1inch NETWORK

1inch Limit Order protocol Security Analysis

by Pessimistic

This report is public

November 1, 2021

Abstract

In this report, we consider the security of smarts contracts of 1inch Limit Order protocol 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. One 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, security audit is not an investment advice.

Summary

In this report, we considered the security of 1inch Limit Order protocol smart contracts. We performed our audit according to the procedure described belows.

The initial audit showed only a few issues of low severity. The code base is of excellent code quality. The project has the documentation, all test pass without any issues.

After the initial auidt, the code base was updated. All the issues were fixed.

General recommendations

We recommend adding missing NatSpec comments and writing additional tests to improve code coverage.

Project overview

Project description

For the audit, we were provided with 1inch Limit Order protocol project on a public GitHub repository, commit 9d118307df7acc3bcef73407f3964acd6aa0f35c.

The documentation for the project was provided as a README.md file in the repository, the code base has comments and some NatSpecs.

All 54 tests pass, the overall code coverage is 76.81%.

The total LOC of audited sources is 651.

Code base update

After the initial auidt, the code base was updated. For the recheck, we were provided with commit 9325553606f56e64dbffe1f2d02f89a3ad0201db.

In this update, all the issues mentioned in the initial audit were fixed.

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 tool 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 often lead to the loss of funds or other catastrophic failures. The contracts should not be deployed before these issues are fixed. We highly recommend fixing them.

The audit showed no critical issues.

Medium severity issues

Medium issues can influence project operation in 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's operations. However, they might lead to various problems in the future versions of the code. We recommend taking them into account.

Code quality (fixed)

- In **OrderMixin** contract, `result.length` is checked to be greater than `0` in:
  - `simulateCalls()` function at line 115.
  - `_makeCall()` function at line 285.

  We recommend verifying that `result.length` is not less than `32` before decoding it to `bool` variable.

- In **OrderRFQMixin** contract, the code for amounts calculation at lines 107 and 111 duplicates the code from `getTakerAmount()` and `getMakerAmount()` functions of **AmountCalculator** contract.

  Consider reusing the code from **AmountCalculator** contract.

- In **OrderRFQMixin** contract, the code at line 44 (`cancelOrderRFQ()` function) does the same that the code at lines 88–93 (`fillOrderRFQTo()` function). In the latter case, the code performs additional checks. Consider moving this functionality to a separate function to minimize code duplication.

- In `decodeBool()` function of **ArgumentsDecoder** contract, unsafe type conversion is used for `bool` value since any non-zero value will be evaluated to `true`.

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

This analysis was performed by Pessimistic:

Igor Sobolev, Security Engineer
Daria Korepanova, Security Engineer
Evgeny Marchenko, Senior Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

November 1, 2021