DeHacker

Code Security Assessment

1inch – Limit Order

April 2nd, 2022

Contents

Summary

DeHacker's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow/underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service/logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Issue Categories

Every issue in this report was assigned a severity level from the following:

Critical severity issues

A vulnerability that can disrupt the contract functioning in a number of scenarios or creates a risk that the contract may be broken.

Major severity issues

A vulnerability that affects the desired outcome when using a contract or provides the opportunity to use a contract in an unintended way.

Medium severity issues

A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.

Minor severity issues

A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.

Informational

A vulnerability that has informational character but is not affecting any of the code.

Overview

Project Summary

Project Name	1inch - Limit Order
Platform	Ethereum
website	https://app.1inch.io/#/1/swap/ETH/DAI
Language	Solidity
Codebase	https://github.com/1inch/limit-order-protocol
Commit	c3f3d6af6c03603df7ec149afcc7bb86a627646 bf190982639e0408d0f8b1bbf4a874ff0810096c 9b5b4cfa174c2ca3e4d4b2f6b08eb73d13f2eefd

Audit Summary

Delivery Date	April 2, 2022
Audit Methodology	Static Analysis, Manual Review

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
Major	4	0	0	3	0	1
Medium	3	0	0	2	0	1
Minor	1	0	0	0	0	1
Informational	4	0	0	0	0	4
Discussion	0	0	0	0	0	0

Audit scope

ID	File	SHA256 Checksum
ACC	helpers/AmountCalculator.sol	193815b0f73cf2e4fc109897f587f9a22399ae829b1a3aaf48d81a46bddf4d63
CCC	helpers/ChainlinkCalculator.sol	a034717629da3b53a5db377cd13722696fa967cc60d49d9ed023f8389f3227e
ERC	helpers/ERC1155Proxy.sol	87599a0aeb6bde8bc9fa4c93184fc7f2f89f89719342cac256c42167f77374f2
ERP	helpers/ERC721Proxy.sol	b8fd60b0d0e3e33b48d00c1c5a37de771844c71b7235639958464836fafe066f
ERS	helpers/ERC721ProxySafe.sol	6dc8954e3ef892f3e8a9e5480db7b1b74c294f520e0f86b91a5fa5dafffc4581
IOC	helpers/ImmutableOwner.sol	4dd0eefcca2b4c029467afce8081e016e2e1ea4d802df247c4158250489d9827
NMC	helpers/NonceManager.sol	8a095c998af1df18a25ea1f4e1a152842d22d691b49618fe0812af83106c7b48
PHC	helpers/PredicateHelper.sol	db314568a0d9757a884a1ff3f76481f05fc47714e40f25e094e67287e3075176
ADC	libraries/ArgumentsDecoder.sol	7f3276bde678376da813d0d7efe70daf059c78d76dc6f6c23303afcee9dbf06c
PCK	libraries/Permitable.sol	7b65219c91eee0c043a08169a28dfab2483660e15b6ee6719402f14aa287f008
RRP	libraries/RevertReasonParser.sol	91311b20562ffccdef5459a345671a9be55b28bfafd07c6d9b1257afdf3fc791
LOP	LimitOrderProtocol.sol	86f81cefee9df58e2fb94cfb5f635fe2dde8db10f6c1d00ca9b7faf83e0b23ec
OMC	OrderMixin.sol	05c51bd187ba3af0a70bd8c1e1ec76e23f05587c588ee96df86911e78b730f47
ORF	OrderRFQMixin.sol	5ab37879b3fda9e37dbb78b36525e5a5b5c016c1078ef5f6b0ba08dfa603e08e

Understandings

Overview

1inch limit order protocol is a set of smart contracts that can work on any EVM-based blockchain (Ethereum, Binance Smart Chain, Polygon, etc.). Key features of the protocol are extreme flexibility and high gas efficiency that is achieved by using two different order types - regular Limit Orders and RFQ Orders.

Dependencies

There are a few depending injection contracts or addresses in the current project:

- `targets[i]`, `order.maker` , `order.makerAsset`, `order.takerAsset`, `order.receiver`, `order.allowedSender`, `token` decoded from `permit`, `interactionTarget` decoded from `order.interaction`, and `target` for the contract `OrderMixin`;
- `order.maker` , `order.makerAsset`, `order.takerAsset`, `order.allowedSender`, and `target` for the contract `OrderRFQMixin`;
- `target` for the contract `AmountCalculator`;
- `oracle`, `oracle1`, and `oracle2` for the contract `ChainlinkCalculator`;
- `IERC721 token` for the contract `ERC721Proxy`;
- `IERC721 token` for the contract `ERC721ProxySafe`;
- `IERC1155 token` for the contract `ERC1155Proxy`;
- `targets[i]` and `target` for the contract `PredicateHelper`;
- `token` for the contract `Permitable`.

We assume these contracts or addresses are valid and non-vulnerable actors and implement proper logicto collaborate with the current project.

Privileged Functions

In the contract ERC721Proxy, the role immutableOwner has the authority over the following function:

- func_602HzuS(), which transfers NFT tokens in an ERC721 contract.

In the contract ERC721ProxySafe, the role immutableOwner has the authority over the following function:

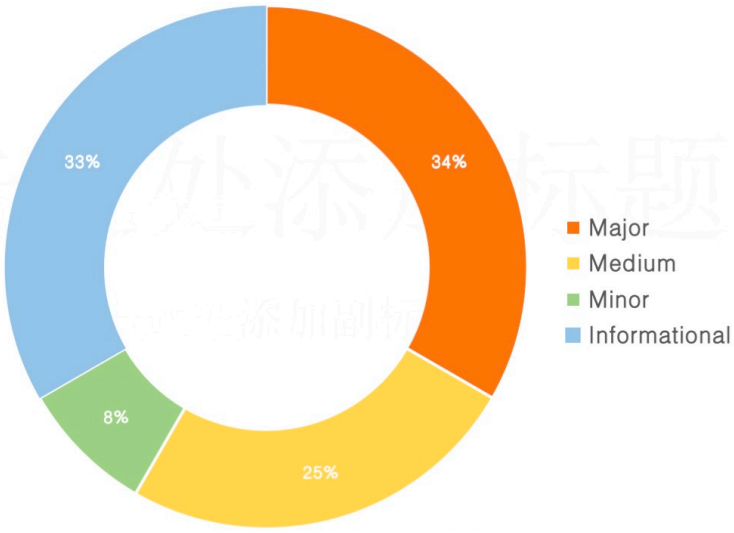
- func_602HzuS(), which transfers NFT tokens in an ERC721 contract.

In the contract ERC1155Proxy, the role immutableOwner has the authority over the following function:

- func_301JL5R() , which transfers tokens in an ERC1155 contract.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified tothe community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of the Timelock contract.

Findings



ID	Title	Category	Severity	Status
ACC-01	Users May Lose Money on Partial Order Fills	Logical Issue	Informational	Resolved
ADC-01	Functionality of decodeBool()	Logical Issue	Minor	Resolved
ERC-01	Centralization Risk	Centralization / Privilege	Major	Acknowledged
ERP-01	Centralization Risk	Centralization / Privilege	Major	Acknowledged
ERS-01	Centralization Risk	Centralization / Privilege	Major	Acknowledged
OMC-01	The _callGetter() Function Can ReturnIncorrect Values	Logical Issue	Major	Resolved
OMC-02	Incorrect Variable Used	Logical Issue	Medium	Resolved
OMC-03	Potential Reentrancy Attack	Logical Issue	Medium	Acknowledged
OMC-04	Misspelled Error Message	Coding Style	Informational	Resolved
ORF-01	Potential Reentrancy Attack	Logical Issue	Medium	Acknowledged
ORF-02	Missing Emit Events	Coding Style	Informational	Resolved
ORF-03	Gas Inefficiency of Validation	Gas Optimization	Informational	Resolved

Major

ERC-01 | Centralization Related Risks

Category	Severity	Location	Status
Centralization / Privilege	Major	helpers/ERC1155Proxy.sol: 21	Acknowledged

Description

In the contract ERC1155Proxy, the role immutableOwner has the authority over the following function:

- func_301JL5R(), which transfers tokens in an ERC1155 contract.

Any compromise to the immutableOwner account may allow the hacker to take advantage of this and gain unauthorized access to token transfers.

Recommendation

We advise the client to carefully manage the immutableOwner account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.
Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:
A time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;

Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[1inch Team]: Proxy owner will be the `LimitOrderProtocol` itself.

[CertiK]: The auditors agree that if the `immutableOwner` is the `LimitOrderProtocol` contract, there will not be risks on the `immutableOwner` account's private key. However, considering the auditors do not know if the deployment will proceed correctly, the status of this issue will be updated after contract deployment upon request.

ERP-01 | Centralization Related Risks

Category	Severity	Location	Status
Centralization / Privilege	Major	helpers/ERC721Proxy.sol: 22	Acknowledged

Description

In the contract ERC721Proxy, the role immutableOwner has the authority over the following function:

- func_602HzuS() , which transfers NFT tokens in an ERC721 contract.

Any compromise to the immutableOwner account may allow the hacker to take advantage of this and gain unauthorized access to token transfers.

Recommendation

We advise the client to carefully manage the immutableOwner account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:

A time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;

Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[1inch Team]: Proxy owner will be the `LimitOrderProtocol` itself.

[CertiK]: The auditors agree that if the `immutableOwner` is the `LimitOrderProtocol` contract, there will not be risks on the `immutableOwner` account's private key. However, considering the auditors do not know if the deployment will proceed correctly, the status of this issue will be updated after contract deployment upon request.

ERS-01 | Centralization Related Risks

Category	Severity	Location	Status
Centralization / Privilege	Major	helpers/ERC721Proxy.sol: 22	Acknowledged

Description

In the contract ERC721Proxy, the role immutableOwner has the authority over the following function:

- func_602HzuS() , which transfers NFT tokens in an ERC721 contract.

Any compromise to the immutableOwner account may allow the hacker to take advantage of this and gain unauthorized access to token transfers.

Recommendation

We advise the client to carefully manage the immutableOwner account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Here are some feasible suggestions that would also mitigate this risk in the short-term and long-term:

- A time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[1inch Team]: Proxy owner will be the `LimitOrderProtocol` itself.

[CertiK]: The auditors agree that if the `immutableOwner` is the `LimitOrderProtocol` contract, there will not be risks on the `immutableOwner` account's private key. However, considering the auditors do not know if the deployment will proceed correctly, the status of this issue will be updated after contract deployment upon request.

OMC-01 | The _callGetter() Function Can Return Incorrect Values

Category	Severity	Location	Status
Logical Issue	Major	OrderMixin.sol	Resolved

Description

The _callGetter() function on line 330 is meant to return a makerAmount when given takerAmount data and similarly return a takerAmount when given makerAmount data.
However, if getter. Length == 0, it actually returns an amount that is of the same type as the input.

```
330     function _callGetter(bytes memory getter, uint256 orderAmount, uint256 amount)
private view returns(uint256) {
331         if (getter.length == 0) {
332             // On empty getter calldata only exact amount is allowed
333             require(amount == orderAmount, "LOP: wrong amount");
334             return orderAmount;
```

Recommendation

We recommend revisiting the logic of this function.

Alleviation

The 1inch Team heeded our advice and resolved this issue by using the correct variable in commit9b5b4cfa174c2ca3e4d4b2f6b08eb73d13f2eefd.

Medium

OMC-02 | Incorrect Variable Used

Category	Severity	Location	Status
Logical Issue	Medium	OrderMixin.sol: 280	Resolved

Description

In the function fillOrderTo() , the function makes a call to order.makerAsset.

```
273         _makeCall(
274             order.makerAsset,
275             abi.encodePacked(
276                 IERC20.transferFrom.selector,
277                 uint256(uint160(order.maker)),
278                 uint256(uint160(target)),
279                 makingAmount,
280                 order.makerAsset
281             )
282         );
```

However, if we understood the Order struct correctly, the order.makerAsset at the end of the abi.encodePacked should instead be order.makerAssetData.

Recommendation

We recommend fixing the typo by changing order.makerAsset in the abi.encodePacked to order.makerAssetData .

Alleviation

The 1inch Team heeded our advice and resolved this issue by using the correct variable in commitcace3ec8f1cb58d2938417bdcfa47a4eea41d8a0.

OMC-03 | Potential Reentrancy Attack

Category	Severity	Location	Status
Logical Issue	Medium	OrderMixin.sol: 190~197	Acknowledged

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - nonReentrant modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

[1inch Team]: The only place with reentrancy is the permit call that happens when order.permit is not empty. We have reentrancy checks there. Other places follow the checks-effects-interactions pattern.

[Certik]: We agree that there is a reentrancy check on order.permit . However, the contract performs other calls, such as to order.takerAsset (line 253), order.makerAsset (line 274), andinteractionTarget (line 267). If these are malicious, they can reenter and bypass the reentrancy check on order.permit.

ORF-01 | Potential Reentrancy Attack

Category	Severity	Location	Status
Logical Issue	Medium	OrderRFQMixin.sol: 91~97	Acknowledged

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

[1inch Team]: The code already follows the checks-effects-interactions pattern.

[Certik Team]: We agree the code follows the checks-effects-interactions pattern for variable states in the contract. However, two transfers are performed on lines 144 and 145 and it is possible for the first transfer to re-enter before changes to the chain state from the second transfer.

Minor

ADC-01 | Functionality Of decodeBool()

Category	Severity	Location	Status
Logical Issue	Minor	libraries/ArgumentsDecoder.sol: 14	Resolved

Description

The decodeBool() function does not check if the bytes input is 0 or 1, but rather if it is non-zero. Any non-zero data will be decoded into true.

Alleviation

The 1inch Team resolved this issue by changing the decodeBool() function to check whether or not the decoded value is 1 in commit 908926a004af4da8c7f6966f652424500f2ac046.

Informational

ACC-01 | Users May Lose Money On Partial Order Fills

Category	Severity	Location	Status
Logical Issue	Informational	helpers/AmountCalculator.sol: 14, 20	Resolved

Description

The `getMakerAmount()` function gives the floor of `swapTakerAmount * orderMakerAmount /orderTakerAmount` , while `getTakerAmount()` gives the ceiling of `swapMakerAmount * orderTakerAmount /orderMakerAmount`.

For users that partially fill orders with small amounts, they may potentially lose money. For example, suppose `orderMakerAmount = 10` and `orderTakerAmount = 3`, so the ratio between makertokens and taker tokens is 10:3.

If a user planned to exchange 1 taker token, then they would receive `getMakerAmount(10,3,1) = 3` makertokens, due to a floor rounding, resulting in a loss of 0.33 maker tokens.

If a user instead wished to obtain 1 maker token, then they would have to pay `getTakerAmount(10,3,1) =1` taker token, due to the ceiling rounding, resulting in an overpay of 0.7 taker tokens.

Alleviation

[1inch Team]: That's intended behavior. Taker can control the rounding.

OMC-04 | Misspelled Error Message

Category	Severity	Location	Status
Coding Style	Informational	OrderMixin.sol: 202	Resolved

Description

One of the error messages in the function `fillOrderTo()` is misspelled.

```
202             require(remainingMakerAmount != 1, "LOP: remaining amoint is 0");
```

Recommendation

We recommend correcting the spelling mistake.

Alleviation

The 1inch Team heeded our advice and resolved this issue by fixing the spelling mistake in commitf6dad7b99c6864acb2236abe954159d37ce9eba3.

ORF-02 | Missing Emit Events

Category	Severity	Location	Status
Coding Style	Informational	OrderRFQMixin.sol: 45	Resolved

Description

The following function affects the status of sensitive variables and should be able to emit events as notifications:

- `OrderRFQMixin.cancelOrderRFQ()` to cancel an RFQ order.

Recommendation

We recommend emitting events for all the essential state variables that are possible to be changed during the runtime.

Alleviation

[1inch Team]: RFQ orders are intended to be used by market makers who probably have their own sophisticated order tracking tools set up. And in that case, they don't need specific event to track cancelation of their own order. Moreover, we've seen no single use of RFQ canceling so far. Thus, we think that there is no need in emitting cancel event for RFQ orders.

ORF-03 | Gas Inefficiency Of Validation

Category	Severity	Location	Status
Coding Style	Informational	OrderRFQMixin.sol: 139~141	Resolved

Description

In the function `fillOrderRFQTo()`, the function checks whether the order is valid near the end of the function call.

```
 139         require(order.allowedSender == address(0) || order.allowedSender ==
msg.sender, "LOP: private order");
 140         bytes32 orderHash =
_hashTypedDataV4(keccak256(abi.encode(LIMIT_ORDER_RFQ_TYPEHASH, order)));
 141         require(SignatureChecker.isValidSignatureNow(maker, orderHash, signature),
"LOP: bad signature");
```

It might be more optimal to locate these checks at the start of the function call so extraneous calculations are not performed.

Recommendation

We recommend moving these checks to the beginning of the function call.

Alleviation

The 1inch Team heeded our advice and resolved this issue by and have moved the validation checks to the beginning of the function in commit bf190982639e0408d0f8b1bbf4a874ff0810096c.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES ORMATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Coding Style

Coding Style findings usually do not affect the generated bytecode but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block. timestamp works.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux “sha256sum" command against the target file.

About

DeHacker is a team of auditors and white hat hackers who perform security audits and assessments. With decades of experience in security and distributed systems, our experts focus on the ins and outs of system security. Our services follow clear and prudent industry standards. Whether it's reviewing the smallest modifications or a new platform, we'll provide an in-depth security survey at every stage of your company's project. We provide comprehensive vulnerability reports and identify structural inefficiencies in smart contract code, combining high-end security research with a real-world attacker mindset to reduce risk and harden code.

BLOCKCHAIINS

Ethereum
Cosmos
Eos
Substrate

TECH STACK

Python
Solidity
Rust
c++

CONTACTS

https://dehacker.io
https://twitter.com/dehackerio
https://github.com/DeHacker-io/audits_public
https://t.me/dehackerio
https://blog.dehacker.io/

DeHacker

April 2022