



SMART CONTRACT AUDIT REPORT

for

1inch Settlement Extension



Prepared By: Xiaomi Huang

PeckShield
December 12, 2023

Document Properties

Client	1inch Protocol
Title	Smart Contract Audit Report
Target	1inch Settlement Extension
Version	1.0
Author	Xuxian Jiang
Auditors	Colin Zhong, Xuxian Jiang
Reviewed by	Xiaomi Huang
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	December 12, 2023	Xuxian Jiang	Final Release
1.0-rc	November 27, 2023	Xuxian Jiang	Release Candidate

Contact

For more information about this document and its contents, please contact PeckShield Inc.

Name	Xiaomi Huang
Phone	+86 183 5897 7782
Email	contact@peckshield.com

Contents

1	Introduction	4
1.1	About 1inch Limit Order Settlement	4
1.2	About PeckShield	5
1.3	Methodology	5
1.4	Disclaimer	7
2	Findings	9
2.1	Summary	9
2.2	Key Findings	10
3	Detailed Results	11
3.1	Improved Validation on Integration Fee Distribution	11
4	Conclusion	13
	References	14

1 | Introduction

Given the opportunity to review the design document and revised smart contract source code of the `1inch Limit Order Settlement` protocol, we outline in the report our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between smart contract code and design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts is well designed and engineered, though it can be further improved by addressing our suggestions. This document outlines our audit results.

1.1 About 1inch Limit Order Settlement

The `1inch Limit Order` protocol facilitates the innovative and flexible limit order swap opportunities in DeFi. The key features of the protocol is extreme flexibility and high gas efficiency that is achieved by using two different order types - regular `Limit Order` and `RFQ Order`. The audited `SettlementExtension` contract acts as an extension to current `Settlement` support and replaces the `Fusion` settlement contract. The basic information of the audited protocol is as follows:

Table 1.1: Basic Information of 1inch Settlement Extension

Item	Description
Name	1inch Protocol
Website	https://app.1inch.io/
Type	EVM Smart Contract
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	December 12, 2023

In the following, we show the Git repositories of reviewed files and the commit hash value used in this audit. Note this audit only covers the following contract – `SettlementExtension.sol`.

- <https://github.com/1inch/limit-order-settlement.git> (ff7909c)

1.2 About PeckShield

PeckShield Inc. [5] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystems by offering top-notch, industry-leading services and products (including the service of smart contract auditing). We are reachable at Telegram (<https://t.me/peckshield>), Twitter (<http://twitter.com/peckshield>), or Email (contact@peckshield.com).

Table 1.2: Vulnerability Severity Classification

Impact	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low
		Likelihood		

1.3 Methodology

To standardize the evaluation, we define the following terminology based on the OWASP Risk Rating Methodology [4]:

- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact measures the technical loss and business damage of a successful attack;
- Severity demonstrates the overall criticality of the risk.

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact and can be classified into four categories accordingly, i.e., *Critical*, *High*, *Medium*, *Low* shown in Table 1.2.

To evaluate the risk, we go through a checklist of items and each would be labeled with a severity category. For one check item, if our tool or analysis does not identify any issue, the contract is considered safe regarding the check item. For any discovered issue, we might further deploy contracts on our private testnet and run tests to confirm the findings. If necessary, we would

Table 1.3: The Full Audit Checklist

Category	Checklist Items
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead Of Transfer
	Costly Loop
	(Unsafe) Use Of Untrusted Libraries
	(Unsafe) Use Of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
Semantic Consistency Checks	Semantic Consistency Checks
Advanced DeFi Scrutiny	Business Logics Review
	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.3.

In particular, we perform the audit according to the following procedure:

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.
- Semantic Consistency Checks: We then manually check the logic of implemented smart contracts and compare with the description in the white paper.
- Advanced DeFi Scrutiny: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [3], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development. Though some categories used in CWE-699 may not be relevant in smart contracts, we use the CWE categories in Table 1.4 to classify our findings. Moreover, in case there is an issue that may affect an active protocol that has been deployed, the public version of this report may omit such issue, but will be amended with full details right after the affected protocol is upgraded with respective fixes.

1.4 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Last but not least, this security audit should not be used as investment advice.


Table 1.4: Common Weakness Enumeration (CWE) Classifications Used in This Audit

Category	Summary
Configuration	Weaknesses in this category are typically introduced during the configuration of the software.
Data Processing Issues	Weaknesses in this category are typically found in functionality that processes data.
Numeric Errors	Weaknesses in this category are related to improper calculation or conversion of numbers.
Security Features	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.)
Time and State	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.
Error Conditions, Return Values, Status Codes	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function.
Resource Management	Weaknesses in this category are related to improper management of system resources.
Behavioral Issues	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
Business Logic	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.
Initialization and Cleanup	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
Arguments and Parameters	Weaknesses in this category are related to improper use of arguments or parameters within function calls.
Expression Issues	Weaknesses in this category are related to incorrectly written expressions within code.
Coding Practices	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an exploitable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained.

2 | Findings

2.1 Summary

Here is a summary of our findings after analyzing the implementation of the `SettlementExtension` smart contract. During the first phase of our audit, we study the smart contract source code and run our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by our tool. We further manually review business logic, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	# of Findings	
Critical	0	
High	0	
Medium	0	
Low	0	
Informational	1	
Total	3	

We have so far identified a list of potential issues: some of them involve subtle corner cases that might not be previously thought of, while others refer to unusual interactions among multiple contracts. For each uncovered issue, we have therefore developed test cases for reasoning, reproduction, and/or verification. After further analysis and internal discussion, we determined a few issues of varying severities need to be brought up and paid more attention to, which are categorized in the above table. More information can be found in the next subsection, and the detailed discussions of each of them are in [Section 3](#).

2.2 Key Findings

Overall, these smart contracts are well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 1 informational recommendation.

Table 2.1: Key 1inch Settlement Extension Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Informational	Improved Validation on Integration Fee Distribution	Coding Practices	Resolved

Beside the identified issues, we emphasize that for any user-facing applications and services, it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms should kick in at the very moment when the contracts are being deployed on mainnet. Please refer to Section 3 for details.



3 | Detailed Results

3.1 Improved Validation on Integration Fee Distribution

- ID: PVE-001
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: SettlementExtension
- Category: Coding Practices [2]
- CWE subcategory: CWE-563 [1]

Description

The SettlementExtension contract has a postInteraction() callback that will be invoked by the main protocol to collect associated fee, including resolver and integrator. While analyzing the integrator fee collection, we notice the current implementation can be improved.

In the following, we show the related code snippet from the postInteraction() routine. We notice the integration fee is sent to integrator on the condition that the integrationFee is positive (line 128). This condition can be improved by additionally validating the recipient integrator is not empty, i.e., `if (integrationFee > 0 and integrator != address(0))`.

```

112     function postInteraction(
113         IOrderMixin.Order calldata order,
114         bytes calldata /* extension */,
115         bytes32 /* orderHash */,
116         address taker,
117         uint256 makingAmount,
118         uint256 takingAmount,
119         uint256 /* remainingMakingAmount */,
120         bytes calldata extraData
121     ) external onlyLimitOrderProtocol {
122         (uint256 resolverFee, address integrator, uint256 integrationFee, bytes calldata
            whitelist) = _parseFeeData(extraData, order.makingAmount, makingAmount,
            takingAmount);

124         if (!_isWhitelisted(whitelist, taker)) revert ResolverIsNotWhitelisted();
125         if (!_isPriorityFeeValid()) revert InvalidPriorityFee();

```

```
127     _chargeFee(taker, resolverFee);
128     if (integrationFee > 0) {
129         IERC20(order.takerAsset.get()).safeTransferFrom(taker, integrator,
130             integrationFee);
131     }
```

Listing 3.1: SettlementExtension::postInteraction()

Recommendation Strengthen the `postInteraction()` routine to have necessary sanity checks on the integrator address.

Status



4 | Conclusion

In this audit, we have analyzed the design and implementation of the revised `1inch Limit Order Settlement` protocol. The `1inch Limit Order` protocol facilitates the innovative and flexible limit order swap opportunities in DeFi. The key features of the protocol is extreme flexibility and high gas efficiency. The audited `SettlementExtension` contract acts as an extension to current `Settlement` support and replaces the `Fusion` settlement contract. The current code base is well structured and neatly organized. Those identified issues are promptly confirmed and addressed.

Moreover, we need to emphasize that [Solidity](#)-based smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.



References

- [1] MITRE. CWE-563: Assignment to Variable without Use. <https://cwe.mitre.org/data/definitions/563.html>.
- [2] MITRE. CWE CATEGORY: Bad Coding Practices. <https://cwe.mitre.org/data/definitions/1006.html>.
- [3] MITRE. CWE VIEW: Development Concepts. <https://cwe.mitre.org/data/definitions/699.html>.
- [4] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.
- [5] PeckShield. PeckShield Inc. <https://www.peckshield.com>.