BlockAura Token 3.1

AUDIT REPORT

Version 1.0.0

Serial No. 20221005000012015

Presented by Fairyproof

October 5, 2022

01. Introduction

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

Audit Start Time:

October 4, 2022

Audit End Time:

October 5, 2022

Audited Code's Github Repository:

https://github.com/blockaura-blockchain/BlockAura-Token-3.1

Audited Code's Github Commit Number When Audit Started:

26bc8ee7e4db355a1e1f026c351c0979f0fe095b

Audited Code's Github Commit Number When Audit Ended:

b56481ac4a0d2426fc440283c4828f1055e306a1

Audited Source Files:

The calculated SHA-256 value for the audited file when the audit was done is as follows:

```
BlockAura-ETH.sol:
0xe2f56b65b480fb42be1ca457b95f62a2ed63bd5a60da7157d2dd6cad377fd5be
```

The source file audited is as follows:

```
contracts/
└── BlockAura-ETH.sol.sol

0 directories, 1 file
```

The goal of this audit is to review BlockAura'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 BlockAura 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

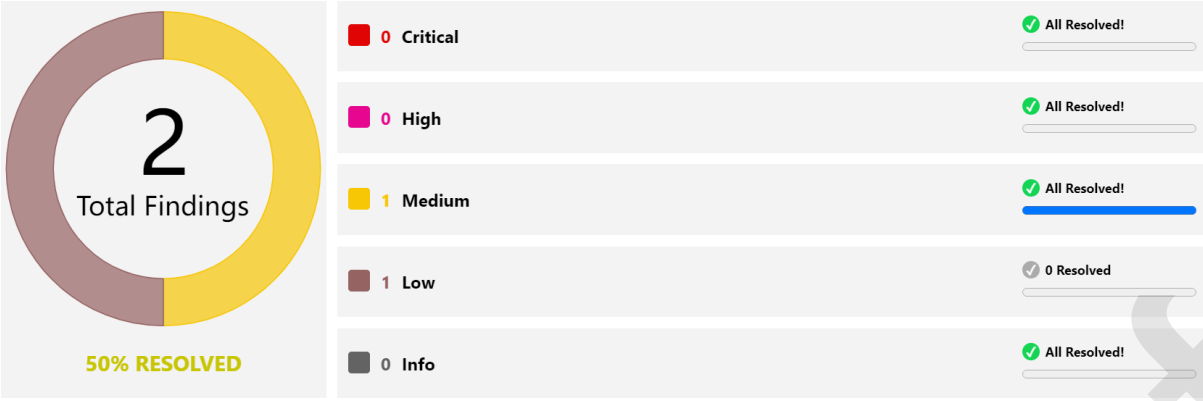
For this audit, we used the following sources of truth about how the token issuance system should work:

Contract Source Code

This was considered the specification, and when discrepancies arose with the actual code behavior, we consulted with the BlockAura team or reported an issue.

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2022100500012015	Fairyproof Security Team	Oct 4, 2022 - Oct 5, 2022	Low 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 and one issue of low-severity were uncovered. The BlockAura team fixed one issue of medium-severity , and acknowledged the remaining one issue of low-severity.

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. Major functions of audited code

The audited code implements an token issuance function and here are the details:

- Token Address: 0x591975253e25101f6E6f0383e13E82B7601D8c59 (Ethereum)
- Token Standard: ERC-20
- Name: BlockAura Token 3.1
- Symbol: TBAC
- Decimals: 8
- Max Supply: 20,000,000
- Burn: Yes
- Blacklist: Yes
- Transfer Can be Paused: Yes

Note: The contract provides the function of burning tokens on approval. Tokens held by blacklisted addresses cannot be transferred.

04. Coverage of issues

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

- Re-entrancy Attack
- Replay Attack
- Reordering Attack
- Miner's Advantage
- Rollback Attack
- DDoS Attack
- Injection Attack
- Transaction Ordering Attack
- Race Condition
- Access Control
- Integer Overflow/Underflow
- Timestamp Attack
- Gas Consumption
- Inappropriate Callback Function
- Function Visibility
- Implementation Vulnerability
- Uninitialized Storage Pointer
- Arithmetic Precision
- Tx.origin
- Fake Deposit
- Shadow Variable
- Design Vulnerability
- Token Issuance
- Admin Rights
- Inappropriate Proxy Design
- Inappropriate Use of Slots
- Asset Security
- Contract Upgrade/Migration
- Code Improvement
- Misc

05. 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.

06. 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 and FP-2 in "08. Issue description".

- Integer Overflow/Underflow

We checked all the code sections, which had arithmetic operations and might introduce integer overflow or underflow if no safe libraries were used. All of them used safe libraries.

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".

We didn't find issues or risks in these functions or areas at the time of writing.

- Token Issuance & Exchange

We checked whether or not the contract files could mint tokens at will.

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 didn't find issues or risks in these functions or areas at the time of writing.

07. List of issues by severity

Index	Title	Issue/Risk	Severity	Status
FP-1	Transfer Can be Paused	Design Vulnerability	Low	Acknowledged
FP-2	Could Transfer Tokens From Blacklisted Addresses	Implementation Vulnerability	Medium	✓ Fixed

08. Issue descriptions

[FP-1] Transfer Can be Paused Low Acknowledged

Issue/Risk: Design Vulnerability

description:

The contract has a pause function. When paused, users can not transfer tokens, which will have a certain impact on user usage.

Recommendation:

Consider pausing transfers in very special cases.

Update/Status:

The BlockAura team has acknowledged this. `Pausable.sol` implements an emergency stop mechanism.

[FP-2] Could Transfer Tokens From Blacklisted Addresses Medium ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

When a user was added to the blacklist, he/she could approve other users to spend his/her tokens, so the approved users could still transfer his/her tokens by calling the `transferFrom` function

Recommendation:

Consider adding a conditional check in the `transferFrom` function as follows:

```
require(!isBlackListed[from],"From is in blacklist");
```

Update/Status:

It has been fixed by the BlockAura team.

09. 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 admin's access control with great care and trasferring it to a multi-sig wallet or DAO when necessary.

10. Appendices

10.1 Unit Test File

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

describe("BlockauraToken", function () {
    let instance;
    let owner,user1,user2,users;
    const init_supply = ethers.utils.parseUnits("2000000",6);
    const max_supply = ethers.utils.parseUnits("2000000000",6);
    before(async () => {
        [owner,user1,user2,...users] = await ethers.getSigners();
    });

    beforeEach(async() => {
        const BlockauraToken = await ethers.getContractFactory("BlockauraToken");
        instance = await
BlockauraToken.deploy("BlockauraToken","BKT",6,init_supply,);
    });

    describe("Meta test", function() {
        it("name | symbol | decimals test", async () => {
            expect(await instance.name()).to.be.equal("BlockauraToken");
            expect(await instance.symbol()).to.be.equal("BKT");
            expect(await instance.decimals()).to.be.equal(6);
        });
    });

    describe("Init status test", function() {
        it("init_supply test", async () => {
            expect(await instance.totalSupply()).to.be.equal(init_supply);
            expect(await
instance.balanceOf(owner.address)).to.be.equal(init_supply);
        });

        it("init pause test", async () => {
            expect(await instance.paused()).to.be.false;
        });
```

```
    });

    describe("approve and allowance test", function() {
        it("approve should change allowance", async () => {
            expect(await
instance.allowance(user1.address,user2.address)).to.be.equal(0);
            await
expect(instance.connect(user1).approve(user2.address,10000)).to.be.emit(
                instance,"Approval"
            ).withArgs(user1.address,user2.address,10000);
            expect(await
instance.allowance(user1.address,user2.address)).to.be.equal(10000);
        });

        it("increaseAllowance/decreaseAllowance should change allowance", async ()
=> {
            expect(await
instance.allowance(user1.address,user2.address)).to.be.equal(0);
            await
expect(instance.connect(user1).increaseAllowance(user2.address,10000)).to.be.emit(
                instance,"Approval"
            ).withArgs(user1.address,user2.address,10000);
            expect(await
instance.allowance(user1.address,user2.address)).to.be.equal(10000);

            await
expect(instance.connect(user1).decreaseAllowance(user2.address,1000)).to.be.emit(
                instance,"Approval"
            ).withArgs(user1.address,user2.address,9000);
            expect(await
instance.allowance(user1.address,user2.address)).to.be.equal(9000);
        });
    });

    describe("transferFrom and transfer test", function() {
        it("transfer should change balance", async () => {
            await expect(instance.transfer(user1.address,1000)).to.be.emit(
                instance,"Transfer"
            ).withArgs(owner.address,user1.address,1000);
            expect(await
instance.balanceOf(owner.address)).to.be.equal(init_supply.sub(1000));
            expect(await instance.balanceOf(user1.address)).to.be.equal(1000);
            expect(await instance.totalSupply()).to.be.equal(init_supply);
        });

        it("transfer to self shouldn't change balance", async () => {
            await instance.transfer(owner.address,1000);
            expect(await
instance.balanceOf(owner.address)).to.be.equal(init_supply);
            expect(await instance.totalSupply()).to.be.equal(init_supply);
        });

        it("transfer should failed while sender has insufficient tokens", async ()
=> {
```

```
            await
expect(instance.connect(user1).transfer(user2.address,10)).to.be.reverted;
        });

        it("transferFrom without approval should be failed", async () => {
            await
expect(instance.connect(user1).transferFrom(owner.address,user2.address,10)).to.be
.reverted;
        });

        it("transferFrom should change balance and allowance", async () => {
            await instance.approve(user1.address, 10000);
            await
expect(instance.connect(user1).transferFrom(owner.address,user2.address,3000)).to.
be.emit(
                instance,"Transfer"
            ).withArgs(owner.address,user2.address,3000);
            expect(await
instance.balanceOf(owner.address)).to.be.equal(init_supply.sub(3000));
            expect(await instance.balanceOf(user1.address)).to.be.equal(0);
            expect(await instance.balanceOf(user2.address)).to.be.equal(3000);
            expect(await instance.totalSupply()).to.be.equal(init_supply);
            expect(await
instance.allowance(owner.address,user1.address)).to.be.equal(7000);
        });
    });

    describe("minter and burn", async () => {

        it("mint beyond supply or while paused should be failed", async () => {
            let value = max_supply.sub(init_supply).add(100);
            await
expect(instance.mint(user1.address,value)).to.be.revertedWith("value exceeds
beyond maximum supply");
            await expect(instance.connect(user1).mint(user1.address,
100)).to.be.revertedWith("you are not minter");
            await instance.pause();
            await expect(instance.mint(user1.address,
100)).to.be.revertedWith("Can not mint contract is paused");

        });

        it("mint should change state and emit event ", async () => {
            await expect(instance.mint(user1.address,10000)).to.be.emit(
                instance,"Transfer"
            ).withArgs(ethers.constants.AddressZero,user1.address,10000);

            expect(await instance.balanceOf(user1.address)).to.be.equal(10000);
            expect(await
instance.totalSupply()).to.be.equal(init_supply.add(10000));
        });

        it("BurnFrom should change state and emit event", async () => {
            await instance.mint(user1.address,10000);
            await instance.connect(user1).approve(owner.address,10000);
```

```
            expect(await instance.balanceOf(user1.address)).to.be.equal(10000);
            expect(await
instance.allowance(user1.address,owner.address)).to.be.equal(10000);
            await expect(instance._burnFrom(user1.address,100)).to.be.emit(
                instance,"Transfer"
            ).withArgs(user1.address,ethers.constants.AddressZero,100);
            expect(await instance.balanceOf(user1.address)).to.be.equal(9900);
            expect(await
instance.allowance(user1.address,owner.address)).to.be.equal(9900);
            await expect(instance._burnFrom(user1.address,10000)).to.be.reverted;
        });
    });

    describe("BlackList test", function() {
        it("only owner can add and remove blacklist", async () => {
            await
expect(instance.connect(user1).addBlackList(user2.address)).to.be.revertedWith("Ow
nable: caller is not the owner");
            await
expect(instance.connect(user1).removeBlackList(user2.address)).to.be.revertedWith(
"Ownable: caller is not the owner");
        });
        it("add/remove blacklist should change state", async () => {
            expect(await instance.isBlackListed(user1.address)).to.be.false;
            expect(await instance.getBlackListStatus(user1.address)).to.be.false;
            await instance.addBlackList(user1.address);
            expect(await instance.isBlackListed(user1.address)).to.be.true;
            expect(await instance.getBlackListStatus(user1.address)).to.be.true;
            await instance.removeBlackList(user1.address);
            expect(await instance.isBlackListed(user1.address)).to.be.false;
            expect(await instance.getBlackListStatus(user1.address)).to.be.false;
        });
    });

    describe("transfer with isBlackListed or Pausable test", function() {
        it("transfer should be failed while paused", async () => {
            await expect(instance.pause()).to.be.emit(
                instance,"Paused"
            ).withArgs(owner.address);
            await expect(instance.transfer(user1.address,100)).to.be.reverted;
            await instance.unpause();
            await instance.transfer(user1.address,100);
        });

        it("transferFrom should be failed while paused", async () => {
            await instance.approve(user1.address,10000)
            await expect(instance.pause()).to.be.emit(
                instance,"Paused"
            ).withArgs(owner.address);
            await
expect(instance.connect(user1).transferFrom(owner.address,user1.address,100)).to.b
e.revertedWith("Can not mint contract is paused");
            await instance.unpause();
            instance.connect(user1).transferFrom(owner.address,user1.address,100);
        });
```

```
        it("transfer should be failed while in blacklist", async () => {
            await instance.mint(user1.address,100);
            await instance.addBlackList(user1.address);
            await
expect(instance.connect(user1).transfer(user2.address,20)).to.be.reverted;

            await instance.removeBlackList(user1.address);
            await instance.connect(user1).transfer(user2.address,20);
            expect(await instance.balanceOf(user1.address)).to.be.equal(80);
            expect(await instance.balanceOf(user2.address)).to.be.equal(20);
        });

        // it("blacklist can not call transferFrom", async ()=> {
        //     await instance.approve(user1.address,10000);
        //     await instance.addBlackList(user1.address);
        //     await
expect(instance.connect(user1).transferFrom(owner.address,user2.address,100)).to.b
e.reverted;
        // });

        it("transferFrom should be failed while from in blacklist", async () => {
            await instance.approve(user1.address,10000)
            await instance.addBlackList(owner.address);
            await
expect(instance.connect(user1).transferFrom(owner.address,user1.address,100)).to.b
e.revertedWith("From is in blacklist");
            await instance.removeBlackList(owner.address);
            await
instance.connect(user1).transferFrom(owner.address,user1.address,100);
            expect(await instance.balanceOf(user1.address)).to.be.equal(100);
        });

        it("_burnFrom should change state", async () => {
            await instance.transfer(user1.address,100);
            expect(await instance.balanceOf(user1.address)).to.be.equal(100);

            await expect(instance._burnFrom(user1.address,20)).to.be.reverted;
            await instance.connect(user1).approve(owner.address,10000);

            expect(await
instance.allowance(user1.address,owner.address)).to.be.equal(10000);
            await instance._burnFrom(user1.address,20);
            expect(await
instance.allowance(user1.address,owner.address)).to.be.equal(9980);

            expect(await instance.totalSupply()).to.be.equal(init_supply.sub(20));
            expect(await instance.balanceOf(user1.address)).to.be.equal(80);
        });
    });

});
```

Output:

```
BlockauraToken
  Meta test
    ✓ name | symbol | decimals test (42ms)
  Init status test
    ✓ init_supply test
    ✓ init pause test
  approve and allowance test
    ✓ approve should change allowance
    ✓ increaseAllowance/decreaseAllowance should change allowance (49ms)
  transferFrom and transfer test
    ✓ transfer should change balance (40ms)
    ✓ transfer to self shouldn't change balance
    ✓ transfer should failed while sender has insufficient tokens (44ms)
    ✓ transferFrom without approval should be failed
    ✓ transferFrom should change balance and allowance (50ms)
  minter and burn
    ✓ mint beyond supply or while paused should be failed (77ms)
    ✓ mint should change state and emit event
    ✓ BurnFrom should change state and emit event (62ms)
  BlackList test
    ✓ only owner can add and remove blacklist
    ✓ add/remove blacklist should change state (46ms)
  transfer with isBlackListed or Pausable test
    ✓ transfer should be failed while paused (49ms)
    ✓ transferFrom should be failed while paused (45ms)
    ✓ transfer should be failed while in blacklist (59ms)
    ✓ transferFrom should be failed while from in blacklist (54ms)
    ✓ _burnFrom should change state (61ms)


20 passing (3s)
```

10.2 External Functional Checkpoints

File:BlockAura-ETH.sol

(An empty field in the table means it is not required or applicable)

contract: BlockauraToken is ERC20Pausable, ERC20Detailed

Index	Function	Visibility	Permission Check	Re-entrancy Check	Injection Check	Unit Test	Notes
1	name()	public				Passed	
2	symbol()	public				Passed	
3	decimals()	public				Passed	
4	transfer(address,uint256)	public				Passed	whenNotPaused
5	transferFrom(address,address,uint256)	public				Passed	whenNotPaused
6	approve(address,uint256)	public				Passed	whenNotPaused
7	increaseAllowance(address,uint)	public				Passed	whenNotPaused
8	decreaseAllowance(address,uint)	public				Passed	whenNotPaused
9	totalSupply()	public				Passed	
10	balanceOf(address)	public				Passed	
11	allowance(address,address)	public				Passed	
12	mint(address,uint256)	public	onlyMinter			Passed	whenNotPaused
13	_burnFrom(address,uint256)	public				Passed	NeedApproval,whenNotPaused
14	paused()	public				Passed	View
15	pause()	public	onlyPauser			Passed	whenNotPaused
16	unpause()	public	onlyPauser			Passed	whenPaused
17	addBlackList	public	onlyOwner			Passed	
18	removeBlackList	public	onlyOwner			Passed	
19	getBlackListStatus	public				Passed	View
20	getOwner	public					redundant,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