Macaron Staking

AUDIT REPORT

Version 1.0.0

Serial No. 20240419000012025

Presented by Fairyproof

April 19, 2024

www.fairyproof.com

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Macaron Staking project.

Audit Start Time:

April 15, 2024

Audit End Time:

April 19, 2024

Audited Source File's Address:

https://github.com/Macaromswap/contract-swap/blob/main/contracts/Staking.sol

The goal of this audit is to review Macaron's solidity implementation for its Staking 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 Macaron 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:

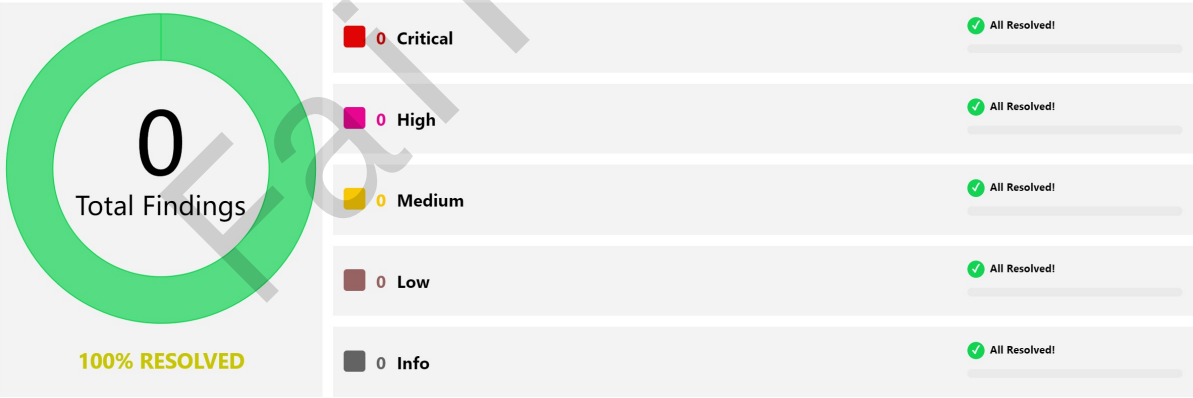
Website:https://macaron.xyz/

Source Code: https://github.com/Macaromswap/contract-swap/blob/main/contracts/Staking.sol

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

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2024041900012025	Fairyproof Security Team	Apr 15, 2024 - Apr 19, 2024	Passed



Summary:

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

02. About Fairyproof

Fairyproof is a leading technology firm in the blockchain industry, providing consulting and security audits for organizations. Fairyproof has developed industry security standards for designing and deploying blockchain applications.

03. Introduction to Macaron

Macaron is a decentralized exchange similar to UniswapV2.

The above description is quoted from relevant documents of Macaron.

04. Major functions of audited code

The audited code mainly implements a single-token staking aggregator. After the users stake specific tokens into the aggregator the aggregator will staking the tokens to a third-party application to earn profits. A portion of the profits will be charged by the aggregator but point rewards will be given.

Note:
The aggregator stakings tokens to third-party applications. This operation has potential risks.
Note:
The aggregator employs the `Transparent Proxy` pattern which means the contract is upgradeable. Before doing a contract upgrade audits need to be done.

05. Coverage of issues

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

- Access Control
- Admin Rights
- Arithmetic Precision
- Code Improvement
- Contract Upgrade/Migration
- Delete Trap
- Design Vulnerability
- DoS Attack

- EOA Call Trap
- Fake Deposit
- Function Visibility
- Gas Consumption
- Implementation Vulnerability
- Inappropriate Callback Function
- Injection Attack
- Integer Overflow/Underflow
- IsContract Trap
- Miner's Advantage
- Misc
- Price Manipulation
- Proxy selector clashing
- Pseudo Random Number
- Re-entrancy Attack
- Replay Attack
- Rollback Attack
- Shadow Variable
- Slot Conflict
- Token Issuance
- Tx.origin Authentication
- Uninitialized Storage Pointer

06. Severity level reference

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

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

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

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

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

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

07. Major areas that need attention

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

- Function Implementation

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

- Access Control

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

- Token Issuance & Transfer

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

- State Update

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

- Asset Security

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

- Miscellaneous

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

08. issues by severity

- N/A

09. Issue descriptions

- N/A

10. Recommendations to enhance the overall security

We list some recommendations in this section. They are not mandatory but will enhance the overall security of the system if they are adopted.

- Consider managing the owner's access control with great care and transfering it to a multi-sig wallet or DAO when necessary.
- In the `updatePool` function, if the third-party application encounters issues, the returned `underlying token` will be less than the quantity recorded in the contract and this will not count the current reward but update the contract's reward record with the returned quantity. Here is the code snippet:

```
 uint256 coinReward = totalBalance > pool.lastTotalBalance ? totalBalance -
pool.lastTotalBalance : 0;

     if (coinReward > 0 && pool.totalBalance > 0) {
         pool.accCoinPerShare += (coinReward * ACC_COIN_PRECISION) /
pool.totalBalance;
     }

     pool.lastTotalBalance = totalBalance;
```

  In case this happens, it will lead to a wrong reward. Consider resetting the transaction to pause the contract when `totalBalance < pool.lastTotalBalance` holds true.

11. Appendices

11.1 Unit Test

1. Staking.t.js

```
const {
    loadFixture,
} = require("@nomicfoundation/hardhat-toolbox/network-helpers");
const { anyValue } = require("@nomicfoundation/hardhat-chai-matchers/withArgs");
const { expect, assert } = require("chai");
const { ethers } = require("hardhat");


describe("Staking Unit Test", function () {
    async function deployStakingFixture() {
        const [owner, alice,bob,...users] = await ethers.getSigners();
        const StandardToken = await ethers.getContractFactory("StandardToken");
        const usdt = await StandardToken.deploy("USDT","USDT",
ethers.parseEther("10000000000000"));

        const MockCErc20 = await ethers.getContractFactory("MockCErc20");
        const cUsdt = await MockCErc20.deploy("cUsdt","cUsdt",0);
        const dUsdt = await MockCErc20.deploy("dUsdt","dUsdt",0);
        await cUsdt.setUnderlying(usdt.target);
        await dUsdt.setUnderlying(usdt.target);
        await usdt.transfer(cUsdt.target,ethers.parseEther("50000"));
        await usdt.transfer(dUsdt.target,ethers.parseEther("50000"));

        const Staking = await ethers.getContractFactory("Staking");
        const staking = await Staking.deploy();

        await staking.initialize(users[0].address);
        await staking.addPool(usdt.target,false);
        // await staking.setTokenToCErc20(usdt.target,cUsdt.target);
        await staking.setProfitPercent([usdt.target],[1000]);

        return {
            staking,usdt,cUsdt,dUsdt,owner,alice,bob,users
        }
    }

    it("metadata test", async () => {
        const {staking,owner,alice,bob,usdt,cUsdt}  = await
loadFixture(deployStakingFixture);
        expect(await staking.addedPool(usdt.target)).true;
        expect(await staking.addedPool(cUsdt.target)).false;
        expect(await staking.tokenToPoolId(usdt.target)).eq(0);
```

```
        expect(await staking.tokenToPoolId(cUsdt.target)).eq(0);
        // expect(await staking.tokenToCErc20(usdt.target)).eq(cUsdt.target);
    });

    it("deposit test without cerc20", async () => {
        const {staking,owner,alice,bob,usdt,cUsdt,users}  = await
loadFixture(deployStakingFixture);

        await usdt.transfer(alice.address,ethers.parseEther("10"));
        await
usdt.connect(alice).approve(staking.target,ethers.parseEther("10000000"));

        await
expect(staking.connect(alice).deposit(0,ethers.parseEther("1.0"))).to.emit(
            staking,"Deposit"
        ).withArgs(
            alice.address,
            usdt.target,
            ethers.parseEther("1.0")
        );

        expect(await
usdt.balanceOf(staking.target)).eq(ethers.parseEther("1.0"));
        await
expect(staking.connect(alice).withdraw(0,ethers.parseEther("1.1"))).to.revertedWi
th(
            "Amount exceeded"
        );

        await
expect(staking.connect(alice).withdraw(0,ethers.parseEther("1.0"))).to.emit(
            staking,"Withdraw"
        ).withArgs(
            alice.address,
            usdt.target,
            ethers.parseEther("1.0"),
            0,
            users[0].address,
            0
        );
        expect(await usdt.balanceOf(staking.target)).eq(0);

    });

    describe("Loan migrate unit test", function() {
        it("setToken should withdraw old and doposit new", async () => {
            const {staking,owner,alice,bob,usdt,cUsdt,dUsdt,users}  = await
loadFixture(deployStakingFixture);
            await usdt.transfer(alice.address,ethers.parseEther("10"));
            await
usdt.connect(alice).approve(staking.target,ethers.parseEther("10000000"));
            await staking.connect(alice).deposit(0,ethers.parseEther("1.0"));
            let pool = await staking.poolInfo(0);
            expect(pool[1]).eq(ethers.parseEther("1.0"));

            await staking.connect(alice).withdraw(0,ethers.parseEther("0.2"));
```

```
            pool = await staking.poolInfo(0);
            expect(pool[1]).eq(ethers.parseEther("0.8"));

            await
expect(staking.setTokenToCErc20(usdt.target,cUsdt.target)).emit(
                usdt,"Transfer"
            ).withArgs(staking.target,cUsdt.target,ethers.parseEther("0.8"));



            pool = await staking.poolInfo(0);
            expect(pool[1]).eq(ethers.parseEther("0.8"));
            expect(await staking.lentAmounts(usdt)).eq(ethers.parseEther("0.8"));

            await
expect(staking.setTokenToCErc20(usdt.target,dUsdt.target)).emit(
                usdt,"Transfer"
            ).withArgs(staking.target,dUsdt.target,ethers.parseEther("0.96"));

            pool = await staking.poolInfo(0);
            expect(pool[1]).eq(ethers.parseEther("0.8"));
            expect(await staking.lentAmounts(usdt)).eq(ethers.parseEther("0.8"));

            await staking.updatePool(0);
            let pool_before = await staking.poolInfo(0);

            await staking.updatePool(0);
            let pool_after = await staking.poolInfo(0);
            expect(pool_after[4]).eq(pool_before[4]);



            let underlying = await
staking.getBalanceOfUnderlying.staticCall(dUsdt.target,staking.target);
            expect(underlying).eq(ethers.parseEther("0" + (0.96 * 1.2)));
            // (0.8 * 1.2 * 1.2) - 0.8 = 0.44
            let per_share = ethers.parseEther("0.44") *
ethers.getBigInt(10000000);
            expect(pool_after[4]).eq(per_share);
            // 0.8 * 1.2 * 1.2
            expect(pool_after[2]).eq(ethers.parseEther("1.152"));
            await dUsdt.setRate(14);
            underlying = await
staking.getBalanceOfUnderlying.staticCall(dUsdt.target,staking.target);
            expect(underlying).eq(ethers.parseEther("1.344"));
            await staking.updatePool(0);
            // (0.8 * 1.2 * 1.4  - 0.8 * 1.2 * 1.2) / 0.8 = 0.24
            //  0.24 + 0.44 = 0.68
            pool_after = await staking.poolInfo(0);
            per_share = ethers.parseEther("0.68") * ethers.getBigInt(10000000);
            expect(pool_after[4]).eq(per_share);
            expect(pool_after[2]).eq(ethers.parseEther("1.344"));
            // 0.96 * 1.4 - 0.8 = 0.544; fee = 0.544 * 0.1;
            let reward = ethers.parseEther("0.544");
            await
expect(staking.connect(alice).withdraw(0,ethers.parseEther("0.4"))).emit(
```

```
                staking,"Withdraw"
            ).withArgs(
                alice.address,
                usdt.target,
                ethers.parseEther("0.4"),
                reward * ethers.getBigInt(9) / ethers.getBigInt(10),
                users[0].address,
                reward/ ethers.getBigInt(10),
            );


            pool_after = await staking.poolInfo(0);
            expect(pool_after[1]).eq(ethers.parseEther("0.4"));
            expect(pool_after[2]).eq(ethers.parseEther("1.344") - reward -
ethers.parseEther("0.4"));
            let user_info = await staking.userInfo(0,alice.address);
            expect(user_info[0]).eq(ethers.parseEther("0.4"));
            expect(user_info[1]).eq(reward / ethers.getBigInt(2));
            expect(user_info[2]).eq(0);

            await
expect(staking.connect(alice).withdraw(0,ethers.parseEther("0.4"))).emit(
                staking,"Withdraw"
            ).withArgs(
                alice.address,
                usdt.target,
                ethers.parseEther("0.4"),
                anyValue,
                users[0].address,
                0
            );

            pool_after = await staking.poolInfo(0);
            expect(pool_after[1]).eq(0);
            expect(pool_after[2]).eq(0);

            user_info = await staking.userInfo(0,alice.address);
            expect(user_info[0]).eq(0);
            expect(user_info[1]).eq(0);
            expect(user_info[2]).eq(0);
        });
    });

});
```

2. StakingFork.t.js

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


describe("Staking Unit Test", function () {
    const holder = "0x938941ff182a3142b2e6c8f4e2e68c0e259a3e3e";
    const dot_address = "0xabd106de1c36b4ac5d51f0c41bd08eecd5c8bca6";
    const c_dot = "0x5C8840E0d935170B75b6EE31265D0c7a42E79481";

    async function deployStakingFixture() {
        const [owner, alice,bob,...users] = await ethers.getSigners();
        const dot = await ethers.getContractAt("StandardToken",dot_address);

        const Staking = await ethers.getContractFactory("Staking");
        const staking = await Staking.deploy();

        await staking.initialize(users[0].address);
        await staking.addPool(dot.target,false);
        await staking.setTokenToCErc20(dot_address,c_dot);
        await staking.setProfitPercent([dot_address],[1000]);

        let rich = await ethers.getImpersonatedSigner(holder);
        await owner.sendTransaction({
            to:holder,
            value:ethers.parseEther("10")
        });
        await dot.connect(rich).approve(staking.target,ethers.MaxUint256);

        return {
            staking,rich,dot,owner,alice,bob,users
        }
    }

    describe("Loan migrate unit test", function() {
        it("deposit and withdraw unit test", async () => {
            const {staking,owner,alice,bob,dot,rich,users}  = await
loadFixture(deployStakingFixture);
            const deposit_value = ethers.parseEther("10.0");
            await staking.connect(rich).deposit(0,deposit_value);
            let pool = await staking.poolInfo(0);
            expect(pool[1]).eq(deposit_value);
            expect(pool[2]).eq(deposit_value);
            expect(pool[4]).eq(0);

            const block = await ethers.provider.getBlockNumber()
            await ethers.provider.send("hardhat_mine",[block + 1000]);
            await staking.updatePool(0);
            pool = await staking.poolInfo(0);
            expect(pool[1]).eq(deposit_value);
```

```
            await staking.connect(rich).withdraw(0,ethers.parseEther("4.0"));
            pool = await staking.poolInfo(0);
            expect(pool[1]).eq(ethers.parseEther("6.0"));


            await staking.connect(rich).withdraw(0,ethers.parseEther("6.0"));
            pool = await staking.poolInfo(0);
            expect(pool[1]).eq(0);

            balance = await
staking.getBalanceOfUnderlying.staticCall(c_dot,staking.target);
            console.log("balance:",balance);
            await staking.updatePool(0);

        });
    });

});
```

3. MockCErc20.sol

```
pragma solidity ^0.8.23;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "./interfaces/ICErc20.sol";

contract MockCErc20 is ICErc20,ERC20 {

    ERC20 public underlying;
    uint public rate = 12;

    constructor(string memory name, string memory symbol, uint initSupply)
ERC20(name, symbol){
        _mint(msg.sender,initSupply);
    }

    function setRate(uint _rate) external {
        rate = _rate;
    }

    function setUnderlying(address _underlying) external {
        underlying = ERC20(_underlying);
    }

    function mint(uint mintAmount) external returns (uint) {
        underlying.transferFrom(msg.sender, address(this), mintAmount);
        _mint(msg.sender,mintAmount);
        return 0;
```

```
    }

    function redeem(uint redeemTokens) external returns (uint) {
        underlying.transfer(msg.sender, redeemTokens * rate / 10);
        _burn(msg.sender,redeemTokens);
        return 0;
    }

    function redeemUnderlying(uint redeemAmount) external returns (uint) {
        underlying.transfer(msg.sender, redeemAmount);
        _burn(msg.sender,redeemAmount * 10 / rate);
        return 0;
    }

    function balanceOf(address owner) public override(ICErc20,ERC20) view returns
(uint256) {
        return ERC20.balanceOf(owner);
    }

    function balanceOfUnderlying(address owner) external view returns (uint) {
        return balanceOf(owner) * rate / 10;
    }
}
```

4. UnitTestOutput

```
  Staking Unit Test
    ✓ metadata test (2096ms)
    ✓ deposit test without cerc20 (96ms)
    Loan migrate unit test
      ✓ setToken should withdraw old and doposit new (227ms)


  3 passing (2s)
```

11.2 External Functions Check Points

1. Staking.sol.md

File: contracts/Staking.sol

contract: Staking is OwnableUpgradeable, ReentrancyGuardUpgradeable

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

Index	Function	StateMutability	Modifier	Param Check	IsUserInterface	Unit Test	Miscellaneous
1	initialize(address)		`initializer`			Passed	
2	poolLength()	view				Passed	
3	getCErc20(address)	view				Passed	
4	getCtokenBalance(address,address)	view				Passed	
5	getBalanceOfUnderlying(address,address)					Passed	
6	setBeneficiary(address)		`onlyOwner`			Passed	
7	setProfitPercent(address[],uint256[])		`onlyOwner`			Passed	
8	setTokenToCErc20(address,address)		`onlyOwner`			Passed	
9	addPool(address,bool)		`onlyOwner`			Passed	
10	massUpdatePools()						
11	updatePool(uint256)					Passed	
12	pending(uint256,address)						
13	deposit(uint256,uint256)		`nonReentrant`		Yes	Passed	
14	withdraw(uint256,uint256)		`nonReentrant`		Yes	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