1inch AggregationRouter V5 Security Analysis

by Pessimistic

This report is public

October 3, 2022

Abstract

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

The initial audit showed A call with no gas limit issue of medium severity and a few low-severity issues.

After the initial audit, the developers updated the code. Most of the issues were fixed.

The code is of high quality. However, the code complexity is above average due to excessive usage of inline assembly and insufficient amount of in-code comments. It poses a risk for future development or any possible integrations.

General recommendations

We recommend providing comments to all assembly blocks.

Project overview

Project description

For the audit, we were provided with the following GitHub repositories:

- 1inch-contract private repo, commit bc00c75f2c99d62e1a206aa2f81c408caba4b370.
- limit-order-protocol public repo, commit 171c5d7bbb280d9f754404828051f2a47fb726df.
- solidity-utils public repo, commit c35dc32fd91ee01f961df13ab7c30faf40be8b89.

The scope of the audit included **AggregationRouterV5** contract and all its dependencies. The most important files are listed below:

- **AggregationRouterV5.sol**, **GenericRouter.sol**, **UnoswapRouter.sol**, **UnoswapV3Router.sol**, and **ClipperRouter.sol** in `1inch-contract` (AggregationRouterV5) repository.
- **LimitOrderProtocol.sol**, **OrderMixin.sol**, **OrderRFQMixin.sol**, **OrderLib.sol**, and **OrderRFQLib.sol** in `limit-order-protocol` repository.
- **UniERC20.sol**, **SafeERC20.sol**, **ECDSA.sol**, and **RevertReasonForwarder.sol** in `solidity-utils` repository.

The documentation for the project included **AggregationRouterV5.docx** file (sha1sum `a993a3a76ca7f3423c9b7c1be2ab161ddb256e59`) with the description of the scope and changes since the previous version.

364 tests pass, 7 pending, 2 failing. The code coverage is 77.42%.

The total LOC of audited sources is 1984.

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.

Inter alia, we check whether:

- Swap operations consume only the specified amount of users' tokens and yield at least the required amount in return.
- Meta-transactions are secure, and the signatures comply with EIP-712.
- Router correctly integrates with Clipper v2.
- Assembly blocks are correct and do not break Solidity compiler conventions.
- Any standard Solidity issues are present in the codebase.

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. A call with no gas limit (fixed)

In **OrderMixin** contract, `fillOrderTo` function makes a low-level call for ether transfer without setting a gas limit. Thus, a malicious user can use it to re-enter the router with another order before `postInteraction` happens.

The issue has been fixed at commit f9ea21f2fc29a62f24c45ff04f4aebc56f4b249b.

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. Possible overflow (fixed)

In **NonceManager** contract, `advanceNonce` public function accepts `uint8` value as an argument. Since contracts can omit compile-time type checks, e.g., via low-level calls, `advanceNonce` function might behave unexpectedly for the caller due to overflow during the typecasting.

The issue has been fixed at commit 5594dae23c43b24f5f0578201c158f922a6c5ec3.

L02. Typo in comments (fixed)

In **LimitOrderProtocol** contract, the comment at line 28 provides a link to a wrong file. The link should point to `OrderRFQMixin.md` instead of `OrderRFQMixin.sol`.

The issue has been fixed at commit 72cd058391d954a897dec5b1fdd9c8ab3cbd2c5d.

L03. Code quality (commented)

We recommend performing flag checking with `data & FLAG == FLAG` expression instead of `... != 0` to improve code readability.

> *Comment from the developers: We have checked gas consumption for both cases and found no difference. So we leave it as is.*

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Ivan Gladkikh, Junior Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

October 3, 2022