IBSClassic Token

AUDIT REPORT

Version 1.0.0

Serial No. 20221214000012017

Presented by Fairyproof

December 14, 2022

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the International Blockchain ServiceToken Issuance project.

Audit Start Time:

December 13, 2022

Audit End Time:

December 14, 2022

Audited Source File's Address:

https://bscscan.com/token/0x99ce9D59568941A623a46e5598515B06862d13eC#code

The goal of this audit is to review International Blockchain Service'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 International Blockchain Service 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

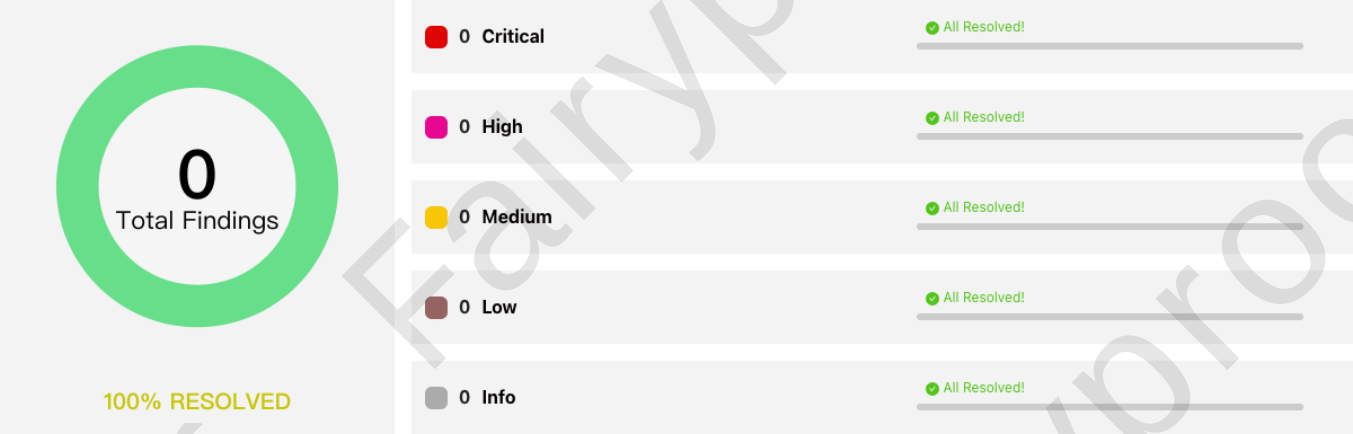
For this audit, we used the following sources of truth about how the token issuance function should work:

Source Code: https://bscscan.com/token/0x99ce9D59568941A623a46e5598515B06862d13eC#code

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

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2022121400012017	Fairyproof Security Team	Dec 13, 2022 - Dec 14, 2022	Passed



Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, no issue 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 International Blockchain Service

IBSclassic is a Native token of IBSA Pvt ltd company, also a Sister concern of International Blockchain service and academy. IBSclassic is basically used for payment gateway of ibs Insurance project and Ibs Realestate project and all upcoming project of IBSA pvt ltd.

04. Major functions of audited code

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

- Token Standard: BEP-20
- Token Address: 0x99ce9D59568941A623a46e5598515B06862d13eC
- Token Name: IBSClassic
- Token Symbol: IBSC
- Decimals: 18
- Max Supply: 100,000,000,000
- Pausable: Yes

Note:

In the current token contract, token transfers can be paused by owner.

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 examine 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 check 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.

- Consider managing the owner's access control with great care and transferring it to a multi-sig wallet or DAO when necessary.

11. Appendices

11.1 Unit Test

1. IBSClassic-test.js

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

describe("Pausable ERC20Token unit test", function () {
    let token;
    let owner,user1,user2,users;
    const MAX_SUPPLY = ethers.utils.parseEther("100000000000");

    beforeEach(async () => {
        [owner,user1,user2,...users] = await ethers.getSigners();
        const IBSClassic = await ethers.getContractFactory("IBSClassic");
        token = await
IBSClassic.deploy("IBSClassic",'IBSC',18,100000000000,owner.address);
    });

    describe("init status test", () => {
        it("meta data and supply check", async () => {
            expect(await token.name()).to.be.equal("IBSClassic");
            expect(await token.symbol()).to.be.equal("IBSC");
            expect(await token.decimals()).to.be.equal(18);
            expect(await token.totalSupply()).to.be.equal(MAX_SUPPLY);
            expect(await token.balanceOf(owner.address)).to.be.equal(MAX_SUPPLY);
            expect(await token.paused()).to.be.equal(false);
            expect(await token.owner()).to.be.equal(owner.address);
        });
    });

    describe("Ownable test", async () => {
        it("transferOwnership should be failed while not owner", async () => {
            await
expect(token.connect(user1).transferOwnership(user2.address)).to.be.reverted;
        });
        it("transferOwnership to zero should be failed ", async() => {
            await
expect(token.transferOwnership(ethers.constants.AddressZero)).to.be.reverted;
        });
        it("transferOwnership should change state and emit event", async () => {
            await expect(token.transferOwnership(user1.address)).to.be.emit(
                token,"OwnershipTransferred"
            ).withArgs(owner.address,user1.address);
            expect(await token.owner()).to.be.equal(user1.address);
        });
    });

    describe("Pausable unit test", () => {
        // pause
        it("pause should change state and emit event",async () => {
```

```
            await expect(token.pause()).to.be.emit(
                token,"Pause"
            );
            expect(await token.paused()).to.be.equal(true);
        });

        it("pause should be failed while has paused", async () => {
            await token.pause();
            // whenNotPaused
            await expect(token.pause()).to.be.reverted;
        })

        // unpause
        it("unpause should be failed while not paused", async () => {
            await expect(token.unpause()).to.be.reverted;
        });

        it("unpause should change state and emit event",async () => {
            // pause first
            await token.pause();
            expect(await token.paused()).to.be.equal(true);
            // unpause
            await expect(token.unpause()).to.be.emit(
                token,"Unpause"
            );
            expect(await token.paused()).to.be.equal(false);
        });
    });

    describe("transfer unit test", () => {
        it("transfer should change state and emit event", async () => {
            // emit event
            await expect(token.transfer(user1.address,10000)).to.be.emit(
                token,"Transfer"
            ).withArgs(owner.address,user1.address,10000);
            // check status
            expect(await
token.balanceOf(owner.address)).to.be.equal(MAX_SUPPLY.sub(10000));
            expect(await token.balanceOf(user1.address)).to.be.equal(10000);
            expect(await token.totalSupply()).to.be.equal(MAX_SUPPLY);
        });

        it("transfer should be failed while paused", async () => {
            await token.pause();
            await expect(token.transfer(user1.address,10000)).to.be.reverted;
        });

        it("transfer to zero should be failed", async () => {
```

```
92              await
expect(token.transfer(ethers.constants.AddressZero,10000)).to.be.reverted;
93          });
94
95          it("transfer zero token should be successful", async () => {
96              await token.connect(user1).transfer(user2.address,0);
97              expect(await token.balanceOf(user1.address)).to.be.equal(0);
98              expect(await token.balanceOf(user2.address)).to.be.equal(0);
99          });
100
101         it("transfer to self should be successful", async () => {
102             await token.transfer(owner.address,10000);
103             expect(await token.totalSupply()).to.be.equal(MAX_SUPPLY);
104             expect(await token.balanceOf(owner.address)).to.be.equal(MAX_SUPPLY);
105         });
106
107         it("transfer beyond balance should be failed", async () => {
108             await token.transfer(user1.address,100);
109             await
expect(token.connect(user1).transfer(user2.address,200)).to.be.reverted;
110         });
111     });
112
113     describe("Allowance  unit test", () => {
114         it("increaseApproval should be failed while paused", async () => {
115             await token.pause();
116             await
expect(token.increaseApproval(user1.address,10000)).to.be.reverted;
117         });
118
119         it("decreaseApproval should be failed while paused", async () => {
120             await token.pause();
121             await
expect(token.decreaseApproval(user1.address,10000)).to.be.reverted;
122         });
123
124         it("approve should be failed while paused", async () => {
125             await token.pause();
126             await expect(token.approve(user1.address,10000)).to.be.reverted;
127         });
128
129         it("increaseApproval and decreaseApproval should change state and emit
event",
130             async () => {
131                 await
expect(token.increaseApproval(user1.address,100)).to.be.emit(
132                     token,"Approval"
133                 ).withArgs(owner.address,user1.address,100);
```

```
                expect(await
token.allowance(owner.address,user1.address)).to.be.equal(100);

                await expect(token.decreaseApproval(user1.address,30)).to.be.emit(
                    token,"Approval"
                ).withArgs(owner.address,user1.address,100 - 30);
                expect(await
token.allowance(owner.address,user1.address)).to.be.equal(100 - 30);

                // decreaseApproval beyond should be zero
                await token.decreaseApproval(user1.address,90);
                expect(await
token.allowance(owner.address,user1.address)).to.be.equal(0);
        });

        it("approve should change state and emit event", async () => {
            await expect(token.approve(user1.address,100)).to.be.emit(
                token,"Approval"
            ).withArgs(owner.address,user1.address,100);
            expect(await
token.allowance(owner.address,user1.address)).to.be.equal(100);
            // approve again
            await token.approve(user1.address,30);
            expect(await
token.allowance(owner.address,user1.address)).to.be.equal(30);
        });
    });

    describe("TransferFrom test", () => {
        it("TransferFrom should be failed while paused", async () => {
            await token.pause();
            await
expect(token.connect(user1).transferFrom(owner.address,user1.address,100)).to.be.r
everted;
        });

      it("TransferFrom should consume allowance and emit event", async () => {
            await
expect(token.connect(user1).transferFrom(owner.address,user1.address,100)).to.be.r
everted;
            await token.increaseApproval(user1.address,10000);
            await
expect(token.connect(user1).transferFrom(owner.address,user1.address,100)).to.be.e
mit(
                token,"Transfer"
            ).withArgs(owner.address,user1.address,100);
            // check state
            expect(await
token.allowance(owner.address,user1.address)).to.be.equal(10000 - 100);
```

```
            expect(await token.totalSupply()).to.be.equal(MAX_SUPPLY);
            expect(await token.balanceOf(user1.address)).to.be.equal(100);
            expect(await
token.balanceOf(owner.address)).to.be.equal(MAX_SUPPLY.sub(100));
        });

        it("Can transfer zero tokens while has no approval", async () => {
            await
expect(token.connect(user1).transferFrom(user2.address,user1.address,0)).to.be.emi
t(
                token,"Transfer"
            ).withArgs(user2.address,user1.address,0);
        });
    });

    describe("Burn test", () => {
        it("Burn should be successful while paused", async () => {
            await token.pause();
            await token.burn(100);
        });

        it("burn should reduce the balance of user and total supply", async () =>
{
            await expect(token.burn(100)).to.be.emit(
                token,"Burn"
            ).withArgs(owner.address,100);
            expect(await token.totalSupply()).to.be.equal(MAX_SUPPLY.sub(100));
            expect(await
token.balanceOf(owner.address)).to.be.equal(MAX_SUPPLY.sub(100));
        });

        it("Burn beyond balance should be failed", async () => {
            await token.transfer(user1.address,100);
            await expect(token.connect(user1).burn(200)).to.be.reverted;
        });
    });
});

```

2. UnitTestOutput

```
Pausable ERC20Token unit test
    init status test
      ✔ meta data and supply check (52ms)
    Ownable test
      ✔ transferOwnership should be failed while not owner
      ✔ transferOwnership to zero should be failed
```

```
      ✔ transferOwnership should change state and emit event
    Pausable unit test
      ✔ pause should change state and emit event
      ✔ pause should be failed while has paused
      ✔ unpause should be failed while not paused
      ✔ unpause should change state and emit event
    transfer unit test
      ✔ transfer should change state and emit event
      ✔ transfer should be failed while paused
      ✔ transfer to zero should be failed
      ✔ transfer zero token should be successful
      ✔ transfer to self should be successful
      ✔ transfer beyond balance should be failed
    Allowance  unit test
      ✔ increaseApproval should be failed while paused
      ✔ decreaseApproval should be failed while paused
      ✔ approve should be failed while paused
      ✔ increaseApproval and decreaseApproval should change state and emit event
(39ms)
      ✔ approve should change state and emit event
    TransferFrom test
      ✔ TransferFrom should be failed while paused
      ✔ TransferFrom should consume allowance and emit event (43ms)
      ✔ Can transfer zero tokens while has no approval
    Burn test
      ✔ Burn should be successful while paused
      ✔ burn should reduce the balance of user and total supply
      ✔ Burn beyond balance should be failed


  25 passing (2s)

```

11.2 External Functions Check Points

1. IBSClassic.sol

File: contracts/IBSClassic.sol

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

contract: IBSClassic is PausableToken

Index	Function	Visibility	Permission Check	Re-entrancy Check	Injection Check	Unit Test	Notes
1	burn(uint256)	public				Passed	
2	transfer(address,uint256)	public				Passed	whenNotPaused
3	transferFrom(address,address,uint256)	public				Passed	whenNotPaused
4	approve(address,uint256)	public				Passed	whenNotPausedwhenNotPaused
5	increaseApproval(address,uint)	public				Passed	whenNotPaused
6	decreaseApproval(address,uint)	public				Passed	whenNotPaused
7	blackListAddress(address,bool)	public	onlyOwner				redundancy
8	pause()	public	onlyOwner			Passed	whenNotPaused
9	unpause()	public	onlyOwner			Passed	whenPaused
10	transferOwnership(address)	public	onlyOwner			Passed	
11	balanceOf(address)	public				Passed	View
12	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