Report

v. 1.0

Customer

1inch

Smart Contract Audit

Limit Order Settlement. Phase II

19th June 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	19.06.23	A. Zveryanskaya	Initial Draft
0.2	19.06.23	A. Zveryanskaya	Minor revision
1.0	19.06.23	A. Zveryanskaya	Release

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

The 1inch Network unites decentralized protocols whose synergy enables the most lucrative, fastest and protected operations in the DeFi space.

3 Project scope

We were asked to review:

- New functionality as a diff to the code
- After-audit fixes

Files:

contract/		
Settlement.sol	FeeBank.sol	FeeBankCharger.sol
DynamicSuffix.sol	FusionDetails.sol	

4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places as well as their visibility scopes and access levels are relevant. At this phase, we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and done properly. At this phase, we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if proper data types are used. We also make sure that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Minor issues** contain code style, best practices and other recommendations.

5 Our findings

We found 3 major, and a few less important issues.

	Active	Fixed
Major	2	1
Moderate	2	9
Minor	10	2

Fixed 12 out of 26 issues

6 Major Issues

CVF-1. INFO

- **Category** Unclear behavior
- **Source** FusionDetails.sol

Recommendation Greater than or equal would be more appropriate here. Also, it would make the code simpler, as “iszero” opcode below wouldn’t be necessary anymore.

```
145 +valid := gt(timestamp(), add(startTime, publicTimeDelay))
```

CVF-2. FIXED

- **Category** Procedural
- **Source** FusionDetails.sol

Description The later “calldataload” opcode is redundant, as the action point bump was already read by the former “calldataload”.

Recommendation Consider extracting the action point bump from the already loaded word.

```
211 +let bump := shr(_AUCTION_POINT_BUMP_BIT_SHIFT, calldataload(ptr))
```

```
213 +let delay := shr(_AUCTION_POINT_DELTA_BIT_SHIFT, calldataload(ptr))
```

CVF-3. INFO

- **Category** Procedural
- **Source** FeeBankCharger.sol

Description Preventing low-level technical errors, such as overflows, via high-level business constraints is a bad practice, as such relations are non obvious and could be broken unintentionally by changes or bugs in business logic.

Recommendation Consider using checked math unless a reason why over-/underflow is impossible is obvious from the nearby code.

```
32 +allowance += amount;  // overflow is impossible due to limited
   ↪ _token supply
```

7 Moderate Issues

CVF-4. FIXED

- **Category** Overflow/Underflow
- **Source** DynamicSuffix.sol

Description Underflow is possible here.

Recommendation Consider performing proper length checks for “cd”.

```
28 +args.length := sub(suffix, args.offset)
```

CVF-5. FIXED

- **Category** Flaw
- **Source** FusionDetails.sol

Description There is no length check for “interaction”, so here data before “interaction” could be loaded from the calldata.

Recommendation Consider adding appropriate length check.

```
95 +let ptr := sub(add(details.offset, details.length),
      ↪ _TAKING_FEE_BYTES_SIZE)
96 +ret := shr(_TAKING_FEE_BIT_SHIFT, calldataload(ptr))
```

CVF-6. FIXED

- **Category** Flaw
- **Source** FusionDetails.sol

Description There is no check to ensure that ”addressPtr” calculated here points inside ”interaction”.

Recommendation Consider adding an appropriate check.

```
106 +let addressPtr := sub(add(interaction.offset, interaction.length),
      ↪ 1)
107 +addressPtr := sub(addressPtr, mul(_RESOLVER_ADDRESS_BYTES_SIZE,
      ↪ byte(0, calldataload(addressPtr))))
```

CVF-7. FIXED

- **Category** Flaw
- **Source** FusionDetails.sol

Description There is no check to ensure that calldata offset calculated here points inside ”interaction”.

Recommendation Consider adding an appropriate check.

```
120 +let resolverRaw := calldataload(add(addressPtr, mul(resolverIndex,
       ↪ _RESOLVER_ADDRESS_BYTES_SIZE)))
```

CVF-8. FIXED

- **Category** Flaw
- **Source** FusionDetails.sol

Description There is no length check for “details”, so here data before “details” could be loaded from the calldata.

Recommendation Consider adding appropriate length check.

```
139 +let flags := byte(0, calldataload(details.offset))
```

CVF-9. FIXED

- **Category** Flaw
- **Source** FusionDetails.sol

Description There is no length check for “details”, so here data before “details” could be loaded from the calldata.

Recommendation Consider adding appropriate length check.

```
143 +let startTime := shr(_START_TIME_BIT_SHIFT, calldataload(add(
       ↪ details.offset, _START_TIME_BYTES_OFFSET)))
```

```
154 +        let resolverIndex := byte(0, calldataload(ptr))
```

CVF-10. FIXED

- **Category** Flaw
- **Source** FusionDetails.sol

Description There is no check to ensure that ”addressPtr” calculated here points inside ”interaction”.

Recommendation Consider adding an appropriate check.

```
151 +let addressPtr := sub(add(interaction.offset, interaction.length),
       ↪ 1)
152 +addressPtr := sub(addressPtr, mul(_RESOLVER_ADDRESS_BYTES_SIZE,
       ↪ byte(0, calldataload(addressPtr))))
```

CVF-11. FIXED

- **Category** Flaw
- **Source** FusionDetails.sol

Description There is no check to ensure that calldata offset calculated here points inside ”interaction”.

Recommendation Consider adding an appropriate check.

```
159 +let account := shr(_RESOLVER_ADDRESS_BIT_SHIFT, calldataload(add(
       ↪ addressPtr, mul(resolverIndex, _RESOLVER_ADDRESS_BYTES_SIZE)))
       ↪ )
```

CVF-12. FIXED

- **Category** Flaw
- **Source** FusionDetails.sol

Description There is no length check for “details”, so here data before “details” could be loaded from the calldata.

Recommendation Consider adding appropriate length check.

```
174 +startBump := shr(_INITIAL_RATE_BUMP_BIT_SHIFT, calldataload(add(
       ↪ details.offset, _INITIAL_RATE_BUMP_BYTES_OFFSET)))
```

```
176 +    shr(_START_TIME_BIT_SHIFT, calldataload(add(details.offset,
       ↪ _START_TIME_BYTES_OFFSET))),
177 +    shr(_AUCTION_DELAY_BIT_SHIFT, calldataload(add(details.offset,
       ↪ _AUCTION_DELAY_BYTES_OFFSET)))
```

```
179 +auctionFinishTime := add(auctionStartTime, shr(
       ↪ _AUCTION_DURATION_BIT_SHIFT, calldataload(add(details.offset,
       ↪ _AUCTION_DURATION_BYTES_OFFSET))))
```

```
200 +    let flags := byte(0, calldataload(details.offset))
```

```
247 +fee := shr(_RESOLVER_FEE_BIT_SHIFT, calldataload(add(details.offset
       ↪ , _RESOLVER_FEE_BYTES_OFFSET)))
```

CVF-13. INFO

- **Category** Overflow/Underflow
- **Source** FusionDetails.sol

Description Overflow, underflow, and division by zero are possible here.

Recommendation Consider using checked math.

Client Comment *Overflow, underflow and division by zero is impossible because of the way the function is used. v1 and v2 values are small, t is always between t1 and t2. Division by zero will not happen because of the early returns before the assembly block.*

```
190 +v := div(
191 +    add(mul(sub(t, t1), v2), mul(sub(t2, t), v1)),
192 +    sub(t2, t1)
193 +)
```

CVF-14. INFO

- **Category** Flaw
- **Source** Settlement.sol

Description This could read after “extraData”.

Recommendation Consider adding proper length checks.

Client Comment *Won’t fix. tokensAndAmounts and all the remainder of the suffix is created by the same contract that is parsing it.*

```
79 +let ptr := add(allTokensAndAmounts, 0x20)
```

8 Minor Issues

CVF-15. INFO

- **Category** Procedural
- **Source** DynamicSuffix.sol

Recommendation Specifying as “^0.8.0” would make this change unnecessary.

```
3  -pragma solidity 0.8.17;
3  +pragma solidity 0.8.19;
```

CVF-16. INFO

- **Category** Procedural
- **Source** DynamicSuffix.sol

Description We didn’t review this file.

```
5  +import "@1inch/solidity-utils/contracts/libraries/AddressLib.sol";
```

CVF-17. INFO

- **Category** Procedural
- **Source** FusionDetails.sol

Description Specifying as particular version makes it harder to migrate to newer versions.

Recommendation Consider specifying as “^0.8.0”.

```
3  +pragma solidity 0.8.19;
```

CVF-18. INFO

- **Category** Procedural
- **Source** FusionDetails.sol

Description We didn’t review this file.

```
5  +import "@1inch/solidity-utils/contracts/libraries/AddressLib.sol";
```

CVF-19. FIXED

- **Category** Suboptimal
- **Source** FusionDetails.sol

Recommendation This could be optimized as: mul(24, shr (7, flags))

```
86  +mul(_TAKING_FEE_BYTES_SIZE, iszero(iszero(and(_HAS_TAKING_FEE_FLAG,
        ↪  flags))))
```

CVF-20. INFO

- **Category** Procedural
- **Source** Settlement.sol

Recommendation Specifying as "^0.8.0" would make this change unnecessary.

```
3   -pragma solidity 0.8.17;
3   +pragma solidity 0.8.19;
```

CVF-21. INFO

- **Category** Unclear behavior
- **Source** Settlement.sol

Description The function always returns true and the semantics of the returned value is unclear.

Recommendation Consider explaining the semantics of the returned value or returning nothing.

Client Comment *Solidity does not do extcodesize before the call if function returns something.*

```
51  +function settleOrders(bytes calldata data) external returns(bool) {
```

```
53  +    return true;
```

CVF-22. FIXED

- **Category** Unclear behavior
- **Source** Settlement.sol

Description The condition “interaction.length < 20” condition is always true as otherwise slicing would fail.

```
149 +bytes calldata fusionDetails = interaction[21:];
```

```
152 +if (interaction.length < 20 || address(bytes20(interaction)) !=
    ↪ address(this)) revert WrongInteractionTarget();
```

CVF-23. INFO

- **Category** Procedural
- **Source** FeeBankCharger.sol

Recommendation Specifying as “^0.8.0” would make this change unnecessary.

```
3  -pragma solidity 0.8.17;
3  +pragma solidity 0.8.19;
```

CVF-24. INFO

- **Category** Procedural
- **Source** FeeBank.sol

Recommendation Specifying as “^0.8.0” would make this change unnecessary.

```
3  -pragma solidity 0.8.17;
3  +pragma solidity 0.8.19;
```

CVF-25. INFO

- **Category** Bad naming
- **Source** FeeBank.sol

Description This error gives no clue regarding which particular address is zero.

Recommendation Consider replacing with three errors: ”ZeroCharger”, ”Zero1Inch”, and ”ZeroAccount”.

```
15 +error ZeroAddress();
```

CVF-26. INFO

- **Category** Suboptimal
- **Source** FeeBank.sol

Recommendation These checks are redundant, as it is anyway possible to pass dead addresses. Forbiding zero address does make sense only when the zero address is used as a special value.

```
23  +if (address(charger_) == address(0)) revert ZeroAddress();
24  +if (address(inch_) == address(0)) revert ZeroAddress();
```

```
114 +if (account == address(0)) revert ZeroAddress();
```

ABDK
Consulting

About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting