OpenZeppelin | security

1inch Protocol Fee Audit

May 16, 2025

Table of Contents

Summary

Type	DeFi
Timeline	From 2025-04-21 To 2025-04-25
Languages	Solidity

Total Issues	8 (7 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	2 (2 resolved)
Notes & Additional Information	6 (5 resolved)

Scope

OpenZeppelin audited the 1inch/fusion-protocol repository at commit c0197a5.

In scope were the following files:

```
contracts
└── SimpleSettlement.sol
```

In addition, the 1inch/limit-order-protocol repository was audited at commit cf8e50e.

In scope were the following files:

```
contracts
└── extensions
    ├── AmountGetterBase.sol
    ├── AmountGetterWithFee.sol
    └── FeeTaker.sol
```

System Overview

This project introduces a settlement contract (`SimpleSettlement.sol`) for decentralized trading, enhancing functionality with a configurable fee-charging mechanism, a Dutch auction-style rate adjustment, and a time-based taker whitelist. The contract integrates with `AmountGetterBase.sol`, `AmountGetterWithFee.sol`, and `FeeTaker.sol` to calculate fees, adjust prices, and verify whitelists.

The settlement contract is called by the resolver (taker) during trade execution in the Limit Order Protocol. It is used to calculate fees, handle Dutch auction price adjustments, and verify time-based taker whitelists, ensuring secure and efficient settlement.

There are two types of fees, configured when an order is created:

- **Integrator Fees**: Optional fees set by third-party applications to compensate for their services, such as providing user interfaces or liquidity.
- **Protocol Fees**: Fees to support the extension's maintenance, development, or governance, directed to a protocol-controlled address.

Both fees are calculated by `AmountGetterWithFee.sol` and transferred by `FeeTaker.sol` during settlement. The taker bears both fees, as the fees either reduce the net tokens received or increase the tokens paid.

`SimpleSettlement.sol` implements a Dutch auction mechanism, where the swap price decreases over time to encourage rapid execution. The price curve is defined by rate bumps—predefined price adjustments (e.g., 100 wei) set at specific timestamps (e.g., 1000 seconds). These are encoded in a `pointsAndTimeDeltas` array in `SimpleSettlement.sol`. During settlement, the auction bump—the effective price adjustment—is calculated by linearly interpolating between two rate bumps based on the current blockchain timestamp.

For example, if one rate bump point is 100 wei at 1000 seconds and the next is 200 wei at 2000 seconds, then at 1500 seconds, the auction bump is 150 wei. Optionally, a gas bump based on estimated gas prices (`block.baseFee`) reduces the effective price paid by the taker. Implemented in `SimpleSettlement.sol`'s `_getAuctionBump`, the gas bump is subtracted from the auction bump, increasing with gas price variability to offset taker losses and encourage prompt order filling, as voted in 1IP-43.

An optional whitelist restricts which addresses can act as takers, with time-based checkpoints controlling access. Each whitelisted address is associated with a timestamp (`allowedTime`), and the number of eligible takers increases as checkpoints are reached. For example, a new address becomes eligible once `block.timestamp` exceeds its `allowedTime`. This is implemented in the `SimpleSettlement.sol` contract's `_isWhitelistedPostInteractionImpl` function.

Security Model and Trusted Assumptions

The security model ensures secure, gas-efficient trading through audited contracts (`SimpleSettlement.sol`, `AmountGetterBase.sol`, `AmountGetterWithFee.sol`, `FeeTaker.sol`) and input validation. Key aspects include:

- Orders are created off-chain, signed per `EIP-712`, and validated on-chain to prevent tampering.
- Only whitelisted takers (resolvers) can execute trades reducing the risk of malicious settlements. Resolvers are incentivized to act honestly through settlement fees and protocol reputation.

Trust Assumptions

The extension's security depends on the following assumptions:

- Takers (resolvers) are whitelisted off-chain and act honestly, executing trades in `SimpleSettlement.sol` without manipulating `whitelistData` (taker eligibility) or `pointsAndTimeDeltas` (auction price curve). Malicious takers could disrupt settlements but are deterred by reputation and fee incentives.
- Proper `pointsAndTimeDeltas` Configuration in `SimpleSettlement._getAuctionBump`:
  - The number of points is less than 255 (`uint8` max).
  - The first byte of `pointsAndTimeDeltas` specifies the number of points.
  - Each point occupies exactly 5 bytes, following the first byte.
  - Points have monotonically decreasing rate bumps, ensuring a Dutch auction-style price reduction (e.g., 200 wei to 100 wei over time).

- Correct `extraData` Structure in `FeeTaker._postInteraction`:
  - If the `_CUSTOM_RECEIVER_FLAG` is set, `extraData` includes a 20-byte `feeCustomReceiver` address that is not otherwise included.

Low Severity

L-01 Missing Docstrings

Throughout the codebase, multiple instances of missing docstrings were identified:

- In `AmountGetterBase.sol`, the `getMakingAmount` function
- In `AmountGetterBase.sol`, the `getTakingAmount` function
- In `IAmountGetter.sol`, the `IAmountGetter` interface
- In `IOrderMixin.sol`, the `IOrderMixin` interface

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 #364.*

L-02 Incomplete Docstrings

Throughout the codebase, multiple instances of incomplete docstrings were identified:

- In `IAmountGetter.sol`:
  - In the `getMakingAmount` and `getTakingAmount` functions, not all return values are documented.
- In `IOrderMixin.sol`:
  - In the `remainingInvalidatorForOrder` and `rawRemainingInvalidatorForOrder` functions, the `maker` parameter is not documented.
- In `SimpleSettlement.sol`:
  - In the `_getAuctionBump` and `_getRateBump` functions, the `tail` return variable is not documented.
  - In the `_getFeeAmounts` function, none of the return variables are documented.

Consider thoroughly documenting all functions/events (and their parameters or return values) that are part of a contract's public API. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Resolved in pull request #365 of "limit-order-protocol" and pull request #192 of "fusion-protocol".*

Notes & Additional Information

N-01 Incorrect Order of Modifiers

Function modifiers should be ordered as follows: `visibility`, `mutability`, `virtual`, `override`, and `custom modifiers`.

The `_getFeeAmounts` function in `SimpleSettlement.sol` has an incorrect order of modifiers.

To improve the project's overall legibility, consider reordering the function modifiers as recommended by the Solidity Style Guide.

Update: *Resolved in pull request #193.*

N-02 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is quite beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for their maintainers to contact the appropriate person about the problem and provide mitigation instructions.

Throughout the codebase, multiple instances of contracts that do not have a security contact were identified:

- The `AmountGetterBase` contract
- The `AmountGetterWithFee` contract
- The `FeeTaker` contract
- The `SimpleSettlement` contract

Consider adding a NatSpec comment containing a security contact above each contract definition. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

Update: *Acknowledged, not resolved. The team stated:*

> *Noted*

N-03 Misleading Documentation

The word "making" in the docstring of the `getTakingAmount` function in `IAmountGetter` interface should be changed to "taking".

Consider implementing the above suggestion to improve the clarity and maintainability of the codebase.

Update: *Resolved in pull request #366.*

N-04 Typographical Errors

In line 93 of `FeeTaker.sol`, "accidently" should be corrected to "accidentally".

Consider correcting the above-mentioned typographical error to improve the readability of the codebase.

Update: *Resolved in pull request #367.*

N-05 Inconsistent Order Within a Contract

Throughout the codebase, multiple instances of contracts deviating from the Solidity Style Guide due to having an inconsistent ordering of functions were identified:

- The `FeeTaker` contract in `FeeTaker.sol`
- The `IOrderMixin` contract in `IOrderMixin.sol`
- The `SimpleSettlement` contract in `SimpleSettlement.sol`

To improve the project's overall legibility, consider standardizing ordering throughout the codebase as recommended by the Solidity Style Guide (Order of Functions).

Update: *Resolved in pull request #368 of "limit-order-protocol" and pull request #194 of "fusion-protocol".*

N-06 Unnecessary Casts

In lines 81 and 84 of `AmountGetterWithFee.sol`, the `bytes1` casting can be simplified by directly specifying a single index instead of using the slicing operator. Similarly, in line 212 of `SimpleSettlement.sol`, the `bytes1` casting can also be simplified by accessing the desired index directly.

Consider simplifying the aforementioned casting operations to improve the clarity and maintainability of the codebase.

Update: *Resolved in pull request #369 of "limit-order-protocol" and pull request #195 of "fusion-protocol".*

Conclusion

The audited codebase introduces an extension to the 1inch Fusion and Limit Order Protocols, enhancing their functionality with a configurable fee-charging mechanism, Dutch auction-style rate adjustment, and a time-based taker whitelist.

No security issues were found during the audit. However, multiple recommendations were nonetheless made to enhance code readability and clarity in order to facilitate future audits, integrations, and development.

The 1inch team is appreciated for being responsive throughout the audit period and providing comprehensive documentation about the audited codebase.