OpenZeppelin | security

1inch Settlement Refactor Audit

April 16, 2024

Table of Contents

Summary

Type	DeFi
Timeline	From 2024-03-22 To 2024-03-28
Languages	Solidity

Total Issues	3 (3 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	0 (0 resolved)
Notes & Additional Information	3 (3 resolved)
Client Reported Issues	0 (0 resolved)

Scope

We audited the `1inch/limit-order-protocol` repository at the `2f87361` commit and the `1inch/limit-order-settlement` repository at the `1405320` commit.

In scope were the following files:

```
limit-order-protocol@2f87361
└── contracts
    └── FeeTaker.sol

limit-order-settlement@1405320
└── contracts
    ├── Settlement.sol
    ├── SimpleSettlement.sol
    └── extensions
        ├── BaseExtension.sol
        ├── ExtensionLib.sol
        ├── IntegratorFeeExtension.sol
        ├── ResolverFeeExtension.sol
        └── WhitelistExtension.sol
```

System Overview

This audit largely covers a refactor of code that had previously been audited by OpenZeppelin. What was `SettlementExtension.sol` at commit `ff7909c` has now been split into several different files. This refactoring allows modularity for developers to build their own extension contracts using only some of the features from what was a monolithic contract before. The different feature sets have been broken up as follows:

Contract	Features
`BaseExtension`	creates empty, overridable pre and post interaction methods along with the `IAmountGetter` methods for fusion orders
`IntegratorFeeExtension`	allows an integrator to charge a percentage of the taking amount from the resolver
`ResolverFeeExtension`	charges the resolver fees from the resolver
`WhitelistExtension`	ensures that only whitelisted resolvers can settle fusion orders
`SimpleSettlement`	wraps all of the above into a single contract
`Settlement`	adds a check for a valid priority fee before any of the other extension logic
`ExtensionLib`	library for parsing data from the bitmask that is the last byte in the `extraData` parameter

These contracts are meant to work with 1inch's Fusion system, where users' swaps are a dutch auction with an exchange rate that increasingly becomes better for the taker over time. A new feature added in this scope is that gas price increases can be factored in the dutch auction prices.

A new `feeTaker` contract has also been added to the core protocol that allows a third party intermediary to take fees in the taking tokens and transfers the rest to the maker.

Security Model and Trust Assumptions

The system relies heavily on the accuracy of off-chain calculation and signing. In order to handle such extensible workflows and yet be gas efficient, untyped bytes are often parsed and passed between contracts and functions. This requires not only the dapp to be incredibly careful when constructing order calls but also requires makers to be diligent with what they are signing. There are a lot of opportunities for functions to behave very different than intended. For example, passing the wrong auction rates and times to the extension could create an auction where things get *more* expensive over time instead of less expensive as intended. Because makers have to sign the data that they want to be executed on-chain, we assume that these users will do so accurately and with knowledge of how their orders will be executed. As for privileged roles within the scoped system, there is only one: the limit order protocol is the only allowed caller of the pre- and post-interaction functions of the Fusion system.

Notes & Additional Information

N-01 Missing Docstrings

Throughout the codebase, docstrings are missing at some places:

- The `FeeTaker` contract declaration
- The `_getAuctionBump` function declaration within `BaseExtension.sol`

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. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Resolved in pull request #311 (commit 76d0640) and pull request #153 (commit 2eef6f8).*

N-02 Documentation suggestions

In `WhitelistExtension`'s `_isWhitelisted` function resolvers are valid only after specific points in time and not before. This is clearly and competently written in the workings of the function. But that this is intended design is not described in the description of the function and led to some confusion on our part. Consider elucidating this detail in the natspec comments of the function so that further readers know how this function is intended to behave.

Update: *Resolved in pull request #153 at commit b599e1d.*

N-03 Code layout not following Solidity Style Guide

To make locating contract elements easier by using a common layout, the Solidity Style Guide recommends adhering to the following order of elements within each contract:

- Type declarations
- State variables
- Events
- Errors
- Modifiers
- Functions

In the `BaseExtension` contract, private functions are declared before internal functions. This deviation from the commonly used layout may impact readability.

Consider moving the internal functions before the private functions.

Update: *Resolved in pull request #153 at commit 51ed3ba.*

Conclusion

This audit largely covered the refactoring of a contract in the limit-order-settlement repository into multiple contracts. This change added modularity and flexibility to the codebase. Additionally, a new extension contract was added to the limit-order-protocol.

No major issues were found during the audit, although some best practice recommendations were made in order to make the codebase even more robust.

The 1inch team was very responsive throughout the engagement, providing us with the necessary insight to expedite our understanding of the implemented changes.