OpenZeppelin | security

Limit Order Protocol Diff Audit

1inch NETWORK

December 15, 2023

Table of Contents

Table of Contents 2

Summary 3

Scope 4

Overview of the Changes 4

System Overview 6

Security Considerations for Users 7

Summary of Findings 8

Medium Severity 9

M-01 Forged Event Emission 9

Low Severity 10

L-01 Makers May Profit off of Takers via Malicious makerPermit Extension 10

L-02 External Calls Have a Hard-Coded Gas Limit 11

L-03 Incomplete Interface 12

Notes & Additional Information 12

N-01 Unused Named Return Variable 12

N-02 Unused State Variable 12

N-03 Missing Named Parameters in Mappings 13

N-04 Missing Security Contact 13

N-05 Code Can Be Simplified 14

N-06 Incorrect or Missing Documentation 14

Conclusion 16

Summary

Type	DeFi
Timeline	From 2023-11-01 To 2023-11-15
Languages	Solidity, Yul

Total Issues	10 (8 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	1 (1 resolved)
Low Severity Issues	3 (2 resolved)
Notes & Additional Information	6 (5 resolved)

Scope

We audited the changes made to the 1inch/limit-order-protocol repository between the `4.0.0-prerelease-8` tag and the `master` branch at that time. The corresponding commits were e9b4573 and aab3364.

In scope were the following contracts:

```
contracts
├── LimitOrderProtocol.sol
├── OrderLib.sol
├── OrderMixin.sol
└── libraries
    ├── ExtensionLib.sol
    ├── MakerTraitsLib.sol
    ├── RemainingInvalidatorLib.sol
    └── TakerTraitsLib.sol
```

Overview of the Changes

The 1inch team incorporated the following changes into the protocol:

- The public interface for filling orders was refactored to have only the following functions: `fillOrder`, `fillOrderArgs`, `fillContractOrder`, `fillContractOrderArgs`. The first pair works with EOA signatures, and the second pair works with smart contract signatures that follow the ERC-1271 standard. The first and third functions work with simple orders while the second and fourth functions support taker interactions and maker extensions.
- Takers can no longer improve the rate for makers. The `NO_IMPROVE_RATE` flag has been removed.
- Takers can now use the `permitAndCall` convenience function to execute a permit and fill an order in one function call. Supported permits include ERC-2612 permits, DAI-like permits, and Permit2.
- All price-getter extensions are now required to follow the `IAmountGetter` interface.
- The remaining invalidator now stores the bitwise `not` of the remaining amount instead of the remaining amount plus 1 wei. A value of zero means that the order is either a new

one or does not exist. The value of `type(uint256).max` means that the order is either fully filled or cancelled.

- The `OrderLib.isValidExtension` internal function now returns a boolean and an error code instead of reverting.

System Overview

The Limit Order Protocol implements an order book with makers creating and signing orders off-chain, and takers filling these orders on-chain. Makers can expose their orders to takers via the Limit Orders API or by any other means. The protocol relies on ERC-2612 and DAI-like permits for EOA signatures, EIP-712 and ERC-1271 for smart contract signatures, and also supports Permit2. Additionally, it comes with plenty of extensions that can be enabled to customize orders.

The `LimitOrderProtocol` contract allows the filling and cancelling of orders along with helper functionality for takers. It is possible to have orders with maker extensions and taker interactions or without them. Furthermore, the maker can be an EOA or a contract. Thus, there are four different functions to fill an order.

Order cancellation comes in three flavours depending on the order parameters:

- For orders that support multiple fills, individual invalidation by order hash is used.
- Additionally, for orders that support multiple fills, mass invalidation by series and epoch can be enabled and used.
- For orders that do not support multiple fills, mass invalidation by setting a bit in a bitmap is used.

Basic orders which do not have any extensions support the following features:

- Specify the receiving wallet for an order which can be different from the maker.
- Choose whether to allow or disallow partial and multiple fills.
- Choose an approval method for token allowance (approve, permit, or Permit2).
- Choose to unwrap WETH.
- Make an order private by specifying the only allowed taker's address. Technically, only the last 10 bytes of the address need to match the taker's address for the order filling to succeed.
- Specify the order's expiration timestamp.
- Specify a nonce or an epoch for the order for later cancellation.

Extensions introduce the following features:

- Define conditions that must be met before execution can proceed (e.g., stop-loss, take-profit orders).

- Specify a pre-interaction (arbitrary maker's code to dynamically procure the funds) to execute before the swap takes place, and a post-interaction (arbitrary maker's code to ensure the order was filled correctly) to be executed at the end of the order filling flow.
- Define a proxy to handle assets that are not ERC-20-compliant, allowing to swap ERC-721 and ERC-1155 tokens.
- Define functions for on-chain exchange rate calculation. These functions can be used to implement Dutch auctions (where the rate decreases over time) or range orders (where the rate depends on the volume already filled), among others.

Additionally, the taker can use taker traits for the following features:

- Specify either the making amount (how much they want to receive) or the taking amount (how much they want to give).
- Choose to unwrap WETH.
- Choose to skip the maker's permit.
- Choose an approval method for token allowance (approve, permit, or Permit2).
- Specify the threshold that refers to either the minimum amount to receive from the maker, or the maximum amount the taker wants to give depending on whether the making amount or the taking amount is specified.

Security Considerations for Users

The following security considerations are not inherently bugs or vulnerabilities but may lead to them if not accounted for.

- By default, orders can be partially filled but cannot be filled multiple times. A malicious taker can fill 1 wei of such an order to effectively invalidate it. In practice, however, most such orders are limited to a single taker so this is not possible. In other cases, the user can disable partial filling or enable multi-fill and send the order again.
- ERC-1271 implies that signature validation can be context-dependent (e.g., time-based or state-based). However, for multi-fill orders, the Limit Order Protocol only checks the signature on the first fill. This can lead to a scenario whereby when the order is partially filled, the signature is invalidated with the expectation that the order cannot be filled any more, whereas the order can still be filled because the signature is only checked on the

first fill. Nevertheless, the Limit Order Protocol allows specifying predicates to achieve behavior similar to what is implied by ERC-1271.

Summary of Findings

We found the codebase to be very well designed, with abundant documentation and highly optimized for gas efficiency. It also offers a great variety of settings to both takers and makers. We are glad to see robust security patterns in place and thoughtful considerations being applied to follow the best practices.

Medium Severity

M-01 Forged Event Emission

Users have two options to cancel their orders:

- Via the `cancelOrder` function, which uses both the `BitInvalidator` and the `RemaningInvalidator` approaches for cancellation, and emits the `OrderCancelled` event.
- Via the `bitsInvalidateForOrder` function, which uses only the `BitInvalidator` approach and does not emit any event regarding order cancellation.

Given that the `BitInvalidator` approach invalidates multiple orders at once and is not bound to a specific order hash, it is natural to skip emitting the `OrderCancelled` event within it. However, when using this approach via the `cancelOrder` function, the event will be emitted nonetheless. Furthermore, a malicious user may leverage this behaviour to confuse off-chain systems by emitting the `OrderCancelled` event for order hashes that are not actually cancelled.

Consider not emitting the `OrderCancelled` event for `BitInvalidator` cancellations within the `cancelOrder` function, and instead emitting another event that contains more relevant information. Moreover, consider refactoring the `cancelOrder` and `bitsInvalidateForOrder` functions to separate `BitInvalidator` and `RemaningInvalidator` approaches. This will also protect users from using incorrect maker traits with the help of the `OrderIsNotSuitableForMassInvalidation` error.

Update: *Resolved in pull request #292 at commit 29199d8. The 1inch team added a new* `BitInvalidatorUpdated` *event, which is triggered when orders with* `useBitInvalidator` *are cancelled. Normal cancellations by* `orderHash` *still trigger the old* `OrderCancelled` *event.*

Low Severity

L-01 Makers May Profit off of Takers via Malicious `makerPermit` Extension

Makers have the option of giving allowance via permit, in which case the `HAS_EXTENSION_FLAG` from the `MarketTraitsLib` library would need to be set and the `extension` should contain the `makerPermit` payload.

When takers fill such an order, if no prior allowance exists, the `SKIP_ORDER_PERMIT_FLAG` needs to be set to zero in order to ensure the permit will be consumed right after verifying the signature.

In order to consume the permit, both the target address and the permit payload need to be extracted from the `extension` bytes. The first 20 bytes of the maker permit will be used as the target address, while the remaining bytes will be used as the payload.

Depending on the permit length, the call will be managed differently, always respecting the first 20 bytes of the `makerPermit` as the address to call, except when the payload indicates that it is a `Permit2`, in which case the proper contract will be called.

Malicious makers can inject a malicious `makerPermit` bytes parameter encoded within the order `extension` so that the `tryPermit` function calls a contract owned by them. When called, this contract can perform arbitrary actions at the expense of the taker, since they are the ones funding the gas fees. Potential gas-heavy actions that could be run for free include:

- Contract deployment
- Filling another limit order
- Minting gas tokens on chains where they are still available in order to later sell them for a profit

This attack will be possible as long as:

- The malicious contract actually ends up consuming the permit or giving the necessary allowance for the order filling to go through.
- The expected profit for the taker will still be positive, even after accounting for the extra gas. This assumes that the takers are considering the total gas spent before the end of the transaction and reverting if a net profit cannot be achieved.

In order to mitigate this attack vector, consider disabling makers from being able to specify the target address within the `makerPermit` bytes. If makers want to use `Permit2`, the `SafeERC20` library is already overriding the target value. For ERC-2612-compatible permits, or exotic permit structures such as with the DAI token, the target address should be the `makerAsset`. This will also help optimize gas consumption when using permits since extensions will use 20 bytes less of `calldata`.

Even though `takers` will still make a profit and this technique could be considered within the "rules of the game", consider following this recommendation in order to make the service fairer to all `takers` while also making the protocol slightly more gas efficient.

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

> *We can't be sure that `makerAsset` is actually the token. When using proxies, for example, `makerAsset` is the address of the proxy and not the asset itself. And also, in that case, `maker` sets the allowance to the proxy address, not to the `LimitOrderProtocol` address.*

L-02 External Calls Have a Hard-Coded Gas Limit

Gas limits on calls are considered a bad practice for the following reasons:

- Gas costs have changed over time (e.g., as per EIP-1884).
- EVM side-chains and L2s can have different gas costs. This has previously caused funds to become stuck as transfers run out of gas and revert.
- When receivers are smart contracts, they might implement further logic that exceeds the gas limit. This will become especially relevant when account abstraction becomes more popular.

These factors could lead to unreliable functionality over time and across chains. A smart contract wallet can have logic in its `receive` or `fallback` function which exceeds the 5000 gas limit, thereby disabling that wallet to use the limit order protocol for swaps where the native currency is involved.

The following calls have a hard-coded gas limit:

- The call `msg.sender.call{value: msg.value - takingAmount, gas: _RAW_CALL_GAS_LIMIT}("")` in the contract `OrderMixin` in `OrderMixin.sol`
- The call `order.getReceiver().call{value: takingAmount, gas: _RAW_CALL_GAS_LIMIT}("")` in the contract `OrderMixin` in `OrderMixin.sol`

To ensure expected behavior and prevent external calls from running out of gas, consider removing the gas limitation or providing an explicit reason in the documentation for such limit.

Update: *Resolved in pull request #292 at commit fcd4732. The 1inch team decided to remove the hard-coded gas limit on both external calls.*

L-03 Incomplete Interface

The function `permitAndCall` in the `OrderMixin` contract is one of the main entry points for takers when filling orders if they want to execute a permit before filling an order. However, it is not part of the `IOrderMixin` interface.

Consider adding this function to the `IOrderMixin` interface in order to have a comprehensive API for third parties looking to integrate with the protocol.

Update: *Resolved in pull request #292 at commit 1800be9.*

Notes & Additional Information

N-01 Unused Named Return Variable

Named return variables are used in a function's body to be returned as the function's output. They are an alternative to explicit in-line `return` statements.

In `ExtensionLib.sol`, the `result` return variable for the `_get` function is unused.

Consider either using or removing any unused named return variables.

Update: *Resolved in pull request #292 at commit 48c2cc1.*

N-02 Unused State Variable

In the `OrderMixin` contract, the `_PERMIT2` state variable is unused.

To improve the overall clarity, intentionality, and readability of the codebase, consider removing any unused state variables.

Update: *Resolved in pull request #292 at commit 69a7a08.*

N-03 Missing Named Parameters in Mappings

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 provides a more transparent representation of a mapping's purpose.

Throughout the codebase, there are multiple mappings without named parameters:

- The `_bitInvalidator` state variable in the `OrderMixin` contract
- The `_remainingInvalidator` state variable in the `OrderMixin` contract

Consider adding named parameters to the mappings to improve the readability and maintainability of the codebase. Particularly, the `_remainingInvalidator` mapping could benefit from being more explicit about the key names.

Update: *Resolved in pull request #292 at commit 154519f.*

N-04 Missing 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 proves 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. Furthermore, if the contract incorporates third-party libraries and a bug surfaces in these, it becomes easier for the maintainers of those libraries to contact the appropriate person about the problem and provide mitigation instructions.

Throughout the codebase, there are contracts that do not have a security contact:

- The `ExtensionLib` contract
- The `LimitOrderProtocol` contract
- The `MakerTraitsLib` contract
- The `OrderLib` contract

- The `OrderMixin` contract
- The `RemainingInvalidatorLib` contract
- The `TakerTraitsLib` contract

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-05 Code Can Be Simplified

The `permitAndCall` function of the `OrderMixin` contract uses the version of `tryPermit` with four parameters. However, there is a shorter version of `tryPermit` which matches the use case.

Consider using the shorter version for conciseness.

Update: *Resolved in pull request #292 at commit c83322c.*

N-06 Incorrect or Missing Documentation

Throughout the codebase, there are several parts that do not have docstrings:

- Docstrings on line 16 in `TakerTraitsLib.sol` claim that four bits are used for flags and the remaining bits are used to store the threshold amount, whereas the threshold amount actually takes fewer bits. The same docstrings in `TakerTraitsLib.sol` are missing documentation for the `_ARGS_HAS_TARGET` flag, the `_ARGS_EXTENSION_LENGTH_OFFSET` and `_ARGS_INTERACTION_LENGTH_OFFSET` values, and bits from 250 to 248.
- Docstrings on lines 14-15 in `MakerTraitsLib.sol` are missing documentation for the 253rd bit. While the bit is related to a removed feature and is not used anymore, this fact should nonetheless be stated explicitly.
- Docstrings on line 460 in `OrderMixin.sol` claim that the function processes taker interaction while it also processes maker extensions. The same docstrings on line 461 in `OrderMixin.sol` claim to revert if the taker permit is invalid, whereas permits are not checked to begin with.
- Docstrings on line 47 in `OrderMixin.sol` are missing.

- Documentation for predicates mentions `predicate2` twice on line 479 instead of mentioning `predicate2` and `predicate3` once.
- Docstrings on line 77 in `IOrderMixin.sol` refer to the old way of storing the remaining amount.
- The main `description.md` file has not been updated to reflect the new changes.

Consider updating the documentation so that it reflects all the newly introduced changes.

Update: *Resolved in pull request #285 at commits b21a2e and ca2158d, and in pull request #292 at commits 1800be9, aa88e62, 46f4795 and 3c9b8ab.*

Conclusion

The audited version of the 1inch Limit Order Protocol enables highly efficient permissionless swaps between makers and takers. It provides a high degree of flexibility and customizability while ensuring safety for all parties involved. We found the codebase to be very well-written, thoroughly documented, and optimized for gas consumption.

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 team was very responsive throughout the engagement, providing us with the necessary insight to expedite our understanding of the changes implemented and their effect on the codebase.