Monstar

AUDIT REPORT

Version 1.0.0

Serial No. 20211117000220l9

Presented by Fairyproof

November 17, 2021

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Monstar project's yield farming function.

Audit Start Time:

November 12, 2021

Audit End Time:

November 16, 2021

Audited Source Files:

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

```
LPPool.sol       :   0xd7417f68065a7ff1f548a10dcf1373f12c559d32328271a7a4a11ab65bf7eb6f
SinglePool.sol   :   0x8903ee47861b281cc913b793c0abd44e637583c4699771960ff086d4b60ca8f6
IERC20.sol       :   0x2dc2e051247adbac96e500ddec5a8ed56900ff65d9aeabd3990ff75fd3738596
IMdexFactory.sol:   0x06c41c4c6bf30b24be40e4c3ac95b5dc111e29f42b96536af827086003d91473
IMdexOracle.sol  :   0x465ddae4f07fa5759f3eb14a485d16b905c7c6bed05b181612bc44ce8846a612
IMdexPair.sol    :   0x263401b1c0360a094179f8ecac62ce48cee646e728c021fa1bf385b13b516ebe
IMdexRouter.sol  :   0xab784c505cb6269473093c2779bc83192c15f092e4e357706e175309e55d062d
Manager.sol      :   0x9c2fabf52d649b553923aec9aafe2fbb1718c0c069aff6f627787bda0072b723
Member.sol       :   0xbd836a0e594211876515b3a797aeda6e1d16848459656c660a319726e17440bd
Ownable.sol      :   0x994cd5c94a9c405c7c4dbc5e26d0557b7f9b8c5e575ff3bdf56fe17615b0fdc2
PriceOracle.sol  :   0x8081e4129f0ca8c1970ff72e49f76f27d2dded227fa94af131d2571af4c97c58
PriceSniffer.sol:   0xaeaf134e49d0c05c60f1535c84a64fe6e824744751e33d4126b8fcad437098e3
Recover.sol      :   0xd78910f7a97a5e150d02ed1f4e0f1c6b0d781261236d22e52d3d1b3ed4985922
```

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

```
contracts/
├── LPPool.sol
├── SinglePool.sol
└── lib
    ├── IERC20.sol
    ├── IMdexFactory.sol
    ├── IMdexOracle.sol
    ├── IMdexPair.sol
    ├── IMdexRouter.sol
    ├── Manager.sol
    ├── Member.sol
    ├── Ownable.sol
    ├── PriceOracle.sol
    ├── PriceSniffer.sol
    └── Recover.sol
```

```
1 directory, 13 files
```

The goal of this audit is to review Monstar's solidity implementation for its blockchain game, 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 Monstar 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 the blockchain game should work:

https://monstar.game/

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

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2021111700022019	Fairyproof Security Team	November 12, 2021 - November 16, 2021	Passed

Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit 1 risk of medium-severity and 1 risk of low-severity were discovered and 1 neutral suggestion was listed.

The risks of low-severity and medium-severity have been fixed and the neutral suggestion was ignored.

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 Monstar

Monstar is a blockchain game. In 2039, humans have colonized the entire Milky Way. In the infinite universe, however, there are incredible creatures apart from human being born with mysterious power. Trainers domesticate these alien monsters to help them live harmoniously with human beings. To make MonStars stronger, the Stella League holds battles and competitions where the monsters fight to hone their skills.

04. Major functions of audited code

The audited code mainly has the following functions:

- Yield Farming with SinglePool

The SingPool is a yield farming pool. Users can stake the platformToken to earn profits. Users can stake a non-platformToken to earn profits as well. But when users stake a non-platformToken, they need to stake the platformToken together. The amount of the platformTokens that are needed for staking is associated with the amount of staking assets, the staking asset's price and the platformToken's price.

- When the staking asset is the platformToken, the amount of platformTokens needed for staking = the amount of the staking assets, and the reward = the amount of the staking assets * the staking asset's price * 200
- When the staking asset is not the platformToken, the amount of platformTokens needed for staking = 1/4 * (the staking asset's price * the amount of the staking assets)/platformToken's price , and the reward = 5/4 * the amount of the staking assets * the staking asset's price * 100

The staking asset's price is retrieved from the PriceOracle contract.

- Yield Farming with LPPool

The LPPool is a yield farming pool. Users can stake ERC-20 tokens to earn profits in an ERC-20 token.

- PriceOracle

The `PriceOracle.sol` is an oracle used as a price feed. It is implemented based on UniswapV2's oracle design.

- Migration of Reward Tokens

The `Recover.sol` is a contract that is used to migrate reward tokens. When emergent cases happen, the admin can migrate reward tokens to a specified address.

- Admin Rights

The admin has the following rights:

- updating the oracle contract's address
- updating the staking asset's whitelist
- pausing yield farming and migrating all the reward tokens in staking pools to a specific address
- updating yield farming's core parameters
- pausing staking of crypto assets

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
- 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
- Design Vulnerability
- Token Issurance
- Admin Rights
- Inappropriate Proxy Design
- Inappropriate Use of Slots
- Asset Security
- Contract Upgrade/Migration
- Code Improvement

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.

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

07. List of issues by severity

Index	Description	Issue/Risk	Severity	Status
N1	Pooled Tokens Could Be Migrated	Admin Rights	Medium	Fixed
N2	Incorrect Algorithm of Calculating `rewardRate`	Implementation Vulnerability	Low	Fixed
N3	Missing Check for Zero-address	Code Improvement	Neutral	Ignored

08. Issue descriptions

[N1] [Medium] Pooled Tokens Could Be Migrated

Risk Severity: Medium

Issue/Risk: Admin Rights

Description:

In the constructors of `LPPool.sol` and `SinglePool.sol` the staking token and the reward token could be defined as the same token. When the staking token and the reward token were the same token, the admin could call the `takeBack` function to withdraw the staking tokens

Recommendation:

1. In the constructor of `LPPool.sol`, consider adding a validity check to prevent the staking token from being the same as the reward token.
2. The `SinglePool.sol` recorded the total amount of staking tokens and the total amount of platformtokens. Consider adding a code section to check whether or not the remaining tokens are sufficient for users to withdraw their staking tokens and reward tokens whenever a withdrawal of reward tokens is executed. If not sufficient, the withdrawal would be reverted.

Status: it has been fixed by the Monstar team.

[N2] [Low] Incorrect Algorithm of Calculating `rewardRate`

Risk Severity: Low

Issue/Risk: Implementation Vulnerability

Description:

In `SinglePool.sol` when the staking token and the reward token were the same, the balance of `rewardToken` was actually the balance of the reward token and the staking token. When the staking token and the reward token were the same token, the balance of `rewardToken` was greater than the real balance of the reward token. In this case the `rewardRate` calculated by the algorithm would be incorrect.

Recommendation:

Consider using the following algorithm to calculate the real balance of the reward token prior to verifying `rewardRate`:

balance of reward token = balance of `rewardToken` - balance of the staking token

Status: it has been fixed by the Monstar team.

[N3] [Neutral] Missing Check for Zero-address

Risk Severity: Neutral

Issue/Risk: Code Improvement

Description:

The `recoverERC20` and `recoverBasic` functions defined in the `Recover.sol` file didn't check whether the input address was a zero-address. If the input address was a zero-address tokens would be transferred to the zero-address.

Recommendation:

Consider adding a check for the input address.

Status: the Monstar team replied that the `transfer` function had such a check so this suggestion was ignored.

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.

- Adding An Event for Emergent Migrations of Reward Tokens

When an admin migrated reward tokens in emergent cases, no event was issued.

Recommendation:

Consider adding an event for this action.

Status: it has been fixed by the Monstar team and an event has been defined for this action.

- Don't Apply A Token with A Callback Function or A Token Whose Amount Will Be Deducted in Transfer To Staking

Staking a token whose transfer function has a callback function would expose the application to re-entrancy attack. A token whose amount will be deducted in a transfer is inappropriate for staking.

Recommendation:

Consider disallowing these kinds of tokens to be staked.

Status: it has been acknowledged by the Monstar team.

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