FRP Token

AUDIT REPORT

Version 1.0.0

Serial No. 20221008000220019

Presented by Fairyproof

October 8, 2022

01. Introduction

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

Audit Start Time:

Oct 7, 2022

Audit End Time:

Oct 8, 2022

Audited Source File's Address:

https://bscscan.com/address/0x8eCA5C1B51A801A822912167153041ed0B92a397#code

The goal of this audit is to review FRP'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 FRP 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 function should work:

Website: https://famerewardplus.ai/

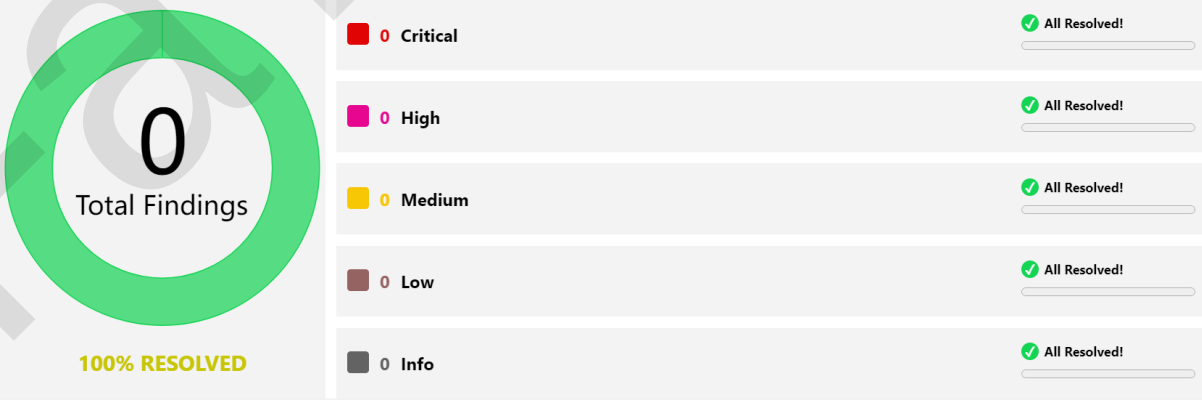
Whitepaper: https://famerewardplus.ai/whitepaper/

Source Code: https://bscscan.com/address/0x8eCA5C1B51A801A822912167153041ed0B92a397#code

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

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2022100800022019	Fairyproof Team	Oct 7, 2022 - Oct 8, 2022	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 FRP

Fame Reward Plus (FRP) is a special and unique community driven token to reward the users of Fame Infinity ecosystem. FAME Infinity Ecosystem has pioneered a new and disruptive approach towards its multi products and services based several digital platforms having cutting edge AI and blockchain technology since its inception. Fame Reward Plus(FRP) is the in-ecosystem digital currency of the Fame Infinity . It helps and enables the users of the ecosystem to earn rewards by doing several activities on our platforms. . At the same time , this can also be traded on supported cryptocurrency exchanges which will facilitate the bigger market access to this in-ecosystem token.

FRP can only be earned in the form of rewards by users of Fame Infinity platform by participating in various activities within Fame Infinity ecosystem. The amount of FRP tokens users will earn/accumulate will solely depend on their performances. This kind of arrangement will inspire users to be a part of this amazing ecosystem to earn FRP tokens.

Nevertheless, earning FRP tokens can be challenging for some users due to certain factors. Those users who find it difficult to earn FRP as rewards, can buy and own FRP tokens from other users at different crypto exchanges.

Monetizing through Fame Infinity ecosystem is manifold and super simple. Earning, holding and trading FRP tokens is one of several ways to make money in a risk free manner in Fame Infinity ecosystem.
FRP aims to derive different kinds of benefits for its users.

04. Major functions of audited code

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

Token Standard: ERC-20

Token Address: 0x8eCA5C1B51A801A822912167153041ed0B92a397 (BNB Chain)

Token Name: FAME REWARD PLUS

Token Symbol: FRP

Decimals: 18

Max Supply: 1,000,000

Misc: No

05. 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
- DoS 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

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.

- 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.

08. List of 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.

- N/A

Appendices

Unit Test

AccountBase.sol

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.6.0;

contract AccountBase {
    address constant public user = address(666);
    address constant public bob = address(444);
    address constant public alice = address(222);
}
```

StandardToken-test.sol

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity 0.8.4;

import "forge-std/Test.sol";
import {StandardToken} from "../src/FRP.sol";
import "./AccountBase.sol";


contract FRPTest is AccountBase, Test {
    StandardToken public token;
    uint constant public INIT_SUPPLY = 100000000000000000000000000;

    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);

    function setUp() public {
        token = new StandardToken("FAME REWARD
PLUS","FRP",18,INIT_SUPPLY,alice,0);
    }

    function testMetaAndInitState() public {
        assertEq(token.name(),"FAME REWARD PLUS");
        assertEq(token.symbol(),"FRP");
        assertEq(token.decimals(),18);
        assertEq(token.owner(),address(this));
        assertEq(token.totalSupply(),INIT_SUPPLY);
        assertEq(token.balanceOf(address(this)),INIT_SUPPLY);
    }

    function testTransferShouldEmitEventAndChangeState() public {
        vm.expectEmit(true,true,false,true,address(token));
        emit Transfer(address(this),alice,10000);
        token.transfer(alice,10000);
        assertEq(token.totalSupply(),INIT_SUPPLY);
        assertEq(token.balanceOf(address(this)),INIT_SUPPLY - 10000);
        assertEq(token.balanceOf(alice),10000);
    }

    function testTransferToZeroAddress() public {
        vm.expectRevert(bytes("ERC20: transfer to the zero address"));
        token.transfer(address(0),10000);
    }

    function testTransferToSelf() public {
        token.transfer(address(this),10000);
        assertEq(token.totalSupply(),INIT_SUPPLY);
        assertEq(token.balanceOf(address(this)),INIT_SUPPLY);
    }

    function testIncreaseAllowanceAndDecreaseAllowance() public {
        vm.expectEmit(true,true,false,true,address(token));
        emit Approval(address(this),alice,12345);
```

```
        token.increaseAllowance(alice,12345);
        assertEq(token.allowance(address(this),alice),12345);

        vm.expectEmit(true,true,false,true,address(token));
        emit Approval(address(this),alice,12345 - 234);
        token.decreaseAllowance(alice,234);
        assertEq(token.allowance(address(this),alice),12345 - 234);
    }

    function testApproveAndTransferFrom() public {
        // should emit event
        vm.expectEmit(true,true,false,true,address(token));
        emit Approval(address(this),alice,12345);
        token.approve(alice,12345);
        // check state
        assertEq(token.allowance(address(this),alice),12345);

        // should be failed
        vm.expectRevert(bytes("ERC20: transfer amount exceeds allowance"));
        vm.prank(alice);
        token.transferFrom(address(this),bob,20000);
        // should emit event
        vm.expectEmit(true,true,false,true,address(token));
        emit Transfer(address(this), bob, 10000);
        vm.prank(alice);
        token.transferFrom(address(this),bob,10000);
        // check state
        assertEq(token.allowance(address(this),alice),12345 - 10000);
        assertEq(token.balanceOf(address(this)),INIT_SUPPLY - 10000);
        assertEq(token.balanceOf(bob),10000);
    }
}
```

output:

```
Running 6 tests for test/StandardToken-test.sol:FRPTest
[PASS] testApproveAndTransferFrom() (gas: 106986)
[PASS] testIncreaseAllowanceAndDecreaseAllowance() (gas: 44312)
[PASS] testMetaAndInitState() (gas: 27416)
[PASS] testTransferShouldEmitEventAndChangeState() (gas: 46009)
[PASS] testTransferToSelf() (gas: 15051)
[PASS] testTransferToZeroAddress() (gas: 9292)
Test result: ok. 6 passed; 0 failed; finished in 1.27ms
```

External Check Points

File:FRP.sol

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

contract: StandardToken is IERC20, Ownable, BaseToken

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	totalSupply()	public				Passed	
5	balanceOf(address)	public				Passed	
6	transfer(address,uint256)	public				Passed	
7	allowance(address,address)	public				Passed	
8	approve(address,uint256)	public				Passed	
9	transferFrom(address,address,uint256)	public				Passed	
10	increaseAllowance(address,uint256)	public				Passed	
11	decreaseAllowance(address,uint256)	public				Passed	
12	owner()	public				Passed	
13	renounceOwnership()	public					
14	transferOwnership(address)	public					

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