Saffron Audit Report

Version 1.0.0

Serial No. 2021080800022025

Presented by Fairyproof

August 8, 2021

FAIRYPROOF

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Saffron project, at the request of the Saffron team.

Audit Start Time:

August 6, 2021

Audit End Time:

August 6, 2021

Audited Code's Github Repository:

https://github.com/saffron-finance/saffron-staking-v2

Audited Code's Github Commit Number When Audit Started:

48cdb2d9683efd6632bf93b34e6cdfb4ec3f15ba

Audited Code's Github Commit Number When Audit Ended:

f4fecd0d59ed0cc0758ea9083947e39a3cf7f27c

Audited Source Files:

The calculated SHA-256 values for the audited files when the audit was done are as follows:

```
SFIRewarder.sol:
0x40a52691411efbdb78c1f350cfd17adb0ec42bfe1fa8dd7988539fe2beeeca6a

SaffronStakingV2.sol:
0xe42aef23397e0ec643f1085e08217aa165f06e8105850088acee804fe6c2cff7

interface/ISFIRewarder.sol:
0x800965c0fc4ea7ed9a05b377f3c407e2b8e3eb256f7ee127c31e793ee97b04a9
```

The source files audited include all the files with the extension "sol" as follows:

```
contracts/
├── SFIRewarder.sol
├── SaffronStakingV2.sol
└── interface
    └── ISFIRewarder.sol
```

The goal of this audit is to review Saffron’s solidity implementation for its staking application, 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 Saffron team for specified versions. Whenever the code, software, materials, settings, enviroment 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 that includes the following
   i. Review of the specifications, sources, and instructions provided to Fairyproof to make sure we understand the size, scope, and functionality of the project's source code.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Fairyproof describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run the test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the source code to improve maintainability, security, and control based on the 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 of truth about how Saffron's staking application should work:

https://github.com/saffron-finance/saffron-staking-v2#saffron-staking

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

— Comments from Auditor

No vulnerabilities with critical, high, medium or low-severity were found in the above source code.

The comments and discovery only apply to the code deployed and run on BSC, HECO, OKExChain and ETH blockchain.

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 Saffron Finance

Saffron is an asset collateralization platform where liquidity providers have access to dynamic exposure by selecting customized risk and return profiles.

04. Major functions of audited code

The audited code implements Saffron's staking application which mainly includes the following functions:

- Staking: users stake specified ERC-20 tokens and will get rewards in the ERC-20 token specified by Saffron before the reward mechanism ends
- Rewards are kept in `SFIRewarder`: rewards are kept in `SFIRewarder` . The staking contract distributes rewards by calling `SFIRewarder` 's interface.
- Admin's access control: parameters that specify the reward mechanism can be modified by the Admin

Note:

- **The third-party libraries the project relies on were not covered by this audit.**
- **The reward token's contract was not covered by this audit.**

05. Key points in audit

During the audit Fairyproof worked closely with the Saffron team and reviewed possible vulnerabilities in the staking functions and here is a finding:

- SaffronStaking.sol

No Need to Require `uint256` `>=0`

Both the `constructor()` function in ine 64 of `SaffronStaking.sol` and the `setRewardPerBlock()` function in line 85 of `SaffronStaking.sol` have the following directive:

```
require(_sfiPerBlock >= 0, "invalid sfiPerBlock");
```

`_sfiPerBlock` is a uint256 variable, its data type ensures it is greater than 0. So this `require` is unnecessary.

Recommendation:

Consider removing `require(_sfiPerBlock >= 0)`

Update:

The directive should be `require( _sfiPerBlock > 0)`. This has been fixed with commit `f4fecd0d59ed0cc0758ea9083947e39a3cf7f27c`.

06. 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
- DDos Attack
- Integer Overflow
- Function Visibility
- Logic Vulnerability
- Uninitialized Storage Pointer
- Arithmetic Precision
- Tx.origin
- Shadow Variable
- Design Vulnerability

- Token Issurance
- Asset Security
- Access Control

07. 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.

08. List of issues by severity

A. Critical

- N/A

B. High

- N/A

C. Medium

- N/A

D. Low

- N/A

09. List of issues by source file

- N/A

10. Issue descriptions

- N/A

11. 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