Zedxion ZEDXION Token

AUDIT REPORT

Version 1.0.0

Serial No. 20230120000022014

Presented by Fairyproof

January 20, 2023

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Zedxion ZEDXION project.

Audit Start Time:

January 18, 2023

Audit End Time:

January 19, 2023

Audited Source File's Address:

https://etherscan.io/address/0xfbc4f3f645c4003a2e4f4e9b51077d2daa9a9341#code

https://bscscan.com/address/0xfbc4f3f645c4003a2e4f4e9b51077d2daa9a9341#code

https://tronscan.io/#/contract/TMNTn2uFAHhkGE3uuM84rhfRRjt6ry9xnL/code

The goal of this audit is to review Zedxion's solidity implementation for its ZEDXION 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 Zedxion 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://www.zedxion.io/en_US/

Source Code:

https://etherscan.io/address/0xfbc4f3f645c4003a2e4f4e9b51077d2daa9a9341#code

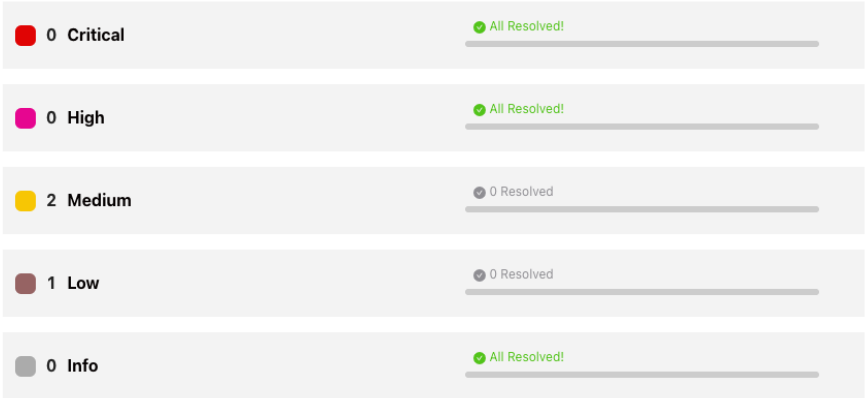
https://bscscan.com/address/0xfbc4f3f645c4003a2e4f4e9b51077d2daa9a9341#code

https://tronscan.io/#/contract/TMNTn2uFAHhkGE3uuM84rhfRRjt6ry9xnL/code

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

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2023012000022014	Fairyproof Security Team	Jan 18, 2023 - Jan 19, 2023	Medium Risk



Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, two issues of medium-severity and one issue of low-severity were uncovered. The Zedxion 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 Zedxion

Zedxion offers a comprehensive solution to the major problems faced by the traditional, fiat-driven monetary system. Building a crypto powered ecosystem comprising Zedxion Token.

The above description is quoted from relevant documents of Zedxion.

04. Major functions of audited code

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

- Blockchain: Multi Blockchain
- Token Standard: ERC20
- Token Address: 0xfbc4f3f645c4003a2e4f4e9b51077d2daa9a9341 (Ethereum)
- Token Address: 0xfbc4f3f645c4003a2e4f4e9b51077d2daa9a9341 (BNB Chain)
- Token Address: TMNTn2uFAHhkGE3uuM84rhfRRjt6ry9xnL (Tron)
- Token Name: Zedxion
- Token Symbol: ZEDXION
- Decimals: 18
- Current Supply: 4,746,558,037
- Max Supply: No Cap
- Burnable: Yes
- Mintable: Yes

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 found one issue, for more details please refer to [FP-3] in "09. Issue description".

- 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 found one issue, for more details please refer to [FP-2] in "09. Issue description".

- 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
FP-2	Owner Transfer Issue	Access Control	Medium	Acknowledged
FP-3	Redundant State Variables	Design Vulnerability	Low	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 Zedxion team prefers to keep it now and will improve the code in the future.

[FP-2] Owner Transfer Issue

Access Control | Medium | Acknowledged

Issue/Risk: Access Control

Description:

The Zedxion contract deployed at 0xFbC4f3f645C4003a2E4F4e9b51077d2DaA9a9341 on the BNB chain has two private `_owner` variables, one defined in the `Ownable` contract and the other defined in the `ZedxionBEP20` contract. And `ZedxionBEP20` inherits `Ownable`. All `owner` related operations are authorized by the `_owner` variable in the `Ownable` contract. All mint and burn operations are authorized by the `_owner` in the `ZedxionBEP20` contract and the the `_owner` cannot be changed since it is also the deployer of the contract. Therefore after calling the functions to transfer the `owner`'s right, the contract's `owner` address will be shown as zero. But the `owner` 's right hasn't been really transferred. This will confuse users.

Recommendation:

Since the contracts have been deployed and cannot be changed, consider not transferring the owner's right or disallowing the transfer of the owner's right to avoid confusing users.

Update/Status:

The Zedxion team has acknowledged this issue.

[FP-3] Redundant State Variables

Design Vulnerability | Low | Acknowledged

Issue/Risk: Design Vulnerability

Description:

The `Zedxion` contract deployed at TMNTn2uFAHhkGE3uuM84rhfRRjt6ry9xnL on Tron has the following redundant variables:

```
uint256 private _totalSupply;
uint256 private _burnToken;
uint256 private _totalMintableTokens;
address private _owner;
```

These variables have been defined in their child contracts and they are not used in `Zedxion`. They are redundant.

Recommendation:

In general a child contract shall not define variables that have the same names as the ones defined in its father contract. Consider removing these redundant variables.

Update/Status:

The Zedxion team has acknowledged this 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.

- Note that the current owner cannot be transferred, and owner permissions need to be carefully kept to prevent loss or disclosure of permissions.

11. Appendices

11.1 Unit Test

1. Zedxion_eth.t.js

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

describe("Zedxion Token deployed on Ethereum unit test", function () {
    let owner,user1,user2,users;
    let instance;
    const init_supply_unit = ethers.constants.WeiPerEther;
    async function deployTokens() {
        const Zedxion = await ethers.getContractFactory("Zedxion");
        instance = await Zedxion.deploy("Zedxion","ZEDXION",18,10);
    }

    beforeEach(async () =>{
        [owner, user1, user2, ...users] = await ethers.getSigners();
        await deployTokens();
```

```
16      });
17
18      describe("init state test", () => {
19          it("init state check", async () => {
20              expect(await instance.burnedTokens()).to.be.equal(0);
21              expect(await
    instance.balanceOf(owner.address)).to.be.equal(init_supply_unit.mul(10));
22              expect(await
    instance.totalSupply()).to.be.equal(init_supply_unit.mul(10));
23          });
24      });
25
26      describe("Mint test", () => {
27          it("Receiver must be owner", async () => {
28              await expect(instance.mint(user1.address,100)).to.be.revertedWith(
29                  "ERC20: mint not allowed this address"
30              );
31          });
32          it("Msg.sender must owner", async () => {
33              await
    expect(instance.connect(user1).mint(owner.address,100)).to.be.revertedWith(
34                  "ERC20: mint not allowed to this address"
35              );
36          });
37          it("Mint should change state", async () => {
38              await instance.mint(owner.address,1);
39              expect(await
    instance.balanceOf(owner.address)).to.be.equal(init_supply_unit.mul(11));
40              expect(await
    instance.totalSupply()).to.be.equal(init_supply_unit.mul(11));
41          });
42
43          it("Mint should have a cap", async () => {
44              let ten = ethers.BigNumber.from("10");
45              let amount = ten.pow(69 - 18).sub(10);
46              await instance.mint(owner.address,amount);
47              expect(await instance.totalSupply()).to.be.equal(ten.pow(69));
48              assert(false,"mint no cap");
49          });
50      });
51
52      describe("Burn test", () => {
53          it("Target must be owner", async () => {
54              await expect(instance.burn(user1.address,1)).to.be.revertedWith(
55                  "ERC20: burn not allowed to this address"
56              );
57          });
58          it("Msg.sender must owner", async () => {
```

```
59              await
    expect(instance.connect(user1).burn(owner.address,1)).to.be.revertedWith(
60                  "ERC20: burn not allowed to this address"
61              );
62          });
63          it("Burn should change state", async () => {
64              await instance.burn(owner.address,1);
65              let burn_tokens = await instance.burnedTokens();
66              let total_supply = await instance.totalSupply();
67              expect(await
    instance.balanceOf(owner.address)).to.be.equal(init_supply_unit.mul(9));
68              expect(total_supply).to.be.equal(init_supply_unit.mul(9));
69              expect(burn_tokens).to.be.equal(init_supply_unit);
70          });
71      });
72
73      describe("Approval test", () => {
74          it("approve should change state and emit event", async () => {
75              await
    expect(instance.connect(user1).approve(user2.address,100)).to.be.emit(
76                  instance,"Approval"
77              ).withArgs(user1.address,user2.address,100);
78              expect(await
    instance.allowance(user1.address,user2.address)).to.be.equal(100);
79          });
80      });
81
82      describe("Transfer test",  () => {
83          it("Transfer to zero should be failed", async () => {
84              await
    expect(instance.transfer(ethers.constants.AddressZero,0)).to.be.rejectedWith(
85                  "ERC20: transfer to the zero address"
86              );
87          });
88
89          it("Transfer zero token should be successfully", async () => {
90              await instance.transfer(user2.address,0);
91          });
92
93          it("Transfer should change balance", async () => {
94              await instance.transfer(user1.address,100);
95              expect(await instance.balanceOf(user1.address)).to.be.equal(100);
96              expect(await
    instance.balanceOf(owner.address)).to.be.equal(init_supply_unit.mul(10).sub(100));
97              expect(await
    instance.totalSupply()).to.be.equal(init_supply_unit.mul(10));
98          });
99
100         it("Transfer beyond balance should be failed", async () => {
```

```
            await instance.transfer(user1.address,100);
            await
expect(instance.connect(user1).transfer(user2.address,101)).to.be.rejectedWith(
                "ERC20: transfer amount exceeds balance"
            );
        });

        it("TransferFrom should transfer token", async () => {
            await instance.transfer(user1.address,10000);
            await instance.connect(user1).approve(user2.address,10000);
            await
instance.connect(user2).transferFrom(user1.address,user2.address,10000);
            expect(await instance.balanceOf(user2.address)).to.be.equal(10000);
            expect(await instance.balanceOf(user1.address)).to.be.equal(0);
            expect(await
instance.totalSupply()).to.be.equal(init_supply_unit.mul(10));

        });

        it("TransferFrom to zero address should be failed", async () => {
            await
expect(instance.transferFrom(user1.address,ethers.constants.AddressZero,100)).to.b
e.reverted;
        });

        it("TransferFrom beyond approval should be failed", async () => {
            await instance.transfer(user1.address,10000);
            await instance.connect(user1).approve(owner.address,1000)
            await
expect(instance.transferFrom(user1.address,user2.address,1002)).to.be.revertedWith
(
                "ERC20: transfer amount exceeds allowance"
            );
        });

        it("TransferFrom beyond balance should be failed", async () => {
            await instance.transfer(user1.address,900);
            await instance.connect(user1).approve(owner.address,1000)
            await
expect(instance.transferFrom(user1.address,user2.address,950)).to.be.rejectedWith(
                "ERC20: transfer amount exceeds balance"
            );
        });

        it("TransferFrom should change approval", async () => {
            await instance.transfer(user1.address,1000);
            await instance.connect(user1).approve(owner.address,1000);
            expect(await
instance.allowance(user1.address,owner.address)).to.be.equal(1000);
```

```
            await instance.transferFrom(user1.address,user2.address,1000);
            expect(await instance.balanceOf(user1.address)).to.be.equal(0);
            expect(await instance.allowance(user1.address,owner.address)).to.be.equal(0);
        });
    });
});

```

2. Zedxion_bsc.t.js

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

describe("Zedxion Token deployed on BNB Chain unit test", function () {
    let owner,user1,user2,users;
    let instance;
    const init_supply_unit = ethers.constants.WeiPerEther;
    async function deployTokens() {
        const ZedxionBEP20 = await ethers.getContractFactory("ZedxionBEP20");
        instance = await ZedxionBEP20.deploy("Zedxion","ZEDXION",18,10);
    }

    beforeEach(async () =>{
        [owner, user1, user2, ...users] = await ethers.getSigners();
        await deployTokens();
    });

    describe("init state test", () => {
        it("init state check", async () => {
            expect(await instance.totalMitable()).to.be.equal(init_supply_unit.mul(10));
            expect(await instance.burnedTokens()).to.be.equal(0);
            expect(await instance.balanceOf(owner.address)).to.be.equal(init_supply_unit.mul(10));
            expect(await instance.totalSupply()).to.be.equal(init_supply_unit.mul(10));
        });
    });

    describe("Confusing owner test", () => {
        it("Contract should have only owner", async () => {
            let first_slot = await ethers.provider.getStorageAt(instance.address,0);
            let first_owner_str = "0x" + first_slot.substring(first_slot.length - 40);
```

```
            let first_owner = ethers.utils.getAddress(first_owner_str);
            let ninth_slot = await ethers.provider.getStorageAt(instance.address,8);
            let second_owner_str = "0x" + ninth_slot.substring(ninth_slot.length - 42,ninth_slot.length -2);
            let second_owner = ethers.utils.getAddress(second_owner_str);
            assert(first_owner == owner.address,"unmatched first owner");
            assert(second_owner == owner.address,"unmatched second owner");
            assert(false,"Has two owner");
        });

        it("Mint should be failed address after renounceOwnership", async () => {
            expect(await instance.getOwner()).to.be.equal(owner.address);
            await instance.renounceOwnership();
            expect(await instance.getOwner()).to.be.equal(ethers.constants.AddressZero);
            await expect(instance.mint(owner.address,100)).to.be.revertedWith(
                "BEP20: mint not allowed this address"
            );
        });
    });

    describe("Mint test", () => {
        it("Receiver must be owner", async () => {
            await expect(instance.mint(user1.address,100)).to.be.revertedWith(
                "BEP20: mint not allowed this address"
            );
        });
        it("Msg.sender must owner", async () => {
            await expect(instance.connect(user1).mint(owner.address,100)).to.be.revertedWith(
                "BEP20: mint not allowed to this address"
            );
        });
        it("Mint should change state", async () => {
            await instance.mint(owner.address,1);
            expect(await instance.totalMitable()).to.be.equal(init_supply_unit.mul(11));
            expect(await instance.balanceOf(owner.address)).to.be.equal(init_supply_unit.mul(11));
            expect(await instance.totalSupply()).to.be.equal(init_supply_unit.mul(11));
        });
        it("Mint should have a cap", async () => {
            let ten = ethers.BigNumber.from("10");
            let amount = ten.pow(69 - 18).sub(10);
            await instance.mint(owner.address,amount);
            expect(await instance.totalSupply()).to.be.equal(ten.pow(69));
            assert(false,"mint no cap");
```

```
        });
    });

    describe("Burn test", () => {
        it("Target must be owner", async () => {
            await expect(instance.burn(user1.address,1)).to.be.revertedWith(
                "BEP20: burn not allowed to this address"
            );
        });
        it("Msg.sender must owner", async () => {
            await
expect(instance.connect(user1).burn(owner.address,1)).to.be.revertedWith(
                "BEP20: burn not allowed to this address"
            );
        });
        it("Burn should change state", async () => {
            await instance.burn(owner.address,1);
            let mint_tokens = await instance.totalMitable();
            let burn_tokens = await instance.burnedTokens();
            let total_supply = await instance.totalSupply();
            expect(mint_tokens).to.be.equal(init_supply_unit.mul(10));
            expect(await
instance.balanceOf(owner.address)).to.be.equal(init_supply_unit.mul(9));
            expect(total_supply).to.be.equal(init_supply_unit.mul(9));
            expect(burn_tokens).to.be.equal(init_supply_unit);
            expect(mint_tokens.sub(burn_tokens)).to.be.equal(total_supply);
        });
    });

    describe("Approval test", () => {
        it("approve should change state and emit event", async () => {
            await
expect(instance.connect(user1).approve(user2.address,100)).to.be.emit(
                instance,"Approval"
            ).withArgs(user1.address,user2.address,100);
            expect(await
instance.allowance(user1.address,user2.address)).to.be.equal(100);
        });

        it("increaseAllowance should change state and emit event", async () => {
            await
expect(instance.connect(user1).increaseAllowance(user2.address,100)).to.be.emit(
                instance,"Approval"
            ).withArgs(user1.address,user2.address,100);
            expect(await
instance.allowance(user1.address,user2.address)).to.be.equal(100);
        });

    });
```

```
116
117     describe("Transfer test",  () => {
118         it("Transfer to zero should be failed", async () => {
119             await
    expect(instance.transfer(ethers.constants.AddressZero,0)).to.be.rejectedWith(
120                 "BEP20: transfer to the zero address"
121             );
122         });
123
124         it("Transfer zero token should be successfully", async () => {
125             await instance.transfer(user2.address,0);
126         });
127
128         it("Transfer should change balance", async () => {
129             await instance.transfer(user1.address,100);
130             expect(await instance.balanceOf(user1.address)).to.be.equal(100);
131             expect(await
    instance.balanceOf(owner.address)).to.be.equal(init_supply_unit.mul(10).sub(100));
132             expect(await
    instance.totalSupply()).to.be.equal(init_supply_unit.mul(10));
133         });
134
135         it("Transfer beyond balance should be failed", async () => {
136             await instance.transfer(user1.address,100);
137             await
    expect(instance.connect(user1).transfer(user2.address,101)).to.be.rejectedWith(
138                 "BEP20: transfer amount exceeds balance"
139             );
140         });
141
142         it("TransferFrom should transfer token", async () => {
143             await instance.transfer(user1.address,10000);
144             await instance.connect(user1).approve(user2.address,10000);
145             await
    instance.connect(user2).transferFrom(user1.address,user2.address,10000);
146             expect(await instance.balanceOf(user2.address)).to.be.equal(10000);
147             expect(await instance.balanceOf(user1.address)).to.be.equal(0);
148             expect(await
    instance.totalSupply()).to.be.equal(init_supply_unit.mul(10));
149
150         });
151
152         it("TransferFrom to zero address should be failed", async () => {
153             await
    expect(instance.transferFrom(user1.address,ethers.constants.AddressZero,100)).to.b
    e.reverted;
154         });
155
156         it("TransferFrom beyond approval should be failed", async () => {
```

```
            await instance.transfer(user1.address,10000);
            await instance.connect(user1).approve(owner.address,1000)
            await
expect(instance.transferFrom(user1.address,user2.address,1002)).to.be.revertedWith
(
                "BEP20: transfer amount exceeds allowance"
            );
        });

        it("TransferFrom beyond balance should be failed", async () => {
            await instance.transfer(user1.address,900);
            await instance.connect(user1).approve(owner.address,1000)
            await
expect(instance.transferFrom(user1.address,user2.address,950)).to.be.rejectedWith(
                "BEP20: transfer amount exceeds balance"
            );
        });

        it("TransferFrom should change approval", async () => {
            await instance.transfer(user1.address,1000);
            await instance.connect(user1).approve(owner.address,1000);
            expect(await
instance.allowance(user1.address,owner.address)).to.be.equal(1000);
            await instance.transferFrom(user1.address,user2.address,1000);
            expect(await instance.balanceOf(user1.address)).to.be.equal(0);
            expect(await
instance.allowance(user1.address,owner.address)).to.be.equal(0);
        });

    });

});


```

3. Zedxion_trc.t.js

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

describe("Zedxion Token deployed on tron unit test", function () {
    let owner,user1,user2,users;
    let instance;
    const init_supply = ethers.utils.parseEther("4746558037.0");
    async function deployTokens() {
        const ZedxionTron = await ethers.getContractFactory("ZedxionTron");
        instance = await ZedxionTron.deploy();
```

```
    }

    beforeEach(async () =>{
        [owner, user1, user2, ...users] = await ethers.getSigners();
        await deployTokens();
    });

    describe("init state test", () => {
        it("init state check", async () => {
            expect(await instance.getOwner()).to.be.equal(owner.address);
            expect(await instance.totalMitable()).to.be.equal(init_supply);
            expect(await instance.burnedTokens()).to.be.equal(0);
            expect(await
instance.balanceOf(owner.address)).to.be.equal(init_supply);
            expect(await instance.totalSupply()).to.be.equal(init_supply);
        });
    });

    describe("Mint test", () => {
        it("Receiver must be owner", async () => {
            await expect(instance.mint(user1.address,100)).to.be.revertedWith(
                "TRC20: mint not allowed this address"
            );
        });
        it("Msg.sender must owner", async () => {
            await
expect(instance.connect(user1).mint(owner.address,100)).to.be.revertedWith(
                "TRC20: mint not allowed to this address"
            );
        });
        it("Mint should change state", async () => {
            await instance.mint(owner.address,1);
            expect(await
instance.totalMitable()).to.be.equal(init_supply.add(ethers.constants.WeiPerEther)
);
            expect(await
instance.balanceOf(owner.address)).to.be.equal(init_supply.add(ethers.constants.We
iPerEther));
            expect(await
instance.totalSupply()).to.be.equal(init_supply.add(ethers.constants.WeiPerEther))
;
        });
        it("Mint should have a cap", async () => {
            let ten = ethers.BigNumber.from("10");
            let amount = ten.pow(69 - 18).sub(4746558037);
            await instance.mint(owner.address,amount);
            expect(await instance.totalSupply()).to.be.equal(ten.pow(69));
            assert(false,"mint no cap");
        });
```

```
52      });
53
54      describe("Burn test", () => {
55          it("Target must be owner", async () => {
56              await expect(instance.burn(user1.address,1)).to.be.revertedWith(
57                  "TRC20: burn not allowed to this address"
58              );
59          });
60          it("Msg.sender must owner", async () => {
61              await
    expect(instance.connect(user1).burn(owner.address,1)).to.be.revertedWith(
62                  "TRC20: burn not allowed to this address"
63              );
64          });
65          it("Burn should change state", async () => {
66              await instance.burn(owner.address,1);
67              let mint_tokens = await instance.totalMitable();
68              let burn_tokens = await instance.burnedTokens();
69              let total_supply = await instance.totalSupply();
70              expect(mint_tokens).to.be.equal(init_supply);
71              expect(await
    instance.balanceOf(owner.address)).to.be.equal(init_supply.sub(ethers.constants.We
    iPerEther));
72
     expect(total_supply).to.be.equal(init_supply.sub(ethers.constants.WeiPerEther));
73              expect(burn_tokens).to.be.equal(ethers.constants.WeiPerEther);
74              expect(mint_tokens.sub(burn_tokens)).to.be.equal(total_supply);
75          });
76      });
77
78      describe("Approval test", () => {
79          it("approve should change state and emit event", async () => {
80              await
    expect(instance.connect(user1).approve(user2.address,100)).to.be.emit(
81                  instance,"Approval"
82              ).withArgs(user1.address,user2.address,100);
83              expect(await
    instance.allowance(user1.address,user2.address)).to.be.equal(100);
84          });
85
86          it("increaseAllowance should change state and emit event", async () => {
87              await
    expect(instance.connect(user1).increaseAllowance(user2.address,100)).to.be.emit(
88                  instance,"Approval"
89              ).withArgs(user1.address,user2.address,100);
90              expect(await
    instance.allowance(user1.address,user2.address)).to.be.equal(100);
91          });
92
```

```
93          it("decreaseAllowance should change state and emit event", async () => {
94              await instance.connect(user1).approve(user2.address,300);
95              await
expect(instance.connect(user1).decreaseAllowance(user2.address,100)).to.be.emit(
96                  instance,"Approval"
97              ).withArgs(user1.address,user2.address,200);
98              expect(await
instance.allowance(user1.address,user2.address)).to.be.equal(200);
99          });
100     });
101
102     describe("Transfer test",  () => {
103         it("Transfer to zero should be failed", async () => {
104             await
expect(instance.transfer(ethers.constants.AddressZero,0)).to.be.rejectedWith(
105                 "ERC20: transfer to the zero address"
106             );
107         });
108
109         it("Transfer zero token should be successfully", async () => {
110             await instance.transfer(user2.address,0);
111         });
112
113         it("Transfer should change balance", async () => {
114             await instance.transfer(user1.address,100);
115             expect(await instance.balanceOf(user1.address)).to.be.equal(100);
116             expect(await
instance.balanceOf(owner.address)).to.be.equal(init_supply.sub(100));
117             expect(await instance.totalSupply()).to.be.equal(init_supply);
118         });
119
120         it("Transfer beyonds balance should be failed", async () => {
121             await instance.transfer(user1.address,100);
122             await
expect(instance.connect(user1).transfer(user2.address,101)).to.be.rejectedWith(
123                 "SafeMath: subtraction overflow"
124             );
125         });
126
127         it("TransferFrom should transfer token", async () => {
128             await instance.transfer(user1.address,10000);
129             await instance.connect(user1).approve(user2.address,10000);
130             await
instance.connect(user2).transferFrom(user1.address,user2.address,10000);
131             expect(await instance.balanceOf(user2.address)).to.be.equal(10000);
132             expect(await instance.balanceOf(user1.address)).to.be.equal(0);
133             expect(await instance.totalSupply()).to.be.equal(init_supply);
134
135         });
```

```
136
137         it("TransferFrom to zero address should be failed", async () => {
138             await
expect(instance.transferFrom(user1.address,ethers.constants.AddressZero,100)).to.b
e.rejectedWith(
139                 "ERC20: transfer to the zero address"
140             );
141         });
142
143         it("TransferFrom beyond approval should be failed", async () => {
144             await instance.transfer(user1.address,10000);
145             await instance.connect(user1).approve(owner.address,1000)
146             await
expect(instance.transferFrom(user1.address,user2.address,1002)).to.be.revertedWith
(
147                 "SafeMath: subtraction overflow"
148             );
149         });
150
151         it("TransferFrom beyond balance should be failed", async () => {
152             await instance.transfer(user1.address,900);
153             await instance.connect(user1).approve(owner.address,1000)
154             await
expect(instance.transferFrom(user1.address,user2.address,950)).to.be.rejectedWith(
155                 "SafeMath: subtraction overflow"
156             );
157         });
158
159         it("TransferFrom should change approval", async () => {
160             await instance.transfer(user1.address,1000);
161             await instance.connect(user1).approve(owner.address,1000);
162             expect(await
instance.allowance(user1.address,owner.address)).to.be.equal(1000);
163             await instance.transferFrom(user1.address,user2.address,1000);
164             expect(await instance.balanceOf(user1.address)).to.be.equal(0);
165             expect(await
instance.allowance(user1.address,owner.address)).to.be.equal(0);
166         });
167     });
168 });
169
170
```

4. output

```
Zedxion Token deployed on BNB Chain unit test
    init state test
      ✔ init state check
    Confusing owner test
      1) Contract should have only owner
      2) Mint should be failed address after renounceOwnership
    Mint test
      ✔ Receiver must be owner
      ✔ Msg.sender must owner
      ✔ Mint should change state
      3) Mint should have a cap
    Burn test
      ✔ Target must be owner
      ✔ Msg.sender must owner
      ✔ Burn should change state
    Approval test
      ✔ approve should change state and emit event
      ✔ increaseAllowance should change state and emit event
    Transfer test
      ✔ Transfer to zero should be failed
      ✔ Transfer zero token should be successfully
      ✔ Transfer should change balance
      ✔ Transfer beyond balance should be failed
      ✔ TransferFrom should transfer token
      ✔ TransferFrom to zero address should be failed
      ✔ TransferFrom beyond approval should be failed
      ✔ TransferFrom beyond balance should be failed
      ✔ TransferFrom should change approval

  Zedxion Token deployed on Ethereum unit test
    init state test
      ✔ init state check
    Mint test
      ✔ Receiver must be owner
      ✔ Msg.sender must owner
      ✔ Mint should change state
      4) Mint should have a cap
    Burn test
      ✔ Target must be owner
      ✔ Msg.sender must owner
      ✔ Burn should change state
    Approval test
      ✔ approve should change state and emit event
    Transfer test
      ✔ Transfer to zero should be failed
      ✔ Transfer zero token should be successfully
      ✔ Transfer should change balance
```

```
      ✔ Transfer beyond balance should be failed
      ✔ TransferFrom should transfer token
      ✔ TransferFrom to zero address should be failed
      ✔ TransferFrom beyond approval should be failed
      ✔ TransferFrom beyond balance should be failed
      ✔ TransferFrom should change approval

  Zedxion Token deployed on tron unit test
    init state test
      ✔ init state check
    Mint test
      ✔ Receiver must be owner
      ✔ Msg.sender must owner
      ✔ Mint should change state
      5) Mint should have a cap
    Burn test
      ✔ Target must be owner
      ✔ Msg.sender must owner
      ✔ Burn should change state
    Approval test
      ✔ approve should change state and emit event
      ✔ increaseAllowance should change state and emit event
      ✔ decreaseAllowance should change state and emit event
    Transfer test
      ✔ Transfer to zero should be failed
      ✔ Transfer zero token should be successfully
      ✔ Transfer should change balance
      ✔ Transfer beyonds balance should be failed
      ✔ TransferFrom should transfer token
      ✔ TransferFrom to zero address should be failed
      ✔ TransferFrom beyond approval should be failed
      ✔ TransferFrom beyond balance should be failed
      ✔ TransferFrom should change approval


  54 passing (4s)
  5 failing

```

11.2 External Functions Check Points

1. output.md

File: contracts/ZedxionBEP20.sol

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

contract: ZedxionBEP20 is Ownable, IBEP20

Index	Function	Visibility	Permission Check	Re-entrancy Check	Injection Check	Unit Test	Notes
1	name()	public					
2	getOwner()	public				Passed	
3	symbol()	public					
4	decimals()	public					
5	totalSupply()	public				Passed	
6	totalMitable()	public				Passed	
7	balanceOf(address)	public				Passed	
8	burnedTokens()	public				Passed	
9	transfer(address,uint256)	public				Passed	
10	transferFrom(address,address,uint256)	public				Passed	
11	allowance(address,address)	public				Passed	
12	increaseAllowance(address,uint256)	public				Passed	
13	approve(address,uint256)	public				Passed	
14	burn(address,uint256)	public	_owner			Passed	
15	mint(address,uint256)	public	_owner			Passed	
16	renounceOwnership()	public					
17	transferOwnership(address)	public					

File: contracts/ZedxionTRC20.sol

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

contract: ZedxionTron is TRC20, TRC20Detailed

Index	Function	Visibility	Permission Check	Re-entrancy Check	Injection Check	Unit Test	Notes
1	name()	public					
2	symbol()	public					
3	decimals()	public					
4	totalSupply()	public				Passed	
5	balanceOf(address)	public				Passed	
6	transfer(address,uint256)	public				Passed	
7	allowance(address,address)	public				Passed	
8	approve(address,uint256)	public				Passed	
9	transferFrom(address,address,uint256)	public				Passed	
10	increaseAllowance(address,uint256)	public				Passed	
11	decreaseAllowance(address,uint256)	public				Passed	
12	totalMitable()	public				Passed	
13	getOwner()	public				Passed	
14	burnedTokens()	public				Passed	
15	burn(address,uint256)	public	_owner			Passed	
16	mint(address,uint256)	public	_owner			Passed	

File: contracts/Zedxion.sol

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

contract: Zedxion is Context, IERC20

Index	Function	Visibility	Permission Check	Re-entrancy Check	Injection Check	Unit Test	Notes
1	name()	public					
2	symbol()	public					
3	decimals()	public					
4	totalSupply()	public					
5	balanceOf(address)	public				Passed	
6	burnedTokens()	public				Passed	
7	transfer(address,uint256)	public				Passed	
8	allowance(address,address)	public				Passed	
9	approve(address,uint256)	public				Passed	
10	transferFrom(address,address,uint256)	public				Passed	
11	burn(address,uint256)	public	_owner			Passed	
12	mint(address,uint256)	public	_owner			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