1inch NETWORK

1inch Farming Security Analysis

by Pessimistic

This report is public

February 08, 2022

Abstract

In this report, we consider the security of farming smart contracts of 1inch project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. A single audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, a security audit is not investment advice.

Summary

In this report, we considered the security of 1inch Farming smart contracts. We performed our audit according to the procedure described below.

The initial audit showed only several issues of low severity. They do not endanger project security.

The overall code quality is high, the code is extremely optimized. However, the codebase lacks NatSpec comments, making the code hard to read and maintain.

Later the developers updated the codebase to fix most issues and simplify the code structure.

General recommendations

We recommend adding NatSpec comments to the codebase.

Project overview

Project description

For the audit, we were provided with 1inch Farming contracts project on a public GitHub repository, commit d191cc14b526e2eab09c266580b84116b2630da8.

The project has README.md file with a detailed high-level description of the project. However, the codebase lacks NatSpec comments.

All 43 tests pass, the code coverage is 99%.

The total LOC of audited sources is 318.

Codebase update

After the initial audit, the codebase was updated. For the recheck, we were provided with commit ae37f32f531cdf470151440e350966643036dc44.

This update improves code structure, addresses most issues, and adds new tests. All tests pass (56 total), new code coverage is 94%.

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 project's code base with automated tools: Slither and SmartCheck.
    - We manually verify (reject or confirm) all the issues found by tools.
- Manual audit
    - We manually analyze code base for security vulnerabilities.
    - We assess 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.

The audit showed no issues of medium severity.

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.

Code quality

- The project codebase is heavily optimized at the expense of the code structure and readability. The lack of NatSpec comments aggravates the situation.

  The codebase was significantly simplified. However, it still lacks NatSpec comments.

- According to OZ documentation, the `_beforeTokenTransfer` function should be declared as `virtual` in **ERC20Farmable** and **FarmingPool** contracts.

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

- Consider declaring the following functions as `external` instead of `public` to improve code readability and optimize gas consumption:
  - `farmingCheckpoint` function of **Farm** contract.
  - `farmedSinceCheckpointScaled` function of **Farm** contract.
  - `farmed` function of **FarmingPool** contract.
  - `exit` function of **FarmingPool** contract.

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

Gas consumption

In **ERC20Farmable** contract, `_beforeTokenTransfer` function contains nested loops that iterate through the lists of users' farms. Consider using another data structure for the list of farms that allow using `O(n)` set comparison.

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Vladimir Tarasov, Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

February 08, 2022