1inch NETWORK

1inch Limit Swap Security Analysis

by Pessimistic

This report is public.

Published: June 1, 2021

Abstract

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

The initial audit showed several issues of low severity, mostly of Code quality type.

The project has no documentation that hinders the auditing process.

The overall code quality is good.

After the initial audit, the code base was updated to the latest version. In this update, some functionality was modified, also one issue (typo) was fixed. The audit of the updated code did not reveal any new issues.

General recommendations

We recommend adding the documentation to the project and fixing the issues mentioned in the report.

Project overview

Project description

For the audit, we were provided with 1inch Limit Swap project on a private GitHub repository, commit fc528b390bad66927b9316470e4c86d06df58563.

The project has no documentation.

The project compiles successfully and has tests, the coverage is above 80%.

The total LOC of audited sources is 452.

Latest version of the code

After the initial audit, the code base was updated. For the recheck, we were provided with commit 5da3a74b178b5081dccd11de8a7308534a2c3697.

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 smart contracts security. We highly recommend fixing them.

The audit showed no critical issues.

Medium severity issues

Medium issues can influence project operation in current implementation. We highly recommend addressing them.

No documentation

The project has no documentation, and the code is lacking natspecs. As a result, it is sometimes unclear for the auditor what is the intention of the code, is its behavior correct, and whether the architecture of the project is appropriate.

Considering the complexity of the project, the documentation is critically important not only for the audit but also for development process. It should explicitly explain the purpose and behavior of the contracts, their interactions, and main design choices.

> *Comment from developers: work in progress.*

Low severity issues

Low severity issues can influence project operation in future versions of code. We recommend taking them into account.

Project management

Scripts in `package.json` require `truffle`, `truffle-flattener`, `solhint`, and `eslint` packages to be installed globally, though they are installed locally.

Code quality

- In **LimitOrderProtocol** contract, there are typos at lines 175 and 180. Should be:

```
175  // Check if order is valid
...  ...
180  // Compute maker and taker assets amount
```

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

- When decoding parameters, consider checking the length of the decoded data. This significantly improves the maintainability of the code and helps to identify the reason in case of revert. The issue occurs in:
  - **LimitOrderProtocol** contract at lines 120, 295, and 311
  - **AmountCalculator** contract at line 23
  - **PredicateHelper** contract at lines 35, 40, and 45
- In **LimitOrderProtocol** contract, the name of `simulateTransferFroms()` function at line 114 is confusing as it performs arbitrary calls, not only transfers.

  The issue has been fixed and is not present in the latest version of the code.
- In **LimitOrderProtocol** contract, functions `fillOrderRFQ()` and `_validate()` require detailed documentation or at least natspec comments as their code is complicated and unclear.

Code logic

In **LimitOrderProtocol** contract, `_maxSelector` variable at line 79 should have value `bytes4(uint32(IERC20.transferFrom.selector) + 5)` according to its usage.

Gas consumption

Reading `.length` property of an array on each iteration of a `for` loop is expensive.

Consider saving value of `.length` property to local variables in:

- **PredicateHelper** contract at lines 12 and 23
- **LimitOrderProtocol** contract at lines 99 and 116.

This analysis was performed by Pessimistic:

Igor Sobolev, Security Engineer
Vladimir Tarasov, Security Engineer
Daria Korepanova, Security Engineer
Evgeny Marchenko, Senior Security Engineer
Boris Nikashin, Analyst

June 1, 2021