
# Meta Finance

## AUDIT REPORT

Version 1.0.0

Serial No. 20220601000420l9

Presented by Fairyproof

June 1, 2022

# 01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Meta Finance project.

**Audit Start Time:**

May 25, 2022

**Audit End Time:**

June 1, 2022

**Audited Code's Github Repository:**

https://github.com/MetaFinanceContract/trigger/tree/main/contracts

**Audited Code's Github Commit Number When Audit Started:**

cf4511a4a903fa85ea41e3d518548104d32f2e35

**Audited Code's Github Commit Number When Audit Ended:**

99c68564e525cfa70b981af405bdb5e7dbeebab8

**Audited Source Files:**

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

```
contracts/
├── MetaFinanceClubInfo.sol
├── MetaFinanceIssuePool.sol
├── MetaFinanceTriggerPool.sol
├── events
|   ├── MfiIssueEvents.sol
|   └── MfiTriggerEvents.sol
├── interfaces
|   ├── MfiIssueInterfaces.sol
|   └── MfiTriggerInterfaces.sol
├── storages
|   ├── MfiClubStorages.sol
|   ├── MfiIssueStorages.sol
|   └── MfiTriggerStorages.sol
└── utils
    ├── MfiAccessControl.sol
    └── MfiContractPorxy.sol

4 directories, 12 files
```

The goal of this audit is to review Meta Finance's solidity implementation for its smart router 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 Meta Finance 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 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

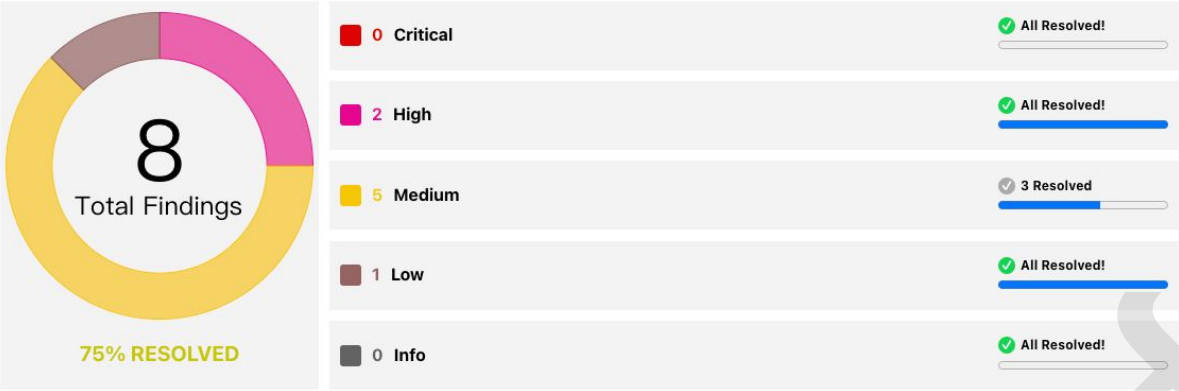
For this audit, we used the following sources of truth about how the smart router function should work:

Smart Contract Files

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

## — Comments from Auditor

| Serial Number | Auditor | Audit Time | Result |
|---|---|---|---|
| 2022060100042019 | Fairyproof Security Team | 2022.05.25 - 2022.06.01 | Medium |




Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, 2 risks of high-severity, 5 risks of medium-severity and 1 risk of low-severity were uncovered. 2 risks of high-severity, 3 risks of medium-severity and 1 risk of low-severity have been fixed, 1 risk of medium-severity has been partially fixed. 1 risk of medium-severity has been acknowledged.

# 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. Major functions of audited code

The audited code mainly implements a smart router on the BNB chain. Users' staked Cake tokens will be invested in third-party applications to earn profits in both Cake and another token.

Note: like all the existing smart router applications, since users' assets will be invested in third-party applications, it is possible users' assets will be lost due to security issues that may happen to these third-party applications.

# 04. 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
- DDoS 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
- Parameter Check
- Design Vulnerability
- Token Issuance
- Admin Rights
- Inappropriate Proxy Design
- Inappropriate Use of Slots
- Asset Security
- Contract Upgrade/Migration
- Code Improvement
- Misc

# 05. 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.

# 06. 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.

## - Implementation of Functions

We checked whether or not all the functions were properly implemented.

We found two issues, for more details, please refer to FP-1 and FP-3 in “08. Issue description”

## - 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 and Transactions

We checked whether or not the contract files that minted tokens or transferred tokens worked normally.

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 found one issue, for more details, please refer to FP-7 in “08. Issue description”

## - Asset Security

We checked whether or not all the functions that transfer assets were safely handled.

We found two issues, for more details, please refer to FP-2 and FP-6 in “08. Issue description”

## - Contract Migration/Upgrade

We checked whether or not the contract files introduced issues or risks associated with contract migration/upgrade.

We found one issue, for more details, please refer to FP-4 in “08. Issue description”.

## - Miscellaneous

We found two issues or risks in other functions or areas at the time of writing, for more details, please refer to FP-5 and FP-8 in “08. Issue description”

# 07. List of issues by severity

| Index | Title | Issue/Risk | Severity | Status |
|---|---|---|---|---|
| FP-1 | Incorrect Algorithm of Reward Calculation | Design Vulnerability | High | ✓ Fixed |
| FP-2 | User Assets Could Be Withdrawn | Admin Rights | High | ✓ Fixed |
| FP-3 | Malfunctioning of Contracts | Design Vulnerability | Medium | Partially Fixed |
| FP-4 | Contract Upgradeable | Contract Upgrade/Migration | Medium | Acknowledged |
| FP-5 | Redundant `receive` Function | Implementation Vulnerability | Medium | ✓ Fixed |
| FP-6 | Missing Interface for Emergent Withdrawal | Implementation Vulnerability | Medium | ✓ Fixed |
| FP-7 | Mutable Address Variables | Admin Rights | Medium | ✓ Fixed |
| FP-8 | Missing Parameter Check | Parameter Check | Low | ✓ Fixed |

# 08. Issue descriptions

## [FP-1] Incorrect Algorithm of Reward Calculation

High ✓ Fixed

Issue/Risk: Design Vulnerability

Description:

In `MetaFinanceIssuePool.sol`, the `earned` function used a formula `new reward = (new reward + existing reward) * coefficient`. When this was called multiple times, the existing reward would be timed by the coefficient multiple times thus causing the actual reward to be less than expected.

The code section was as follows:

```
function earned(address account_) public view returns (uint256) {
    uint256 userEarned =
(_balances[account_].mul(rewardPerToken().sub(userRewardPerTokenPaid[account_]))
.div(1e18).add(rewards[account_]));
    if (metaFinanceClubInfo.userClub(account_) ==
metaFinanceClubInfo.treasuryAddress())
        return
userEarned.mul(metaFinanceClubInfo.noClub()).div(metaFinanceClubInfo.proportion(
));
    return
userEarned.mul(metaFinanceClubInfo.yesClub()).div(metaFinanceClubInfo.proportion
());
}
```

Recommendation:

Consider changing the formula to: `new reward = new reward * coefficient + existing reward`.

Update/Status:

It has been fixed by the Meta Finance team.

## [FP-2] User Assets Could Be Withdrawn

High ✓ Fixed

Issue/Risk: Admin Rights

Description:

In `MetaFinanceTriggerPool.sol`, the `claimTokens` function could only be called by the admin. This function could be used to withdraw surplus ERC-20 tokens. However this function didn't have restrictions on the type of the ERC-20 token that could be withdrawn, therefore users' staked tokens might be withdrawn as well. In emergent cases when users staked assets were transferred to this contract by calling `uploadMiningPool` the admin could take away these assets.

Recommendation:

Consider adding a constraint such that users' staked tokens couldn't be withdrawn.

Update:

The Meta Finance team added the following code:

```
if (token != address(cakeTokenAddress))
      IERC20Metadata(token).safeTransfer(to, amount);
```

Status:

It has been fixed by the Meta Finance team.

## [FP-3] Malfunctioning of Contracts

Medium Partially Fixed

Issue/Risk: Design Vulnerability

Description:

In `MetaFinanceTriggerPool.sol`, there was a precondition for external access: all the staked assets had been invested in third-party applications. When no third-party application address was set or the address was set to 0 (the length of `smartChefArray` was 0), the assets wouldn't be invested. In this case the contract didn't allow operations such as staking, withdrawal or setting `smartChefArray`.

Recommendation:

Consider changing the design by adding an implementation to handle this case.

Update:

The Meta Finance team added some conditional checks such that when the length of `smartChefArray` is 0, the assets will not be invested. In addition, the team will prevent this issue from happening by acting with more care such as avoiding setting `smartChefArray` to a NULL array.

Status:

It has been partially fixed and the team plans to fix it in future upgrades.

## [FP-4] Contract Upgradeable

Medium Acknowledged

Issue/Risk: Contract Upgrade/Migration

Description:

In order for the implementation to be more scalable and adaptable, the contracts were designed as upgradeable. However this might introduce issues or risks if a contract upgrade is not handled properly.

Recommendation:

Consider handling contract upgrade with great care and conducting a contract audit prior to a contract upgrade.

Update:

The purpose of designing contracts as upgradeable was to handle emergent cases, optimize code and make the contracts adaptable to changing situations. The Meta Finance team will handle contract upgrade with great care and conduct a contract audit prior to a contract upgrade.

Status:

It has been acknowledged by the Meta Finance team.

## [FP-5] Redundant `receive` Function

Medium ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

In `MetaFinanceTriggerPool.sol`, `receive() external payable {}` could accept ETHs sent by users. However this function is redundant. If a user mistakenly sent ETHs to it, the ETHs would never be taken back.

Recommendation:

Consider removing this function.

Update/Status:

It has been fixed by the Meta Finance team.

## [FP-6] Missing Interface for Emergent Withdrawal

Medium ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

In `MetaFinanceTriggerPool`, the `projectPartyEmergencyWithdraw` function was designed to withdraw assets in emergent cases, however it didn't have a variable to record the withdrawal status. Therefore users could still stake or withdraw in emergent cases, and might not be able to withdraw staked assets in emergent cases.

Recommendation:

Consider adding an interface for withdrawing tokens in emergent cases. When emergent cases happen, users can only withdraw staked assets, not rewards.

Update:

The Meta Finance team added a variable to record the withdrawal status and an interface for withdrawing tokens in emergent cases.

Status:

It has been fixed by the Meta Finance team.

## [FP-7] Mutable Address Variables

Medium ✓ Fixed

Issue/Risk: Admin Rights

Description:

In `MetaFinanceTriggerPool`, the values of `MetaFinanceIssuePool` and `MetaFinanceClubInfo` could be updated by the `setExternalContract` function. However after the values were updated, calculation of users' rewards would be prone to errors.

Recommendation:

Consider removing the function.

Update:

The function has been removed.

Status:

It has been fixed by the Meta Finance team.

## [FP-8] Missing Parameter Check Low ✓ Fixed

Issue/Risk: Parameter Check

Description:

Some functions didn't have parameter checks such that when some parameters were improperly set, these functions wouldn't work normally. Here were the cases:

- In `MetaFinanceClubInfo.sol`, the `boundClub` function didn't check whether or not `clubAddress_` was a zero-address. When it was, the function could be called repeatedly thus generating redundant dirty data.
- In `MetaFinanceClubInfo.sol`, the `setClubProportion` function didn't check the equality of the two parameters.
- In `MetaFinanceTriggerPool.sol`, the `setFeeRatio` function didn't validate the `newTreasuryRatio_` parameter.

Recommendation:

Consider adding checks for these parameters.

Update/Status:

It has been fixed by the Meta Finance team.

# 09. 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.

1. In `MetaFinanceClubInfo.sol`, consider adding functions to read the lengths of `userArray` and `clubArray`.

   Update: it has been fixed.

2. In the `rewardPerToken` function defined in `MetaFinanceIssuePool.sol`, consider changing `block.timestamp < lastUpdateTime` to `block.timestamp <= lastUpdateTime`.

   Update: it has been fixed.

3. In the `setProportion` function defined in `MetaFinanceIssuePool.sol`, consider removing `difference = difference != 0 ? difference : 1;` and changing the third `if` to `else if`.

Update: the redundant code has been removed

4. In the `notifyRewardAmount` function defined in `MetaFinanceIssuePool.sol`, when `startingTime_` is greater than 0, it shouldn't be less than `lastUpdateTime`.

   Update: `startingTime_` will not be set to 0 only when it is initialized. Other than that it will alway be set to 0.

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