OpenZeppelin | security

Limit Order Settlement Audit

March 01, 2023

This security assessment was prepared by OpenZeppelin.

Table of Contents

Summary

Type	DeFi
Timeline	From 2023-02-06 To 2023-02-13
Languages	Solidity

Total Issues	4 (2 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	1 (1 resolved)
Notes & Additional Information	3 (1 resolved)

Scope

We audited a diff of the 1inch/limit-order-settlement repository with `0e8189419c5fbbc67416a1d9e7721cb06b51a38b` being the base commit and `d738a07f4b131344ce0320e8e9d1679b72483379` being the head.

In scope were the following contracts:

```
contracts
├── FeeBank.sol
├── FeeBankCharger.sol
├── PowerPod.sol
├── Settlement.sol
├── St1inch.sol
├── StakingFarmingPod.sol
├── WhitelistRegistry.sol
├── helpers
│   ├── ResolverMetadata.sol
│   ├── St1inchPreview.sol
│   ├── VotingPowerCalculator.sol
│   └── WhitelistHelper.sol
├── interfaces
│   ├── IFeeBank.sol
│   ├── IFeeBankCharger.sol
│   ├── IResolver.sol
│   ├── ISettlement.sol
│   ├── ISt1inch.sol
│   └── IVotable.sol
└── libraries
    ├── DynamicSuffix.sol
    ├── OrderSaltParser.sol
    ├── OrderSuffix.sol
    ├── TakingFee.sol
    └── TokensAndAmounts.sol
```

Overview

The `limit-order-settlement` protocol is designed to provide a marketplace for settling limit orders from 1inch's `limit-order-protocol`. Only a select group of resolvers will be allowed to settle limit orders intended for the `Settlement` contract. These resolvers are selected by being promoted on the `WhitelistRegistry`.

Limit orders targeted for the `Settlement` contract cannot be filled without using the `Settlement` contract itself. The `Settlement` contract adds a few extra features to normal limit orders by encoding data in the salt. The extra data consists of a fee that is collected by 1inch and information to run a Dutch auction. There is also a taker fee encoded in the signed limit order which is paid throughout the settlement process.

Each limit order will run a Dutch auction on the `takingAmount`, decreasing the amount required to fill an order over time. This allows for competition between resolvers, as resolvers who wait for larger profits risk losing out completely if competing resolvers fill orders at lower margins.

An important part of this protocol is the `WhitelistRegistry`. The `WhitelistRegistry` is a leaderboard which tracks the addresses with the highest delegated staked 1inch tokens. Only the top addresses are allowed to promote users to settle limit orders on the `Settlement` contract. The whitelist is used to promote addresses across multiple chains. 1inch enforces the whitelist by requiring users to encode the current promoted users into the order, which they check during the settlement process. However, since this is user-defined data, the real restriction is 1inch omitting orders with incorrect data from their API.

Staked 1inch is tracked via the `St1inch` contract and delegated balances are tracked via an instance of a `RewardableDelegationPodWithVotingPower` pod, which enables users to delegate their voting power to different addresses, counting towards the delegated address's `WhitelistRegistry` credibility.

Since `St1inch` inherits from the `ERC20Pods` contract, it is possible for other pods to be attached to this staked 1inch contract. If the `RewardableDelegationPodWithVotingPower` contract is used, it is also possible to distribute rewards to users that delegate to specific addresses.

The `ResolverMetadata` contract allows registered delegatees to associate a URL with their delegate share.

Summary of Updates

The `Settlement` contract passes traded tokens and amounts as a part of the suffix to ease parsing on the resolver's side. The contract also uses the 0x4 precompile, also known as the Identity precompile, to copy parts of the calldata.

The `FeeBank` contract uses unchecked math in some statements relying on other parts of the system to check overflow and underflow issues.

The `St1inch` contract now supports the default farm logic that uses the `StakingFarmingPod` contract.

There are other smaller changes across the contracts.

Privileged Roles

The `owner` role in the `St1inch` contract is used to set the `emergencyExit` boolean, rescue tokens mistakenly sent to the contract, and configure parameters such as maximum loss ratio and the fee receiver address. When the `emergencyExit` flag is set, users can withdraw their staked 1inch tokens before their lock period expires.

The `owner` role in the `WhitelistRegistry` contract is used to transfer funds out of the contract and set various limits on it.

The `owner` role in the `FeeBank` contract is solely used for collecting fees that are owed to the contract.

Although these are not permanent roles, the `WhitelistRegistry` lists addresses that are promoted by whitelisted addresses. These whitelisted addresses are the ones able to settle limit orders on the `Settlement` contract.

Findings

Here we present our findings.

Low Severity

L-01 Condition always evaluates to true

The `not(valid)` assembly expression always evaluates to true because the `not` operator works as bitwise negation of x (every bit of x is negated). There is no visible security impact, however this behavior increases gas costs for orders that can be publicly resolved.

Consider changing the logic of the expression.

Update: *Resolved at commit* `3b92467`.

Notes & Additional Information

N-01 Unused named return variables

Named return variables are a way to declare variables that are meant to be used inside a function body and returned at the end of the function. This is an alternative to the explicit `return` statement to provide function outputs.

In the `FeeBank.sol` contract, the following functions contain instances of unused named return variables:

- `deposit`
- `depositFor`
- `depositWithPermit`
- `depositForWithPermit`
- `withdraw`
- `withdrawTo`

Consider using any unused named return variables. For example, the `return _depositFor(msg.sender, amount);` line becomes `totalAvailableCredit = _depositFor(msg.sender, amount)`.

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

> *Will not fix. It is debatable whether this change improves readability or not.*

N-02 Missing docstrings

There are several parts of the codebase that can benefit from having docstrings. For instance:

- All custom data structures; for example `tokensAndAmounts`
- Parsing of custom data structures should reference the custom data structure documentation. For example, the `_settleOrder` function could mention where to find docs for the order and dynamic suffixes
- All the functions that are part of the contract's public API

Consider documenting the aforementioned behavior and functions. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

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

> *Noted. We will fix the NatSpec docs in the future.*

N-03 Unused imports

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

- Import `DynamicSuffix` of `IResolver.sol`
- Import `AddressLib` of `DynamicSuffix.sol`
- Import `AddressLib` of `TokensAndAmounts.sol`

Consider removing unused imports to avoid confusion that could reduce the overall clarity and readability of the codebase.

Update: *Resolved at commit* `b3f6f9e`*. In regards to the* `AddressLib` *imports, the 1inch team stated:*

> *Other imports are needed for the "Address" type definition.*

Conclusions

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

Appendix

Monitoring Recommendations

While audits help in identifying potential security risks, the 1inch team is encouraged to also incorporate automated monitoring of on-chain contract activity into their operations. Ongoing monitoring of deployed contracts helps in identifying potential threats and issues affecting the production environment.

The internally tracked balance of 1inch tokens should always reflect the amount of tokens held by the contract. If the internally tracked balance is higher than the actual balance, there is an accounting error or an actively exploited vulnerability.

The protocol also contains an instance of the ERC-20 pods contract (`St1inch`). Attached pods should not revert during normal operation. When they do revert, their balance tracking will not correctly reflect the true balances of the underlying ERC-20 pods contract. This could signal either malicious activity or oversight from pod developers and users.

The whitelist can change over time as delegations change, resulting in a whitelisted user dipping below the `resolverThreshold`. In this case, it is possible for the on-chain representation of the whitelist to not accurately reflect the list of addresses that should be whitelisted. Monitoring the whitelist for inaccuracies can reveal if one of the update functions needs to be called.

We also recommend monitoring the entries and exits of addresses on the whitelist. Under normal use, it is not expected for an address to enter and exit the whitelist repeatedly. Detecting a large number of repeat entries and exits to the whitelist can signal either a need for a larger whitelist or possibly a vulnerability that an attacker is exploiting.

The `Settlement` contract does not support any use by users that are not registered on the whitelist. Monitoring for non-reverted calls from addresses that are not on the whitelist could lead to detecting an actively exploited vulnerability.

Since the `FeeBank` holds user funds, it is recommended that proper accounting via off-chain monitoring is in place to ensure balances properly reflect deposited funds.

The `St1inch`, `FeeBank`, and `WhitelistRegistry` all have owners associated with them that are capable of executing privileged actions. In the rare case of keys being stolen,

monitoring activity from owners can detect any unexpected activity and alert the protocol owners of the stolen keys.