Boom DAO Wrapped eNaira

AUDIT REPORT

Version 1.0.0

Serial No. 20230525000120222

Presented by Fairyproof

May 25, 2023

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Boom DAO's token issuance project.

Audit Start Time:

May 25, 2023

Audit End Time:

May 25, 2023

Audited Source File's Address:

https://etherscan.io/token/0x025404b8be27f52988aca2b4415b89f6269fed83#code

The goal of this audit is to review Boom DAO'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 Boom DAO 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

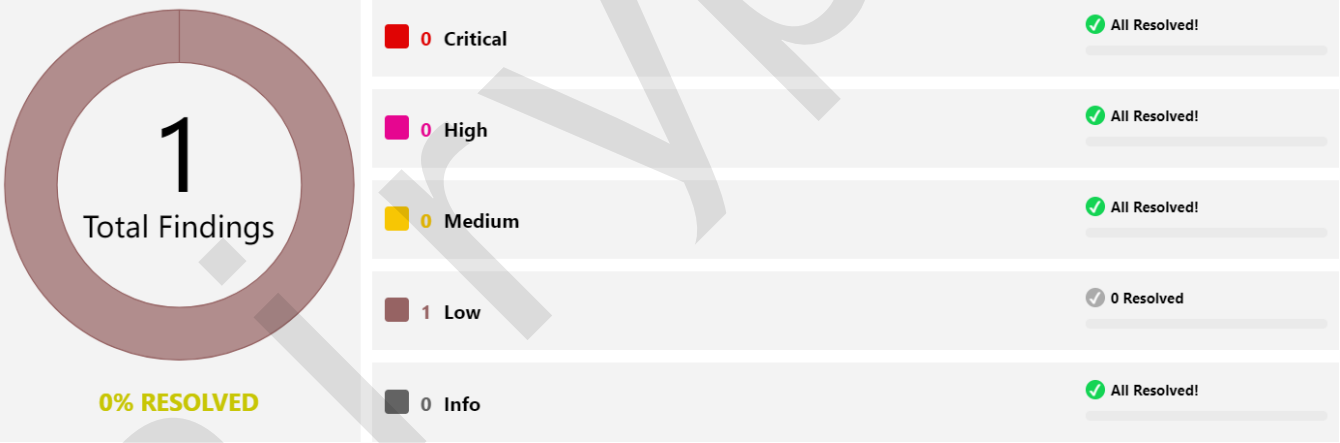
For this audit, we used the following source(s) of truth about how the token issuance function should work:

Source Code: https://etherscan.io/token/0x025404b8be27f52988aca2b4415b89f6269fed83#code

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

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2023052500012022	Fairyproof Security Team	May 25, 2023 - May 25, 2023	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 low-severity was uncovered. The Boom DAO team acknowledged all the issues.

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 eNaira

eNaira is a Central Bank of Nigeria-issued digital currency that provides a unique form of money denominated in Naira. WrappedENaira is the ERC20 token issued by eNaira.

The above description is quoted from relevant documents of Boom DAO.

04. Major functions of audited code

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

- Blockchain: Ethereum
- Token Standard: ERC-20
- Token Address: 0x025404b8bE27F52988ACA2b4415b89f6269fED83
- Token Name: Wrapped eNaira
- Token Symbol: WeNGN
- Decimals: 18
- Current Supply: 1,300,000,000,000
- Max Supply: No Cap
- Burnable: Yes
- Mintable: Yes

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 found one issue, for more details please refer to [FP-1] in "09. Issue description".

- State Update

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

- Asset Security

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

- Miscellaneous

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

08. List of issues by severity

Index	Title	Issue/Risk	Severity	Status
FP-1	Unlimited Token Issuance	Token Issuance	Low	Acknowledged

09. Issue descriptions

[FP-1] Unlimited Token Issuance

Token Issuance Low Acknowledged

Issue/Risk: Token Issuance

Description:

In the current contract, tokens can be issued additionally and there is no cap on issuance,which may cause losses to token holders in certain scenarios.

Recommendation:

Consider setting a cap on token issuance.

Update/Status:

The WrappedENaira team replied that the wrapped eNaira has unlimited supply because it is a wrapped version of a CBDC. Hence the more the government mints cash, the more the supply of the CBDC and therefore the Wrapped version since the wrapped is pegged 1:1 to the CBDC. Thus unlimited supply is a feature of Fiat that they are mimicking and not a flaw.

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 transfering it to a multi-sig wallet or DAO when necessary.

11. Appendices

11.1 Unit Test

1. WrappedENaira.t.js

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

describe("WrappedENaira Test", function () {
  let owner, user1;
  const initialAmount = ethers.utils.parseEther("1300000000000");
  const ZERO_ADDRESS = ethers.constants.AddressZero;

  async function deployOneWrappedENaira() {
    [owner, user1] = await ethers.getSigners();

    const WrappedENaira = await ethers.getContractFactory("WrappedENaira");
    const tokenInstance = await WrappedENaira.deploy();

    return { tokenInstance };
  }

  describe("Initial state uint test", function () {
    it("Initial state should equal with the params of constructor", async function () {
      const { tokenInstance } = await loadFixture(deployOneWrappedENaira);
      expect(await tokenInstance.name()).to.equal("WrappedENaira");
      expect(await tokenInstance.symbol()).to.equal("WENGN");
      expect(await tokenInstance.decimals()).to.equal(18);
      expect(await tokenInstance.totalSupply()).to.equal(initialAmount);
      expect(await tokenInstance.balanceOf(owner.address)).to.equal(initialAmount)
      expect(await tokenInstance.owner()).to.equal(owner.address)
    });
  });

  describe("Mint test", async () => {
    it(("Mint will emit Transfer event"), async () => {
      const { tokenInstance } = await loadFixture(deployOneWrappedENaira);
      expect(await deployOneWrappedENaira()).to
        .emit(tokenInstance, "Transfer")
```

```
        .withArgs(ZERO_ADDRESS, owner.address, initialAmount);
    })

    it("Mint can only called by owner", async () => {
      const { tokenInstance } = await loadFixture(deployOneWrappedENaira);
      await expect(tokenInstance.connect(user1).mint(user1.address, initialAmount))
        .to.revertedWith("Ownable: caller is not the owner")
    })

    it("Mint to zero address will fail", async () => {
      const { tokenInstance } = await loadFixture(deployOneWrappedENaira);
      await expect(tokenInstance.mint(ZERO_ADDRESS, initialAmount))
        .to.revertedWith("ERC20: mint to the zero address")
    })

    it("Warning: Owner can mint as much as he can", async () => {
      const { tokenInstance } = await loadFixture(deployOneWrappedENaira);
      expect(await tokenInstance.totalSupply()).to.equal(initialAmount);
      await expect(tokenInstance.mint(user1.address, initialAmount))
        .emit(tokenInstance, "Transfer")
        .withArgs(ZERO_ADDRESS, user1.address, initialAmount);
      expect(await tokenInstance.totalSupply()).to.equal(initialAmount.mul(2));
    })
  })

  describe("Burn test", async () => {
    it("Burn will emit Transfer event", async () => {
      const { tokenInstance } = await loadFixture(deployOneWrappedENaira);
      expect(await tokenInstance.balanceOf(owner.address)).to.equal(initialAmount)
      expect(await tokenInstance.totalSupply()).to.equal(initialAmount);
      await expect(tokenInstance.burn(initialAmount.div(2)))
        .emit(tokenInstance, "Transfer")
        .withArgs(owner.address, ZERO_ADDRESS, initialAmount.div(2));
      expect(await tokenInstance.balanceOf(owner.address)).to.equal(initialAmount.div(2));
      expect(await tokenInstance.totalSupply()).to.equal(initialAmount.div(2));
    });

    it("Burn can only called by anyone who own enough token", async () => {
      const { tokenInstance } = await loadFixture(deployOneWrappedENaira);
      expect(await tokenInstance.balanceOf(owner.address)).to.equal(initialAmount)
      await expect(tokenInstance.connect(user1).burn(initialAmount.div(2)))
        .to.revertedWith("ERC20: burn amount exceeds balance")
      await tokenInstance.transfer(user1.address, initialAmount.div(2))
      await expect(tokenInstance.connect(user1).burn(initialAmount.div(2)))
        .emit(tokenInstance, "Transfer")
        .withArgs(user1.address, ZERO_ADDRESS, initialAmount.div(2));
      expect(await tokenInstance.balanceOf(user1.address)).to.equal(0)
      expect(await tokenInstance.totalSupply()).to.equal(initialAmount.div(2));
    });
  });

  describe("Transfer test", async () => {
    it("Transfer will emit Transfer event", async () => {
```

```
      const { tokenInstance } = await loadFixture(deployOneWrappedENaira);
      expect(await tokenInstance.balanceOf(owner.address)).to.equal(initialAmount)
      await expect(tokenInstance.transfer(user1.address, initialAmount.div(2)))
        .emit(tokenInstance, "Transfer")
        .withArgs(owner.address, user1.address, initialAmount.div(2));
      expect(await tokenInstance.balanceOf(owner.address)).to.equal(initialAmount.div(2));
      expect(await tokenInstance.balanceOf(user1.address)).to.equal(initialAmount.div(2));
      expect(await tokenInstance.totalSupply()).to.equal(initialAmount);
    });

    it(("Transfer should have enough amount"), async () => {
      const { tokenInstance } = await loadFixture(deployOneWrappedENaira);
      expect(await tokenInstance.balanceOf(owner.address)).to.equal(initialAmount)
      await expect(tokenInstance.connect(user1).transfer(owner.address,
initialAmount.div(2)))
        .to.revertedWith("ERC20: transfer amount exceeds balance")
    });
  });

  describe("TransferFrom test", async () => {
    it(("TransferFrom should have enough allowance"), async () => {
      const { tokenInstance } = await loadFixture(deployOneWrappedENaira);
      expect(await tokenInstance.balanceOf(owner.address)).to.equal(initialAmount)
      expect(await tokenInstance.allowance(user1.address, owner.address)).to.equal(0)
      await expect(tokenInstance.connect(user1).transferFrom(owner.address, user1.address,
initialAmount.div(2)))
        .to.revertedWith("ERC20: insufficient allowance")
      await tokenInstance.approve(user1.address, initialAmount.div(2))
      expect(await tokenInstance.allowance(owner.address,
user1.address)).to.equal(initialAmount.div(2))
      await expect(tokenInstance.connect(user1).transferFrom(owner.address, user1.address,
initialAmount.div(2)))
        .emit(tokenInstance, "Transfer")
        .withArgs(owner.address, user1.address, initialAmount.div(2));
      expect(await tokenInstance.balanceOf(owner.address)).to.equal(initialAmount.div(2));
      expect(await tokenInstance.balanceOf(user1.address)).to.equal(initialAmount.div(2));
      expect(await tokenInstance.allowance(owner.address, user1.address)).to.equal(0)
    });

    it(("TransferFrom will emit event"), async () => {
      const { tokenInstance } = await loadFixture(deployOneWrappedENaira);
      await tokenInstance.approve(user1.address, initialAmount.div(2))
      await expect(tokenInstance.connect(user1).transferFrom(owner.address, user1.address,
initialAmount.div(2)))
        .emit(tokenInstance, "Transfer")
        .withArgs(owner.address, user1.address, initialAmount.div(2));
      expect(await tokenInstance.allowance(owner.address, user1.address)).to.equal(0)
    });
  });
});
```

2. 2. Output

```
WrappedENaira Test
  Initial state uint test
    ✓ Initial state should equal with the params of constructor (170ms)
  Mint test
    ✓ Mint will emit Transfer event (44ms)
    ✓ Mint can only called by owner
    ✓ Mint to zero address will fail
    ✓ Warning: Owner can mint as much as he can
  Burn test
    ✓ Burn will emit Transfer event
    ✓ Burn can only called by anyone who own enough token
  Transfer test
    ✓ Transfer will emit Transfer event
    ✓ Transfer should have enough amount
  TransferFrom test
    ✓ TransferFrom should have enough allowance (39ms)
    ✓ TransferFrom will emit event


11 passing (437ms)
```

11.2 External Functions Check Points

1. File: contracts/WrappedENaira.sol

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

contract: WrappedENaira is ERC20, ERC20Burnable, Ownable

Index	Function	Visibility	StateMutability	Permission Check	IsUserInterface	Unit Test	Notes
1	mint(address,uint256)	public		onlyOwner		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