FAIRYPROOF

Tala Token Issuance

AUDIT REPORT

Version 1.0.0

Serial No. 20250911000012023

Presented by Fairyproof

September 11, 2025

01. Introduction

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

Audit Start Time:

September 10, 2025

Audit End Time:

September 11, 2025

Audited Source File's Address:

https://bscscan.com/address/0x9Cac9837155A851e3d08639a40aE5bF18d968cd3#code

https://zedscan.net/address/0x162C0e6D40c53CC7C1F286d3E397bE5418694B11?tab=contract

Audited Code's Github Repository:

https://github.com/talatoken/Tala-Smart-Contrat

Audited Code's Github Commit Number When Audit Started:

de460212586cb03a2f3710c3276aefbf610b421d

Audited Code's Github Commit Number When Audit Ended:

de460212586cb03a2f3710c3276aefbf610b421d

The goal of this audit is to review Tala'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 Tala 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://talatoken.com/

Source Code:

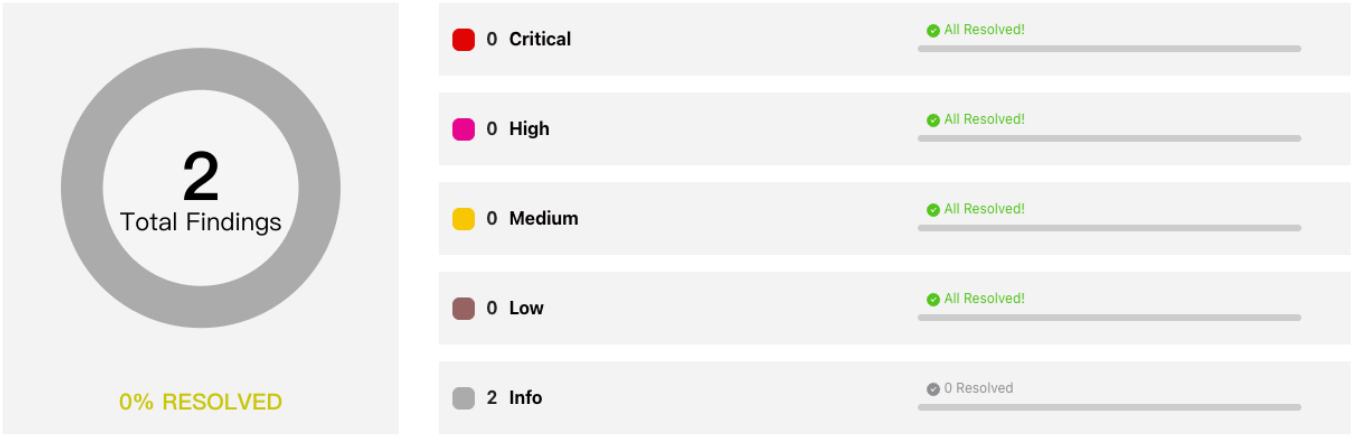
https://bscscan.com/address/0x9Cac9837155A851e3d08639a40aE5bF18d968cd3#code

https://zedscan.net/address/0x162C0e6D40c53CC7C1F286d3E397bE5418694B11?tab=contract

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

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2025091100012023	Fairyproof Security Team	Sep 10, 2025 - Sep 11, 2025	Info Risk



Summary:

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

TALA (1OZT) defines authority in asset-backed finance — secured beyond 100%, accelerated on BNB Chain, and built for DeFi. Every milestone expands trust, liquidity, and usability.

The above description is quoted from relevant documents of Tala.

04. Major functions of audited code

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

Token On BNB Chain:

- Blockchain: BNB Chain

- Token Standard: BEP-20
- Token Address: 0x9Cac9837155A851e3d08639a40aE5bF18d968cd3
- Token Name: Tala
- Token Symbol: 1ozt
- Decimals: 6
- Current Supply: 3,220,000
- Max Supply: 3,220,000
- Taxable: Yes

Note:

The Tala Token on BNB chain is taxed on transfers at a fixed rate of 0.2%.

Token On ZEDX Chain:

- Blockchain: ZEDX Chain
- Token Standard: BEP-20
- Token Address: 0x162C0e6D40c53CC7C1F286d3E397bE5418694B11
- Token Name: Tala from BNB
- Token Symbol: 1ozt
- Decimals: 6
- Current Supply: 4,989.501
- Max Supply: NoCap
- Taxable: No
- Mintable: Yes
- Burnable: Yes

Note:

The Tala Token on ZEDX chain is its bridge token on the BNB chain.The issuance of this token is controlled by the bridge contract: `0xa3F37D00C47F4B0FDd5acE310632489A21779a6b`.

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 some issues, for more details please refer to [FP-1,FP-2] 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 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. List of issues by severity

Index	Title	Issue/Risk	Severity	Status
FP-1	Recursive Fee Collection on Fee Recipient Transfers	Code Improvement	Info	Acknowledged
FP-2	Unnecessary Zero-Amount Transfers	Code Improvement	Info	Acknowledged

09. Issue descriptions

[FP-1] Recursive Fee Collection on Fee Recipient Transfers

Code Improvement Info Acknowledged

Issue/Risk: Code Improvement

Description:

The current implementation charges fees on transfers to the feeRecipient address, creating a recursive fee collection scenario.

Details:

When fees are transferred to feeRecipient via `super._transfer(sender, feeRecipient, fee)`, the `_transfer` function is called again.
This results in an additional fee being charged on the fee transfer itself
The feeRecipient receives slightly less than the intended fee amount

Recommendation:

Update/Status:

The Tala team has known the issue.

[FP-2] Unnecessary Zero-Amount Transfers

Code Improvement Info Acknowledged

Issue/Risk: Code Improvement

Description:

The contract performs transfers even when the calculated fee amount is zero.
Details:
For very small transfer amounts, the fee calculation `(amount * FEE_PERCENTAGE) / 100000` may result in zero due to integer division
The contract still executes super._transfer(sender, feeRecipient, 0) even when fee is zero

Recommendation:

Update/Status:

The Tala team has known 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. Tala.js

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

describe("TALA", function () {
  // We define a fixture to reuse the same setup in every test.
  async function deployTALAFixture() {
    // Contracts are deployed using the first signer/account by default
    const [owner, feeRecipient, user1, user2] = await ethers.getSigners();

    const TALA = await ethers.getContractFactory("TALA");
    const tala = await TALA.deploy(feeRecipient.address, owner.address);

    return { tala, owner, feeRecipient, user1, user2 };
  }

  describe("Deployment", function () {
    it("Should set the right name and symbol", async function () {
      const { tala } = await loadFixture(deployTALAFixture);

      expect(await tala.name()).to.equal("Tala");
      expect(await tala.symbol()).to.equal("1ozt");
    });

    it("Should set the right decimals", async function () {
      const { tala } = await loadFixture(deployTALAFixture);

      expect(await tala.decimals()).to.equal(6);
    });

    it("Should set the right total supply", async function () {
      const { tala } = await loadFixture(deployTALAFixture);

      const expectedTotalSupply = 3_220_000 * 10 ** 6;
      expect(await tala.totalSupply()).to.equal(expectedTotalSupply);
    });

    it("Should set the right owner", async function () {
      const { tala, owner } = await loadFixture(deployTALAFixture);
```

```
    expect(await tala.owner()).to.equal(owner.address);
  });

  it("Should set the right fee recipient", async function () {
    const { tala, feeRecipient } = await loadFixture(deployTALAFixture);

    expect(await tala.feeRecipient()).to.equal(feeRecipient.address);
  });

  it("Should mint total supply to owner", async function () {
    const { tala, owner } = await loadFixture(deployTALAFixture);

    const expectedTotalSupply = 3_220_000 * 10 ** 6;
    expect(await tala.balanceOf(owner.address)).to.equal(expectedTotalSupply);
  });

  it("Should set the right fee percentage", async function () {
    const { tala } = await loadFixture(deployTALAFixture);

    expect(await tala.FEE_PERCENTAGE()).to.equal(200);
  });

  it("Should fail if fee recipient is zero address", async function () {
    const [owner] = await ethers.getSigners();
    const TALA = await ethers.getContractFactory("TALA");

    await expect(
      TALA.deploy(ethers.ZeroAddress, owner.address)
    ).to.be.revertedWith("Invalid fee recipient");
  });

  it("Should fail if owner is zero address", async function () {
    const [, feeRecipient] = await ethers.getSigners();
    const TALA = await ethers.getContractFactory("TALA");

    await expect(
      TALA.deploy(feeRecipient.address, ethers.ZeroAddress)
    ).to.be.revertedWith("Invalid owner address");
  });
});

describe("Fee Recipient Management", function () {
  it("Should allow owner to update fee recipient", async function () {
    const { tala, owner, user1 } = await loadFixture(deployTALAFixture);

    await tala.connect(owner).setFeeRecipient(user1.address);
    expect(await tala.feeRecipient()).to.equal(user1.address);
  });

  it("Should emit FeeRecipientUpdated event", async function () {
    const { tala, owner, user1 } = await loadFixture(deployTALAFixture);
```

```
    await expect(tala.connect(owner).setFeeRecipient(user1.address))
      .to.emit(tala, "FeeRecipientUpdated")
      .withArgs(user1.address);
  });

  it("Should revert when non-owner tries to update fee recipient", async function () {
    const { tala, user1, user2 } = await loadFixture(deployTALAFixture);

    await expect(
      tala.connect(user1).setFeeRecipient(user2.address)
    ).to.be.revertedWith("Ownable: caller is not the owner");
  });

  it("Should revert when setting fee recipient to zero address", async function () {
    const { tala, owner } = await loadFixture(deployTALAFixture);

    await expect(
      tala.connect(owner).setFeeRecipient(ethers.ZeroAddress)
    ).to.be.revertedWith("Invalid address");
  });
});

describe("Transfers with Fee", function () {
  it("Should transfer tokens with correct fee deduction", async function () {
    const { tala, owner, feeRecipient, user1 } = await loadFixture(deployTALAFixture);

    const transferAmount = ethers.parseUnits("1000", 6); // 1000 tokens
    const expectedFee = (transferAmount * 200n) / 100000n; // 0.2% fee
    const expectedAmountAfterFee = transferAmount - expectedFee;

    // Transfer tokens to user1
    await tala.connect(owner).transfer(user1.address, transferAmount);

    // Check balances
    expect(await tala.balanceOf(user1.address)).to.equal(expectedAmountAfterFee);
    expect(await tala.balanceOf(feeRecipient.address)).to.equal(expectedFee);
  });

  it("Should emit TransferWithFee event", async function () {
    const { tala, owner, user1 } = await loadFixture(deployTALAFixture);

    const transferAmount = ethers.parseUnits("1000", 6);
    const expectedFee = (transferAmount * 200n) / 100000n;
    const expectedAmountAfterFee = transferAmount - expectedFee;

    await expect(tala.connect(owner).transfer(user1.address, transferAmount))
      .to.emit(tala, "TransferWithFee")
      .withArgs(owner.address, user1.address, expectedAmountAfterFee, expectedFee);
  });

  it("Should calculate fee correctly for different amounts", async function () {
```

```
      const { tala, owner, feeRecipient, user1, user2 } = await
loadFixture(deployTALAFixture);

      // Test with 100 tokens
      const amount1 = ethers.parseUnits("100", 6);
      const expectedFee1 = (amount1 * 200n) / 100000n; // 0.2 tokens

      await tala.connect(owner).transfer(user1.address, amount1);
      expect(await tala.balanceOf(feeRecipient.address)).to.equal(expectedFee1);

      // Test with 50000 tokens
      const amount2 = ethers.parseUnits("50000", 6);
      const expectedFee2 = (amount2 * 200n) / 100000n; // 100 tokens

      await tala.connect(owner).transfer(user2.address, amount2);
      expect(await tala.balanceOf(feeRecipient.address)).to.equal(expectedFee1 +
expectedFee2);
    });

    it("Should handle transfers between users (not from owner)", async function () {
      const { tala, owner, feeRecipient, user1, user2 } = await
loadFixture(deployTALAFixture);

      // First, transfer some tokens to user1
      const initialAmount = ethers.parseUnits("10000", 6);
      await tala.connect(owner).transfer(user1.address, initialAmount);

      // Reset fee recipient balance for clear testing
      const feeRecipientBalance = await tala.balanceOf(feeRecipient.address);

      // Now user1 transfers to user2
      const transferAmount = ethers.parseUnits("1000", 6);
      const expectedFee = (transferAmount * 200n) / 100000n;
      const expectedAmountAfterFee = transferAmount - expectedFee;

      await tala.connect(user1).transfer(user2.address, transferAmount);

      expect(await tala.balanceOf(user2.address)).to.equal(expectedAmountAfterFee);
      expect(await tala.balanceOf(feeRecipient.address)).to.equal(feeRecipientBalance +
expectedFee);
    });

    it("Should handle small amounts that result in zero fee", async function () {
      const { tala, owner, feeRecipient, user1 } = await loadFixture(deployTALAFixture);

      // Transfer very small amount (1 token = 1e6 units, fee = 0.2%)
      const smallAmount = 1n; // 1 unit (smallest possible)
      const expectedFee = (smallAmount * 200n) / 100000n; // Should be 0
      const expectedAmountAfterFee = smallAmount - expectedFee;

      const initialFeeBalance = await tala.balanceOf(feeRecipient.address);
```

```
      await tala.connect(owner).transfer(user1.address, smallAmount);

      expect(await tala.balanceOf(user1.address)).to.equal(expectedAmountAfterFee);
      expect(await tala.balanceOf(feeRecipient.address)).to.equal(initialFeeBalance +
expectedFee);
    });
  });

  describe("Standard ERC20 Functions", function () {
    it("Should approve and transferFrom correctly with fees", async function () {
      const { tala, owner, feeRecipient, user1, user2 } = await
loadFixture(deployTALAFixture);

      const transferAmount = ethers.parseUnits("1000", 6);
      const expectedFee = (transferAmount * 200n) / 100000n;
      const expectedAmountAfterFee = transferAmount - expectedFee;

      // Owner approves user1 to spend tokens
      await tala.connect(owner).approve(user1.address, transferAmount);

      // user1 transfers from owner to user2
      await tala.connect(user1).transferFrom(owner.address, user2.address,
transferAmount);

      expect(await tala.balanceOf(user2.address)).to.equal(expectedAmountAfterFee);
      expect(await tala.balanceOf(feeRecipient.address)).to.equal(expectedFee);
    });

    it("Should handle allowance correctly", async function () {
      const { tala, owner, user1 } = await loadFixture(deployTALAFixture);

      const approvalAmount = ethers.parseUnits("5000", 6);

      await tala.connect(owner).approve(user1.address, approvalAmount);
      expect(await tala.allowance(owner.address, user1.address)).to.equal(approvalAmount);
    });
  });

  describe("Edge Cases", function () {
    it("Should handle maximum transfer amount", async function () {
      const { tala, owner, feeRecipient, user1 } = await loadFixture(deployTALAFixture);

      const maxAmount = await tala.balanceOf(owner.address); // All tokens
      const expectedFee = (maxAmount * 200n) / 100000n;
      const expectedAmountAfterFee = maxAmount - expectedFee;

      await tala.connect(owner).transfer(user1.address, maxAmount);

      expect(await tala.balanceOf(user1.address)).to.equal(expectedAmountAfterFee);
      expect(await tala.balanceOf(feeRecipient.address)).to.equal(expectedFee);
      expect(await tala.balanceOf(owner.address)).to.equal(0);
    });
```

```
    it("Should revert on insufficient balance", async function () {
      const { tala, user1, user2 } = await loadFixture(deployTALAFixture);

      const transferAmount = ethers.parseUnits("100", 6);

      await expect(
        tala.connect(user1).transfer(user2.address, transferAmount)
      ).to.be.revertedWith("ERC20: transfer amount exceeds balance");
    });
  });

  describe("Constants", function () {
    it("Should have correct constant values", async function () {
      const { tala } = await loadFixture(deployTALAFixture);

      expect(await tala.FEE_PERCENTAGE()).to.equal(200);
      expect(await tala.TOTAL_SUPPLY()).to.equal(3_220_000 * 10 ** 6);
    });
  });
});
```

2. TalaUnitTestOutPut

```
  TALA
    Deployment
      ✔ Should set the right name and symbol (927ms)
      ✔ Should set the right decimals
      ✔ Should set the right total supply
      ✔ Should set the right owner
      ✔ Should set the right fee recipient
      ✔ Should mint total supply to owner
      ✔ Should set the right fee percentage
      ✔ Should fail if fee recipient is zero address
      ✔ Should fail if owner is zero address
    Fee Recipient Management
      ✔ Should allow owner to update fee recipient
      ✔ Should emit FeeRecipientUpdated event
      ✔ Should revert when non-owner tries to update fee recipient
      ✔ Should revert when setting fee recipient to zero address
    Transfers with Fee
      ✔ Should transfer tokens with correct fee deduction
      ✔ Should emit TransferWithFee event
      ✔ Should calculate fee correctly for different amounts
      ✔ Should handle transfers between users (not from owner)
      ✔ Should handle small amounts that result in zero fee
```

```
    Standard ERC20 Functions
      ✔ Should approve and transferFrom correctly with fees
      ✔ Should handle allowance correctly
    Edge Cases
      ✔ Should handle maximum transfer amount
      ✔ Should revert on insufficient balance
    Constants
      ✔ Should have correct constant values


  23 passing (992ms)
```

11.2 External Functions Check Points

1. contracts/TALA.sol

File: contracts/TALA.sol

contract: TALA is ERC20, Ownable

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

Index	Function	StateMutability	Modifier	Param Check	IsUserInterface	Unit Test	Miscellaneous
1	decimals()	pure				Passed	
2	setFeeRecipient(address)		`onlyOwner`		False	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