1INCH LIMIT ORDER PROTOCOL SMART CONTRACT AUDIT

September 06, 2021

MixBytes()

CONTENTS

1. INTRODUCTION

1.1 DISCLAIMER

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of 1Inch. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 PROJECT OVERVIEW

1inch is a DeFi aggregator and a decentralized exchange with smart routing. The core protocol connects a large number of decentralized and centralized platforms in order to minimize price slippage and find the optimal trade for the users.

1.3 SECURITY ASSESSMENT METHODOLOGY

A group of auditors are involved in the work on the audit who check the provided source code independently of each other in accordance with the methodology described below:

01 Project architecture review:
> Reviewing project documentation
> General code review
> Reverse research and study of the architecture of the code based on the source code only
> Mockup prototyping

Stage goal:
Building an independent view of the project's architecture and identifying logical flaws in the code.

02 Checking the code against the checklist of known vulnerabilities:
> Manual code check for vulnerabilities from the company's internal checklist
> The company's checklist is constantly updated based on the analysis of hacks, research and audit of the clients' code
> Checking with static analyzers (i.e Slither, Mythril, etc.)

Stage goal:
Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flashloan attacks, etc.)

03 Checking the code for compliance with the desired security model:
> Detailed study of the project documentation
> Examining contracts tests
> Examining comments in code
> Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit
> Exploits PoC development using Brownie

Stage goal:
Detection of inconsistencies with the desired model

04 Consolidation of interim auditor reports into a general one:
> Cross-check: each auditor reviews the reports of the others
> Discussion of the found issues by the auditors
> Formation of a general (merged) report

Stage goal:
Re-check all the problems for relevance and correctness of the threat level and provide the client with an interim report.

05 Bug fixing & re-check:
> Client fixes or comments on every issue
> Upon completion of the bug fixing, the auditors double-check each fix and set the statuses with a link to the fix

Stage goal:
Preparation of the final code version with all the fixes

06 Preparation of the final audit report and delivery to the customer.

Findings discovered during the audit are classified as follows:

FINDINGS SEVERITY BREAKDOWN

Level	Description	Required action
Critical	Bugs leading to assets theft, fund access locking, or any other loss funds to be transferred to any party	Immediate action to fix issue
Major	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.	Implement fix as soon as possible
Warning	Bugs that can break the intended contract logic or expose it to DoS attacks	Take into consideration and implement fix in certain period
Comment	Other issues and recommendations reported to/acknowledged by the team	Take into consideration

Based on the feedback received from the Customer's team regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The project team is aware of this finding. Recommendations for this finding are planned to be resolved in the future. This finding does not affect the overall safety of the project.
No issue	Finding does not affect the overall safety of the project and does not violate the logic of its work.

1.4 EXECUTIVE SUMMARY

The audited scope implements a protocol of limit orders with two types:

- orderRFQ is a simple limit order with gas optimized option;
- order is a limit order has complex option with significant configuration variability.

with supported tokens ERC20, EC721, ERC1155 with some new adds like: taker permit.
The code is written in a very gas-efficient manner for cheap usage by end-users.

1.5 PROJECT DASHBOARD

Client	1Inch
Audit name	Limit Order Protocol
Initial version	a14bde6a260458de5083cee117d734221e1cbc05
Final version	4f9986a1b2dbc580f683f06a1519d9c554b72933
Date	August 20, 2021 - September 06, 2021
Auditors engaged	2 auditors

FILES LISTING

LimitOrderProtocol.sol	https://github.com/1inch/limit-order-protocol/tree/a14bde6a260458de5083cee117d734221e1cbc05/contracts/LimitOrderProtocol.sol
AmountCalculator.sol	https://github.com/1inch/limit-order-protocol/tree/a14bde6a260458de5083cee117d734221e1cbc05/contracts/helpers/AmountCalculator.sol
ChainlinkCalculator.sol	https://github.com/1inch/limit-order-protocol/tree/a14bde6a260458de5083cee117d734221e1cbc05/contracts/helpers/ChainlinkCalculator.sol
ERC1155Proxy.sol	https://github.com/1inch/limit-order-protocol/tree/a14bde6a260458de5083cee117d734221e1cbc05/contracts/helpers/ERC1155Proxy.sol
ERC20Proxy.sol	https://github.com/1inch/limit-order-protocol/tree/a14bde6a260458de5083cee117d734221e1cbc05/contracts/helpers/ERC20Proxy.sol

ERC721Proxy.sol	https://github.com/1inch/limit-order-protocol/tree/a14bde6a260458de5083cee117d734221e1cbc05/contracts/helpers/ERC721Proxy.sol
ImmutableOwner.sol	https://github.com/1inch/limit-order-protocol/tree/a14bde6a260458de5083cee117d734221e1cbc05/contracts/helpers/ImmutableOwner.sol
NonceManager.sol	https://github.com/1inch/limit-order-protocol/tree/a14bde6a260458de5083cee117d734221e1cbc05/contracts/helpers/NonceManager.sol
PredicateHelper.sol	https://github.com/1inch/limit-order-protocol/tree/a14bde6a260458de5083cee117d734221e1cbc05/contracts/helpers/PredicateHelper.sol
ArgumentsDecoder.sol	https://github.com/1inch/limit-order-protocol/tree/a14bde6a260458de5083cee117d734221e1cbc05/contracts/libraries/ArgumentsDecoder.sol
SilentECDSA.sol	https://github.com/1inch/limit-order-protocol/tree/a14bde6a260458de5083cee117d734221e1cbc05/contracts/libraries/SilentECDSA.sol
UncheckedAddress.sol	https://github.com/1inch/limit-order-protocol/tree/a14bde6a260458de5083cee117d734221e1cbc05/contracts/libraries/UncheckedAddress.sol

FINDINGS SUMMARY

Level	Amount
Critical	0
Major	0
Warning	7
Comment	7

CONCLUSION

Smart contract has been audited and several suspicious places were found. During the audit no critical or major issues were spotted. Several issues were marked as warnings and comments. After working on audit report all issues were fixed or acknowledged by the client. Thus, contract is assumed as secure to use according to our security criteria.Final commit identifier with all fixes: `4f9986a1b2dbc580f683f06a1519d9c554b72933`

2. FINDINGS REPORT

2.1 CRITICAL

Not Found

2.2 MAJOR

Not Found

2.3 WARNING

WRN-1	Possible call for zero address
File	LimitOrderProtocol.sol
Severity	Warning
Status	Acknowledged

DESCRIPTION

Zero address don't checked in the following function:
LimitOrderProtocol.sol#L145
LimitOrderProtocol.sol#L349

RECOMMENDATION

We recommend to add the following check:

```
require(targets[i] != address(0), "LOP: incorrect address");
```

CLIENT'S COMMENTARY

First function always reverts. And second one will just do nothing on wrong address. So we don't see benefits in those requires.

WRN-2	Cancel of already cancelled order
File	LimitOrderProtocol.sol
Severity	Warning
Status	Fixed at 56116071

DESCRIPTION

User can cancel already filled order, so incorrect event would emit:
LimitOrderProtocol.sol#L161

RECOMMENDATION

We recommend to add the following check:

```
require(_remaining[orderHash] != 1, "LOP: already filled");
```

WRN-3	Possible reentrancy
File	LimitOrderProtocol.sol
Severity	Warning
Status	Acknowledged

DESCRIPTION

In the following function reentrancy can occur:
LimitOrderProtocol.sol#L280-L345

RECOMMENDATION

We recommend to add `nonReentrant` modificator.

CLIENT'S COMMENTARY

There's a problem with reentrancy only if reentrancy will under permit, and from this protection is built in. If reentrancy will be when token transfer this is no problem because all all state updates take place before transfers.

WRN-4	Possible filling of already filled order
File	LimitOrderProtocol.sol
Severity	Warning
Status	No Issue

DESCRIPTION

User can try to fill already filled order by mistake:
LimitOrderProtocol.sol#L294

RECOMMENDATION

We recommend to add the following check:

```
require(remainingMakerAmount > 0, "LOP: already filled");
```

CLIENT'S COMMENTARY

It is enforced indirectly as we do not allow empty fills.

WRN-5	Possible ddos attack
File	LimitOrderProtocol.sol
Severity	Warning
Status	Acknowledged

DESCRIPTION

Malicious maker can block bots for filling order if he calls unbounded operation (infinite loop) in `notifyFillOrder`:
LimitOrderProtocol.sol#L338

RECOMMENDATION

We recommend to use gas limit for this function.

CLIENT'S COMMENTARY

This will lead to a single unfillable limit order which is fine.

WRN-6	Incorrect signature can lead to incorrect call
File	LimitOrderProtocol.sol
Severity	Warning
Status	No Issue

DESCRIPTION

In the following function, if signature is incorrect and maker is a user, then this function would revert because of static call for user:
LimitOrderProtocol.sol#L397

RECOMMENDATION

We recommend to change function like this:

```
if (signature.length != 65 && signature.length != 64) { ... }
else {require(SilentECDSA.recover(orderHash, signature) == maker), "LOP: incorrect signature
```

CLIENT'S COMMENTARY

Static calls to EOA are allowed. isValidSignature call will just return 0 bytes which will then revert with LOP: bad signature.

WRN-7	Unclear check
File	LimitOrderProtocol.sol
Severity	Warning
Status	Fixed at 4f9986a1

DESCRIPTION

If user passes incorrect amount to filling the order, then this function reverts because of static call with zero parameters:
LimitOrderProtocol.sol#L441
LimitOrderProtocol.sol#L452

RECOMMENDATION

We recommend to change the function like this:

```
require(order.getMakerAmount.length != 0, "LOP: incorrect amount");
```

2.4 COMMENT

CMT-1	Unclear field
File	LimitOrderProtocol.sol
Severity	Comment
Status	Fixed at 9d32dd53

DESCRIPTION

Meaning of bits in `info` field in `OrderRFQ` structure is unclear:
LimitOrderProtocol.sol#L59

RECOMMENDATION

We recommend to add a comment like this:

```
uint256 info; // unused(128bit) expiration(64bit) slot(56bit) shift(8bit)
```

CMT-2	`_MAX_SELECTOR` is too big
File	LimitOrderProtocol.sol
Severity	Comment
Status	No Issue

DESCRIPTION

Selector with maximum value which used in this contract = uint32(IERC20.transferFrom.selector) + 4:
LimitOrderProtocol.sol#L88

RECOMMENDATION

We recommend to reduce `_MAX_SELECTOR`.

CLIENT'S COMMENTARY

It is reserved for future token types.

CMT-3	Syntax inconsistency
File	LimitOrderProtocol.sol
Severity	Comment
Status	Fixed at 9617c704

DESCRIPTION

RFQ order shift calculation is different:
LimitOrderProtocol.sol#L167
LimitOrderProtocol.sol#L211

RECOMMENDATION

We recommend to use the same calculation to increase readability ((orderInfo & 0xff) or (uint8(orderinfo)))

CMT-4	Incorrect message in require
File	ERC721Proxy.sol
Severity	Comment
Status	Fixed at b170b3a6

DESCRIPTION

Message in require is incorrect:
ERC721Proxy.sol#L15

RECOMMENDATION

We recommend to change the message in require.

CMT-5	Incorrect event field
File	LimitOrderProtocol.sol
Severity	Comment
Status	No Issue

DESCRIPTION

In `OrderFilled` event first field should be `taker` LimitOrderProtocol.sol#L48

RECOMMENDATION

We recommend to rename `maker` to `taker`.

CLIENT'S COMMENTARY

This is done so that makers can get collect their stats. takers can collect their stats directly from their transactions.

CMT-6	Possibility of cancel non-existing orders
File	LimitOrderProtocol.sol
Severity	Comment
Status	No Issue

DESCRIPTION

User can cancel any order with `makerAssetData` containing his address at `FROM_INDEX` index LimitOrderProtocol.sol#L157-L163

RECOMMENDATION

We recommend to check order existing.

CLIENT'S COMMENTARY

This is by design. This is for orders that exists offchain but were not yet posted onchain.

CMT-7	User can deccrease allowance
File	LimitOrderProtocol.sol
Severity	Comment
Status	No Issue

DESCRIPTION

User can decrease allowance before fillOrder will be executed

RECOMMENDATION

Add additional require for checking allowance for user before next lines.
LimitOrderProtocol.sol#L206
LimitOrderProtocol.sol#L305
It saves some gas in this case. Also it possible to add this checking on user side before transaction executing.

CLIENT'S COMMENTARY

It saves some gas on fails but it also add additional gas overhead for successful transactions. Considering the tradeoff we'll leave the contract unchanged.

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build open-source solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

BLOCKCHAINS

- Ethereum
- Cosmos
- EOS
- Substrate

TECH STACK

- Python
- Solidity
- Rust
- C++

CONTACTS

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://t.me/MixBytes

https://twitter.com/mixbytes