FAIRYPROOF

Macaron Stable Exchange

AUDIT REPORT

Version 1.0.0

Serial No. 2024051400012021

Presented by Fairyproof

May 14, 2024

www.fairyproof.com

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Macaron Stable Coin Exchange project.

Audit Start Time:

May 11, 2024

Audit End Time:

May 14, 2024

Audited Code's Github Repository:

https://github.com/Macaromswap/contract-swap/tree/main/contracts/StableSwap

Audited Code's Github Commit Number When Audit Started:

85e9f8f1e4148139bfed13658e848a9c8a7f17bc

Audited Code's Github Commit Number When Audit Ended:

082f5449c3169e2c3374d86f44e9007d0f6d84c5

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

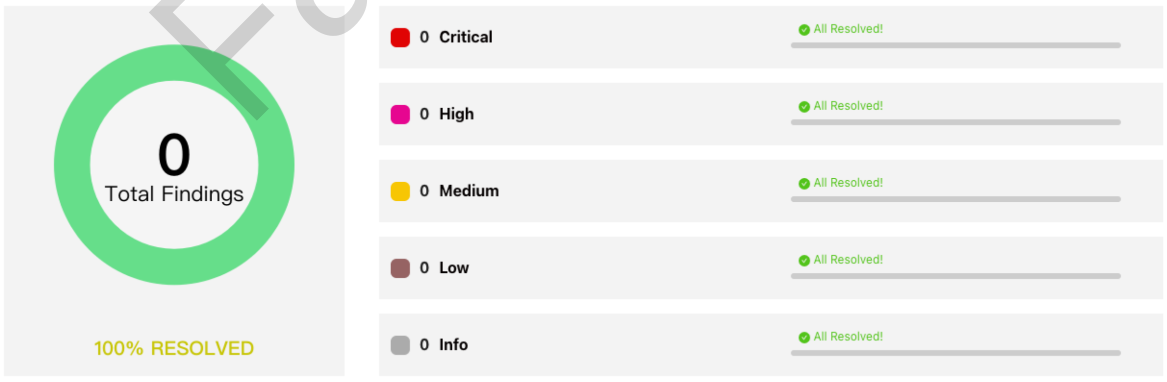
Website:https://macaron.xyz/

Source Code: https://github.com/Macaromswap/contract-swap/tree/main/contracts/StableSwap

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
2024051400012021	Fairyproof Security Team	May 11, 2024 - May 14, 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 deployed on Bitlayer block chain.

The above description is quoted from relevant documents of Macaron.

04. Major functions of audited code

The audited code mainly implements a stable-coin exchange which uses algorithms similar to Curve's.
The trading pairs in the exchange can only be created by the Macaron team. Users can add liquidity with single stable-coins or stable-coin pairs, or exchange stable-coins.

The algorithms used in this exchange improve exchange efficiency compared to conventional DEXs and are specifically good for high volume stable-coin transactions.

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.

11. Appendices

11.1 Unit Test

1. pool.t.js

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

describe("Two Pool Unit Test", function () {

    async function deployFixture() {
        const [owner, alice,bob,...users] = await ethers.getSigners();

        // deploy stable token
        const MacaronStableSwapLP = await
ethers.getContractFactory("MacaronStableSwapLP");
        let usdt = await MacaronStableSwapLP.deploy();
        await usdt.mint(owner.address, ethers.parseEther("100000"));

        let usdc = await MacaronStableSwapLP.deploy();
        await usdc.mint(owner.address, ethers.parseEther("100000"));

        let usde = await MacaronStableSwapLP.deploy();
        await usde.mint(owner.address, ethers.parseEther("100000"));

        // deploy dex
        const MacaronStableSwapFactory = await
ethers.getContractFactory("MacaronStableSwapFactory");
        const factory = await MacaronStableSwapFactory.deploy();

        // create usdt/usdc
        await factory.createSwapPair(
            usdt.target,usdc.target,10000, 5 * 10**8, 1 * 10**8
        );
        // addLiquid
        let pool_address = await factory.allPairs(0);
        const MacaronStableSwapTwoPool = await
ethers.getContractFactory("MacaronStableSwapTwoPool");
        let pool = MacaronStableSwapTwoPool.attach(pool_address);
        await usdt.approve(pool.target,ethers.parseEther("10000"));
        await usdc.approve(pool.target,ethers.parseEther("10000"));
        await pool.add_liquidity(
            [ethers.parseEther("1000"),ethers.parseEther("900")],
            1
        );

        let lp_address = await pool.token();
        let lp = MacaronStableSwapLP.attach(lp_address);

        return {usdt,usdc,usde,owner,alice,bob,factory,pool,lp};
    }

    it("pool info unit test", async () => {
        const {usdt,usdc,usde,owner,alice,bob,factory,pool,lp} = await
loadFixture(deployFixture);
```

```
        let [tokenA,tokenB] = await factory.sortTokens(usdt.target,usdc.target);
        let {swapContract,token0,token1,LPContract} = await
factory.stableSwapPairInfo(usdt.target,usdc.target);
        assert(tokenA === token0,"token0 unmatched");
        assert(tokenB === token1,"token1 unmatched");
        assert(swapContract === pool.target,"pool unmatched");
        assert(lp.target === LPContract,"lp unmatched");
    });

    describe("Liquid unit test", function() {
        it("Add and remove liquid test", async () => {
            const {usdt,usdc,usde,owner,alice,bob,factory,pool,lp} = await
loadFixture(deployFixture);
            await usdt.mint(alice.address,ethers.parseEther("10"));
            await usdc.mint(alice.address,ethers.parseEther("10"));
            await
usdc.connect(alice).approve(pool.target,ethers.parseEther("10000"));
            await
usdt.connect(alice).approve(pool.target,ethers.parseEther("10000"));
            let totalSupply_before = await lp.totalSupply();
            console.log("totalSupply_before:",totalSupply_before);
            await pool.connect(alice).add_liquidity(
                [ethers.parseEther("10"),ethers.parseEther("10")],
                1
            );
            let totalSupply_after = await lp.totalSupply();
            expect(await usdt.balanceOf(alice.address)).eq(0);
            expect(await usdc.balanceOf(alice.address)).eq(0);
            let lp_balance = await lp.balanceOf(alice.address);
            let balance0 = await pool.balances(0);
            let balance1 = await pool.balances(1);
            let [token0,token1] = await
factory.sortTokens(usdt.target,usdc.target);
            let usdt_balance = token0 === usdt.target ? balance0 : balance1;
            let usdc_balance = token0 === usdt.target ? balance1 : balance0;
            let usdt_withdraw = usdt_balance * lp_balance / totalSupply_after;
            let usdc_withdraw = usdc_balance * lp_balance / totalSupply_after;
            await lp.connect(alice).approve(pool.target,ethers.MaxUint256);
            await expect(pool.connect(alice).remove_liquidity(lp_balance,
[1,1])).to.emit(
                pool,"RemoveLiquidity"
            );
            expect(await usdt.balanceOf(alice.address)).eq(usdt_withdraw);
            expect(await usdc.balanceOf(alice.address)).eq(usdc_withdraw);

            // add liquid with only one token
            let add_amount = token0 === usdt.target ? usdt_withdraw :
usdc_withdraw;
            await expect(pool.connect(alice).add_liquidity(
                [add_amount,0],
                1
            )).emit(
                pool,"AddLiquidity"
            );
            // remove liquid to another token;
            lp_balance = await lp.balanceOf(alice.address);
            let target_index = token0 === usdt.target ? 1 : 0;
            await expect(pool.connect(alice).remove_liquidity_one_coin(
                lp_balance,target_index,1
            )).emit(
                pool,"RemoveLiquidityOne"
            );
            console.log(await usdt.balanceOf(alice.address));
            console.log(await usdc.balanceOf(alice.address));
        });
    });

    describe("Swap unit test", function() {
        it("Swap on usdt/usdc", async () => {
            const {usdt,usdc,usde,owner,alice,bob,factory,pool,lp} = await
loadFixture(deployFixture);
            let init_value = ethers.parseEther("10");
            await usdt.mint(alice.address,init_value);
            await usdt.connect(alice).approve(pool.target,ethers.MaxUint256);
            let [token0,token1] = await
factory.sortTokens(usdt.target,usdc.target);
            let usdt_index = token0 === usdt.target ? 0 : 1;
            let usdc_index = token0 === usdt.target ? 1 : 0;
            let usdc_amount = await
pool.get_dy(usdt_index,usdc_index,init_value);
            await
expect(pool.connect(alice).exchange(usdt_index,usdc_index,init_value,1)).emit(
                pool,"TokenExchange"

).withArgs(alice.address,usdt_index,init_value,usdc_index,usdc_amount);

            expect(await usdt.balanceOf(alice.address)).eq(0);
            expect(await usdc.balanceOf(alice.address)).eq(usdc_amount);

            let usdt_amount = await
pool.get_dy(usdc_index,usdt_index,usdc_amount);
            await usdc.connect(alice).approve(pool.target,ethers.MaxUint256);
            await
expect(pool.connect(alice).exchange(usdc_index,usdt_index,usdc_amount,1)).emit(
                pool,"TokenExchange"

).withArgs(alice.address,usdc_index,usdc_amount,usdt_index,usdt_amount);

            expect(await usdt.balanceOf(alice.address)).eq(usdt_amount);
            expect(await usdc.balanceOf(alice.address)).eq(0);
            console.log(ethers.formatEther(usdt_amount));

        });
    });

});
```

2. UnitTestOutput

```
  Two Pool Unit Test
    ✓ pool info unit test (1413ms)
    Liquid unit test
totalSupply_before: 1899999736137607287534n
0n
19454989305970286135n
      ✓ Add and remove liquid test (184ms)
    Swap unit test
9.024999976502749134
      ✓ Swap on usdt/usdc (116ms)


  3 passing (2s)
```

11.2 External Functions Check Points

1. MacaronStableSwapFactory.sol

File: contracts/StableSwap/MacaronStableSwapFactory.sol

contract: MacaronStableSwapFactory is Ownable

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

Index	Function	StateMutability	Modifier	Param Check	IsUserInterface	Unit Test	Miscellaneous
1	allPairsLength()	view					
2	createSwapPair(address,address,uint256,uint256,uint256)		`onlyOwner`	Yes		Passed	
3	getPairInfo(address,address)	view					
4	sortTokens(address,address)	pure					

contract: MacaronStableSwapTwoPool is Ownable, ReentrancyGuard

(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[N_COINS],uint256,uint256,uint256,address,address)			Yes		Passed	OnlyOnce
2	A()	view					
3	get_virtual_price()	view					
4	calc_token_amount(uint256[N_COINS],bool)	view					
5	add_liquidity(uint256[N_COINS],uint256)	payable	`nonReentrant`		Yes	Passed	
6	get_dy(uint256,uint256,uint256)	view				Passed	
7	get_dy_underlying(uint256,uint256,uint256)	view					
8	exchange(uint256,uint256,uint256,uint256)	payable	`nonReentrant`		Yes	Passed	
9	remove_liquidity(uint256,uint256[N_COINS])		`nonReentrant`		Yes	Passed	
10	remove_liquidity_imbalance(uint256[N_COINS],uint256)		`nonReentrant`		Yes		
11	calc_withdraw_one_coin(uint256,uint256)	view					

Index	Function	StateMutability	Modifier	Param Check	IsUserInterface	Unit Test	Miscellaneous
12	remove_liquidity_one_coin(uint256,uint256,uint256)		`nonReentrant`		Yes	Passed	
13	ramp_A(uint256,uint256)		`onlyOwner`				
14	stop_rampget_A()		`onlyOwner`				
15	commit_new_fee(uint256,uint256)		`onlyOwner`				
16	apply_new_fee()		`onlyOwner`				
17	revert_new_parameters()		`onlyOwner`				
18	admin_balances(uint256)	view					
19	withdraw_admin_fees()		`onlyOwner`				
20	donate_admin_fees()		`onlyOwner`				
21	kill_me()		`onlyOwner`				
22	unkill_me()		`onlyOwner`				

contract: MacaronStableSwapLP is ERC20

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

Index	Function	StateMutability	Modifier	Param Check	IsUserInterface	Unit Test	Miscellaneous
1	setMinter(address)		`onlyMinter`		False	Passed	
2	mint(address,uint256)		`onlyMinter`		False	Passed	
3	burnFrom(address,uint256)		`onlyMinter`		False	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