OpenZeppelin | security

1inch Aggregation Router V6 Audit

April 18, 2023

This security assessment was prepared by OpenZeppelin.

Table of Contents

Summary

Type	DeFi
Timeline	From 2023-03-06 To 2023-03-24
Languages	Solidity

Total Issues	6 (3 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	1 (0 resolved)
Notes & Additional Information	5 (3 resolved)

Scope

We audited the `1inch/1inch-contract` repository at the `d1bfd3dc3752edca2092bc6b522f4f692188fa5a` commit.

In scope were the following contracts:

```
contracts
├── AggregationRouterV6.sol
├── helpers
│   └── RouterErrors.sol
├── libs
│   └── ProtocolLib.sol
└── routers
    ├── ClipperRouter.sol
    ├── GenericRouter.sol
    └── UnoswapRouter.sol
```

System Overview

The `1inch-contract` repository is home to the `AggregationRouterV6`, which was the focus of this audit. This aggregation router is an extremely efficient collection of popular swapping protocols. Since this router is a single entry point to multiple swapping protocols, 1inch uses it to route efficient trades from their aggregator's front end.

The main supported swapping protocols are Uniswap V3, Uniswap V2, Clipper, and Curve. Additionally, the `GenericRouter` can be used to generically support other swap protocols. With it, it is possible to create `AggregationExecutors` to execute swaps on previously unsupported protocols.

1inch has gone to great lengths to create some of the most gas-efficient routers. As a result, the codebase features a lot of low-level assembly and many handwritten components, which other protocols would normally import.

The aggregation router does not hold user funds, but rather works on approvals. Because of this, they have implemented a way to rescue user funds that were accidentally sent to the protocol. However, this rescue function is not a perfect solution, as both the generic and Uniswap routers provide a way for normal users to withdraw accidentally deposited funds.

Security Model & Trust Assumptions

The aggregation router has an owner, but it is only used for rescuing funds and destroying the protocol in the case of an exploit. Other than that, there are no privileged roles. Under normal use, user funds should not be stuck in the router. The largest risk to users is the allowances granted to the router.

Low Severity

L-01 `selfdestruct` may be deprecated

At the time of writing, EIP-4758 is being deliberated and appears on track to be implemented. It will re-map the `SELFDESTRUCT` opcode to `SENDALL`, which will only send the ETH balance, leaving the contract intact.

Since the `selfdestruct` call in the `AggregationRouterV6` contract is intended to destroy the contract if an exploit is found, it is crucial that this functionality continues to work well into the future. The proposed `SENDALL` opcode (which does not destroy the contract) would be useless in this case.

Consider implementing some universal pause functionality instead, which would make all calls to any public API functions revert. Accompanying this functionality, there could be additional functions that are only callable when the system is paused, which would transfer ETH or tokens out of the contract. Alternatively, consider a schema in which a proxy contract is used for `AggregationRouterV6`. In the case of an exploit, the implementation for the contract could be set to some inaccessible address like the zero address, effectively "deleting" the contract code.

Update: *Acknowledged, not resolved. The 1inch team states:*

> *It'll be fine until at least the next hardfork. We'll mitigate this in the next release.*

Notes & Additional Information

N-01 UnoswapRouter definitions do not follow the Solidity Style Guide

The UnoswapRouter defines constants in the middle of the contract, on lines 198-206, and on lines 308-336.

For the sake of clarity and readability, consider following the de-facto standard of declaring all constants together at the top of the contract.

Update: *Acknowledged, not resolved. The 1inch team states:*

> *We'll keep those as those constants are only relevant to the methods directly after them. It actually increases readability in that case.*

N-02 Repeated Permit2 address check

Within the UnoswapRouter contract, a check exists to verify that there is a contract at the hard-coded `_PERMIT2` address.

This check will be done every time `uniswapV3SwapCallback` is called, but this is unnecessary. Since the Permit2 contract exists at the same address and cannot be self-destructed, once this check has returned `true` one time, it can be assumed to be `true` afterwards.

Consider implementing this check in the constructor rather than in the `uniswapV3SwapCallback` function. This will save gas every time uniswapV3 is used for swaps.

Update: *Acknowledged, not resolved. The 1inch team states:*

> *We'll keep this in case we will deploy the contract on networks before the Permit2.*

N-03 Duplication of library code

In the `ProtocolLib` contract, the `AddressLib` is used to wrap addresses with additional information, which is passed in the unused space within the `address` type. The `AddressLib` provides library code to query the information stored in the high bits. Instead of using the logic provided by the `AddressLib`, the `ProtocolLib` duplicates the `AddressLib`'s functionality.

For example, the `shouldUnwrapWeth`, `shouldWrapWeth`, and `usePermit2` functions duplicate the logic provided by the `getFlag` function.

Additionally, in the `ClipperRouter` contract, some code from the `SafeERC20` library is duplicated. In case the `srcToken` passed into `clipperSwapTo` is `wETH`, the

`ClipperRouter` transfers and converts `wETH` from a user before interacting with the clipper. `SafeERC20` supports both `transferFrom` and `wETH` `withdraw`.

Later on, when sending the funds back to the user, the router implements a method to convert `ETH` to `wETH` and transfer it. Again, `SafeERC20` has `deposit` and `transfer` functionality.

The use of library code reduces complexity and decreases the possibility that updates will only make it to some instances of duplicated, out-of-date code. Consider changing all instances of duplicated library code in favor of the functions provided by the libraries.

Update: *Resolved in pull request #236 at commit* `040e37d6a6fe409396838eecb5675ce35c6bb392`.

N-04 TODOs in code

The following TODO comment was found in the codebase. This type of comment should be tracked in the project's issue backlog and resolved before the system is deployed:

- The TODO comment on line 92 in UnoswapRouter.sol

During development, having well-described TODO/Fixme comments will make the process of tracking and solving them easier. Without this information, these comments may age and important information for the security of the system could be forgotten by the time it is released to production.

Consider removing all instances of TODO/Fixme comments and instead tracking them in the issues backlog. Alternatively, consider linking each inline TODO/Fixme to the corresponding issues backlog entry.

Update: *Resolved in pull request #236 at commit* `ba660dc87f806fed8f723eeab1073aa2c788a3b2`.

N-05 Unused imports

Throughout the codebase imports on the following lines are unused and could be removed:

- Import `SafeCast` of `UnoswapRouter.sol`
- Import `IUniswapV2Pair` of `UnoswapRouter.sol`

Consider removing unused imports to improve the overall clarity and readability of the codebase.

Update: *Resolved in pull request #236 at commit* `8c278b173b38520fb64e20acca6d3d0c49e905e4`.

Conclusion

One issue was reported to help future-proof the contract, along with a few issues regarding best practices. The codebase may benefit from utilizing battle-tested code and from including more extensive documentation. Ensuring the codebase follows a standardized style and is easily understandable will lead to safer code in the future.

Appendix

Monitoring Recommendations

To keep the protocol secure and operating as intended, we recommend monitoring the following:

- Monitor the mempool for direct calls to the `uniswapV3SwapCallback` function. This may indicate an attack or malicious actions. Monitor transactions for internal calls to `uniswapV3SwapCallback` as well.
- Monitor for accidental transfers of funds to the Aggregation Router, and create an automated system to recover those lost funds.
- Monitor for transaction reverts within the Aggregation Router. Monitor which tokens are being affected in these cases to detect malicious or non-compliant tokens.
- Have a plan in place for activating the `destroy` function. Ensure that the plan is unambiguous, so it is very clear which circumstances authorize the use of this function. Ensure that at any time, including weekends or off-work hours, the plan can be followed quickly. Consider also communicating this plan to users for feedback.