Secure Decentralized Solutions

1Inch Exchange Smart Contract Audit

November 11, 2020 Mariana Soffer No Comments

Reading Time: 3 minutes

Contents

Introduction

CoinFabrik was asked to audit the contracts for the 1Inch Exchange. First we will provide a summary of our discoveries and then we will show the details of our findings.

Summary

Initially we received a snapshot of the contracts and we were indicated the scope of our analysis should include the following archives:

File	Sha256
OneInchExchange.sol	5e12bcf4f35d47d889d70d5466a499a66fc779cff04e62c2621465aa447f4fea
GasDiscountCalculator.sol	91237a64ec44539c46bab8affac07587fbd007d4c9098b1546d3c4fa484c6634

OneInchFlags.sol	996358af581cd5a842964d5a63e81e8cd11e0fa86af78d24b1feaa59a3e277d9
helpers/RevertReasonParser.sol	2ade75345efdd2d74cabcf38bc9589a5694324a59f611903a17f01cba7acd26d
helpers/UniERC20.sol	2c7ceb502077357a0f657217fa4e07d15bd875788af9faaaee3d523bfd852333

The audit should focus on these requirements:

- Users' approval on OneInchExchage contract should be safe
- The swap function itself is safe to use

The main function *swap* from OneInchExchange.sol accepts an array of interactions with other contracts. It is used to perform exchange operations between third party contracts in a single transaction optimizing gas costs. Additionally it has the option to further reduce gas costs by burning Chi tokens that take advantage of Ethereum refund mechanism.

While performing the audit a new version of the contracts was deployed at https://etherscan.io/address/0x111111125434b319222cdbf8c261674adb56f3ae#code and we were requested to analyze that version instead.

The most important change in the updated version was the separation of using Chi tokens to refund gas from the exchange operation so a failure in the exchange operation doesn't leave tokens stranded in OneInchCaller. Most other changes were minor and didn't affect the functionality being audited.

Analyses performed:

- Misuse of the different call methods
- Integer overflow errors
- Division by zero errors
- Outdated version of Solidity compiler
- Front running attacks
- Reentrancy attacks
- Softlock denial of service attacks
- Functions with excessive gas cost
- Missing or misused function qualifiers

- Insufficient validation of the input parameters

Detailed findings

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. They must be fixed **immediately**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of but can be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.
- **Enhancement:** These kinds of findings do not represent a security risk. They are best practices that we suggest to implement.

This classification is summarized in the following table:

SEVERITY	EXPLOITABLE	ROADBLOCK	TO BE FIXED
Critical	Yes	Yes	Immediately
Medium	In the near future	Yes	As soon as possible
Minor	Unlikely	No	Eventually
Enhancement	No	No	Eventually

Issues Found by Severity

Critical severity

No issues of critical severity have been found.

Medium severity

No issues of medium severity have been found.

Minor severity

No issues of minor severity have been found.

Enhancements

Minimal gas improvement in _toHex function

In the function _toHex gas can be saved by replacing

```
bytes memory alphabet = "0123456789abcdef";
```

With

```
bytes32 alphabet = 0x3031323334353637383961626364656600000000000000000000000000000000;
```

The code is only used in *RevertReasonParser.parse* in the extraordinary case of a contract failure. In the common case of a successful exchange it is never used so it would not be an improvement.

Other specific analyses performed

- Analysis of IERC20Permit interface to determine if it is safe to use with arbitrary tokens. We concluded that it is safe to use:
    - An ERC20 token that does not implement IERC20Permit interface will ignore the call and make no changes to allowance
    - An ERC20 token that implements IERC20Permit interface should accept the call and make the required changes
    - Most proxies forward calls to the token implementation with DELEGATECALL will behave similar to calling the token directly

    - In case of selector collisions due to IERC20Permit complexity most contracts will revert because of invalid data
- Attempts to spend more gas than the amount of CHI token burned
    - The formula used is not entirely precise and we tried to take advantage of this using precision decimals to round it up
- Validate that certain flag combinations have a reasonable behavior.
    - When _SHOULD_CLAIM and _PARTIAL_FILL options are both enabled the correct balance is calculated considering the transferred amount, and the received amount is in proportion to minimum returned amount configured

Conclusion

The contracts do not have much documentation. We would recommend adding some documentation so it is easier to understand the expected behavior.

The contracts are pretty simple for the task of interacting with third party contracts making them easy to review and understand. They are well written and we haven't found any issue that can be exploited.

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the 1 Inch Exchange. Moreover, it does not provide a smart contract code faultlessness guarantee.