OpenZeppelin | security

Limit Order Settlement Audit

December 15, 2023

Table of Contents

Summary

Type	DeFi
Timeline	From 2023-11-24 To 2023-11-29
Languages	Solidity

Total Issues	14 (11 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	2 (2 resolved)
Low Severity Issues	3 (2 resolved)
Notes & Additional Information	9 (7 resolved)

Scope

We audited the 1inch/limit-order-settlement repository at commit ff7909c.

In scope were the following files:

```
contracts
├── SettlementExtension.sol
└── WhitelistRegistry.sol
```

System Overview

The Limit Order Settlement repository introduces a set of smart contracts which are part of 1inch Fusion. Built on top of the Limit Order Protocol, 1inch Fusion creates a separate market where anyone can make orders but only a set of whitelisted addresses called resolvers can take them. This set of resolvers is maintained by the `WhitelistRegistry` contract. In order to become a resolver, an address needs to own a certain amount of tokens set by the `WhitelistRegistry` contract. Resolvers can choose to delegate their right to take orders to other addresses, with a separate delegate on each supported chain. The goal of the system is to enable better prices and fees for users and resolvers.

The extension of the Limit Order Protocol introduced by the codebase implements functionalities which include customizable Dutch auctions, resolver fees, integrator fees, and the base fee market regulation defined in 1IP-43.

Makers would usually interact with the protocol via a front-end (e.g., web or mobile app) by specifying the making and taking token addresses, amounts, and the Dutch auction parameters describing how the price changes over time. The front-end would then add an optional resolver and integrator fee, and present the order for signing. Once the order is signed, it is made available to resolvers via the Limit Orders API or by some other means.

Privileged Roles

There is only one privileged role in the system, which is the owner of the `WhitelistRegistry` contract. The owner can set the amount of tokens that resolvers need to own, as well as sweep tokens mistakenly sent to the contract. It is worth noting, however, that the contract is not supposed to hold any funds.

Security Model and Trust Assumptions

When an order is passed to the settlement extension, the last 10 bytes of the addresses of whitelisted resolvers are added as extra data for verification against the taker address. While computing a 10-byte collision would take significant computational power, the fact that resolver addresses tend to be long-lived and that the potential computing power available to attackers keeps increasing may one day allow a determined attacker to generate a collision with a resolver address and bypass the whitelist. A similar concern has also been expressed by Vitalik Buterin in the context of address collisions. However, the resources needed to conduct such an attack on 1inch should currently outweigh the potential profits, especially taking into account that a mitigation is as simple as changing the resolver's address.

An implicit assumption of the Fusion mode is that the resolver market is competitive enough to provide users with good prices. This assumption may break, for example, if the owner sets the threshold required to become a resolver too high to allow for competition to emerge. In addition, it is expected that, in practice, most of the orders will be fetched off-chain by resolvers through the Limit Orders API. Users using the Fusion mode to get the best prices are thus implicitly trusting that this API will maintain a high availability and be equally accessible by all resolvers.

Medium Severity

M-01 No Masking of Addresses in `SafeERC20` Library

The `SafeERC20` library is used to implement several ERC-20 functions (`transfer`, `transferFrom`, `approve` etc.) in a way that is both gas-efficient and maximally compatible with different ERC-20 token implementations. The library uses assembly for efficiency.

However, in multiple functions, the `Address` arguments are directly copied to memory without being masked in assembly blocks. Note that because the library functions are called internally in the codebase, the function calls are replaced by JUMP instructions but no bit cleaning operation is inserted by the compiler. Depending on the context in which the library is used, this could result in unexpected reverts.

For example, an address with dirty upper bits could be computed using assembly, after which the assembly block could be closed and `safeTransferFrom` called (assuming that the bits would get cleaned since it is a function call in Solidity). However, the call to `transferFrom` would revert with an "EvmError: Revert" message. A similar issue was reported in the Solmate library by several users. Libraries can be used in a wide variety of contexts and should aim to work well in all of them.

Consider masking the `Address` arguments in the `SafeERC20` library preemptively to avoid such issues.

Update: *Resolved at commit bcb0ace. Warnings that make the behavior explicit were added. This resolves the issue because the 1inch contracts are not affected by this behavior while potential users of the library are now explicitly aware of it. The 1inch team stated:*

> *For now we have decided to only comment the behavior and add a warning everywhere it is relevant. We might rethink our approach in the future.*

M-02 Fee Regulation Might Not Work as Expected on Non-Mainnet Chains

The `_isPriorityFeeValid` function implements the fee regulation check proposed in 1IP-43. This check constrains the priority fee when interacting with the protocol, which should reduce the gas competition between resolvers and improve prices for users. Given the multi-chain nature of the protocol, however, the check might not work as expected on chains other than the Ethereum mainnet.

For example, Aurora, which is currently supported by 1inch, sets the base fee to zero for all blocks which would lead to reverts in the `_isPriorityFeeValid` function. Some other chains such as Scroll, which are not currently supported but might be in the future, revert when the base fee is accessed. Other chains like Optimism might have significantly different base fee dynamics which makes values hardcoded in the protocol less robust.

Before deploying the fee regulation on a new chain, consider validating that the check done by the `_isPriorityFeeValid` function works as intended. Alternatively, consider introducing the fee regulation only on the Ethereum mainnet.

Update: *Resolved. The 1inch team stated:*

> *The `priorityFee` limitation applies only to the mainnet, so the extension will be deployed on other chains without that check.*

Low Severity

L-01 Low Granularity of Voting Threshold

The `WhitelistRegistry` contract maintains a list of addresses which can act as resolvers. These are required to maintain a certain percentage ownership of the `token` supply, defined by the `resolverPercentageThreshold`. As explained in the associated comment, a `resolverPercentageThreshold` of `10_000` translates to 100% supply ownership, while `1000` corresponds to 10% and `100` to 1%. The smallest possible non-zero value for the threshold being `1` means that resolvers will always need to maintain at least a 0.01% ownership of the supply of `token`. This also means that 0.01% is the smallest amount by which the required percentage ownership for resolvers can be incremented/decremented.

However, assuming that `token` is the 1inch token with a total supply of 1,500,000,000, a 0.01% ownership at current prices would cost roughly $51,000. While the resolver role is likely to be restricted to a small set, $51,000 as the minimum increment seems high and the protocol may prefer a higher granularity when adjusting `resolverPercentageThreshold`.

Consider increasing the number of decimals of the `BASIS_POINTS` variable to increase the granularity with which the threshold can be changed and be more future-proof.

Update: *Acknowledged, will resolve. The 1inch team stated:*

> *Good find, we will increase granularity with the next whitelist release. However, the token that the whitelist is tracking is actually Delegated-Staked-1INCH and its total supply is unlikely to reach 1INCH total supply.*

L-02 Lack of Input Parameter Validation

The `resolverPercentageThreshold` variable stores the percentage of the total supply that an address has to stake in order to become a resolver. This percentage should never be higher than 100% but this is not enforced by the protocol. In extreme cases, this might prevent the whitelist from functioning. For example, if `resolverPercentageThreshold` is set to `type(uint256).max`, it is not possible to clean the whitelist anymore as the check overflows.

Consider adding a check that the percentage is less than 100% when setting it.

Update: *Resolved in pull request #138 at commits 3196ff8 and 921a6b3. The* `resolverPercentageThreshold` *variable can no longer be set to a value larger than 100%.*

L-03 Discrepancy Between Proposal and Implementation

The accepted 1IP-43 proposal states that "for blocks with baseFee >104.1 gwei - priorityFee is capped at 65% of the block's baseFee". However, the implementation applies the cap if the base fee is greater than or equal to 104.1 instead of strictly being greater.

Consider adjusting either the proposal or the implementation to eliminate this discrepancy.

Update: *Resolved in pull request #138 at commit 5e4652a.*

Notes & Additional Information

N-01 Unused Interface

The `WhitelistRegistry` contract maintains a list of whitelisted addresses which can act as resolvers. These are required to maintain a certain percentage ownership of the `token` supply. In the code, `token` is defined as an `IVotable`, where `IVotable` is an interface extending the ERC-20 interface with a `votingPowerOf` function. However, this function is not used, making the use of this interface confusing.

If there is no need to use `IVotable`, consider replacing it with the regular ERC-20 interface to make the code clearer.

Update: *Resolved in pull request #138 at commit 7f697df. The 1inch team decided to replace the unused `IVotable` interface with the more natural `IERC20` interface.*

N-02 Forged Event Emission

According to the documentation of the `promote` function, the whitelist registry allows resolvers to register a worker to fill orders on their behalf. At the end of the process, the `Promotion` event is emitted. However, one does not need to be a whitelisted resolver in order to register a worker. This allows anyone to emit `Promotion` events, which might confuse off-chain systems.

Consider only allowing resolvers to promote a worker.

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

> *We want to allow resolvers to properly set up their workers even if they do not yet have enough balance to be whitelisted.*

N-03 Inconsistent Ordering of Functions

The `SettlementExtension` contract deviates from the Solidity style guide and has an inconsistent ordering compared to the rest of the codebase.

To improve the project's overall legibility, consider reordering the functions of this contract.

Update: *Resolved in pull request #138 at commit e7d9771.*

N-04 Unused Error

The `NotWhitelisted` error defined in `WhitelistRegistry.sol` is unused.

To improve the overall clarity and readability of the codebase, consider either using or removing any currently unused errors.

Update: *Resolved in pull request #138 at commit 95d6279.*

N-05 Typographical Errors

We identified the following typographical errors in the codebase:

- "intergrationFee" should be "integrationFee"
- "stores introduces" should be "introduces"

Consider correcting the identified errors to improve the readability of the codebase.

Update: *Resolved in pull request #138 at commit c8d6302.*

N-06 Missing Named Parameters in Mapping

Since Solidity 0.8.18, developers can utilize named parameters in mappings. This means mappings can take the form of `mapping(KeyType KeyName? => ValueType ValueName?)`. This updated syntax can provide a more transparent representation of the mapping's purpose.

The `promotions` mapping state variable in the `WhitelistRegistry` contract could be better documented by using named parameters.

Consider adding named parameters to the mappings to improve the readability and maintainability of the codebase.

Update: *Resolved in pull request #138 at commit df5874e.*

N-07 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 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. Additionally, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for the maintainers of those libraries to make contact with the appropriate person about the problem and provide mitigation instructions.

However, the `SettlementExtension` and `WhitelistRegistry` contracts do not provide a security contact.

Consider adding a NatSpec comment containing a security contact above the 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, will resolve. The 1inch team stated that they will follow this practice in the future.*

N-08 Missing Docstrings

We identified several instances where additional documentation would make the code easier to understand:

- The `_clean` function in the `WhitelistRegistry` contract
- Most of the functions in the `SettlementExtension` contract (notably `_getRateBump` and `_parseFeeData`)

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 #138 at commit 517b115.*

N-09 Non-Explicit Imports

The use of non-explicit imports in the codebase can decrease code clarity and may create naming conflicts between locally defined and imported variables. However, throughout the codebase, global imports are being used exclusively.

Following the principle that clearer code is better code, consider using named import syntax (`import {A, B, C} from "X"`) to explicitly declare which contracts are being imported.

Update: *Resolved in pull request #138 at commit 4529ed8.*

Conclusion

The Limit Order Settlement codebase is part of 1inch Fusion and enables a market of resolvers to compete to become takers of users' limit orders, in order to provide them with the best prices. Resolvers are tracked in the `WhitelistRegistry` contract, whose role is to validate token ownership and whitelist addresses with enough tokens to become a resolver.

We found the codebase to be well-written but, as pointed out above, it would benefit from further documentation. We also recommend the 1inch team to validate their assumptions regarding 1IP-43 before deploying the system on chains other than Ethereum mainnet.