FAIRYPROOF

BlockAura Token

AUDIT REPORT

Version 1.0.0

Serial No. 2022091200012021

Presented by Fairyproof

September 12, 2022

01. Introduction

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

Audit Start Time:

September 5, 2022

Audit End Time:

September 10, 2022

Audited Code's Github Repository:

https://github.com/blockaura-blockchain/BlockAura-Token-2.1

Audited Code's Github Commit Number When Audit Started:

88309d4d3c59ad6e00cf55c2adf32d7fe569fd4d

Audited Code's Github Commit Number When Audit Ended:

2505e4e6fef8cd9b9eafd51f9a0980d6964ed8a4

Audited Source Files:

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

```
BlockAura-ETH.sol:
0xa32e7ef43381e938e1a13d9ab0bc162e0bad2a69b9ab09cffd11e7bc36b9a369
```

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

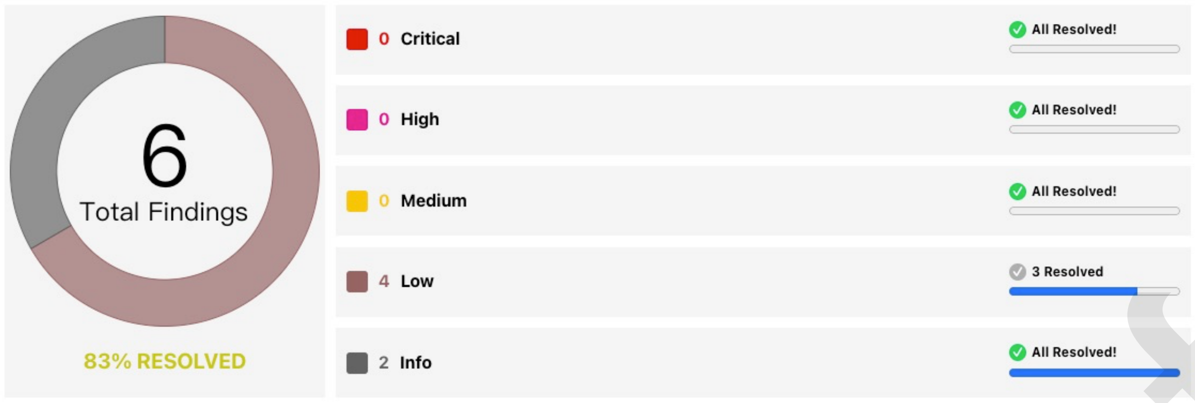
For this audit, we used the following sources of truth about how the token issuance system should work:

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
2022091200012021	Fairyproof Security Team	Sep 5, 2022 - Sep 10, 2022	Low Risk



Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, four issues of low-severity and two issues of informational-severity were uncovered. The BlockAura team fixed three issues of low-severity and two issues of informational-severity, and acknowledged the remaining one issues 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 Standard: ERC-20
- Blockchain: Ethereum
- Address: 0x85800a01809B9a778c78d5Ea70bDddDb166DA65C
- Name: BlockAura Token 3.0
- Symbol: TBAC
- Decimals: 8
- Total Supply: 20,000,000
- Burn: No
- Transfer Pausable: Yes

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 found one issue, for more details please refer to FP-3 in "08. Issue description".

- 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 found some issues, for more details please refer to FP-4, FP-5 and FP-6 in "08. Issue description".

07. List of issues by severity

Index	Title	Issue/Risk	Severity	Status
FP-1	Transfer Can be Paused	Design Vulnerability	Low	Acknowledged
FP-2	Can Mint Token while Paused	Design Vulnerability	Low	✓ Fixed
FP-3	Inappropriate Role Control	Design Vulnerability	Low	✓ Fixed
FP-4	Better Use Ownable Contract	Design Vulnerability	Low	✓ Fixed
FP-5	Unnecessary Function Visibility	Design Vulnerability	Info	✓ Fixed
FP-6	Missing Prompt Messages	Code Improvement	Info	✓ 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] Can Mint Token While Paused Low ✓ Fixed

Issue/Risk: Design Vulnerability

Description:

When paused, `transer` and `approve` were prohibited, but `mint` and `_burnFrom` were still allowed. This is an inconsistent in design.

Recommendation:

Consider adding a modifier `whenNotPaused` on functions `mint` and `_burnFrom`.

Update/Status:

It has been fixed by the BlockAura team.

[FP-3] Inappropriate Role Control Low ✓ Fixed

Issue/Risk: Design Vulnerability

Description:

In the implementation, the executing `minter` can add or remove other minters. Typically, a role with higher privileges is used to perform such operations.

The `Pauser` had this issue as well.

Recommendation:

Consider using `owner` to add or remove `minter` or `pauser`.

Update/Status:

It has been fixed by the BlockAura team.

[FP-4] Better Use Ownable Contract Low ✓ Fixed

Issue/Risk: Design Vulnerability

Description:

Contract `MinterRole` inherited `Ownable` , but the `onlyOwner` was unused.

Recommendation:

Consider using `owner` to add or remove `minter` or `pauser`.

Update/Status:

It has been fixed by the BlockAura team.

[FP-5] Unnecessary Function Visibility Info ✓ Fixed

Issue/Risk: Design Vulnerability

Description:

The name and comments of function `_burnFrom` indicate that it was an internal function, yet it was implemented as a public function.

Recommendation:

Consider not making `burn` external unless it is really necessary.

Update/Status:

It has been fixed by the BlockAura team.

[FP-6] Missing Prompt Messages Info ✓ Fixed

Issue/Risk: Code Improvement

Description:

It is better to add a prompt message for some `require` statements, which is convenient for checking the cause when an error occurs.

For example , change `require(isMinter(msg.sender));` to `require(isMinter(msg.sender), "not minter");`

Recommendation:

Consider adding a prompt message for each of the necessary `require` statements.

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 max_supply = ethers.utils.parseUnits("2000000000",6);
    const init_supply = ethers.utils.parseUnits("1000000000",6);
    before(async () => {
        [owner,user1,user2,...users] = await ethers.getSigners();
    });

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

    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 and 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,4000)).to.be.emit(
                instance,"Approval"
            ).withArgs(user1.address,user2.address,6000);
            expect(await
instance.allowance(user1.address,user2.address)).to.be.equal(6000);
        });

        it("user can approve self", async () => {
            await instance.approve(owner.address,10000);
            expect(await
instance.allowance(owner.address,owner.address)).to.be.equal(10000);
        });
    });

    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("Role and Pause test", function() {
        it("Only owner can add and remove minter", async () => {
            expect(await instance.isMinter(owner.address)).to.be.true;
            expect(await instance.isMinter(user1.address)).to.be.false;
            expect(await instance.isMinter(user2.address)).to.be.false;
            await
expect(instance.connect(user1).addMinter(user2.address)).to.be.reverted;
            await instance.addMinter(user2.address);
            expect(await instance.isMinter(user2.address)).to.be.true;
            await
expect(instance.connect(user2).removeMinter(owner.address)).to.be.reverted;
            await instance.removeMinter(user2.address);
            expect(await instance.isMinter(user2.address)).to.be.false;
        });

        it("Minter can remove self", async () => {
            await instance.addMinter(user2.address);
            await instance.renounceMinter();
            await instance.connect(user2).renounceMinter();
            expect(await instance.isMinter(owner.address)).to.be.false;
            expect(await instance.isMinter(user2.address)).to.be.false;
        });

        it("Only owner can add and remove a pauser", async () => {
            expect(await instance.isPauser(owner.address)).to.be.true;
            expect(await instance.isPauser(user1.address)).to.be.false;
            expect(await instance.isPauser(user2.address)).to.be.false;
            await
expect(instance.connect(user1).addPauser(user2.address)).to.be.reverted;
            await instance.addPauser(user2.address);
            expect(await instance.isPauser(user2.address)).to.be.true;
            await
expect(instance.connect(user1).addPauser(user2.address)).to.be.reverted;
        });

        it("Pauser can renounce self", async () => {
            await instance.addPauser(user2.address);
            await instance.renouncePauser();
            await instance.connect(user2).renouncePauser();
            expect(await instance.isPauser(owner.address)).to.be.false;
            expect(await instance.isPauser(user2.address)).to.be.false;
        });

        it("only pauser can pause/unpause", async () => {
            expect(await instance.paused()).to.be.false;
            await expect(instance.connect(user1).pause()).to.be.reverted;
            await instance.pause();
            expect(await instance.paused()).to.be.true;
            await expect(instance.connect(user1).unpause()).to.be.reverted;
        });

        it("pause/unpause can emit event", async () => {
            await expect(instance.pause()).to.be.emit(
                instance,"Paused"
            ).withArgs(owner.address);
            expect(await instance.paused()).to.be.true;

            await expect(instance.unpause()).to.be.emit(
                instance,"Unpaused"
            ).withArgs(owner.address);
            expect(await instance.paused()).to.be.false;
        });

        it("only pause/unpause while not in the same status", async () => {
            await instance.pause();
            expect(await instance.paused()).to.be.true;
            await expect(instance.pause()).to.be.reverted;
            await instance.unpause();
            expect(await instance.paused()).to.be.false;
            await expect(instance.unpause()).to.be.reverted;
        });
    });

    describe("Pause and transfer or approve test", function() {
        it("approve and transfer are abandon while paused", async () => {
            await instance.approve(user1.address,10000);
            await instance.pause();
            await expect(instance.transfer(user1.address,100)).to.be.reverted;
            await
expect(instance.connect(user1).transferFrom(owner.address,user2.address,100)).to.b
e.reverted;
            await
expect(instance.increaseAllowance(user1.address,100)).to.be.reverted;
            await
expect(instance.decreaseAllowance(user1.address,100)).to.be.reverted;
            await
expect(instance.connect(user1).approve(user2.address,1000)).to.be.reverted;
        });
    });

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

        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(owner.address)).to.be.equal(init_supply);
```

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

        it("mint beyond max_supply should be failed", async () => {
            await
expect(instance.mint(user1.address,max_supply.sub(init_supply).add(1))).to.be.reve
rted;
        });

        it("mint are abandon while paused", async () => {
            await instance.pause();
            await expect(instance.mint(user1.address,100)).to.be.reverted;
        });

    });
});
```

Output:

```
  BlockauraToken
    Meta test
      ✓ name | symbol | decimals test
    Init status test
      ✓ init_supply test
    approve and allowance test
      ✓ approve should change allowance (38ms)
      ✓ increaseAllowance and decreaseAllowance should change allowance (67ms)
      ✓ user can approve self
    transferFrom and transfer test
      ✓ transfer should change balance
      ✓ transfer to self shouldn't change balance
      ✓ transfer should failed while sender has insufficient tokens (40ms)
      ✓ transferFrom without approval should be failed
      ✓ transferFrom should change balance and allowance (50ms)
    Role and Pause test
      ✓ Only owner can add and remove minter (68ms)
      ✓ Minter can remove self (39ms)
      ✓ Only owner can add and remove a pauser (52ms)
      ✓ Pauser can renounce self (46ms)
      ✓ only pauser can pause/unpause (47ms)
      ✓ pause/unpause can emit event
      ✓ only pause/unpause while not in the same status (45ms)
    Pause and transfer or approve test
      ✓ approve and transfer are abandon while paused (68ms)
    minter and burn
      ✓ mint should change state and emit event
      ✓ mint beyond max_supply should be failed
      ✓ mint are abandon while paused


   21 passing (2s)
```

10.2 Functional Checkpoints

File:BlockAura-ETH.sol

contract: BlockauraToken is ERC20Pausable, ERC20Detailed

Index	Function	Visibility	Re-entrancy Check	Permission Check	Unit Test	Notes
1	name()	public	No Need	No Need	Passed	view
2	symbol()	public	No Need	No Need	Passed	view
3	decimals()	public	No Need	No Need	Passed	view
4	transfer(address,uint256)	public	No Need	No Need	Passed	notPaused
5	transferFrom(address,address,uint256)	public	No Need	No Need	Passed	notPaused
6	approve(address,uint256)	public	No Need	No Need	Passed	notPaused
7	increaseAllowance(address,uint)	public	No Need	No Need	Passed	notPaused
8	decreaseAllowance(address,uint)	public	No Need	No Need	Passed	notPaused
9	paused()	public	No Need	No Need	Passed	view
10	pause()	public	No Need	onlyPauser	Passed	notPaused
11	unpause()	public	No Need	onlyPauser	Passed	onlyPaused
12	isPauser(address)	public	No Need	No Need	Passed	view
13	addPauser(address)	public	No Need	onlyOwner	Passed	
14	removePauser	public	No Need	onlyOwner	Passed	
15	renouncePauser()	public	No Need	onlyPauser	Passed	
16	totalSupply()	public	No Need	No Need	Passed	view
17	balanceOf(address)	public	No Need	No Need	Passed	view
18	allowance(address,address)	public	No Need	No Need	Passed	view
19	mint(address,uint256)	public	No Need	onlyMinter	Passed	onlyPaused
20	isMinter(address)	public	No Need	No Need	Passed	view
21	addMinter(address)	public	No Need	onlyOwner	Passed	
22	removeMinter(address)	public	No Need	onlyOwner	Passed	
23	renounceMinter()	public	No Need	onlyMinter	Passed	
24	owner()	public	No Need	No Need	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