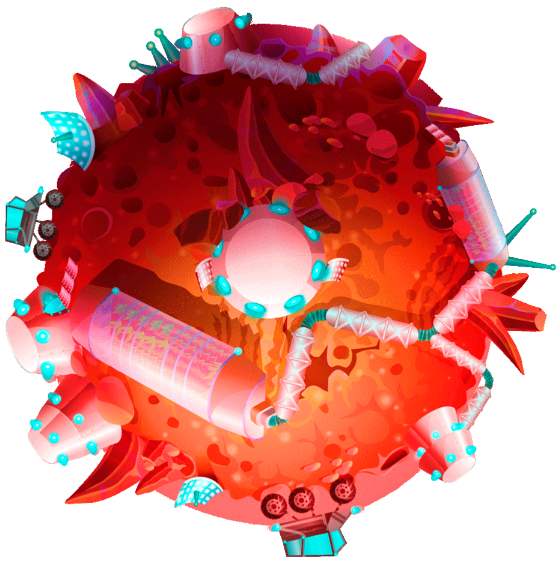

Mars Colony Liquidity Mining Security Analysis

by Pessimistic

This report is public

March 24, 2022

Abstract

In this report, we consider the security of smart contracts of Mars Colony Liquidity Mining project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

Summary

In this report, we considered the security of Mars Colony Liquidity Mining smart contracts. We performed our audit according to the procedure described below.

The audit showed several issues of medium severity, including a Bug, ERC20 standard violation, and Tests issues. Also, several low-severity issues were found.

After the initial audit, the codebase was updated. In this update, the developers fixed most of the issues and added new tests.

General recommendations

We recommend using multisig and implementing CI to run tests, calculate code coverage, and analyze code with linters and security tools.

Project overview

Project description

For the audit, we were provided with Mars Colony Liquidity Mining project on a public GitHub repository, commit 07880b29f9147b18e90483945b30850897b8f6f8.

The scope of the audit included only **ColonyChef.sol** file.

The project has a **README.md** file with a short description of the project. The code contains useful comments.

Three tests out of 66 do not pass. The overall code coverage of the scope is 85.07%.

The total LOC of audited sources is 143.

Codebase update

After the initial audit, the codebase was updated. For the recheck, we were provided with commit 870abf2a09c5044b0fce89c1df23ce6444467ae5.

In this update, the developers fixed most of the issues and added new tests. All 80 tests pass, the code coverage for the scope of the audit increased up to 98.53%.

Procedure

In our audit, we consider the following crucial features of the code:

1. Whether the code is secure.
2. Whether the code corresponds to the documentation (including whitepaper).
3. Whether the code meets best practices.

We perform our audit according to the following procedure:

- Automated analysis
  - We scan the project's codebase with the automated tool Slither.
  - We manually verify (reject or confirm) all the issues found by the tool.

- Manual audit
  - We manually analyze the codebase for security vulnerabilities.
  - We assess the overall project structure and quality.

- Report
  - We reflect all the gathered information in the report.

Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

The audit showed no critical issues.

Medium severity issues

Medium issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

M01. Bug (fixed)

The `deposit` function verifies at line 120 that `_amount >= 0`. However, this check allows adding users to the `providers` list without making a deposit. Also, the error message (`zero deposit`) for this check hints that `_amount` should be greater than `0`.

The issue has been fixed and is not present in the latest version of the code.

M02. ERC20 standard violation (fixed)

ERC-20 standard states:

```
Callers MUST handle false from returns (bool success). Callers MUST
NOT assume that false is never returned!
```

However, the returned values from `transfer` calls are not checked at lines 191 and 193.

The issue has been fixed and is not present in the latest version of the code.

M03. Tests issues (fixed)

The project has tests. However, three tests out of 66 do not pass. The overall code coverage is 85.07%.

Testing is crucial for code security. An audit does not replace tests in any way. We highly recommend covering the codebase with tests and ensuring that all tests pass and the code coverage is sufficient.

The developers added new tests. The code coverage of the scope increased up to 98.53%.

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. Code quality (fixed)

`updatePool` function violates CEI pattern since it updates storage variables after external call to `clnyToken.safeTransferFrom()` at line 113.

The issue has been fixed and is not present in the latest version of the code.

L02. Code quality (fixed)

Consider declaring the following functions as `external` instead of `public` to improve code readability and optimize gas consumption:

1. `providerCount` function.
2. `getProvider` function.
3. `getProviders` function.
4. `deposit` function.
5. `withdraw` function.
6. `emergencyWithdraw` function.

The issues have been fixed and are not present in the latest version of the code.

L03. Project management (fixed)

Optimization is switched off in **truffle.js** config file. The `runs` value is not specified.

The issue has been fixed and is not present in the latest version of the code.

Notes

N01. Overpowered role

The owner of the contract has excessive powers. The owner can:

1. Change `clnyPerSecond` value. This value represents the transfer rate of CLNY tokens from the pool to the contract. Users' rewards depend on this value.

2. Null users' rewards. If the owner updates `clnyPerSecond` variable without prior calling of `fixRewards` function, all users' rewards since the last update will be erased.

3. Change the address of the colony pool. Incorrect address of the pool will stop the transfer of CLNY tokens to the contract and break the whole system.

    > *Comment from developers: Incorrect address will only block deposits and regular withdrawals. However, when the owner changes it to a correct address and give an approve, the contract will accrue all the rewards, so users will not loose anything. Also, users will be able to withdraw deposits using `emergencyWithdrawal` function.*

In the current implementation, the system depends heavily on the owner of the contract. Thus, there are scenarios that can lead to undesirable consequences for the project and its users, e.g., if the owner's private keys become compromised. We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

> *Comment from developers: We are considering our own solution or a solution based on Aragon to manage our smart contracts. https://multisig.harmony.one/ does not fit our requirements for quality management and upgrading contracts, so we are currently using a secure wallet while we explore other solutions.*

This analysis was performed by Pessimistic:

Daria Korepanova, Security Engineer
Evgeny Marchenko, Senior Security Engineer
Nikita Kirillov, Junior Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

March 24, 2022