FAIRYPROOF

WikiCat Token

AUDIT REPORT

Version 1.0.0

Serial No. 2023011200012014

Presented by Fairyproof

January 12, 2023

01. Introduction

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

Audit Start Time:

January 10, 2023

Audit End Time:

January 11, 2023

Audited Source File's Address:

https://bscscan.com/token/0x6ec90334d89dbdc89e08a133271be3d104128edb#code

The goal of this audit is to review WikiCat'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 WikiCat 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:

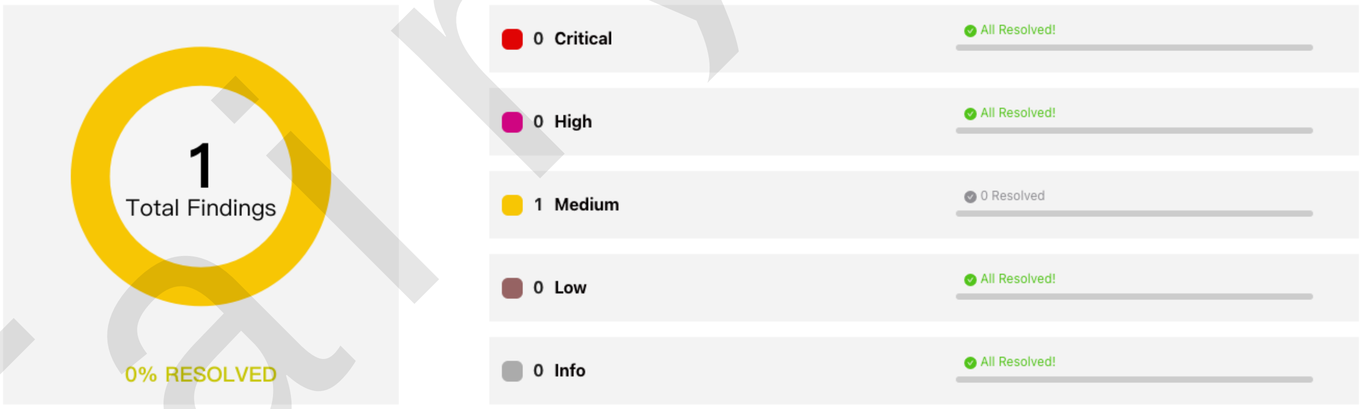
Website:https://wikicatcoin.com/

Source Code: https://bscscan.com/token/0x6ec90334d89dbdc89e08a133271be3d104128edb#code

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

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2023011200012014	Fairyproof Security Team	Jan 10, 2023 - Jan 11, 2023	Medium Risk



Summary:

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

02. About Fairyproof

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

03. Introduction to WikiCat

Wiki Cat was created as a tutorial token by Sir Mapy for SMC DAO.

With the ownership of the token renounced to a strong and vibrant community, the goal of the project is to establish a NFT focused club called Wiki Cat Club, where the users can hold, buy and sell unique NFTs in p2p way.

The above description is quoted from relevant documents of WikiCat.

04. Major functions of audited code

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

- Blockchain: BNB Chain
- Token Standard: BEP-20
- Token Address: 0x6Ec90334d89dBdc89E08A133271be3d104128Edb
- Token Name: WIKI CAT
- Token Symbol: WKC
- Decimals: 18
- Max Supply: 1,000,000,000,000
- Burnable: Yes

Note:

The owner address has been transferred to the black hole address `0x0000000000000000000000000000000000000001`. This has revoked the owner's privilege.
Charges are applied when tokens are transferred: 1% of the transfer amount is burned directly, and another 1% of the transfer amount is sent to `FeeAddress`.

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-1] 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	Allowance Deduction Error in In `transferFrom`	Implementation Vulnerability	Medium	Acknowledged

09. Issue descriptions

[FP-1] Allowance Deduction Error In `transferFrom`

Implementation Vulnerability Medium Acknowledged

Issue/Risk: Implementation Vulnerability

Description:

In the function `transferFrom`, when `txFee > 0` or `burnFee>0`, the `_value` amount of tokens of the address `_from` is deducted. The spender's deductible allowance for the operation is implemented as `_value - fee`. Based on this implementation, `transferFrom` cannot deduct all the allowance and the spender can transfer tokens whose amount exceeds the approved allowance.

Recommendation:

Consider moving the following code `allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);` from where it is to after the code `balances[_from] = balances[_from].sub(_value);`

Update/Status:

The Wiki Cat 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.

- N/A

11. Appendices

11.1 Unit Test

1. WKC-test.js

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

describe("WIKI CAT Token unit test", function () {
    let owner,user1,user2,users;
    let instance;
    let MAX_SUPPLY = 100000000;
    let supply = ethers.utils.parseEther("" + MAX_SUPPLY);
    let address_one = ethers.constants.AddressZero.substring(0,41) + "1"

    async function deployTokens() {
        const CoinToken = await ethers.getContractFactory("CoinToken");
        instance = await CoinToken.deploy("WIKI 
CAT","WKC",18,MAX_SUPPLY,1,1,owner.address,owner.address);
    }

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

    describe("Init status test", () => {
        it("Token metadata check", async () => {
            expect(await instance.totalSupply()).to.be.equal(supply);
            expect(await instance.balanceOf(owner.address)).to.be.equal(supply);
            expect(await instance.paused()).to.be.false;
            expect(await instance.txFee()).to.be.equal(1);
            expect(await instance.burnFee()).to.be.equal(1);
            expect(await instance.FeeAddress()).to.be.equal(owner.address);
        });
    });

    describe("Burn test", () => {
        it("Burn should change supply and balance", async () => {
            await instance.transfer(user1.address,1000000);
            expect(await instance.totalSupply()).to.be.equal(supply);
            expect(await instance.balanceOf(user1.address)).to.be.equal(1000000);
            await instance.connect(user1).burn(100);
            expect(await instance.totalSupply()).to.be.equal(supply.sub(100));
            expect(await instance.balanceOf(user1.address)).to.be.equal(1000000 - 
100);
        });

        it("Burn should be failed while holding insufficient tokens", async () => 
{
            await expect(instance.connect(user1).burn(100)).to.be.reverted;
        });
    });

    describe("Mint test", () => {
        it("Mint should change supply and balance", async () => {
            await instance.mint(user1.address,1000000);
            expect(await instance.totalSupply()).to.be.equal(supply.add(1000000));
            expect(await instance.balanceOf(user1.address)).to.be.equal(1000000);
        });

        it("Burn should be failed while not owner", async () => {
            await 
expect(instance.connect(user1).mint(user1.address,1000000)).to.be.reverted;
        });
    });

    describe("updateFee test", () => {
        it("updateFee should change status", async () => {
            let args = [
                2,3,user2.address
            ]
            await instance.updateFee(...args);
            expect(await instance.txFee()).to.be.equal(2);
            expect(await instance.burnFee()).to.be.equal(3);
```

```
      expect(await instance.FeeAddress()).to.be.equal(user2.address);
    });

    it("updateFee should be failed while not owner", async () => {
      let args = [
        2,3,user2.address
      ]
      await
expect(instance.connect(user1).updateFee(...args)).to.be.reverted;
    });
  });

  describe("Change 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("increaseApproval should change state and emit event", async () => {
      await
expect(instance.connect(user1).increaseApproval(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("decreaseApproval should change state and emit event", async () => {
      await instance.connect(user1).approve(user2.address,100);
      await
expect(instance.connect(user1).decreaseApproval(user2.address,20)).to.be.emit(
        instance,"Approval"
      ).withArgs(user1.address,user2.address,100-20);
      await
expect(instance.connect(user1).decreaseApproval(user2.address,100)).to.be.emit(
        instance,"Approval"
      ).withArgs(user1.address,user2.address,0);
    });
  });

  describe("transferOwnership test", () => {
    it("transferOwnership should be failed while not owner", async () => {
      await instance.transferOwnership(address_one);
      expect(await instance.owner()).to.be.equal(address_one);
```

```
            await 
expect(instance.transferOwnership(user2.address)).to.be.reverted;
        });
    })

    describe("blackListAddress test", () => {
        it("blackListAddress should be failed while not owner", async () => {
            await instance.transferOwnership(address_one);
            await 
expect(instance.blackListAddress(user2.address,true)).to.be.reverted;
        });
    })

    describe("pause and unpause test", () => {
        it ("pause and unpause should change state", async () => {
            await instance.pause()
            expect(await instance.paused()).to.be.true;
            await instance.unpause()
            expect(await instance.paused()).to.be.false;
        })
        it("pause should be failed while not owner", async () => {
            await expect(instance.connect(user1).pause()).to.be.reverted;
            await instance.pause()
            await expect(instance.connect(user1).unpause()).to.be.reverted;
        });
    });

    describe("Transfer test",  () => {
        it("Transfer to zero should be failed", async () => {
            await 
expect(instance.transfer(ethers.constants.AddressZero,0)).to.be.reverted;
        });

        it("Transfer zero token should be successfully", async () => {
            await instance.mint(user1.address,100);
            await instance.connect(user1).transfer(user2.address,0);
        });
        it("Transfer beyond balance should be failed", async () => {
            await instance.mint(user1.address,100);
            expect(await instance.balanceOf(user1.address)).to.be.equal(100);
            await 
expect(instance.connect(user1).transfer(user2.address,101)).to.be.reverted;
        });
        it("Transfer has no fee while from is FeeAddress", async () => {
            await expect(instance.transfer(user1.address,10000)).to.be.emit(
                instance,"Transfer"
            ).withArgs(owner.address,user1.address,10000);
            expect(await 
instance.balanceOf(owner.address)).to.be.equal(supply.sub(10000));
```

```
            expect(await instance.balanceOf(user1.address)).to.be.equal(10000);
            expect(await instance.totalSupply()).to.be.equal(supply);
        });
        it("Transfer should has tax while from is not feeAddress", async () => {
            await instance.mint(user1.address,10000);
            expect(await instance.balanceOf(owner.address)).to.be.equal(supply);
            expect(await instance.balanceOf(user1.address)).to.be.equal(10000);
            expect(await instance.totalSupply()).to.be.equal(supply.add(10000));
            await instance.connect(user1).transfer(user2.address,1000);
            expect(await instance.balanceOf(user1.address)).to.be.equal(10000 - 1000);
            let dev_fee = 1000 * 1 / 100;
            let burn_fee = 1000 * 1 / 100;
            expect(await instance.balanceOf(user2.address)).to.be.equal(1000 - dev_fee - burn_fee);
            expect(await instance.balanceOf(owner.address)).to.be.equal(supply.add(dev_fee));
            expect(await instance.totalSupply()).to.be.equal(supply.add(10000).sub(burn_fee));
        });

        it("Transfer from blacklist should be failed", async () => {
            await instance.mint(user1.address,1000000);
            await instance.blackListAddress(user1.address,true);
            await expect(instance.connect(user1).transfer(user2.address,100)).to.be.reverted;
        });

        it("Transfer to self should has tax", async () => {
            await instance.mint(user1.address,10000);
            expect(await instance.balanceOf(owner.address)).to.be.equal(supply);
            expect(await instance.balanceOf(user1.address)).to.be.equal(10000);
            expect(await instance.totalSupply()).to.be.equal(supply.add(10000));
            await instance.connect(user1).transfer(user1.address,1000);
            let dev_fee = 1000 * 1 / 100;
            let burn_fee = 1000 * 1 / 100;
            expect(await instance.balanceOf(user1.address)).to.be.equal(10000 - dev_fee - burn_fee);
            expect(await instance.balanceOf(owner.address)).to.be.equal(supply.add(dev_fee));
            expect(await instance.totalSupply()).to.be.equal(supply.add(10000).sub(burn_fee));
        });

        it("TransferFrom should transfer token", async () => {
            await instance.mint(user1.address,10000);
            await instance.connect(user1).approve(user2.address,10000);
            await instance.connect(user2).transferFrom(user1.address,user2.address,10000);
            let dev_fee = 10000 * 1 / 100;
            let burn_fee = 10000 * 1 / 100;
            expect(await instance.balanceOf(user2.address)).to.be.equal(10000 - dev_fee - burn_fee);
            expect(await instance.balanceOf(user1.address)).to.be.equal(0);
            expect(await instance.balanceOf(owner.address)).to.be.equal(supply.add(dev_fee));
            expect(await instance.totalSupply()).to.be.equal(supply.add(10000).sub(burn_fee));
        });

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

        it("TransferFrom beyond approval should be failed", async () => {
            await instance.connect(user1).approve(owner.address,1000)
            await expect(instance.transferFrom(user1.address,user2.address,1002)).to.be.reverted;
        });

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

        it("TransferFrom should change approval", async () => {
            await instance.mint(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);
        });

        it("TransferFrom from blacklist should be failed", async () => {
            await instance.mint(user1.address,1000);
            await instance.connect(user1).approve(owner.address,1000);
            await instance.blackListAddress(user1.address,true);
            await expect(instance.transferFrom(user1.address,user2.address,1000)).to.be.reverted;
        });

```

```
    });
});


```

2. UnitTestResult

```
  WIKI CAT Token unit test
    Init status test
      ✔ Token metadata check (61ms)
    Burn test
      ✔ Burn should change supply and balance (65ms)
      ✔ Burn should be failed while holding insufficient tokens
    Mint test
      ✔ Mint should change supply and balance
      ✔ Burn should be failed while not owner
    updateFee test
      ✔ updateFee should change status
      ✔ updateFee should be failed while not owner
    Change Approval test
      ✔ approve should change state and emit event
      ✔ increaseApproval should change state and emit event
      ✔ decreaseApproval should change state and emit event
    transferOwnership test
      ✔ transferOwnership should be failed while not owner
    blackListAddress test
      ✔ blackListAddress should be failed while not owner
    pause and unpause test
      ✔ pause and unpause should change state
      ✔ pause should be failed while not owner
    Transfer test
      ✔ Transfer to zero should be failed
      ✔ Transfer zero token should be successfully
      ✔ Transfer beyond balance should be failed
      ✔ Transfer has no fee while from is FeeAddress
      ✔ Transfer should has tax while from is not feeAddress (72ms)
      ✔ Transfer from blacklist should be failed (42ms)
      ✔ Transfer to self should has tax (61ms)
      ✔ TransferFrom should transfer token (63ms)
      ✔ TransferFrom to zero address should be failed
      ✔ TransferFrom beyond approval should be failed
      ✔ TransferFrom beyond balance should be failed
      1) TransferFrom should change approval
      2) TransferFrom from blacklist should be failed


  25 passing (3s)
  2 failing

  1) WIKI CAT Token unit test
       Transfer test
         TransferFrom should change approval:
     AssertionError: Expected "20" to be equal 0
      at Context.<anonymous> (test/WKC-test.js:221:81)

  2) WIKI CAT Token unit test
       Transfer test
         TransferFrom from blacklist should be failed:

      AssertionError: Expected transaction to be reverted
      + expected - actual

      -Transaction NOT reverted.
      +Transaction reverted.

```

11.2 External Functions Check Points

1. CoinToken.sol

File: contracts/CoinToken.sol

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

contract: CoinToken is PausableToken

Index	Function	Visibility	Permission Check	Re-entrancy Check	Injection Check	Unit Test	Notes
1	burn(uint256)	public				Passed	
2	updateFee(uint256,uint256,address)	public	onlyOwner			Passed	
3	mint(address,uint256)	public	onlyOwner			Passed	
4	transfer(address,uint256)	public				Passed	
5	transferFrom(address,address,uint256)	public				Failed	Approval calculation error
6	approve(address,uint256)	public				Passed	
7	increaseApproval(address,uint)	public				Passed	
8	decreaseApproval(address,uint)	public				Passed	
9	blackListAddress(address,bool)	public	onlyOwner				
10	pause()	public	onlyOwner	Passed		Passed	
11	unpause()	public	onlyOwner	Passed		Passed	
12	transferOwnership(address)	public	onlyOwner	Passed		Passed	
13	balanceOf(address)	public				Passed	View
14	allowance(address,address)	public				Passed	View

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