1inch NETWORK × PESSIMISTIC

ATOMIC SWAPS V2 SECURITY ANALYSIS

by Pessimistic

This report is public

August 28, 2024

ABSTRACT

In this report, we consider the security of smart contracts of 1inch cross-chain atomic swaps v2 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 cross-chain atomic swaps v2 smart contracts. We described the audit process in the section below.

The initial audit showed one issue of medium severity: Reuse of secret. Also, one low-severity issue was found.

After the audit, the developers provided us with a new version of the code. In that version, they fixed the found medium and low-severity issues.

The overall code quality is good.

GENERAL RECOMMENDATIONS

We do not have any recommendations.

PROJECT OVERVIEW

Project description

For the audit, we were provided with 1inch cross-chain atomic swaps v2 project on a private GitHub repository, commit b3acc0ff66c3d5665e389464a7f082a18fcb2774.

The scope of the audit included everything.

The documentation for the project included 1inch Fusion Atomic Swaps.

All 79 tests pass successfully. The code coverage is 90.12%.

The total LOC of audited sources is 665.

Codebase update

After the audit, the developers provided us with a new commit: ef6956a527528ff6c842dbdcbbd00a812436cd1f. In that update, they fixed all of the found issues.

83 tests pass successfully. However, GitHub action for ZKSync chain fails.

AUDIT PROCESS

We started the audit on July 22, 2024, and finished on July 26, 2024.

We inspected the materials provided for the audit. After, we performed preliminary research.

During the work, we stayed in touch with the developers and discussed confusing or suspicious parts of the code.

We manually analyzed all the contracts within the scope of the audit and checked their logic. Among other, we verified the following properties of the contracts:

- Whether the integration with ZkSync is correct;
- Potential arithmetic issues;
- The correctness of the new multiple fills functionality.

We scanned the project with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules.

We ran tests and calculated the code coverage.

We combined in a private report all the verified issues we found during the manual audit or discovered by automated tools.

After the initial audit, we discussed the results with the developers. On August 7, 2024 the developers provided us with an updated version of the code. In this update, they fixed all of the issues from our report.

We reviewed the updated codebase, run the aforementioned tools and updated the report accordingly.

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 severity 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. Reuse of secret (fixed)

It is possible that the resolver can fill an already-filled index in the Merkle tree by providing custom data and bypassing checks in the `takerInteraction` function of the **MerkleStorageInvalidator** contract. As a result, the resolver is able to reuse the secret if they get the same `calculatedIndex` in the **BaseEscrowFactory** contract.

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

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. Code clarity (fixed)

The **BaseEscrowFactory** contract contains logic for obtaining hashlock from the Merkle tree. We recommend clarifying the check for the scenario when the `remainingMakingAmount` is filled by substituting `calculatedIndex + 2` with `secretsAmount + 1` at line 85.

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

Notes

N01. Possible calculation inaccuracy (fixed)

When the `secretsAmount` is significantly greater than the `order.makingAmount`, calculation inaccuracy is accrued at line 83 of the **BaseEscrowFactory** contract. In the case, when `order.makingAmount < secretsAmount`, `calculatedIndex + 2` will not be representing the last secret (`secretsAmount +1`), when the order is filled fully.

> The issue has been fixed at commit acdd2659d9d435f63ad484e1821fd93b50669986.

N02. Privileged role (commented)

The swap process requires adherence to a specific timeline. For instance, there exists a period during which only the resolver can unlock tokens on the destination chain. However, there is no check on the minimum length of this period. The project relies on the relayer role for such timeline checks.

> `Comment from the developers:`
> Not an issue. On the one hand, on a frontend that maintains 1inch, timelocks will be taken from a reliable service and given to the user to sign. On the other hand, Resolver is also motivated to validate timelocks, as the user can sign the intervals deliberately favourable to them.

N03. The process of escrow creation (commented)

In the current implementation, it is essential for the resolver to deposit funds into the escrow and fill the order within the same transaction. Otherwise, funds become stuck on the address of an undeployed contract. For a similar reason, the resolver should first create the escrow on the source chain.

> `Comment from the developers:`
> Not an issue. The logic described fits within the protocol design context. Draft for a possible implementation of **Resolver** contract with sending a deposit within a single transaction is implemented in https://github.com/1inch/cross-chain-swap/pull/52. The requirement to first deploy escrow on the source chain and then on the destination chain is reflected in the documentation.

N04. Time constraints might not work properly in case of L2 networks (commented)

Several L2 networks have mechanisms to combat censorship and resolve disputes. Note that these processes may take longer than the time constraints imposed by the cross-chain swap flow. For example, a swap could be completed on the source chain while still pending on the L2 network.

> `Comment from the developers:`
>
> Not an issue. We will adjust the timelocks for all supported L2 chains to reflect possible differences in their finalisation periods.

This analysis was performed by Pessimistic:

Pavel Kondratenkov, Senior Security Engineer
Yhtyyar Sahatov, Security Engineer
Konstantin Zherebtsov, Business Development Lead
Irina Vikhareva, Project Manager
Alexander Seleznev, CEO

August 28, 2024