Blaize.Security

November 5th, 2021 / V. 1.0

1INCH LIMIT ORDER SMART CONTRACT AUDIT

TABLE OF CONTENTS

AUDIT RATING

The 1inch Limit Order smart contract's source code was taken from one repository provided by the 1inch team.

SCORE **10**/10

The scope of the project is **Limit Order** set of contracts**:**

1/ AmountCalculator.sol

2/ ChainlinkCalculator.sol

3/ ERC721Proxy.sol

4/ ERC721ProxySafe.sol

5/ ERC1155Proxy.sol

6/ ImmutableOwner.sol

7/ NonceManager.sol

8/ PredicateHelper.sol

9/ ArgumentsDecoder.sol

10/ Permitable.sol

11/ RevertReasonParser.sol

12/ LimitOrderProtocol.sol

13/ OrderMixin.sol

14/ OrderRFQMixin.sol

The scope of the audit is the code at the main branch with commit:

- https://github.com/1inch/limit-order-protocol/commit/568e0718eaabd932e6ff5c8395702a5d48dadd68

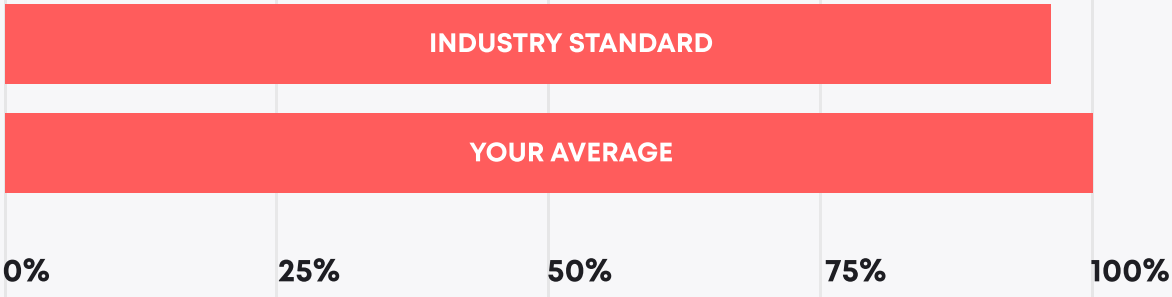
Post-audit scope for validation includes the code at the main branch with commit:

- https://github.com/1inch/limit-order-protocol/commit/568e0718eaabd932e6ff5c8395702a5d48dadd68

TECHNICAL SUMMARY

In this report, we consider the security of the contracts for Limit Order protocol. Our task is to find and describe security issues in the smart contracts of the platform. This report presents the findings of the security audit of **1inch Limit Order** smart contracts conducted between **October 27th, 2021 - November 05th, 2021.**

Testable code



The testable code is 100%, which is above the industry standard of 95%.

The scope of the audit includes the unit test coverage, that bases on the smart contracts code, documentation and requirements presented by the 1inch team. Coverage is calculated based on the set of Truffle framework tests and scripts from additional testing strategies. Though, in order to ensure a security of the contract Blaize.Security team recommends the 1inch team put in place a bug bounty program to encourage further and active analysis of the smart contracts.

THE GRAPH OF VULNERABILITIES DISTRIBUTION:

The table below shows the number of found issues and their severity. A total of 9 problems were found. 8 issues were fixed or verified by the 1inch team.

	FOUND	FIXED/VERIFIED
Critical	0	0
High	0	0
Medium	0	0
Low	6	6
Lowest	3	2

SEVERITY DEFINITION

Critical

A system contains several issues ranked as very serious and dangerous for users and the secure work of the system. Needs immediate improvements and further checking.

High

A system contains a couple of serious issues, which lead to unreliable work of the system and migh cause a huge information or financial leak. Needs immediate improvements and further checking.

Medium

A system contains issues which may lead to mediumfinancial loss or users' private information leak. Needs immediate improvements and further checking.

Low

A system contains several risks ranked as relatively small with the low impact on the users' information and financial security. Needs improvements.

Lowest

A system does not contain any issue critical to the secure work of the system, yet is relevant for best

AUDITING STRATEGYAND TECHNIQUES APPLIED \ PROCEDURE

We have scanned this smart contract for commonly known and more specific vulnerabilities:

- Unsafe type inference;
- Timestamp Dependence;
- Reentrancy;
- Implicit visibility level;
- Gas Limit and Loops;
- Transaction-Ordering Dependence;
- Unchecked external call - Unchecked math;
- DoS with Block Gas Limit;
- DoS with (unexpected) Throw;
- Byte array vulnerabilities;
- Malicious libraries;
- Style guide violation;
- ERC20 API violation;
- Uninitialized state/storage/ local variables;
- Compile version not fixed.

Procedure

In our report we checked the contract with the following parameters:

- Whether the contract is secure;
- Whether the contract corresponds to the documentation;
- Whether the contract meets best practices in efficient use of gas, code readability;

Automated analysis:

Scanning contract by several public available automated analysis tools such as Mythril, Solhint, Slither and Smartdec. Manual verification of all the issues found with tools.

Manual audit:

Manual analysis of smart contracts for security vulnerabilities. Checking smart contract logic and comparing it with the one described in the documentation.

EXECUTIVE SUMMARY

According to the assessment, the Customer's smart contracts have no critical security problems. All unclear or suspicious functionality was verified with 1inch team and with addtional tests. Several places require more detailed comments about the actions performed, since part of security is performed on the side of the dApp. Nevertheless, overall quality of the code is high and the functionality is well-documented and optimized. We described issues and added Customer's comments according to the conclusion of these documents. Please read the whole document to estimate the risks well.

	RATING
Security	9.8
Gas usage and logic optimization	10
Code quality	9.6
Test coverage	10
Total	9.9

COMPLETE ANALYSIS

LOW	✔ Verified

Use safeTransfer functionality for unvalidated tokens

OrderMix.sol, line 256 and 276, fillOrderTo()
Consider usage of safeTransfer for the tokens which do not inherit IERC20 and do not implement canonical interface (like USDT)

Recommendation:

Use safeTransfer operations.

Post-audit:

Security verified both with the 1inch team and with additional tests.

LOW	✔ Verified

Unclear functionality

OrderMix.sol, fillOrderTo()
The function provides unreasonable subtraction and addition of 1 to the remainingMakerAmount (thus to the _remaining[orderHash]). Verify the functionality and provide the appropriate comment.

Recommendation:

Verify the functionality and provide the appropriate comment.

Post-audit:

Functionality verified with 1inch team - it serves for filled orders distinguishing. Nevertheless, it is recommended to add comments to all places with such arithmetical operations.

LOW ✔ Verified

Better choice for the type

OrderRFQMixin.sol, OrderRFQ structure, info field.
The contract utilizes uint256 type for the order.info field. Though, throughout the contract, the highest 128 bit are not used:
- lowest 64 bits for the order if (8 lowest bits - shift for the invalidator bit, 56 bits - order slot)
- next 64 bits for the expiration number
- 128 unused bits
In order to go further in gas optimisation (since we already have packing into the number by bits shifts) uint128 may be used.
The same can be considered for the _invalidator[address][invalidatorSlot] since invalidatorSlot occupies 64 bits, but uint256 is used.

Recommendation:

Consider another type usage for the storage decreasing.

Post-audit:

After the conversation with the 1inch team and additional checks the functionality is verified as storage-efficient.

LOW

✔ Verified

Unused and unconnected storage

OrderMixin contract inherits NonceManager functionality. Though its storage (nonce mapping) and connected functions are not used throughout the code. It seems that these storage and functions are supposed to be used by external sources or directly from the app. Though, “nonce” keyword and all connected functions have unclear usage in the LimitOrder contract and may mislead users or developers.
Nevertheless, since this functionality has no connection to LimitOrder functionality it is recommended to separate it into another contract. Thus it will decrease the contracts size and remove the unclarified functionality.

Recommendation:

Remove unused storage and functionality (unused inheritance) or verify the usage of the functionality.

Post-audit:

Functionality was verified with 1inch team as integrated for gas-optimisation purpose.

LOW ✔ Verified

Unused structure field

OrderMixin.sol and OrderRFQMixin.sol, allowedSender field of structures.
allowedSender field is used in checks against 0 address and msg.sender, but it is not stored anywhere in the contract and can be sent as any value, since all participation functions are public. Thus, the usage of this field and checks against it are useless, unless this field is complexly included from the dApp. Though, since any value can be sent and it is not connected with other functionality, there is no need in its usage from smart contracts point of view.

Recommendation:

Remove unused structure field or verify its usage.

Post-audit:

Security verified both with the 1inch team and with additional tests.

LOW ✔ Verified

Duplicate permit possibility

OrderMix.sol, line 179, fillOrderToWithPermit()
OrderMix.sol, line 209, fillOrderTo()
There is a lack of checks to prevent double permit. Or there is a possibility of a missed duplicated permit for makerAsset or takerAsset, since there is no sign of which token is permitted in both situations.

Recommendation:

Verify the functionality or prevent double permission in order to provide gas savings.

Post-audit:

Security verified both with the 1inch team and with additional tests.

LOWEST	✔ Verified

Missing security check

OrderMix.sol, line 179, fillOrderToWithPermit()
OrderMix.sol, line 209, fillOrderTo()
The function for the order fulfillment has the check against reentrancy and there is no sense to give permit for the tokens other than makerAsset and takerAsset. Though, since there are a lot of known cases of exploits through fake tokens and since the verification of the security of the code against all such attacks needs a round of fuzzy testing and edge cases testing, it is recommended to prevent the usage of unchecked token's contracts. Or at least provide all necessary security checks, such as the check, that ony makerAsset and takerAsset can be used for permits.
Since the chance of such an exploit is very low and possible impact is very low the issue is marked as info. But because of the complexity of proof that such an exploit is impossible, the issue cannot be omitted.

Recommendation:

Add security checks for the token contract verification.

Post-audit:

The functionality was verified with the 1inch team.

LOWEST	✔ Verified

Unused functionality

Functionality from ChainlinkCalculator.sol inherited in OrderMixin contract is not used throughout the code. Looks like it is needed for the further development or for the aggregated calls from the frontend app. Remove the inheritance in order to decrease the contract's size or verify the usage in further development.

Recommendation:

Remove unused inheritance (into the separate contract, for example) or verify the functionality is needed in the contract.

Post-audit:

Functionality was verified with 1inch team as integrated for gas-optimisation purpose.

LOWEST	✔ Unresolved

Avoid direct bool comparison

OrderMix.sol, line 223, fillOrderTo()
(takingAmount == 0) == (makingAmount == 0)
Use boolean operations instead of direct comparison.

Recommendation:

Use boolean operations instead of direct comparison.

	Amount Calculator	Chainlink Calculator	ERC721Proxy
✓ Re-entrancy	Pass	Pass	Pass
✓ Access Management Hierarchy	Pass	Pass	Pass
✓ Arithmetic Over/Under Flows	Pass	Pass	Pass
✓ Delegatecall Unexpected Ether	Pass	Pass	Pass
✓ Default Public Visibility	Pass	Pass	Pass
✓ Hidden Malicious Code	Pass	Pass	Pass
✓ Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
✓ External Contract Referencing	Pass	Pass	Pass
✓ Short Address/ Parameter Attack	Pass	Pass	Pass
✓ Unchecked CALL Return Values	Pass	Pass	Pass
✓ Race Conditions / Front Running	Pass	Pass	Pass
✓ General Denial Of Service (DOS)	Pass	Pass	Pass
✓ Uninitialized Storage Pointers	Pass	Pass	Pass
✓ Floating Points and Precision	Pass	Pass	Pass
✓ Tx.Origin Authentication	Pass	Pass	Pass
✓ Signatures Replay	Pass	Pass	Pass
✓ Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

	ERC721ProxySafe	ERC1155Proxy	Immutable Owner
✓ Re-entrancy	Pass	Pass	Pass
✓ Access Management Hierarchy	Pass	Pass	Pass
✓ Arithmetic Over/Under Flows	Pass	Pass	Pass
✓ Delegatecall Unexpected Ether	Pass	Pass	Pass
✓ Default Public Visibility	Pass	Pass	Pass
✓ Hidden Malicious Code	Pass	Pass	Pass
✓ Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
✓ External Contract Referencing	Pass	Pass	Pass
✓ Short Address/ Parameter Attack	Pass	Pass	Pass
✓ Unchecked CALL Return Values	Pass	Pass	Pass
✓ Race Conditions / Front Running	Pass	Pass	Pass
✓ General Denial Of Service (DOS)	Pass	Pass	Pass
✓ Uninitialized Storage Pointers	Pass	Pass	Pass
✓ Floating Points and Precision	Pass	Pass	Pass
✓ Tx.Origin Authentication	Pass	Pass	Pass
✓ Signatures Replay	Pass	Pass	Pass
✓ Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

	NonceManager	PredicateHelper	Arguments Decoder
✓ Re-entrancy	Pass	Pass	Pass
✓ Access Management Hierarchy	Pass	Pass	Pass
✓ Arithmetic Over/Under Flows	Pass	Pass	Pass
✓ Delegatecall Unexpected Ether	Pass	Pass	Pass
✓ Default Public Visibility	Pass	Pass	Pass
✓ Hidden Malicious Code	Pass	Pass	Pass
✓ Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
✓ External Contract Referencing	Pass	Pass	Pass
✓ Short Address/ Parameter Attack	Pass	Pass	Pass
✓ Unchecked CALL Return Values	Pass	Pass	Pass
✓ Race Conditions / Front Running	Pass	Pass	Pass
✓ General Denial Of Service (DOS)	Pass	Pass	Pass
✓ Uninitialized Storage Pointers	Pass	Pass	Pass
✓ Floating Points and Precision	Pass	Pass	Pass
✓ Tx.Origin Authentication	Pass	Pass	Pass
✓ Signatures Replay	Pass	Pass	Pass
✓ Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

	Permitable	RevertReason Parser	LimitOrder Protocol
✓ Re-entrancy	Pass	Pass	Pass
✓ Access Management Hierarchy	Pass	Pass	Pass
✓ Arithmetic Over/Under Flows	Pass	Pass	Pass
✓ Delegatecall Unexpected Ether	Pass	Pass	Pass
✓ Default Public Visibility	Pass	Pass	Pass
✓ Hidden Malicious Code	Pass	Pass	Pass
✓ Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
✓ External Contract Referencing	Pass	Pass	Pass
✓ Short Address/ Parameter Attack	Pass	Pass	Pass
✓ Unchecked CALL Return Values	Pass	Pass	Pass
✓ Race Conditions / Front Running	Pass	Pass	Pass
✓ General Denial Of Service (DOS)	Pass	Pass	Pass
✓ Uninitialized Storage Pointers	Pass	Pass	Pass
✓ Floating Points and Precision	Pass	Pass	Pass
✓ Tx.Origin Authentication	Pass	Pass	Pass
✓ Signatures Replay	Pass	Pass	Pass
✓ Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

	OrderMixin	OrderRFQMixin
✓ Re-entrancy	Pass	Pass
✓ Access Management Hierarchy	Pass	Pass
✓ Arithmetic Over/Under Flows	Pass	Pass
✓ Delegatecall Unexpected Ether	Pass	Pass
✓ Default Public Visibility	Pass	Pass
✓ Hidden Malicious Code	Pass	Pass
✓ Entropy Illusion (Lack of Randomness)	Pass	Pass
✓ External Contract Referencing	Pass	Pass
✓ Short Address/ Parameter Attack	Pass	Pass
✓ Unchecked CALL Return Values	Pass	Pass
✓ Race Conditions / Front Running	Pass	Pass
✓ General Denial Of Service (DOS)	Pass	Pass
✓ Uninitialized Storage Pointers	Pass	Pass
✓ Floating Points and Precision	Pass	Pass
✓ Tx.Origin Authentication	Pass	Pass
✓ Signatures Replay	Pass	Pass
✓ Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Contract: LimitOrderProtocol

Order test

- ✓ swap fully on signature
- ✓ should swap fully based on RFQ signature
- ✓ should swap half based on signature
- ✓ should floor maker amount
- ✓ should ceil taker amount
- ✓ should fail on floor maker amount = 0
- ✓ should fail with 0 amount
- ✓ should not swap with bad signature
- ✓ should not swap with bad signature RFQ
- ✓ should not fill (1,1)
- ✓ should not fill below threshold
- ✓ should fail when both amounts are zero
- ✓ should fail when both amounts are not zero with OrderRFQ
- ✓ should fail when amount in hash order lower than ordered
- ✓ order without getTakerAmount
- ✓ order without getMakerAmount
- ✓ should fail with wrong maker amount in order without getTakerAmount
- ✓ should fail with wrong taker amount in order without getMakerAmount
- ✓ should fail with too high taking amount in order without getMakerAmount
- ✓ usdt taker test
- ✓ usdt maker test

Permit tests
fillOrderToWithPermit function test
✓ DAI => WETH
✓ DAI => WETH with permit in order
✓ rejects reused signature
fillOrderRFQToWithPermit
✓ DAI => WETH
✓ rejects expired permit
Order Cancelation
✓ should cancel own order
✓ should not cancel foreign order
OrderRFQ Cancellation
✓ should cancel own order
✓ should not fill cancelled order
Private Orders
✓ should fill with correct taker
✓ should not fill with incorrect taker
✓ should not fill with incorrect taker RFQ
Predicate
✓ ȯr`should pass
✓ ȯr`should fail
✓ nonce + ts example
✓ advance nonce
✓ ȧnd`should fail
Expiration
✓ should fill when not expired
✓ should not fill when expired
✓ should fill RFQ order when not expired
✓ should partial fill RFQ order
✓ should fully fill RFQ order
✓ should not partial fill RFQ order when 0
✓ should not fill RFQ order when expired
✓ should fill partially if not enough coins (taker)
✓ should fill partially if not enough coins
(maker)

```
  Interaction
✓ should fill and unwrap token
  Simulate call
✓ simulate simple function
✓ simulate failed transaction
✓ should fail with parameters length mismatch
  Remaining
✓ should faik when checking remaining for unknown
  order
✓ check remaining for single order
✓ check remaining for multiple orders

  55 passing (54s)
```

TEST COVERAGE RESULTS

FILE	% STMTS	% BRANCH	% FUNCS
OrderMixin.sol	100.00	90	100.00
OrderRFQMixin.sol	100.00	100	100.00
All files	**100**	**92.5**	**100**

DISCLAIMER

The information presented in this report is an intellectual property of the customer including all presented documentation, code databases, labels, titles, ways of usage as well as the information about potential vulnerabilities and methods of their exploitation. This audit report does not give any warranties on the absolute security of the code. Blaize.Security is not responsible for how you use this product and does not constitute any investment advice.

Blaize.Security does not provide any warranty that the working product will be compatible with any software, system, protocol or service and operate without interruption. We do not claim the investigated product is able to meet your or anyone else requirements and be fully secure, complete, accurate and free of any errors and code inconsistency.

We are not responsible for all subsequent changes, deletions and relocations of the code within the contracts that are the subjects of this report.

You should perceive Blaize.Security as a tool which helps to investigate and detect the weaknesses and vulnerable parts that may accelerate the technology improvements and faster error elimination.