1inch MerkleDrop Security Analysis

by Pessimistic

This report is public.

Published: September 6, 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

Summary

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

The audit showed a few issues of low severity. However, the owner of the contract has full control over tokens on the contract.

The code is of good quality, it is significantly optimized to minimize gas consumption.

However, the project has no public documentation, and the code is lacking NatSpecs.

After the initial audit, the code base was updated, some fixes were applied.

General recommendations

We recommend adding public documentation to the project and providing the code with detailed NatSpecs.

Project overview

Project description

For the audit, we were provided with 1inch MerkleDrop project on a private GitHub repository, commit 1f8b2a6ed27d1b2d18cf8475e42eece60f41c896.

The project has no public documentation, the code is lacking NatSpecs.

The project compiles successfully. All 39 tests pass, the coverage is 100%.

The total LOC of audited sources is 217.

Code base update

After the initial audit, the code base was updated. For the recheck, we were provided with commit 1026fe507d829ac1708a3393f85f7b91620b0ed8.

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 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.

The audit showed no issues of medium severity.

Low severity issues

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

Code quality

- Avoid using numbers of different radices in assembly block, e.g., at line 94 of **CumulativeMerkleDrop128** and **CumulativeMerkleDrop160** contracts. We recommend using the same radix for numbers all over the project.

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

- Consider removing commented out code.
- Consider removing unused `OZ/MerkleProof` imports from contracts **CumulativeMerkleDrop128** and **CumulativeMerkleDrop160**.

No public documentation

The project has no public documentation, the code base is lacking NatSpecs. Considering the complexity of the code, the lack of the documentations makes the code hard to interact with.

The documentation is a critical part of the project that helps to improve security and reduce risks. Proper documentation should explicitly describe the purpose and behavior of the contracts, their interactions, and main design choices. It is also essential for any further integrations.

Notes

Overpowered owner

The owner of **CumulativeMerkleDrop** contract redistribute rewards arbitrary, remove users from the tree, manipulate their balances, etc.

In the current implementation, the system depends heavily on the owner of the contract. Thus, there are scenarios that may lead to undesirable consequences for users, e.g., if the owner's private keys become compromised.

We recommend designing contracts in a trustless manner or implementing proper key management, e.g., multisig.

> *Comment from developers: the owner is not considered to become a threat since his role is to distribute tokens, not to grab them. To minimize the chance of private keys compromise, the owner account is controlled via multisig.*

This analysis was performed by Pessimistic:

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

September 6, 2021