OpenZeppelin | security

1inch Limit Order Protocol 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	8 (6 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	3 (1 resolved)
Notes & Additional Information	5 (5 resolved)

Scope

We audited the `1inch/limit-order-protocol` repository at the `e8539926bd8bf91b293ce89b8311fb0875b90070` commit.

In scope were the following contracts:

```
contracts
├── LimitOrderProtocol.sol
├── OrderLib.sol
├── OrderMixin.sol
├── helpers
│   ├── AmountCalculator.sol
│   ├── OrderIdInvalidator.sol
│   ├── PredicateHelper.sol
│   └── SeriesEpochManager.sol
└── libraries
    ├── BitInvalidatorLib.sol
    ├── Errors.sol
    ├── ExtensionLib.sol
    ├── MakerTraitsLib.sol
    ├── OffsetsLib.sol
    ├── RemainingInvalidatorLib.sol
    └── TakerTraitsLib.sol
```

System Overview

This audit included the core capabilities of the `limit-order-protocol`, which has been updated to release version 4. Compared to previous releases, this release combines RFQ, standard, and pro limit orders into a single lightweight, extensible limit order type.

In order to support all order types, a dynamic data structure has been implemented to efficiently provide only the required data. The previous order styles are not named like they used to be, but the functionality of the previous styles remains in the new protocol.

The Limit Order Protocol provides a method for users to create and fill limit orders on-chain. Orders themselves are represented as hashes of their components. 1inch publicly hosts standard limit orders off-chain to allow users to search through and fill orders. Since limit orders are created with signatures, they are not put on chain until they are either filled or canceled. This is a highly efficient method and puts the gas burden of the order on the taker.

Limit orders contain many optional parameters for both makers and takers. All maker traits are decided during the construction of the limit order and cannot be changed without canceling and creating a new limit order. For taker traits, parameters are decided during the filling of the order and are not signed by the maker.

A unique and useful part of the Limit Order Protocol is its support for order interactions. Each order has the ability to run pre and post-interactions which are specified by the maker of an order. This allows the maker to hook limit orders before or after they are filled. Similarly, the taker can specify a taker interaction, which allows them to execute code after obtaining the maker's funds but before paying the maker for them.

The Limit Order Protocol also provides a lot of control over orders. One built-in feature is the `PredicateHelper`. The predicate helper allows makers to have more control over when an order is valid. There can also be a timeout associated with an order, specified in the maker traits.

Security Model & Trust Assumptions

The Limit Order Protocol does not have an owner and there are no privileged roles. Any user is allowed to open orders by signing a valid limit order. Conversely, anyone is able to fill any valid order. Depending on the parameters set by the maker, it is possible to restrict limit orders to specific accounts. Orders can only be canceled by the users who created them.

Users should be aware that the protocol does not limit which tokens are accepted for trading. Therefore, any contract exhibiting the ERC-20 interface can be interfaced with by the limit order protocol, potentially having unexpected results (for example, tokens that do not transfer the expected amount when calling `transferFrom`, or tokens that revert for certain accounts when attempting transfers). Users should only interact with orders that deal with tokens that they trust, and should be wary of any orders involving untrusted tokens.

Low Severity

L-01 Maker's pre-interaction disallows short addresses

Within the limit order protocol, before the maker's pre-interaction, there is some code that allows makers to use a different address from their own for the pre-interaction.

As the data for the call (specified after the address) will always be assumed to start at byte 20, it should not be possible to provide an address with a length lower than 19 bytes. However, addresses with a length greater than `0` but lower than `20` are currently supported.

While there may be ways to safely utilize this code with addresses specified in less than 20 bytes, doing so risks corrupting any data passed along with the address.

Since the code assumes that the data will be at least 20 bytes long, consider changing the `if` condition to `data.length > 19`. Additionally, consider warning users to pad all "short" addresses for maker pre-interactions to 20 bytes.

Update: *Resolved in pull request #241 at commit `82f1050ff3bd54ce6802b899fc2e5317177942af`.*

L-02 Missing docstrings

Throughout the codebase there are several parts that do not have docstrings. For example:

- Line 13 in `OrderLib.sol`
- Line 154 of `OrderMixin.sol`
- Line 5 of `BitInvalidatorLib.sol`

Note that this list is not exhaustive. In general, consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. Additionally, document all contracts above their definitions, explaining their purpose and referencing authors and related contracts. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

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

> *We'll work on this in the future.*

L-03 Hard-coded gas limits

`_RAW_CALL_GAS_LIMIT` is hard-coded to `5000`.

While this is currently safe, in the future, gas prices may change. Thus, consider making this value changeable, or creating a procedure to update this value by redeploying this contract. Having a procedure in place, with well-defined limits such as disallowing `SSTORE`s or external calls, will reduce any downtime caused by future changes to gas schedules.

Update: *Acknowledged. The 1inch team states:*

> *We'll redeploy the contract if there will be planned hardfork that affects this.*

Notes & Additional Information

N-01 Unused named return variables

Named return variables are a way to declare variables that are meant to be used within a function body for the purpose of being returned as the function's output. They are an alternative to explicit in-line `return` statements.

Within the `OrderMixin` contract there are multiple instances of unused named return variables:

- The `remaining` return variable in the `remainingInvalidatorForOrder` function.
- The `remainingRaw` return variable in the `rawRemainingInvalidatorForOrder` function.
- The `makingAmount`, `takingAmount`, and `orderHash` return variables in the `fillOrder` function.
- The `makingAmount`, `takingAmount`, and `orderHash` return variables in the `fillOrderExt` function.

- The `makingAmount`, `takingAmount`, and `orderHash` return variables in the `fillOrderTo` function.
- The `makingAmount`, `takingAmount`, and `orderHash` return variables in the `fillOrderToWithPermit` function.
- The `makingAmount`, `takingAmount`, and `orderHash` return variables in the `fillContractOrder` function.

Consider either using or removing any unused named return variables.

Update: *Resolved in pull request #241 at commit `9a8a08be61d996dbae0f992868630b1f5777a6e8`.*

N-02 Unused imports

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

- Import `AmountCalculator` of `OrderMixin.sol`
- Import `ECDSA` of `ExtensionLib.sol`

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

Update: *Resolved in pull request #241 at commit `38789628a650585a7824445d169b39c1f8197ca4`.*

N-03 TODO comments

The following instances of TODO comments were found in the codebase. These comments should be tracked in the project's issue backlog and resolved before the system is deployed:

- The `TODO` comment on line 449 in OrderMixin.sol
- The `TODO` comment on line 466 in OrderMixin.sol
- The `TODO` comment on line 82 in MakerContract.sol

During development, having well-described TODO 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 comments and instead tracking them in the issues backlog. Alternatively, consider linking each inline TODO to the corresponding issues backlog entry.

Update: *Resolved in pull request #241 at commit* `d8748c8d7e1e681ba74c2bbdae31d50dc6818740`.

N-04 Unnecessary redirection in `fillOrder`

Within the `OrderMixin` contract, the `fillOrder` function calls `fillOrderTo`, which then calls `fillOrderToExt`.

Instead, consider directly calling `fillOrderToExt`. This will match what happens in `fillOrderExt` and `fillOrderTo`. This will also make the code clearer for reviewers, and reduce the gas consumption associated with an additional function call.

Update: *Resolved in pull request #241 at commit* `cbf86ed97c5e58e716bf9a2b5b8009cab0e178cf`.

N-05 Type used before defined

In the `OrderMixin` contract, the definition of the `WrappedCalldata` type comes after its first use. Even though defining a type wherever it is most used can be useful, defining it after its first use leads to code that is difficult to read.

Consider moving the definition of the `WrappedCalldata` type and adding a comment where it is used in the `_wrap` and `_unwrap` functions.

Update: *Resolved in pull request #241 at commit* `3cf8e0f4474352b59e095740e7f7c4cee6c1890f` *by removing the* `WrappedCalldata` *type completely.*

Conclusion

Several suggestions were made to improve the quality and readability of the codebase. Additionally, a few suggestions were made to reduce the surface for edge-case errors. The code would benefit from more documentation, specifically within assembly blocks and external

docs. The documentation existing online will need to be updated to reflect this newer version of the codebase.

Appendix

Monitoring Recommendations

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

- Monitor for new orders created without the 1inch front-end. This may expose shortcomings of the front-end or exploits to the protocol which require custom interactions.
- Monitor for users that frequently cancel limit orders to expose griefing attacks.
- Monitor for assets that are frequently involved in canceled orders to expose malicious tokens.
- Monitor for limit orders or swaps that are executed at significantly different prices than the market rate. Consider corroborating this data with multiple on-chain and off-chain data sources for low-liquidity assets.
- Monitor for token approvals to the limit order protocol to identify approval-fishing attacks.
- Monitor for the creation of orders that allow multiple fills, but disallow partial fills. These conditions may indicate a problem with order configuration.