FAIRYPROOF

BITX Token

AUDIT REPORT

Version 1.0.0

Serial No. 2023120100012014

Presented by Fairyproof

December 1, 2023

01. Introduction

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

Audit Start Time:

December 1, 2023

Audit End Time:

December 1, 2023

Audited Source File's Address:

https://bscscan.com/token/0x668935b74cd1683c44dc3e5dfa61a6e0b219b913#code

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

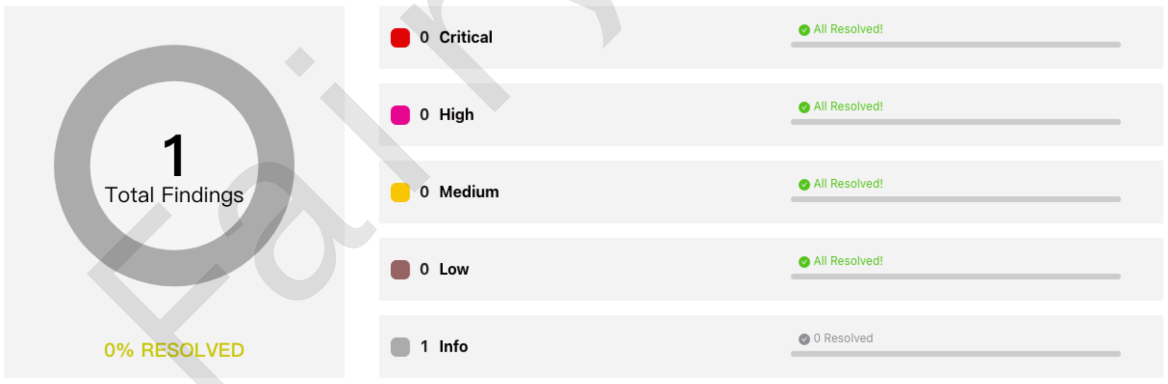
Website:https://www.bitxglobal.org/

Source Code: https://bscscan.com/token/0x668935b74cd1683c44dc3e5dfa61a6e0b219b913#code

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

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2023120100012014	Fairyproof Security Team	Dec 1, 2023 - Dec 1, 2023	Info Risk



Summary:

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

02. About Fairyproof

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

03. Introduction to BITX

BitX Global Exchange Discover why stands as the definitive choice for cryptocurrency trading and investment

The above description is quoted from relevant documents of BITX.

04. Major functions of audited code

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

- Blockchain: BSC
- Token Standard: BEP20
- Token Address: 0x668935b74cD1683c44Dc3E5dfa61A6E0B219B913
- Token Name: BitX Exchange
- Token Symbol: BitX Exchange
- Decimals: 9
- Current Supply: 420,000,000,000,000,000
- Max Supply: 420,000,000,000,000,000
- Taxable: Yes

Note:

This token is deployed on the BNB chain. Taxes are changed for the token transactions. For a token exchange transaction, the seller is charged by 5% and the buyer is charged by 5%. For a regular transfer, 10% of the transaction amount is charged.
Note: the charged taxes are kept in its contract. When the quantity of tokens kept in the contract exceeds a certain threshold, the tokens will be swapped for BNBs and sent to a specified address. In addition, the contract's owner rights have been revoked.

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	Lack of Restriction for Caller of `receive`	Code Improvement	Info	Acknowledged

09. Issue descriptions

[FP-1] Lack of Restriction for Caller of `receive`

Code Improvement Info Acknowledged

Issue/Risk: Code Improvement

Description:

The implementation only allows an address specified by `router` to receive BNBs, therefore it should have `msg.sender == address(router)` in the `receive` function to prevent unexpected transfers of BNBs.

Recommendation:

N/A.

Update/Status:

The BITX 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.

- N/A

11. Appendices

11.1 Unit Test

1. BITX.t.js

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

describe("BITX Token Unit Test", function () {

    const marketingWallet = "0x3eef41c57070115792A3867F1d71b4d002BBecAA";
    const deadWallet =  "0x000000000000000000000000000000000000dEaD";
    const tTotal = ethers.parseUnits("420" + "0".repeat(6),18);
    const init_liquid = ethers.parseUnits("10000000",18);

    async function deployTokenFixture() {
        const [owner, alice,bob,...users] = await ethers.getSigners();

        const WETH9 = await ethers.getContractFactory("WETH9");
        const weth = await WETH9.deploy();

        const UniswapV2Factory = await
ethers.getContractFactory("UniswapV2Factory");
        const factory = await UniswapV2Factory.deploy(users[0].address);
        const UniswapV2Router02 = await
ethers.getContractFactory("UniswapV2Router02");
        const router = await
UniswapV2Router02.deploy(factory.target,weth.target);

        const BITXToken = await ethers.getContractFactory("BITX");
        const instance = await BITXToken.deploy(router.target);

        await instance.approve(router.target,ethers.MaxUint256);
        await
router.addLiquidityETH(instance.target,init_liquid,1,1,owner.address,9876543210,{
            value:ethers.parseEther("10")
        });

        return {owner,alice,bob,users,instance,factory,router,weth};
    }

    it("meta and init supply unit test", async () => {
        const {instance,owner,factory,router} = await
loadFixture(deployTokenFixture);

        expect(await instance.name()).eq("BitX Exchange");
        expect(await instance.symbol()).eq("BITX");
        expect(await instance.decimals()).eq(9);
        expect(await instance.totalSupply()).eq(tTotal);
        expect(await instance.balanceOf(owner.address)).eq(tTotal - init_liquid);
        expect(await instance.router()).eq(router.target);
        let weth = await router.WETH();
        let pair = await factory.getPair(weth,instance.target);
        expect(await instance.pair()).eq(pair);
        expect(await instance.balanceOf(pair)).eq(init_liquid);
    });

    it("Transfer and rate uint test", async () => {
        const {instance,owner,users,alice,bob,router,weth} = await
loadFixture(deployTokenFixture);
        // check rate
        let custom_t_amount = ethers.parseEther("1.0");
        let custom_r_amount = await
instance.reflectionFromToken(custom_t_amount,false);

        let value = ethers.parseEther("20000000");
        await instance.transfer(alice,value);
        expect(await instance.balanceOf(alice)).eq(value);
        let pair = await instance.pair();
        expect(await instance.balanceOf(pair)).eq(init_liquid);
        expect(await
instance.tokenFromReflection(custom_r_amount)).eq(custom_t_amount);

        // alice transfer token to bob
        await instance.connect(alice).transfer(bob.address,value);
        let fee = value / ethers.getBigInt(10);
        expect(await instance.balanceOf(instance.target)).eq(fee);
        expect(await instance.balanceOf(bob.address)).eq(value - fee);
        expect(await instance.balanceOf(pair)).eq(init_liquid);
        expect(await
instance.tokenFromReflection(custom_r_amount)).eq(custom_t_amount);

        // bob transfer token to alice
        await instance.connect(bob).transfer(alice.address,value - fee);
        expect(await instance.balanceOf(pair)).eq(init_liquid + fee);
        let new_fee = (value - fee) * ethers.getBigInt(1) / ethers.getBigInt(10);
        expect(await instance.balanceOf(alice.address)).eq(value -fee - new_fee);
        expect(await instance.balanceOf(instance.target)).eq(new_fee);
        let swapTokensAtAmount = await instance.swapTokensAtAmount();
        assert(swapTokensAtAmount < new_fee,"unexpected");
        expect(await
instance.tokenFromReflection(custom_r_amount)).eq(custom_t_amount);

        // bob sell token
        let amountIn = (value -fee - new_fee) / getBigInt(10000);
        new_fee = amountIn * getBigInt(5) / getBigInt(100);
        await instance.connect(alice).approve(router.target,ethers.MaxUint256);
        await
router.connect(alice).swapExactTokensForETHSupportingFeeOnTransferTokens(
            amountIn,
            1,
            [instance.target,weth.target],
            alice.address,
            9876543210
        );
        expect(await instance.balanceOf(instance.target)).eq(new_fee);
        expect(await
instance.tokenFromReflection(custom_r_amount)).eq(custom_t_amount);
        let balance_alice = await instance.balanceOf(alice.address);

        assert(swapTokensAtAmount > new_fee,"unexpected");
        // alice buy token
        await
router.connect(alice).swapExactETHForTokensSupportingFeeOnTransferTokens(
            1,
            [weth.target,instance.target],
            users[2].address,
            9876543210,
            {
                value:ethers.parseEther("1")
            }
        );
        let balance_this = await instance.balanceOf(instance.target);
        new_fee = balance_this - new_fee;
        let balance = await instance.balanceOf(users[2].address);
        expect(balance).eq(new_fee * ethers.getBigInt(19));
        expect(await
instance.tokenFromReflection(custom_r_amount)).eq(custom_t_amount);

        expect(await instance.balanceOf(owner.address)).eq(tTotal - init_liquid -
value);
        expect(await instance.balanceOf(alice.address)).eq(balance_alice);
    });

    it("rescueBNB unit test", async () => {
        const {instance,owner,alice,bob} = await loadFixture(deployTokenFixture);
        await owner.sendTransaction({
            from:owner.address,
            to:instance.target,
            value:ethers.parseEther("1.0")
        });
        expect(await
ethers.provider.getBalance(instance.target)).eq(ethers.parseEther("1.0"));
        await
expect(instance.connect(alice).rescueBNB(ethers.parseEther("1.0"))).to.revertedWi
th(
            "Ownable: caller is not the owner"
        );
    });
});
```

11.2 External Functions Check Points

1. BITX_output.md

File: contracts/safemoon/BITX.sol

contract: BITX is Context, IBEP20, 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	name()	pure				Passed	
2	symbol()	pure				Passed	
3	decimals()	pure				Passed	
4	totalSupply()	view				Passed	
5	balanceOf(address)	view				Passed	
6	allowance(address,address)	view					
7	approve(address,uint256)				Yes		
8	transferFrom(address,address,uint256)				Yes		
9	increaseAllowance(address,uint256)				Yes		
10	decreaseAllowance(address,uint256)				Yes		
11	transfer(address,uint256)				Yes	Passed	
12	isExcludedFromReward(address)	view					
13	reflectionFromToken(uint256,bool)	view				Passed	
14	tokenFromReflection(uint256)	view				Passed	
15	excludeFromReward(address)		`onlyOwner`				
16	excludeFromFee(address)		`onlyOwner`				
17	isExcludedFromFee(address)	view					
18	rescueBNB(uint256)		`onlyOwner`			Passed	

Index	Function	StateMutability	Modifier	Param Check	IsUserInterface	Unit Test	Miscellaneous
19	receive()	payable				Passed	
20	owner()	view					
21	renounceOwnership()		`onlyOwner`				
22	transferOwnership(address)		`onlyOwner`				

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