SimpsonsInu Token

AUDIT REPORT

Version 1.0.0

Serial No. 20230526000120 23

Presented by Fairyproof

May 26, 2023

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the SimpsonsInu Token project.

Audit Start Time:

May 24, 2023

Audit End Time:

May 26, 2023

Audited Source File's Address:

https://bscscan.com/address/0x5cc97dAb7bc2c01556FbE3d06a09b8C559Dff7d5#code

The goal of this audit is to review SimpsonsInu's solidity implementation for its Token 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 SimpsonsInu 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://thesimpsonsinu.com/

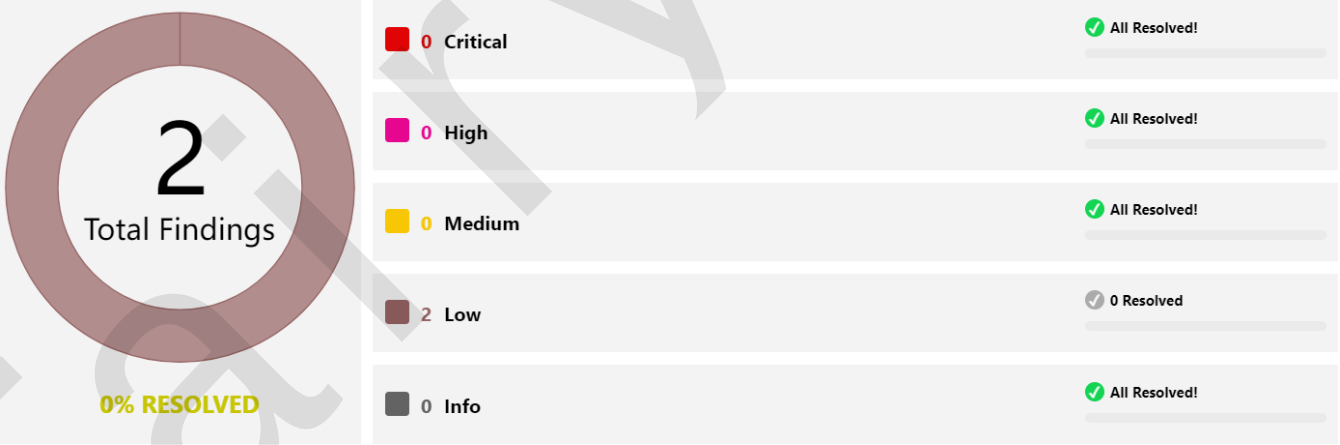
Whitepaper:https://thesimpsonsinu.com/wp-content/uploads/2023/05/WhitePaperV1.pdf

Source Code: https://bscscan.com/address/0x5cc97dAb7bc2c01556FbE3d06a09b8C559Dff7d5#code

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

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2023052600012023	Fairyproof Security Team	May 24, 2023 - May 26, 2023	Low Risk



Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, two issues of low-severity were uncovered. The SimpsonsInu team acknowledged all the issues.

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 SimpsonsInu

The Simpsons Inu is a brand new BSC Project for 2023 that focuses on bringing the latest developmental dAPPs as well as being a great meme focused community token.

The above description is quoted from relevant documents of SimpsonsInu.

04. Major functions of audited code

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

- Blockchain: BSC
- Token Standard: ERC20
- Token Address: 0x5cc97dAb7bc2c01556FbE3d06a09b8C559Dff7d5
- Token Name: The Simpsons Inu
- Token Symbol: SIMPSONSINU
- Decimals: 9
- Current Supply: 407917405321925985343791
- Max Supply: 420690000000000000000000
- Burnable: Yes
- Taxable: Yes

Note:

SimpsonsInu charges three kinds of fees: tax fee, market fee and buyback fee. All the fees are charged in token transfers and the total fee rate is 9%. The token that is sent to specified UniswapPairs is charged by 3% tax fee, 3% buyback fee and 5% market fee, and other token transfers charge 0% tax fee, 0% buyback fee and 9% market fee.

Tokens that are charged as buyback fees are burned.

Tokens that are charged as market fees are exchanged to BNBs and sent to a specific address.

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 found some issues, for more details please refer to [FP-1,FP-2] in "09. Issue description".

08. List of issues by severity

Index	Title	Issue/Risk	Severity	Status
FP-1	Possibly Transferring Tokens to Zero Address	Code Improvement	Low	Acknowledged
FP-2	Incorrect Algorithm in reflectionFromToken	Design Vulnerability	Low	Acknowledged

09. Issue descriptions

[FP-1] Possibly Transferring Tokens to Zero Address

Code Improvement Low Acknowledged

Issue/Risk: Code Improvement

Description:

The `_transfer` function doesn't check if `to != address(0)` holds true therefore tokens may be transferred to the zero address.

Recommendation:

Consider adding a requirement to check if it sends tokens to the zero address.

Update/Status:

The SIMPSONSINU team has acknowledged the issue.

[FP-2] Incorrect Algorithm in reflectionFromToken

Design Vulnerability | Low | Acknowledged

Issue/Risk: Design Vulnerability

Description:

The `reflectionFromToken` function uses `_taxFee`, `_marketingFee` and `_buybackFee` to calculate `tAmount`. However the value of these variables changes when `_tokenTransfer` changes and `reflectionFromToken` gets a result dependent on `_tokenTransfer`.

Recommendation:

Consider using parameters passed onto the function to calculate rather than using these variables to calculate.

Update/Status:

The SIMPSONSINU team has acknowledged the issue.

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. SimpsonsInu.t.js

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

describe("Unit test of SimpsonsInu token", () => {
    let owner, user1, user2, users;
    let simpsonsInu;

    const tokenName = "The Simpsons Inu";
    const tokenSymbol = "SIMPSONSINU";
    const tokenDecimals = 9;
    const tokenTotalSupply = ethers.utils.parseUnits("420690000000000", tokenDecimals);
    const marketingWallet = "0x1478fDbC46F432e6c48866bCC7a21285C00f31Ed";
    const uniswapV2Router = "0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D";
    const tradingEnabled = false;
    const swapTokensAtAmount = tokenTotalSupply.div(5000);
    const pinklock = "0x407993575c91ce7643a4d4cCACc9A98c36eE1BBE";
    const dead = "0x000000000000000000000000000000000000dead";
    const maxUint256 =
ethers.BigNumber.from("0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff")
;

    const ZERO_ADDRESS = ethers.constants.AddressZero;


    beforeEach(async () =>{
        [owner, user1, user2, ...users] = await ethers.getSigners();
        const simpsonsInu_contract = await ethers.getContractFactory('SIMPSONSINU', owner);
        simpsonsInu = await simpsonsInu_contract.deploy();
    });

    describe("Token init unit test", () => {
        it("Initial state should equal with the params of constructor", async () => {
            expect(await simpsonsInu.name()).to.be.equal(tokenName);
            expect(await simpsonsInu.symbol()).to.be.equal(tokenSymbol);
            expect(await simpsonsInu.decimals()).to.be.equal(tokenDecimals);
            expect(await simpsonsInu.owner()).to.be.equal(owner.address);
            expect(await simpsonsInu.totalSupply()).to.be.equal(tokenTotalSupply);

            expect(await simpsonsInu.marketingWallet()).to.be.equal(marketingWallet);
            expect(await simpsonsInu.uniswapV2Router()).to.be.equal(uniswapV2Router);
            expect(await simpsonsInu.tradingEnabled()).to.be.equal(tradingEnabled);
            expect(await simpsonsInu.swapTokensAtAmount()).to.be.equal(swapTokensAtAmount);

            expect(await
simpsonsInu.balanceOf(owner.address)).to.be.equal(tokenTotalSupply);
        });

        it("Initial state of _isExcluded and _isExcludedFromFees", async () => {
```

```
            expect(await
simpsonsInu.isExcludedFromReward(simpsonsInu.address)).to.be.equal(true);
            expect(await simpsonsInu.isExcludedFromReward(dead)).to.be.equal(true);
            expect(await simpsonsInu.isExcludedFromReward(pinklock)).to.be.equal(true);
            expect(await simpsonsInu.isExcludedFromReward(await
simpsonsInu.uniswapV2Pair())).to.be.equal(true);

            expect(await simpsonsInu.isExcludedFromFee(owner.address)).to.be.equal(true);
            expect(await simpsonsInu.isExcludedFromFee(dead)).to.be.equal(true);
            expect(await simpsonsInu.isExcludedFromFee(pinklock)).to.be.equal(true);
            expect(await
simpsonsInu.isExcludedFromFee(simpsonsInu.address)).to.be.equal(true);
        });
    });

    describe("Owner and OnlyOwner unit test", () => {
        it("Account permissions only controled by owner", async () => {
            await
expect(simpsonsInu.connect(user1).excludeFromReward(user1.address)).to.be.revertedWith("Own
able: caller is not the owner");
            await
expect(simpsonsInu.connect(user1).enableTrading()).to.be.revertedWith("Ownable: caller is
not the owner");
            await expect(simpsonsInu.connect(user1).excludeFromFees(user1.address,
true)).to.be.revertedWith("Ownable: caller is not the owner");
            await
expect(simpsonsInu.connect(user1).renounceOwnership()).to.be.revertedWith("Ownable: caller
is not the owner");
        });
        it("renounceOwnership should emit the event", async () => {
            await expect(simpsonsInu.renounceOwnership()).to.emit(
                simpsonsInu, "OwnershipTransferred"
            ).withArgs(owner.address, ethers.constants.AddressZero);
        });

        it("transfer should failed before enableTrading", async () => {
            await simpsonsInu.transfer(user1.address, 10000);
            await expect(simpsonsInu.connect(user1).transfer(user2.address,
10000)).to.be.revertedWith("Trading is not enabled yet");
        });
    });

    describe("Approve and transferFrom unit test", () => {
        it("Approve should change allowance and change state", async () => {
            expect(await
simpsonsInu.allowance(owner.address,user1.address)).to.be.equal(0);

            await expect(simpsonsInu.approve(user1.address, 10000)).to.be.emit(
                simpsonsInu, "Approval"
            ).withArgs(owner.address,user1.address, 10000);
            expect(await
simpsonsInu.allowance(owner.address,user1.address)).to.be.equal(10000);
```

```
            await expect(simpsonsInu.increaseAllowance(user1.address, 10000)).to.be.emit(
                simpsonsInu, "Approval"
            ).withArgs(owner.address,user1.address, 20000);
            expect(await
simpsonsInu.allowance(owner.address,user1.address)).to.be.equal(20000);

            await expect(simpsonsInu.decreaseAllowance(user1.address, 10000)).to.be.emit(
                simpsonsInu, "Approval"
            ).withArgs(owner.address,user1.address, 10000);
            expect(await
simpsonsInu.allowance(owner.address,user1.address)).to.be.equal(10000);
        });

        it("Transfer from should change allowance", async () => {
            await expect(simpsonsInu.connect(user1).transferFrom(owner.address,
user1.address, 1000)).to.be.reverted;
            await simpsonsInu.connect(owner).approve(user1.address, 10000);
            await expect(simpsonsInu.connect(user1).transferFrom(owner.address,
user1.address, 1000)).to.be.emit(
                simpsonsInu, "Transfer"
            ).withArgs(owner.address,user1.address, 1000);
            expect(await simpsonsInu.allowance(owner.address,
user1.address)).to.be.equal(10000 - 1000);
        });
    });

    describe("Reflection unit test", function() {
        it("reflectionFromToken and tokenFromReflection ", async () => {
            const max = ethers.constants.MaxUint256;
            const rTotal = max.sub(max.mod(tokenTotalSupply));
            const rate = rTotal.div(tokenTotalSupply);

            let tAmount = ethers.BigNumber.from("1000000");
            let rAmount = rate.mul(tAmount);

            expect(await simpsonsInu.reflectionFromToken(tAmount,
false)).to.equal(rAmount);
            expect(await simpsonsInu.reflectionFromToken(tAmount, true)).to.equal(rAmount);

            // Transfer change `_taxFee`、`_marketingFee`、`_buybackFee`
            await simpsonsInu.enableTrading();
            await simpsonsInu.transfer(user1.address, 10000);
            await simpsonsInu.connect(user1).transfer(user2.address, 10000)

            expect(await simpsonsInu.reflectionFromToken(tAmount,
false)).to.equal(rAmount);
            expect(await simpsonsInu.reflectionFromToken(tAmount, true)).to.equal(rAmount);

            expect(await simpsonsInu.tokenFromReflection(rAmount)).to.equal(tAmount);
        });
    });

    describe("Transfer unit test", () => {
```

```
        it("Transfer to zero address should be failed", async () => {
            await simpsonsInu.enableTrading();
            await expect(simpsonsInu.transfer(ZERO_ADDRESS,
100)).to.be.revertedWith("BEP20: transfer to the zero address");
        });

        it("Transfer zero value should be failed", async () => {
            expect(await simpsonsInu.balanceOf(user1.address)).to.be.equal(0);
            await expect(simpsonsInu.connect(user1).transfer(user2.address,
0)).to.be.revertedWith(
                "Transfer amount must be greater than zero"
            );
        });

        it("Transfer token beyond balance should be failed", async () => {
            await simpsonsInu.enableTrading();
            await
expect(simpsonsInu.connect(user1).transfer(user2.address,10)).to.be.reverted;
        });

        it("Transfer from address of from excludeFee and to excludeFee", async () => {
            expect(await simpsonsInu.balanceOf(user1.address)).to.equal(0);
            expect(await simpsonsInu.balanceOf(user2.address)).to.equal(0);

            const value = ethers.utils.parseUnits("10000", 9);
            await simpsonsInu.transfer(user1.address, value);
            expect(await simpsonsInu.balanceOf(user1.address)).to.equal(value);

            await simpsonsInu.excludeFromFees(user2.address, true);
            await simpsonsInu.connect(user1).transfer(user2.address, value)
            expect(await simpsonsInu.balanceOf(user2.address)).to.equal(value);
        });

        it("Common Transfer", async () => {
            expect(await simpsonsInu.balanceOf(user1.address)).to.equal(0);
            expect(await simpsonsInu.balanceOf(user2.address)).to.equal(0);

            const value = ethers.utils.parseUnits("10000", 9);
            await simpsonsInu.transfer(user1.address, value);

            await simpsonsInu.enableTrading();
            await simpsonsInu.connect(user1).transfer(user2.address, value)
            const marketingFee = value.mul(9).div(100);
            expect(await simpsonsInu.balanceOf(user2.address)).to.equal(value -
marketingFee);
        });

        it("Transfer with reward and without reward", async () => {
            const value = ethers.utils.parseUnits("10000", 9);
            await simpsonsInu.excludeFromReward(user2.address);

            await simpsonsInu.transfer(user1.address, value);
            await simpsonsInu.transfer(user2.address, value);
```

```
            const user1BalanceBefore = await simpsonsInu.balanceOf(user1.address);
            const user2BalanceBefore = await simpsonsInu.balanceOf(user2.address);

            const uniswapV2Pair = simpsonsInu.uniswapV2Pair();
            await simpsonsInu.enableTrading();
            await simpsonsInu.transfer(users[0].address, ethers.utils.parseUnits("1000000",
9));
            // make rFee
            await simpsonsInu.connect(users[0]).transfer(uniswapV2Pair,
ethers.utils.parseUnits("1000000", 9));

            expect(await simpsonsInu.balanceOf(user1.address)).to.gt(user1BalanceBefore);
            expect(await
simpsonsInu.balanceOf(user2.address)).to.equal(user2BalanceBefore);
        });

        it("Transfer with swap", async () => {
            const router = await ethers.getContractAt('IUniswapV2Router02',
uniswapV2Router);
            await simpsonsInu.approve(router.address, tokenTotalSupply.div(10));
            await router.addLiquidityETH(
                simpsonsInu.address,
                tokenTotalSupply.div(10),
                0,
                0,
                owner.address,
                maxUint256,
                {value: ethers.utils.parseUnits("100", 18)}
            );

            const swapTokensAtAmount = await simpsonsInu.swapTokensAtAmount();
            await simpsonsInu.transfer(user1.address, tokenTotalSupply.div(2));
            await simpsonsInu.enableTrading();

            await simpsonsInu.connect(user1).transfer(user2.address,
tokenTotalSupply.div(2));
            const balance = await simpsonsInu.balanceOf(simpsonsInu.address);
            expect(balance).gt(swapTokensAtAmount);

            const tokenBalanceOfDeadBefore = await simpsonsInu.balanceOf(dead);
            const wethBalanceOfMarket = await ethers.provider.getBalance(marketingWallet);

            expect(tokenBalanceOfDeadBefore).to.equal(0);
            expect(wethBalanceOfMarket).to.equal(0);

            const uniswapV2Pair = simpsonsInu.uniswapV2Pair();
            await simpsonsInu.transfer(uniswapV2Pair, 10000);

            expect(await simpsonsInu.balanceOf(dead)).to.gt(tokenBalanceOfDeadBefore);
            expect(await
ethers.provider.getBalance(marketingWallet)).to.gt(wethBalanceOfMarket);

        });
```

```
    });
});
```

2. UnitTesthHardhatOutput.md

```
  Unit test of SimpsonsInu token
    Token init unit test
      ✓ Initial state should equal with the params of constructor
      ✓ Initial state of _isExcluded and _isExcludedFromFees
    Owner and OnlyOwner unit test
      ✓ Account permissions only controled by owner (39ms)
      ✓ renounceOwnership should emit the event
      ✓ transfer should failed before enableTrading
    Approve and transferFrom unit test
      ✓ Approve should change allowance and change state
      ✓ Transfer from should change allowance
    Reflection unit test
      1) reflectionFromToken and tokenFromReflection
    Transfer unit test
      2) Transfer to zero address should be failed
      ✓ Transfer zero value should be failed
      ✓ Transfer token beyond balance should be failed
      ✓ Transfer from address of from excludeFee and to excludeFee
      ✓ Common Transfer
      ✓ Transfer with reward and without reward (50ms)
      ✓ Transfer with swap (127ms)


  13 passing (4s)
  2 failing
```

11.2 External Functions Check Points

1. SimpsonsInu.sol_output.md

File: contracts/SimpsonsInu.sol

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

contract: SIMPSONSINU is Context, IERC20, Ownable

Index	Function	Visibility	StateMutability	Permission Check	IsUserInterface	Unit Test	Notes

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