FAIRYPROOF

Bobcoin Token

AUDIT REPORT

Version 1.0.0

Serial No. 2023042200012018

Presented by Fairyproof

April 22, 2023

01. Introduction

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

Audit Start Time:

April 22, 2023

Audit End Time:

April 22, 2023

Audited Source File's Addresses:

https://polygonscan.com/token/0x590eb2920486486c2d9bb3eb651f73b81df87bcf

https://etherscan.io/token/0x590eb2920486486c2d9bb3eb651f73b81df87bcf

https://bscscan.com/token/0x590eb2920486486c2d9bb3eb651f73b81df87bcf

https://snowtrace.io/token/0x590eb2920486486c2d9bb3eb651f73b81df87bcf

The goal of this audit is to review bobcoin'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 Bobcoin 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:

Source Code:

https://polygonscan.com/token/0x590eb2920486486c2d9bb3eb651f73b81df87bcf

https://etherscan.io/token/0x590eb2920486486c2d9bb3eb651f73b81df87bcf

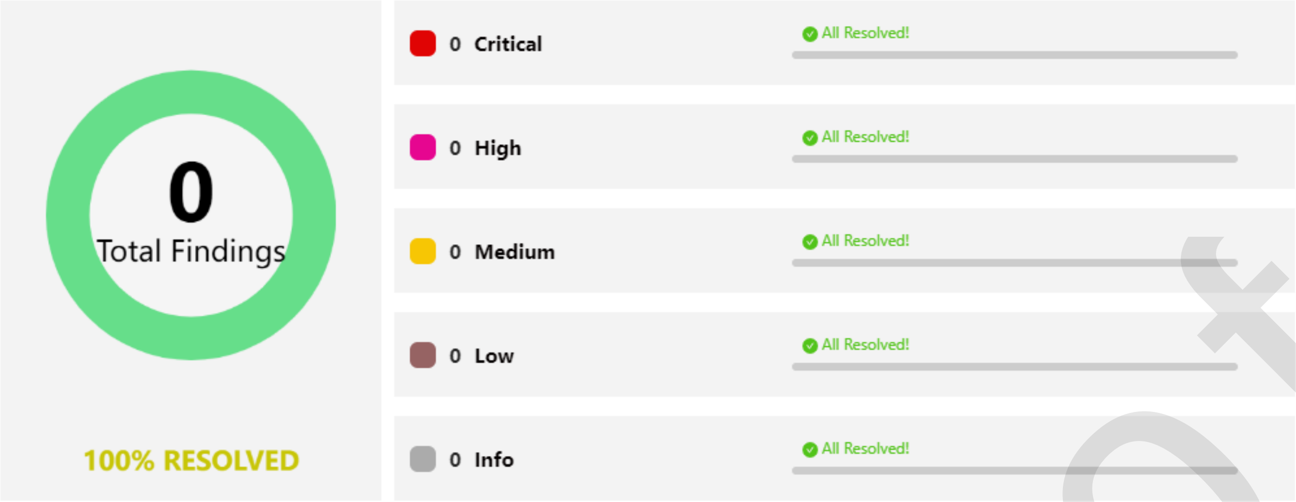
https://bscscan.com/token/0x590eb2920486486c2d9bb3eb651f73b81df87bcf

https://snowtrace.io/token/0x590eb2920486486c2d9bb3eb651f73b81df87bcf

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

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2023042200012018	Fairyproof Security Team	Apr 22, 2023 - Apr 22, 2023	Passed



0
Total Findings

100% RESOLVED

🟥	0 Critical	✓ All Resolved!
🟪	0 High	✓ All Resolved!
🟨	0 Medium	✓ All Resolved!
🟫	0 Low	✓ All Resolved!
⬜	0 Info	✓ All Resolved!

Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, no issues were uncovered.

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 bobcoin Token

bobcoin is an ERC-20 token that will be deployed on multiple blockchains

04. Major functions of audited code

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

- Blockchain: Ethereum, Polygon, AVAX, BNB Chain
- Token Standard: ERC-20
- Token Address:

  Ethereum: 0x590eb2920486486c2d9bb3eb651f73b81df87bcf

  Polygon: 0x590eb2920486486c2d9bb3eb651f73b81df87bcf

  BNB Chain: 0x590eb2920486486c2d9bb3eb651f73b81df87bcf

  Snow: 0x590eb2920486486c2d9bb3eb651f73b81df87bcf
- Token Name: bobcoin
- Token Symbol: BOBC
- Burnable: Yes

Note:

This is the token (launched), tokens have been distributed and the original contract has been put in a Timelock Contract, secured by a multi-sig.

Token holders can burn their own tokens. And authorized addresses can burn another address' tokens as well.

05. Coverage of issues

The issues that the Fairyproof team covered when conducting the audit include but are not limited to the following ones:

- Access Control
- Admin Rights
- Arithmetic Precision
- Code Improvement
- Contract Upgrade/Migration
- Delete Trap
- Design Vulnerability
- DoS Attack
- EOA Call Trap
- Fake Deposit
- Function Visibility
- Gas Consumption
- Implementation Vulnerability
- Inappropriate Callback Function
- Injection Attack
- Integer Overflow/Underflow
- IsContract Trap

- Miner's Advantage
- Misc
- Price Manipulation
- Proxy selector clashing
- Pseudo Random Number
- Re-entrancy Attack
- Replay Attack
- Rollback Attack
- Shadow Variable
- Slot Conflict
- Token Issuance
- Tx.origin Authentication
- Uninitialized Storage Pointer

06. Severity level reference

Every issue in this report was assigned a severity level from the following:

Critical severity issues need to be fixed as soon as possible.

High severity issues will probably bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Informational is not an issue or risk but a suggestion for code improvement.

07. Major areas that need attention

Based on the provided source code the Fairyproof team focused on the possible issues and risks related to the following functions or areas.

- Function Implementation

We checked whether or not the functions were correctly implemented.
We didn't find issues or risks in these functions or areas at the time of writing.

- Access Control

We checked each of the functions that could modify a state, especially those functions that could only be accessed by owner or administrator
We didn't find issues or risks in these functions or areas at the time of writing.

- Token Issuance & Transfer

We examined token issuance and transfers for situations that could harm the interests of holders.
We didn't find issues or risks in these functions or areas at the time of writing.

- State Update

We checked some key state variables which should only be set at initialization.
We didn't find issues or risks in these functions or areas at the time of writing.

- Asset Security

We checked whether or not all the functions that transfer assets were safely handled.
We didn't find issues or risks in these functions or areas at the time of writing.

- Miscellaneous

We checked the code for optimization and robustness.
We didn't find issues or risks in these functions or areas at the time of writing.

08. issues by severity

- N/A

09. Issue descriptions

- N/A

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. testToken.t.js

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

describe("testToken", function () {
  let owner, addr1;
  const totalSupply = ethers.utils.parseUnits("350000000", 18);

  async function deployToken() {
    [owner, addr1] = await ethers.getSigners();
    const TestToken = await ethers.getContractFactory("testToken");
    const instance = await TestToken.deploy(350000000);
    return { instance };
  }

  describe("Deployment test", function () {
```

```
  it("Should set the correct metadata", async function () {
   const { instance } = await deployToken();

   expect(await instance.totalSupply()).equal(totalSupply);
   expect(await instance.balanceOf(owner.address)).equal(totalSupply);
   expect(await instance.name()).equal("token");
   expect(await instance.symbol()).equal("TK");
   expect(await instance.decimals()).equal(18);
  });
 });

 describe("Transactions test", function () {
  it("Should transfer tokens between accounts", async function () {
   const { instance } = await deployToken();
   const transferAmount = "5000";

   expect(await instance.transfer(addr1.address, transferAmount))
    .be.emit(instance, "Transfer").withArgs(owner.address, addr1.address,
transferAmount);
   expect(await instance.balanceOf(addr1.address)).to.equal(transferAmount);
  });

  it("Should fail if sender doesn’t have enough tokens", async function () {
   const { instance } = await deployToken();
   const initialOwnerBalance = await instance.balanceOf(owner.address);
   await expect(instance.connect(addr1).transfer(owner.address,
1)).to.be.revertedWith("ERC20: transfer amount exceeds balance");
   expect(await instance.balanceOf(owner.address)).to.equal(initialOwnerBalance);
  });
 });

 describe("Allowance test", function () {
  it("Should update the allowance when approving", async function () {
   const { instance } = await deployToken();
   const approveAmount = "1000"

   expect(await instance.approve(addr1.address, approveAmount))
    .to.be.emit(instance, "Approval").withArgs(owner.address, addr1.address,
approveAmount);
   const allowance = await instance.allowance(owner.address, addr1.address);
   expect(allowance).to.equal(approveAmount);
   // increse allowance again
   expect(await instance.increaseAllowance(addr1.address, approveAmount))
    .to.be.emit(instance, "Approval").withArgs(owner.address, addr1.address,
allowance.add(approveAmount));
   expect(await instance.allowance(owner.address,
addr1.address)).to.equal(allowance.add(approveAmount));
  });
 });

 describe("Burn test", function () {
  it("Should burn tokens correctly", async function () {
   const { instance } = await deployToken();
```

```
      const initialSupply = await instance.totalSupply();
      const burnAmount = "1000";

      await instance.burn(burnAmount);
      expect(await instance.totalSupply()).to.equal(initialSupply.sub(burnAmount));
      expect(await
instance.balanceOf(owner.address)).to.equal(initialSupply.sub(burnAmount));
    });

    it("Should burn tokens by anyone himself", async function () {
      const { instance } = await deployToken();
      const initialSupply = await instance.totalSupply();
      const burnAmount = "1000";

      await instance.transfer(addr1.address, burnAmount)
      expect(await instance.balanceOf(addr1.address)).to.equal(burnAmount);
      await instance.connect(addr1).burn(burnAmount);
      expect(await instance.totalSupply()).to.equal(initialSupply.sub(burnAmount));
      expect(await instance.balanceOf(addr1.address)).to.equal(0);
    });

    it("Should burnFrom tokens correctly", async function () {
      const { instance } = await deployToken();
      const initialSupply = await instance.totalSupply();
      const burnAmount = "1000";

      await instance.transfer(addr1.address, burnAmount)
      await instance.connect(addr1).approve(owner.address, burnAmount);
      await instance.burnFrom(addr1.address, burnAmount);
      expect(await instance.totalSupply()).to.equal(initialSupply.sub(burnAmount));
      expect(await instance.balanceOf(addr1.address)).to.equal(0);
    });
  });

  describe("Ownership test", function () {
    it("Should transfer and renounce ownership correctly", async function () {
      const { instance } = await deployToken();

      expect(await instance.owner()).to.equal(owner.address);
      await instance.transferOwnership(addr1.address);
      expect(await instance.owner()).to.equal(addr1.address);

      await instance.connect(addr1).renounceOwnership();
      expect(await instance.owner()).to.equal(ethers.constants.AddressZero);
    });
  });

  describe("claimStuckedER20 test", function () {
    it("Should allow the owner to claim stuck tokens", async function () {
      const { instance } = await deployToken();
      const StuckToken = await ethers.getContractFactory("testToken");
      const stuckTokenInstance = await StuckToken.deploy("1000000");
      await stuckTokenInstance.deployed();
      const ownerBalance = await stuckTokenInstance.balanceOf(owner.address);
      expect(ownerBalance).to.equal(ethers.utils.parseUnits("1000000", 18));
      // Send stuck tokens to the testToken contract
      await stuckTokenInstance.transfer(instance.address, "5000");
      expect(await
stuckTokenInstance.balanceOf(owner.address)).to.equal(ownerBalance.sub("5000"));
      expect(await stuckTokenInstance.balanceOf(instance.address)).to.equal("5000");
      // Claim stuck tokens
      await instance.claimStuckedER20(stuckTokenInstance.address);
      expect(await stuckTokenInstance.balanceOf(instance.address)).to.equal("0");
      // Check the owner's balance after claiming
      expect(await
stuckTokenInstance.balanceOf(owner.address)).to.equal(ethers.utils.parseUnits("1000000",
18));
    });

    it("Should only owner can claim stuck tokens", async function () {
      const { instance } = await deployToken();
      const StuckToken = await ethers.getContractFactory("testToken");
      const stuckTokenInstance = await StuckToken.deploy("1000000");
      await stuckTokenInstance.deployed();

      await stuckTokenInstance.transfer(instance.address, "5000")
      await
expect(instance.connect(addr1).claimStuckedER20(stuckTokenInstance.address)).to.be.reverted
With("Ownable: caller is not the owner");
    });
  });
});
```

2. UnitTestOutput

```
testToken
    Deployment test
      ✓ Should set the correct metadata (266ms)
    Transactions test
      ✓ Should transfer tokens between accounts (53ms)
      ✓ Should fail if sender doesn’t have enough tokens (67ms)
    Allowance test
      ✓ Should update the allowance when approving (49ms)
    Burn test
      ✓ Should burn tokens correctly (45ms)
      ✓ Should burn tokens by anyone himself (47ms)
      ✓ Should burnFrom tokens correctly (50ms)
    Ownership test
      ✓ Should transfer and renounce ownership correctly (41ms)
    claimStuckedER20 test
      ✓ Should allow the owner to claim stuck tokens (68ms)
```

```
        ✓ Should only owner can claim stuck tokens (57ms)
```

11.2 External Functions Check Points

1. Token.sol

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

contract: testToken is ERC20, ERC20Burnable, Ownable

Index	Function	Visibility	Permission Check	Re-entrancy Check	Injection Check	Unit Test	Notes
1	claimStuckedER20(address)	external	onlyOwner			Passed	
2	owner()	public				Passed	view
3	renounceOwnership()	public	onlyOwner			Passed	
4	transferOwnership(address)	public	onlyOwner			Passed	
5	burn(uint256)	public				Passed	
6	burnFrom(address,uint256)	public				Passed	
7	name()	public				Passed	view
8	symbol()	public				Passed	view
9	decimals()	public				Passed	view
10	totalSupply()	public				Passed	view
11	balanceOf(address)	public				Passed	view
12	transfer(address,uint256)	public				Passed	
13	allowance(address,address)	public				Passed	view
14	approve(address,uint256)	public				Passed	
15	transferFrom(address,address,uint256)	public				Passed	
16	increaseAllowance(address,uint256)	public				Passed	
17	decreaseAllowance(address,uint256)	public				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