceOf(addr1.address)).to.equal(0);
    });
  });


  describe("Allowance test", function () {
    it("Should update the allowance when approving", async function () {
      const { instance } = await deployToken();
      const approveAmount = 1000

      await expect(instance.approve(addr1.address, approveAmount))
        .to.emit(instance, "Approval").withArgs(owner.address, addr1.address,
approveAmount);
      const allowance = await instance.allowance(owner.address, addr1.address);
      expect(allowance).to.equal(approveAmount);
      // increse allowance again
      await expect(instance.increaseAllowance(addr1.address, approveAmount))
        .to.emit(instance, "Approval").withArgs(owner.address, addr1.address, approveAmount
* 2);
      expect(await instance.allowance(owner.address, addr1.address)).to.equal(approveAmount
* 2);
      // decrease allowance
      await expect(instance.decreaseAllowance(addr1.address, approveAmount))
        .to.emit(instance, "Approval").withArgs(owner.address, addr1.address,
approveAmount);
    });
  });


  describe("Mint test", function () {
    it("State _mintable should be true", async function () {
      const { instance } = await deployToken();

      expect(await instance.mintable()).to.equal(true);
      await instance.mint(1);
    });

    it("Onlyowner can mint", async function () {
      const { instance } = await deployToken();

      await instance.mint(1);
      expect(await instance.totalSupply()).equal(totalSupply + BigInt("1"));
      expect(await instance.balanceOf(owner.address)).equal(totalSupply + BigInt("1"));
    });
  });

  describe("Burn test", function () {
    it("Allows users to burn their own tokens", async function () {
      const { instance } = await deployToken();

      await instance.transfer(addr1.address, 1000);
      expect(await instance.balanceOf(addr1.address)).to.equal(1000);
      await instance.connect(addr1).burn(1000);
      expect(await instance.balanceOf(addr1.address)).to.equal(0);
      expect(await instance.totalSupply()).equal(totalSupply - (BigInt("1000")));
    });
  });
});
```

2. output:

```
Binance Peg Dogecoin Test
  Deployment test
    ✓ Should set the correct metadata (871ms)
  Ownership test
    ✓ Should transfer ownership correctly (87ms)
    ✓ Should lose ownership if the owner renounces ownership (52ms)
  Transactions test
    ✓ Should transfer tokens between accounts
    ✓ Should be failed if sender doesn’t have enough tokens (43ms)
    ✓ Should be failed if sender transfer to zero address (46ms)
    ✓ Should be successful if sender transfer to himself (47ms)
    ✓ Should be successful if sender transfer zero amount (41ms)
    ✓ TransferFrom should need enough allowance (60ms)
  Allowance test
    ✓ Should update the allowance when approving (48ms)
  Mint test
    ✓ State _mintable should be true
    ✓ Onlyowner can mint
  Burn test
    ✓ Allows users to burn their own tokens (51ms)


13 passing (1s)
```

11.2 External Functions Check Points

1. File: contracts/BEP20TokenImplementation.sol

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

contract: BEP20TokenImplementation is Context, IBEP20, Initializable

Index	Function	Visibility	StateMutability	Permission Check	IsUserInterface	Unit Test	Notes
1	initialize(string,string,uint8,uint256,bool,address)	public					OnlyOnce
2	renounceOwnership()	public		onlyOwner		Passed	
3	transferOwnership(address)	public		onlyOwner		Passed	
4	mintable()	external	view			Passed	
5	getOwner()	external	view			Passed	
6	decimals()	external	view			Passed	
7	symbol()	external	view			Passed	
8	name()	external	view			Passed	
9	totalSupply()	external	view			Passed	
10	balanceOf(address)	external	view			Passed	
11	transfer(address,uint256)	external			Yes	Passed	

Index	Function	Visibility	StateMutability	Permission Check	IsUserInterface	Unit Test	Notes
12	allowance(address,address)	external	view			Passed	
13	approve(address,uint256)	external			Yes	Passed	
14	transferFrom(address,address,uint256)	external			Yes	Passed	
15	increaseAllowance(address,uint256)	public			Yes	Passed	
16	decreaseAllowance(address,uint256)	public			Yes	Passed	
17	mint(uint256)	public		onlyOwner		Passed	
18	burn(uint256)	public			Yes		

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