1inch Exchange Audit

OPENZEPPELIN SECURITY | NOVEMBER 13, 2020

Security Audits

The 1inch Exchange is a DEX aggregator that allows users to perform complex trades using multiple decentralized exchanges. We reviewed part of the on-chain infrastructure used to manage the external calls and redeem CHI tokens as desired.

The review was undertaken by two auditors over one week. The audited commit is `72f2812837fdd73ec2d32c8988811df361e80985` and the scope included the following contracts:

– `OneInchExchange.sol`
– `RevertReasonParser.sol`
– `UniERC20.sol`

All other components of the system were assumed to behave as documented. Importantly, this includes arbitrary calls to external contracts that are expected to perform the token transfers.

Summary

Overall, we are happy with the structure and implementation of the code. Our main recommendation is to introduce thorough documentation to better communicate the expected behavior.

System Overview

The main exchange contract is simply a scaffold around a set of arbitrary calls, which are used to interact with various decentralized exchange contracts to perform the desired token swaps. It

We also reviewed two helper contracts. One parses and adds context to the error messages associated with failed transactions, and the other provides a unifying interface for ERC20 tokens and native ETH operations.

Privileged Roles

The `OneInchExchange.sol` contract has an owner address, controlled by the 1inch team, that can pause the exchange. It can also withdraw any funds that were mistakenly sent to the contract itself.

Here we present our findings.

Critical severity

None.

High severity

None.

Medium severity

[M01] Cannot unpause exchange

The `OneInchExchange.sol` contract exposes a mechanism for the owner to pause the contract. This disables the `swap` functionality. However, there is no corresponding mechanism to unpause the contract.

Consider introducing a mechanism for the owner to unpause the contract. Alternatively, if the current behavior is expected, consider renaming the `pause` function to `shutdown` or something similar that implies the contract will be permanently disabled.

Update: *Fixed in commit 0b89110a. The `pause` function has been renamed to `shutdown`.*

allowance granted to a specified address. However, if the new value is non-zero and the existing allowance is greater than the new value, the request is discarded. This means the token allowance cannot be decreased to a non-zero value, which differs from the typical ERC20 `approve` behavior. Consider introducing a `decreaseAllowance` function to handle this case, or documenting the new behavior and its rationale in the function comments.

This issue is related to **[M03] Incorrect safeApprove usage** and any mitigation should consider both simultaneously.

Update: *Fixed in commit bdbda2c7. The function replicates standard `approve` functionality, whether or not the token prevents changing allowances between non-zero values. The code intentionally disregards the possible frontrunning attack in the interest of universal applicability.*

[M03] Incorrect safeApprove usage

The `safeApprove` function of the OpenZeppelin `SafeERC20` library prevents changing an allowance between non-zero values to mitigate a possible front-running attack. Instead, the `safeIncreaseAllowance` and `safeDecreaseAllowance` functions should be used. However, the `UniERC20` library simply bypasses this restriction by first setting the allowance to zero. The reintroduces the front-running attack and undermines the value of the `safeApprove` function. Consider introducing an `increaseAllowance` function to handle this case.

This issue is related to **[M02] Cannot decrease allowance to non-zero value** and any mitigation should consider both simultaneously.

Update: *Fixed in commit bdbda2c7. The function replicates standard `approve` functionality, whether or not the token prevents changing allowances between non-zero values. The code intentionally disregards the possible frontrunning attack in the interest of universal applicability.*

Low severity

[L01] Transfer may fail

- the withdrawer smart contract does not implement a payable fallback or receive function
- the smart contract implements a payable fallback function that uses more than 2300 gas, potentially due to a proxy that increases the call's gas usage.

To prevent unexpected behavior, consider explicitly warning users about these shortcomings. Additionally, note that the `sendValue` function available in OpenZeppelin Contract's `Address` library can be used to transfer Ether without being limited to 2300 gas units. Risks of reentrancy stemming from the use of this function can be mitigated by tightly following the "Check-effects-interactions" pattern and using OpenZeppelin Contract's `ReentrancyGuard` contract. For further reference on why using Solidity's `transfer` is no longer recommended, refer to these articles:

- Stop using Solidity's transfer now
- Reentrancy after Istanbul

Notes & Additional Information

[N01] Missing docstrings and references

The contracts and functions in the reviewed code base lack documentation. This hinders reviewers' understanding of the code's intention, which is fundamental to correctly assess not only security, but also correctness. Additionally, docstrings improve readability and ease maintenance. They should explicitly explain the purpose or intention of the functions, the scenarios under which they can fail, the roles allowed to call them, the values returned and the events emitted.

Consider thoroughly documenting all functions (and their parameters) that are part of the contracts' public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Furthermore, the code comments should reference any relevant standards. In particular:

- The `RevertReasonParser` contract handles the case where the data was encoded as a call to a function with the signature `Panic(uint256)`, which matches the encoding of a

These should be explicitly noted.

[N02] Mixing roles

The `OneInchExchange` contract has a mechanism for the owner to withdraw funds held by the contract to their own account. However, the role of the owner and the funds recipient are conceptually distinct and implicitly combining them can introduce additional complexity if the owner role is transferred to a governance mechanism. Consider allowing the owner to specify a recipient address when withdrawing funds from the contract.

[N03] Safety check is manipulable

The `swap` function of the `OneInchExchange` contract allows users to perform a sequence of arbitrary calls and respond to potential failures in order to execute a complicated token swap across multiple decentralized exchanges. However, there are many ways that this function can be abused to subvert its intended effect. Some examples include:

- the caller can set the `_SHOULD_CLAIM` flag to false and perform swaps without transferring in the expected amount of tokens.
- the caller could use the arbitrary calls to transfer tokens in or out of the destination address, manipulating the calculation of the returned tokens.
- the caller could use the arbitrary calls to reenter the swap function so the balance change calculations will be performed on multiple swaps.

This should not pose a safety risk because the function is intended to provide consistency checks for the caller, so these manipulations only undermine that goal. All security considerations around the token swaps themselves should be handled by the external exchanges. However, this does imply that the emitted event may not accurately reflect the executed operation. If this is undesirable, consider validating the consistency of the parameters and the external calls.

Conclusions

Related Posts

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits

Threat Monitoring
Incident Response
Operation and Automation

Zero Knowledge Proof Practice

Blog

Company

About us
Jobs
Blog

Contracts Library

Docs

© Zeppelin Group Limited 2023