OpenZeppelin | security

1inch AggregationRouterV5 and LimitOrderProtocol Audit

September 22, 2022
This security assessment was prepared by OpenZeppelin.

Table of Contents

Summary

Type	DeFi
Timeline	From 2022-08-01 To 2022-09-02
Languages	Solidity

Total Issues	14 (5 resolved, 2 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	2 (0 resolved)
Low Severity Issues	7 (3 resolved, 1 partially resolved)
Notes & Additional Information	5 (2 resolved, 1 partially resolved)

Scope

We audited 1inch's `1inch-contract` repository at commit `28a12d4` as well as their `limit-order-protocol` at commit `2cdbb9f`.

In scope were the following contracts:

```
1inch-contract/contracts/
├── helpers
│   └── Errors.sol
├── interfaces
│   ├── IAggregationExecutor.sol
│   ├── IClipperExchangeInterface.sol
│   ├── IUniswapV3Pool.sol
│   └── IUniswapV3SwapCallback.sol
├── routers
│   ├── ClipperRouter.sol
│   ├── GenericRouter.sol
│   ├── UnoswapRouter.sol
│   └── UnoswapV3Router.sol
└── AggregationRouterV5.sol
```

```
limit-order-protocol/contracts/
├── helpers
│   ├── AmountCalculator.sol
│   ├── NonceManager.sol
│   ├── PredicateHelper.sol
├── interfaces
│   ├── IOrderMixin.sol
│   └── NotificationReceiver.sol
├── libraries
│   ├── ArgumentsDecoder.sol
│   └── Errors.sol
├── OrderLib.sol
├── OrderMixin.sol
├── OrderRFQLib.sol
└── OrderRFQMixin.sol
```

System Overview

`1inch-contract`

The `1inch-contract` repository houses the `AggregationRouterV5` contract. This contract allows users to complete highly efficient swaps across multiple decentralized exchanges. Supported routers include Uniswap v2, Uniswap v3, Clipper, and a generic router which can enable swaps on arbitrary exchanges. Commit `28a12d4` is the fifth version of the aggregator, the successor to the currently public v4.

One of the most notable changes between version 4 and version 5 is the generic router. The generic router uses individually deployed executors, each created specifically for an exchange. Although outside of the scope of this audit, the repository includes some executor extensions that work with the generic router. Other changes include updates to the `limit-order-protocol`, inheriting `OrderMixin`, and some additional gas savings via assembly across the routers.

Instead of using the public routers for the exchanges, 1inch has rewritten the routers mostly in assembly to save users gas with code optimizations. While their implementation is clean and efficient, it is hard to comprehend due to the lack of documentation. Assembly is naturally more difficult to understand than solidity and requires more documentation. In this repository we found very little documentation, particularly in areas written in assembly.

Documentation is important for both users and developers. Without it, code can become indecipherable, hindering user's understanding. Additionally, it increases the chances of misinterpreting the code which can lead to potentially dangerous upgrades. This issue is worsened when the code is complex, like the assembly used across this contract.

Otherwise, the code was clean and efficient. The 1inch team was very responsive and helpful throughout the `1inch-contract` audit.

`limit-order-protocol`

The `limit-order-protocol` repository contains the `LimitOrderProtocol` contract. As mentioned above, the `LimitOrderProtocol` is inherited by the `AggregationRouterV5`. This contract facilitates an orderbook that allows users to take and make swaps. All orders are saved as hashes onchain, so 1inch's frontend is indispensable for retrieving order contents.

Orders are created by signing the hash of an order. They never are put on-chain until they are either filled or canceled. Canceling involves invalidating a specific order hash. The off-chain-until-filled implementation favors the maker, as it costs nothing to create an order. This contract also supports permits, allowing makers to put the gas burden of both approving tokens and finalizing orders on the taker.

The audited code increases the version from 2 to 3. With the version update comes large changes to the `OrderMixin`, including a new `OrderLib` - a library which implements interaction hooks as well as order function customization.

Similar to the `AggregationRouterV5`, the `LimitOrderProtocol` is under-documented. There are many functions that lack natspec documentation. There also are instances of complicated code with insufficient supporting documentation.

Other than the lack of documentation, the quality of the code is very high. There were not any blatant errors, leading us to perceive this code to be mature. The 1inch team was also very responsive and helpful through the `LimitOrderProtocol` audit.

Trust Assumptions

The `AggregationRouterV5` is very flexible, especially with the addition of the generic router. Transaction data proposed to users is impossible to decode without a deep comprehension of the contract, so users must trust the 1inch frontend to serve accurate transactions.

Findings

Here we present our findings.

Update: *The 1inch team applied several fixes based on our recommendations and provided us with a set of commits that target each respective issue found. We have addressed each of these resolutions in this report.*

Medium Severity

M-01 Return value is not accurate

The Clipper exchange does not return the output value after a swap takes place. To remedy this, the `ClipperRouter` contract returns the `outputAmount` parameter which is the expected output amount passed into the swap.

It is possible for Clipper to return less than expected output amounts when the full proposed input balance is not provided. This can lead to the `ClipperRouter` returning a larger output amount than what is passed back to the user, which is confusing and potentially dangerous if relied upon.

This occurs both in Clipper's `swap` and `sellTokenForEth` functions. The `sellEthForToken` function will revert on partial swaps.

Similar to the Clipper exchange, consider either calculating the exact output amount or omitting the `outputAmount` return all together to avoid any potentially misleading returns.

Update: *Acknowledged, not resolved. 1inch's statement for this issue:*

> *Won't fix. As far as the Clipper contract logic goes, their fairOutput always equals to outputAmount in case when actualInput equals to inputAmount.*

M-02 Misleading or incomplete inline documentation

Throughout the codebase, there is a pattern of incomplete inline documentation for libraries, functions, events, errors, and constants that should be resolved. A few examples are:

- The `AggregationRouterV5` and the `NonceManager` contracts completely lack docstrings.
- The interfaces from `NotificationReceiver.sol`.
- The `interaction` parameter does not show up in the inline documentation.
- The unnamed return value has an explicit name in the inline documentation from the `hashOrder` function of the `IOrderMixin` interface.

- There is no documentation stating the scenarios in which the `owner` of the `AggregationRouterV5` would destroy the contract and what measures will be taken to protect protocols that rely on this contract.

Furthermore, across the `limit-order-protocol` repository, we identified some instances of misleading documentation:

- In the `OrderRFQMixin` contract, the documentation claims bits 0-252 is used for the amount to swap but actually bit 252 is used to signal if ether should be returned wrapped.
- In the `OrderRFQMixin` contract, the documentation claims that `permit` is the `abi.encoded` data of the token's address plus the permit's data, but it does not contain the address.

Clear inline documentation is fundamental for outlining the intentions of the code. Mismatches between the inline documentation and the implementation can lead to serious misconceptions about how the system is expected to behave and hinders reviewers and users from understanding the intention of the code, 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 fixing all instances of misleading documentation and 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).

Update: *Acknowledged, not resolved. 1inch's statement for this issue:*

> *We'll work on better docs in the future.*

Low Severity

L-01 Errors are not standardized

There are a few ways errors are declared across the `1inch-contract` and the `limit-order-protocol` repositories. Both repositories have a rather sparse `Errors` contracts (1) (2), and they differ between implementations drastically:

- The `1inch-contract` `Errors` contract lives in the `helpers` folder while the `limit-order-protocol` `Errors` contract lives in the `libraries` folder.
- The `1inch-contract` `Errors` contract defines errors in a library while the `limit-order-protocol` `Errors` contract lists them in the global scope.
- In both repositories, many other errors are defined in the contracts they are used in instead of in the `Errors` contract.
- The `OrderMixin` contract of the `limit-order-protocol` repository defines a single error far from where the rest of the errors are defined.
- The `ERC20TransferFailed` error of the `Errors` contract in the `1inch-contract` repository is not used, but rather hardcoded in both the `UnoswapRouter` and the `UnoswapV3Router` contracts.

Consider standardizing error definitions by either removing the `Errors` contracts or moving all error definitions into them. Standardizing errors would improve the readability of the code as well as make it easier to update. Consider standardizing errors across each repository as well.

Update: *Partially resolved in pull request 172 on commit* `d5f76dc30720b218d394715553f0d7d94324d0f9`*. The* `Errors.sol` *from the* `1inch-contract` *repository still lies in the* `helpers` *folder.*

L-02 Implicit casting

The following instances of implicit casting between variable types were identified in the codebase:

- In the `NonceManager` contract, the `amount` parameter (`uint8`) is added to the current nonce for the address (`uint256`).

- In the `UnoswapRouter` contract, the `callvalue` value is used as a boolean instead of converting it first, as it was done in another part of the code.

When a different type of variable is needed, consider either checking and casting the variable into the desired type or using OpenZeppelin's `SafeCast` library which performs overflow checks when casting from one type of number to another.

Update: *Acknowledged, not resolved. 1inch's statement for this issue:*

> *Won't fix.*

L-03 Mismatch between interface and implementation

The name of the `thresholdAmount` parameter from the `IOrderMixin` interface does not match the one in the implementation in the `OrderMixin` contract suggesting that the purpose of the parameter has changed. This not only creates confusion about the usage of such parameter, but could cause semantic overload.

Consider applying the necessary changes to the interface and contract to be consistent across the codebase.

Update: *Resolved in pull request 166 on commit* `bfd748635e105241e3218582f0bf7e36d18aad01`.

L-04 Lack of validation

While many of the functions in the codebase verify external inputs, there are some that lack sufficient input validation. For instance:

- In the `NonceManager` contract, the `advanceNonce` function can be called with a zero value, which will not change storage but will trigger the `NonceIncreased` event, polluting the off-chain indexing systems.

To avoid errors and unexpected system behavior, consider implementing the respective checks to prevent unexpected behaviors.

Update: *Resolved in pull request 168 on commit* `ae25bcbaa6685ddae1294a5db6bb7d7500293259`.

L-05 Magic numbers are used

Although constants are generally used correctly throughout the codebase, there are a few occurrences of literal values being used with unexplained meaning. For example:

- The `0x120` and `5` values are hardcoded in the `ClipperRouter` contract
- All the numbers in the `_unoswap` function from the `UnoswapRouter` contract
- The `100` value in the `_selfStaticCall` function from the `PredicateHelper` contract
- The `x` character in the `OrderLib` library

To improve the code's readability and facilitate refactoring, consider defining a constant for every magic number and give it a clear and self-explanatory name. For complex values, consider adding an inline comment explaining how they were calculated or why they were chosen.

Update: *Acknowledged, not resolved. 1inch's statement for this issue:*

> *Won't fix. Error selectors are not supported as compile time selectors in solidity so far so we'll keep those as inline constants until Solidity supports proper constants. Offsets, length and 'x' are pretty self-explanatory so we'll keep them as is.*

L-06 Library function redefinition

In the `solidity-utils` repository, the `RevertReasonForwarder` library provides the `reRevert` function. This function will revert with the revert message of a failed call, passing the error down the call chain. The `reRevert` functionality is used nearly everywhere a low level `call` exists, but instead of using the library function, it is hard coded.

We have identified multiple instances of this behavior:

- In the `UnoswapRouter` contract:
  - `reRevert` is redefined on line 94
- In the `UnoswapV3Router` contract:
  - `reRevert` is redefined on line 131
- In the `ClipperRouter` contract:
  - line 113
  - line 120
  - line 150
  - line 181

    - line 198
    - line 206
    - line 233
- In the `GenericRouter` contract:
    - line 115
- In the `UniswapV2Extension` contract:
    - line 60
    - line 105
- In the `UniswapV3Extension` contract:
    - line 74
- In the `CallDescription` library:
    - imports `RevertReasonForwarder` on line 7, but remains unused
    - line 202

Consider using the `reRevert` function provided by the `RevertReasonForwarder` library to favor standardized, updatable, and reusable library code.

Update: *Acknowledged, not resolved. 1inch's statement for this issue:*

> *Won't fix. All those hardcoded references are made inside the inline assembly. We don't know the way of calling internal solidity function from inline assembly.*

L-07 Reasonable swaps could fail

The `GenericRouter` contract uses external `executor` contracts to facilitate swaps. To improve the gas consumption during execution, the protocol saves one wei of the destination token the first time that token was exchanged and keeps it in the contract.

This means that on the first use, the returned balance to the user will be 1 wei less than expected. If the user passes the exact value they should receive, and the swap returns exactly that value, the swap will revert.

In order to prevent reverted swaps and reduce the confusion to users, consider either reducing one wei from the `minReturnAmount` value when validating the outcome on first time swaps or documenting that swapping with a new token may return slightly less than the `minReturnAmount` value.

Update: *Resolved in pull request 171 on commit* `28d32b9e04768a3d57e4a4cc05675831864f1297`*. More documentation has been added.*

Notes & Additional Information

N-01 Unnecessary function visibility

Inside the `OrderRFQMixin` contract, the `cancelOrderRFQ` function's visibility is set to `public`, however it is unused anywhere else in the contract. It is good practice to reduce function visibility when possible, reducing gas costs, code-size, and attack surface.

To follow best practices, consider reducing the visibility of the `cancelOrderRFQ` function from `public` to `external`.

Update: *Resolved in pull request 169 on commit* `ac5f958f1a62ab7fc66e7438a0cc244228f08763`*. More documentation has been added.*

N-02 Implicit limitation of predicates

The `PredicateHelper` contract allows the Limit Order protocol to incorporate diverse conditions when executing an order.

The contract implicitly limits the `and` and `or` methods to join a maximum of 8 predicates as the `offsets` parameter can only fit 8 32-bit shifts, yet this is undocumented.

Consider documenting this limitation to users along with how to construct the `offsets` and `data` parameters.

Update: *Acknowledged, not resolved. 1inch's statement for this issue:*

> *We'll add these docs in the future.*

N-03 Inconsistent coding style

Across the codebase, there are several places where the code style is inconsistent. Some examples are:

- In the `OrderRFQMixin` contract, a set of constants are defined after external functions.
- In the `OrderMixin` contract, an error is defined in a different location from the rest of the errors' definitions.
- In the `OrderMixin` contract, when calculating the `thresholdAmount` value from the parameter, the negated permit's mask is used instead of creating a particular mask for the threshold value.
- In the `UnoswapRouter` contract, the `WETH` address is hardcoded instead of setting it during contract creation.
- In the `OrderRFQMixin` contract, the signature's length is not being validated as it is being done in the `fillOrderRFQTo` function.
- In the `UnoswapV3Router` contract, `WETH` wrapping and unwrapping operations are implemented with simple function calls while every other instance is written in assembly.
- In the `ClipperRouter` contract, the functions have named return outputs although those explicitly return a value.

Consider reviewing the entire codebase to improve consistency and refactor inconsistent code where possible. It is recommended to use the Solidity Style Guide as a guideline.

Update: *Partially resolved in commits `cb41321803a6fa809587d88ed4acd68fe9b4dbbb`, `d44e8208c5d8ee06876b4889cc1a5114c5d59bec`, and `0204b0a601b84b8a1186448ecc1693c47870aa72` from pull request 171. WETH address is still hardcoded in the `UnoswapRouter` contract even though a cast before the assembly block is being done in the `ClipperRouter` contract for a similar functionality. 1inch's statement for this issue:*

> *we can not set WETH address at the constructor because immutable are not supported in inline assembly*

For the signature's length validation, 1inch's statement for the issue:

> *flag here is being used to select which isValidSignature method to call*

For the WETH wrapping process, 1inch's statement for the issue:

> *that's because the whole function is written in solidity*

For the named return outputs, 1inch's statement for the issue:

> *we’ll keep it as it is*

N-04 Typographical errors

The following typographical errors were identified:

- In line 37 from `IOrderMixin.sol`, "standart" should be "standard".
- In line 133 from `OrderRFQMixin.sol`, "Same" should be "same".
- In line 130 from `UnoswapRouter.sol`, "numeratoar" should be "numerator".

Consider correcting the above typographical errors.

Update: *Resolved in commit `2ea1a622af9266dbf2cdbf9582d93c38a1b3764c` from pull request 170 and commit `2c8688d349ecd953487ef0aacdd65add1d96932b` from pull request 173.*

N-05 Unnecessarily complex code

In the `UnoswapRouter` contract, there is an assembly implementation to properly order reserves based on the reversed flag. The code first sets the reserves in the non-reversed order, checks if they should be reversed, and reverses them with a temporary variable if they are supposed to be reversed.

This code would be much cleaner if the initial assignment of the reserves occurred with a `switch-case` statement on the reversed value. Consider adding a `switch-case` block to avoid unnecessary temporary variables and improve the code's readability.

Update: *Acknowledged, not resolved. Issue has been adapted to reflect the team's feedback. 1inch's statement for this issue:*

> *Won’t fix as there is no else block in assembly and switch-case construction actually increases the complexity.*

Conclusion

No critical or high severity issues were found. Recommendations have been given to ensure production readiness of the code, improve quality, and minimize errors.

Many of the optimizations resulted in assembly implementations of highly complex and critical parts of the code. This, combined with a lack of documentation, made the code challenging to read, understand, and conceptualize. Providing documentation, particularly for these areas, would greatly increase the quality of the code.

Appendix

Monitoring Recommendation

Because of the strong trust assumption towards the front-end, as the transaction data is indecipherable on its own, we recommend implementing monitoring for all deviated-from-the-norm actions. For instance, adding monitoring to look out for suspicious inputs or calls of the following functions:

- `NonceManager.advanceNonce`
- `GenericRouter.swap`
- `UnoswapRouter.unoswapToWithPermit`
- `ClipperRouter.clipperSwapToWithPermit`
- `UnoswapV3Router.uniswapV3SwapToWithPermit`
- `AggregationRouterV5.transferOwnership`
- `UnoswapV3Router.uniswapV3SwapCallback`
- `AggregationRouterV5.rescueFunds`
- `AggregationRouterV5.destroy`
- `EthReceiver.receive`

Suspicious activity could include an unusually high swap `amount`, re-used permits, highly unbalanced input and output values, repetitively swapping from the same `taker`, unusual predicate/interactions shape, unknown `executor` addresses, interactions with unusual assets, anomalous structure of transactions, or atypical pool attributes. Any suspicious activity could be an indication that a malicious actor may be compromising users.

Furthermore, all functions that are maintainable by the contract owner should be monitored in order to keep track of operator and configuration changes. This would enable the team to react quickly if there are incidents as well as be alerted in cases where developers are required to step in.