Limit Order Protocol

SMART CONTRACT AUDIT

06.06.2021

Made in Germany by Chainsulting.de

Table of contents

1. Disclaimer 3
2. About the Project and Company 4
   2.1 Project Overview 5
3. Vulnerability & Risk Level 6
4. Auditing Strategy and Techniques Applied 7
   4.1 Methodology 7
   4.2 Used Code from other Frameworks/Smart Contracts 8
   4.3 Tested Contract Files 9
   4.4 Metrics / CallGraph 10
   4.6 Metrics / Capabilities 12
   4.7 Metrics / Source Unites in Scope 13
5. Scope of Work 14
   5.1 Manual and Automated Vulnerability Test 15
   5.2. SWC Attacks 16
6. Executive Summary 20
7. Deployed Smart Contract 20

1. Disclaimer

The audit makes no statements or warrantees 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 Exchange. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (01.04.2021)	Layout
0.5 (04.04.2021)	Unit tests
0.6 (05.04.2021)	Testing SWC Checks
0.8 (07.04.2021)	Automated Security Testing Manual Security Testing
0.9 (08.04.2021)	Summary and Recommendation
1.0 (10.04.2021)	Final document
1.2 (06.06.2021)	Re-check after updates
1.3 (TBA)	Added deployed contract

2. About the Project and Company

Company address:

1Inch Limited
Quijano Chambers, P.O. Box 3159, Road Town
Tortola, British Virgin Islands

Sergej Kunz Co-Founder & Chief Executive Officer
Anton Bukov Co-Founder & Chief Technology Officer

Discord: https://discord.gg/FZADkCZ

Blog: https://blog.1inch.io

Medium: https://medium.com/@1inch.exchange

Website: https://app.1inch.io

Twitter: https://twitter.com/1inchExchange

Reddit: https://www.reddit.com/r/1inch_exchange

Telegram: https://t.me/OneInchExchange

Forum: https://gov.1inch.io

2.1 Project Overview

1inch is a so-called DEX aggregator, which means that it scrapes a handful of decentralized exchanges for the cheapest prices and reroutes its customers' trades between them to try and ensure that they're getting the best prices. This is particularly important when exchanging large token amounts as it reduces the price slippage, ensuring trades are optimized for the best price.

1inch was founded by Sergej Kunz and Anton Bukov in 2019 during ETHNewYork's hackathon. Since then, 1inch has raised about $15 million in funding from companies such as Binance Labs, Galaxy Digital and Pantera Capital. As of January 2021, 1inch's exchange trades about $155 million a day.

3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
   i. Review of the specifications, sources, and instructions provided to Chainsulting to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Chainsulting describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

4.2 Used Code from other Frameworks/Smart Contracts (direct imports)

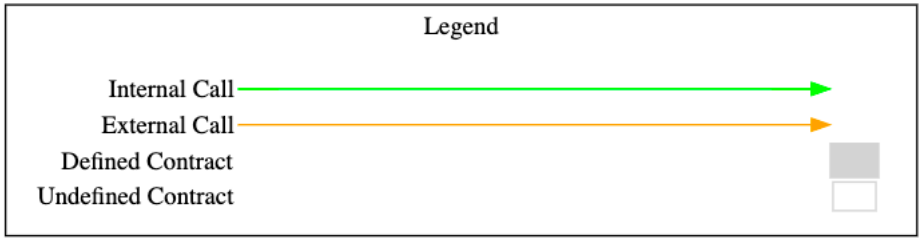
Dependency / Import Path	Source
@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/extensions/draft-ERC20Permit.sol
@openzeppelin/contracts/utils/math/SafeMath.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/math/SafeMath.sol
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/utils/SafeERC20.sol
@openzeppelin/contracts/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/IERC20.sol
@openzeppelin/contracts/token/ERC721/IERC721.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/IERC721.sol
@openzeppelin/contracts/token/ERC1155/IERC1155.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC1155/IERC1155.sol

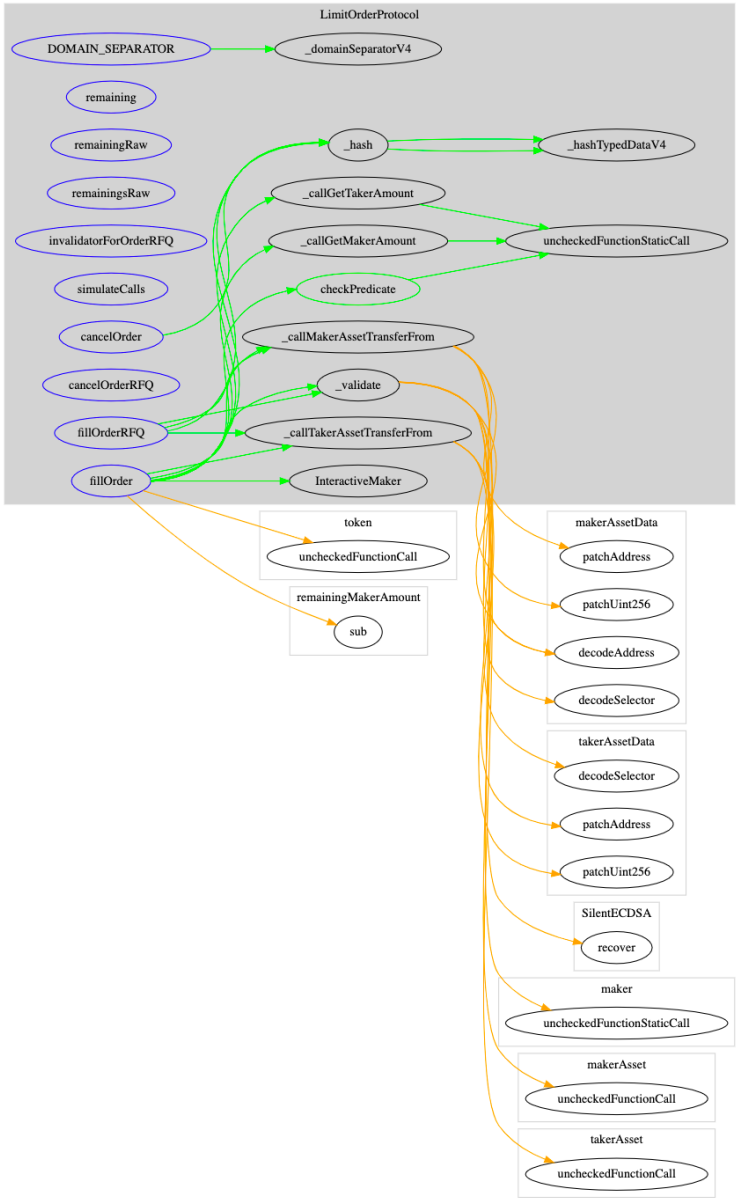
4.3 Tested Contract Files

The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

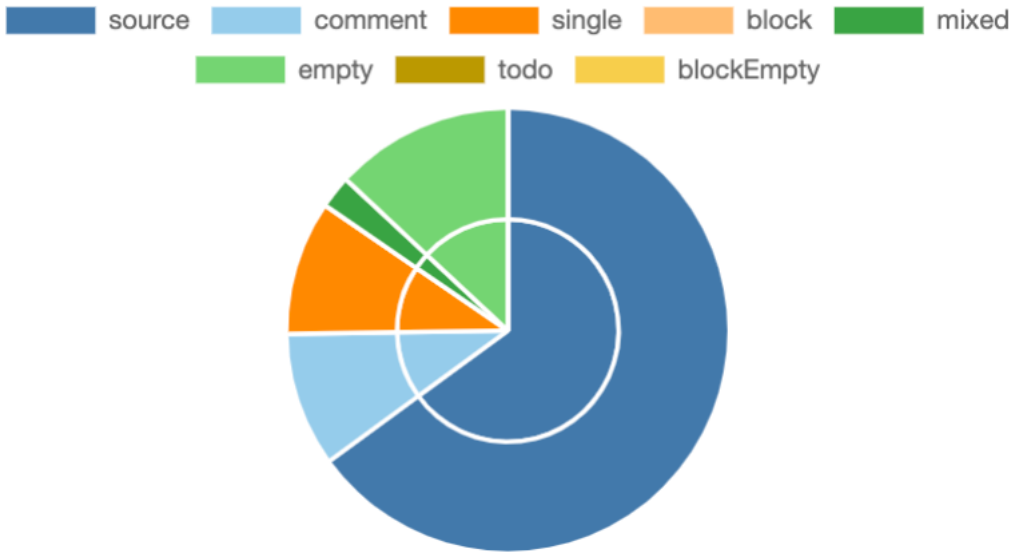
File	Fingerprint (MD5)
LimitOrderProtocol.sol	c3a67acd9bb129885c03bc19718143cb
./libraries/ArgumentsDecoder.sol	0991a750f78c14fdc01d43e18f0f6d5b
./libraries/UncheckedAddress.sol	a5f723dad51a7046470ffdf5df6a3d13
./libraries/SilentECDSA.sol	ede3aec76ededc4d8ac5b7074e5b4e08
./interfaces/AggregatorV3Interface.sol	1140ecccd2cb513c243a1c5e144f146a
./interfaces/InteractiveMaker.sol	5ebc6825022d4839d7f040a099440800
./helpers/AmountCalculator.sol	e9129d2d02616c95372389ed1efa17ff
./helpers/ChainlinkCalculator.sol	e8b539759b8e682732c29ae5c90a56b7
./helpers/ERC20Proxy.sol	ba24c0eff24709a8a7ae767f917021b7
./helpers/ERC721Proxy.sol	c5d76a8093b357525504ca4a3305ea61
./helpers/ERC1155Proxy.sol	870764e5a5e8c5c40ae1949a26600b40
./helpers/ImmutableOwner.sol	f7a07545b7fd3a6e1e0f81d70058018e
./helpers/NonceManager.sol	245ac5d6bf34343d1b367d3fc61c5fbc
./helpers/PredicateHelper.sol	b29178b094ba4428212159c936647f17

4.4 Metrics / CallGraph





4.5 Metrics / Source Lines



4.6 Metrics / Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
`^0.8.0`			**** (0 asm blocks)	

📩 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧱 Uses Hash Functions	🪚 ECRecover	🌀 New/Create/Create2
			`yes`		

🌐Public	💰Payable
11	0

External	Internal	Private	Pure	View
10	14	0	0	11

StateVariables

Total	🌐Public
8	2

4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	contracts/LimitOrderProtocol.sol	1		359	359	273	41	221	
	Totals	**1**		**359**	**359**	**273**	**41**	**221**	

Legend: [-]

- **Lines**: total lines of the source unit
- **nLines**: normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC**: normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines**: lines containing single or block comments
- **Complexity Score**: a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

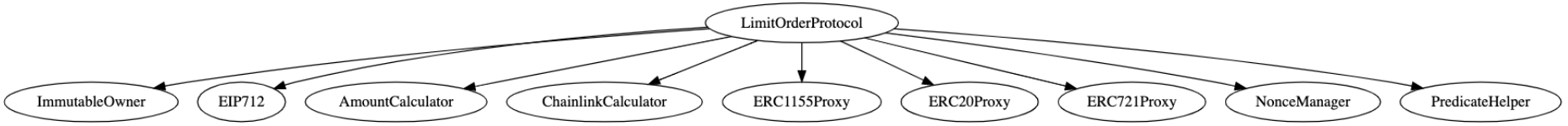
5. Scope of Work

The 1inch Team provided us with the files that needs to be tested. The scope of the audit is the Limit Order Protocol contract.

Following contracts with the direct imports has been tested:

- LimitOrderProtocol.sol

The main goal of this audit was to verify the overall smart contract security. The auditors can provide additional feedback on the code upon the client's request.



5.1 Manual and Automated Vulnerability Test

CRITICAL ISSUES
During the audit, Chainsulting‘s experts found **no Critical issues** in the code of the smart contract.

HIGH ISSUES
During the audit, Chainsulting’s experts found **no High issues** in the code of the smart contract.

MEDIUM ISSUES
During the audit, Chainsulting’s experts found **no Medium issues** in the code of the smart contract.

LOW ISSUES
During the audit, Chainsulting’s experts found **no Low issues** in the code of the smart contract.

5.2. SWC Attacks

ID	Title	Relationships	Test Result
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	✅
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	✅
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	✅
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	✅
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	✅
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	✅
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	✅
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	✅

ID	Title	Relationships	Test Result
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	✅
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	✅
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	✅
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	✅
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	✅
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	✅
SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✅
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	✅
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	✅

ID	Title	Relationships	Test Result
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	✅
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✅
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	✅
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	✅
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	✅
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	✅
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	✅
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	✅
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	✅
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	✅

ID	Title	Relationships	Test Result
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	X
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	✅
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	✅
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	✅

6. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase. The overall code quality of the project is very good. It implemented the newest versions of the widely-used and reviewed contracts from OpenZeppelin.

The main goal of the audit was to verify the claims regarding the security of the smart contract. During the audit, no critical issues were found, after the manual and automated security testing. The auditors have been very satisfied with the documentation and overall code quality.

7. Deployed Smart Contract

PENDING

Smart Contract is deployed here:

0xFFF