# ABDK CONSULTING

SMART CONTRACT
AUDIT

## 1inch

## OrderMixin

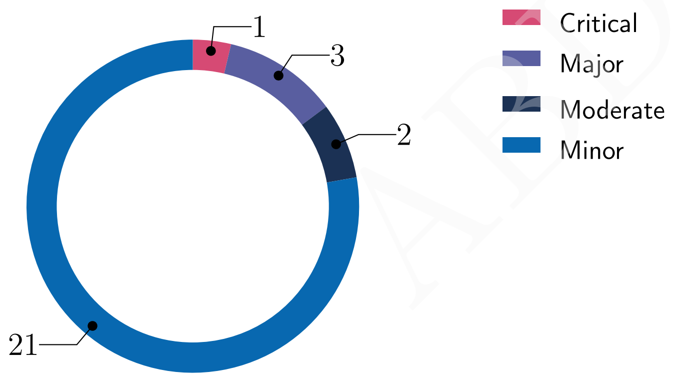
Solidity

abdk.consulting

# SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
12th November 2021

We've been asked to review the 10 files in a Github repo. We found 1 critical, 3 major, and a few less important issues. One critical and one major issues were fixed.



## Findings

| ID | Severity | Category | Status |
|---|---|---|---|
| CVF-1 | Minor | Suboptimal | Info |
| CVF-2 | Minor | Suboptimal | Info |
| CVF-3 | Minor | Bad naming | Info |
| CVF-4 | Minor | Procedural | Info |
| CVF-5 | Minor | Suboptimal | Info |
| CVF-6 | Minor | Suboptimal | Info |
| CVF-7 | Minor | Documentation | Info |
| CVF-8 | Minor | Suboptimal | Info |
| CVF-9 | Minor | Suboptimal | Info |
| CVF-10 | Minor | Bad naming | Fixed |
| CVF-11 | Minor | Suboptimal | Info |
| CVF-12 | Critical | Flaw | Fixed |
| CVF-13 | Minor | Suboptimal | Fixed |
| CVF-14 | Minor | Readability | Fixed |
| CVF-15 | Minor | Procedural | Info |
| CVF-16 | Minor | Procedural | Info |
| CVF-17 | Minor | Documentation | Info |
| CVF-18 | Minor | Procedural | Info |
| CVF-19 | Moderate | Flaw | Fixed |
| CVF-20 | Minor | Bad datatype | Fixed |
| CVF-21 | Minor | Suboptimal | Fixed |
| CVF-22 | Minor | Suboptimal | Info |
| CVF-23 | Major | Procedural | Info |
| CVF-24 | Major | Flaw | Info |
| CVF-25 | Major | Flaw | Fixed |
| CVF-26 | Moderate | Flaw | Fixed |
| CVF-27 | Minor | Flaw | Fixed |

# Contents

# 1 Document properties

## Version

| Version | Date | Author | Description |
|---|---|---|---|
| 0.1 | November 12, 2021 | D. Khovratovich | Initial Draft |
| 0.2 | November 12, 2021 | D. Khovratovich | Minor revision |
| 1.0 | November 12, 2021 | D. Khovratovich | Release |

## Contact

D. Khovratovich

khovratovich@gmail.com

# 2 Introduction

The following document provides the result of the audit performed by ABDK Consulting 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. We have reviewed the contracts at audit tag:

- LimitOrderProtocol.sol
- OrderMixin.sol
- OrderRFQMixin.sol
- helpers/AmountCalculator.sol
- helpers/ChainlinkCalculator.sol
- helpers/NonceManager.sol
- helpers/PredicateHelper.sol
- libraries/ArgumentDecoder.sol
- libraries/Permitable.sol
- libraries/RevertReasonParser.sol

The fixes were provided in commit f110eaa97e.

## 2.1 About ABDK

ABDK Consulting, 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.

## 2.2 Disclaimer

Note that the performed audit represents current best practices and smart contract standards which are relevant at the date of publication. After fixing the indicated issues the smart contracts should be re-audited.

## 2.3 Methodology

The methodology is not a strict formal procedure, but rather a collection of methods and tactics that combined differently and tuned for every particular project, depending on the project structure and and used technologies, as well as on what the client is expecting from the audit. In current audit we use:

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows code best 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 and that 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 is 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 that code actually does what it is supposed to do, that algorithms are optimal and correct, and that proper data types are used. We also check 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.

# 3 Detailed Results

## 3.1 CVF-1

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** Permitable.sol

**Description** This function is redundant as it anyway reads the whole "permit" bytes array into the memory.
**Recommendation** Consider just using the "_permitMemory" function instead.
**Client Comment** Won't fix. It results in 900 gas overhead.

Listing 1:

```
+function _permit(address token, bytes calldata permit) internal
    ↪ {
```

## 3.2 CVF-2

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** Permitable.sol

**Recommendation** Thus code could be simplified: bytes4 selector; if (permit.length == 32 * 7) selector = IERC20Permit.permit.selector; else if (permit.length == 32 * 8) selector = IDaiLikePermit.permit.selector; else revert ("Wrong permit length"); (bool success, bytes memory result) = token.call(abi.encodePacked (selector, permit));

Listing 2:

```
+bool success;
+bytes memory result;
+if (permit.length == 32 * 7) {
+    // solhint-disable-next-line avoid-low-level-calls
+    (success, result) = token.call(abi.encodePacked(
   ↪ IERC20Permit.permit.selector, permit));
+} else if (permit.length == 32 * 8) {
+    // solhint-disable-next-line avoid-low-level-calls
+    (success, result) = token.call(abi.encodePacked(
   ↪ IDaiLikePermit.permit.selector, permit));
+} else {
+    revert("Wrong permit length");
+}

+bool success;
+bytes memory result;
+if (permit.length == 32 * 7) {
+    // solhint-disable-next-line avoid-low-level-calls
+    (success, result) = token.call(abi.encodePacked(
   ↪ IERC20Permit.permit.selector, permit));
+} else if (permit.length == 32 * 8) {
+    // solhint-disable-next-line avoid-low-level-calls
+    (success, result) = token.call(abi.encodePacked(
   ↪ IDaiLikePermit.permit.selector, permit));
+} else {
+    revert("Wrong permit length");
+}
```

## 3.3 CVF-3

- **Severity** Minor
- **Category** Bad naming
- **Status** Info
- **Source** OrderMixin.sol

**Recommendation** Events are usually named via nouns, such as "OrderFill" and "OrderCancellation", or just "Fill" and "Cancel".

Listing 3:

```
+event OrderFilled(

+event OrderCanceled(
```

## 3.4 CVF-4

- **Severity** Minor
- **Category** Procedural
- **Status** Info
- **Source** OrderMixin.sol

**Recommendation** These parameters should probably be indexed.

Listing 4:

```
+bytes32 orderHash,

+bytes32 orderHash
```

## 3.5 CVF-5

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** OrderMixin.sol

**Recommendation** This function wouldn't be necessary if the "_remaining" mapping would be public.

Listing 5:

```
+function remainingRaw(bytes32 orderHash) external view returns(
    ↪ uint256) {
```

## 3.6 CVF-6

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** OrderMixin.sol

**Recommendation** It would be cheaper to return a bit mask.
**Client Comment** Won't fix. This is view only function, so we don't care that much about gas usage here.

Listing 6:

```
104 +* @notice Calls every target with corresponding data. Then
    ↪ reverts with CALL_RESULTS_0101011 where zeroes and ones
```

## 3.7 CVF-7

- **Severity** Minor
- **Category** Documentation
- **Status** Info
- **Source** OrderMixin.sol

**Recommendation** It would be good to explain what is "success" and what is "failure" here, as the function not only analyzes the execution status of a call but also its returned value.

Listing 7:

```
104 +* @notice Calls every target with corresponding data. Then
    ↪ reverts with CALL_RESULTS_0101011 where zeroes and ones
    +* denote failure or success of the corresponding call
```

## 3.8 CVF-8

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** OrderMixin.sol

**Description** The called contract is allowed to spend all the available gas. Thus, if some call will fail by spending all the gas, e.g. by executing an invalid bytecode, other targets will not be tested.
**Recommendation** Consider limiting the gas amount for calls.
**Client Comment** Won't fix. Same as in 6. We don't care that much about gas usage here.

Listing 8:

```
114 +(bool success, bytes memory result) = targets[i].call(data[i]);
```

## 3.9 CVF-9

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** OrderMixin.sol

**Description** This checks makes it redundant to pass the "order.maker" field within function arguments, as its value could be derived from "msg.sender".
**Recommendation** Consider allowing not to pass this field at all (this would require a separate structure definition) or to pass zero inside this field to make transaction cheaper.

Listing 9:

```
127 +require(order.maker == msg.sender, "LOP: Access denied");
```

## 3.10 CVF-10

- **Severity** Minor
- **Category** Bad naming
- **Status** Fixed
- **Source** OrderMixin.sol

**Description** The semantics of the returned values is not documented.
**Recommendation** Consider giving descriptive names to the returned values and/or explaining them in the documentation comment.

Listing 10:

```
147 +) external returns(uint256, uint256) {
```

## 3.11 CVF-11

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** OrderMixin.sol

**Description** This kind of reentrancy protection is error-prone as it relies on business-level state changes.
**Recommendation** Consider using generic reentrancy guard or refactoring the code in such a way that all the state changes occur before untrusted external calls.
**Client Comment** Won't fix. Permit is the only external call that happens before state changes, so we decided to limit reentrancy protection for this case only

Listing 11:

```
185 +require(_remaining[orderHash] == 0, "LOP: reentrancy detected")
    ↪ ;
```

## 3.12 CVF-12

- **Severity** Critical
- **Category** Flaw
- **Status** Fixed
- **Source** OrderMixin.sol

**Recommendation** Should be "makerAssetData" instead of "makerAsset".

Listing 12:

```
+order.makerAsset
```

## 3.13 CVF-13

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** OrderMixin.sol

**Description** These two functions have much in common.
**Recommendation** Consider refactoring them to remove code duplication.

Listing 13:

```
+function _callGetMakerAmount(Order memory order, uint256
  ↪ takerAmount) private view returns(uint256 makerAmount) {

+function _callGetTakerAmount(Order memory order, uint256
  ↪ makerAmount) private view returns(uint256 takerAmount) {
```

## 3.14 CVF-14

- **Severity** Minor
- **Category** Readability
- **Status** Fixed
- **Source** OrderMixin.sol

**Description** The code below looks like it is always executed, while actually it is executed only when certain condition is false.
**Recommendation** Consider putting the rest of the functions into explicit "else" branches for readability.

Listing 14:

```
+}

+}
```

## 3.15 CVF-15

- **Severity** Minor
- **Category** Procedural
- **Status** Info
- **Source** OrderRFQMixin.sol

**Recommendation** This parameter should be indexed.

Listing 15:

```
+bytes32 orderHash,
```

## 3.16 CVF-16

- **Severity** Minor
- **Category** Procedural
- **Status** Info
- **Source** OrderRFQMixin.sol

**Recommendation** Solidity is able to pack several narrow values into a single word. Usually there is no need to pack/unpack manually.
**Client Comment** Wont' fix. Not for calldata though.

Listing 16:

```
+uint256 info; // lowest 64 bits is the order id, next 64 bits
    ↪ is the expiration timestamp
```

## 3.17 CVF-17

- **Severity** Minor
- **Category** Documentation
- **Status** Info
- **Source** OrderRFQMixin.sol

**Description** The semantics of the keys and values of this mapping is unclear.
**Recommendation** Consider adding a documentation comment.
**Client Comment** Noted.

Listing 17:

```
+mapping(address => mapping(uint256 => uint256)) private
    ↪ _invalidator;
```

## 3.18 CVF-18

- **Severity** Minor
- **Category** Procedural
- **Status** Info
- **Source** OrderRFQMixin.sol

**Recommendation** This function probably should log an event.

Listing 18:

```
+function cancelOrderRFQ(uint256 orderInfo) external {
```

## 3.19 CVF-19

- **Severity** Moderate
- **Category** Flaw
- **Status** Fixed
- **Source** RevertReasonParser.sol

**Description** This may ready after the "data" array.
**Recommendation** Consider performing a length check to ensure that "data" as at least 4 bytes before reading the selector.

Listing 19:

```
+selector := mload(add(data, 0x20))
```

## 3.20 CVF-20

- **Severity** Minor
- **Category** Bad datatype
- **Status** Fixed
- **Source** RevertReasonParser.sol

**Recommendation** The hardcoded selectors should be turned into named constants. These constants could be initialized like this: bytes4 public constant ERROR_SELECTOR = bytes4 (keccak256 ("Error(string)")); bytes4 public constant PANIC_SELECTOR = bytes4 (keccak256 ("Panic(uint256)"));

Listing 20:

```
+if (selector == 0x08c379a0 && data.length >= 68) {

+else if (selector == 0x4e487b71 && data.length == 36) {
```

### 3.21 CVF-21

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** RevertReasonParser.sol

**Description** This trick only works well for correctly formatted "data".
**Recommendation** Consider using the "abi.decode" instead like this: https://ethereum.stackexchange.com/questions/83528/how-can-i-get-the-revert-reason-of-a-call-in-solidity-so-that-i-can-use-it-in-th/110428#110428

Listing 21:

```
+string memory reason;
+// solhint-disable no-inline-assembly
+assembly {
+    // 68 = 32 bytes data length + 4-byte selector + 32 bytes
    ↪ offset
+    reason := add(data, 68)
+}

+uint256 code;
+// solhint-disable no-inline-assembly
+assembly {
+    // 36 = 32 bytes data length + 4-byte selector
+    code := mload(add(data, 36))
+}
```

### 3.22 CVF-22

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** RevertReasonParser.sol

**Description** This function is suboptimal. See the following answer for how it could be optimized: https://stackoverflow.com/questions/67893318/solidity-how-to-represent-bytes32-as-string/69266989#69266989
**Client Comment** Noted.

Listing 22:

```
+function _toHex(uint256 value) private pure returns(string
    ↪ memory) {
```

### 3.23 CVF-23

- **Severity** Major
- **Category** Procedural
- **Status** Info
- **Source** ArgumentsDecoder.sol

**Description** This can read the memory after the end of the "data" array.
**Recommendation** Consider adding an explicit length check to ensure that "data" is at least 32 bytes long.
**Client Comment** We do that check before all calls.

Listing 23:

```
+value := mload(add(data, 0x20))
```

### 3.24 CVF-24

- **Severity** Major
- **Category** Flaw
- **Status** Info
- **Source** ArgumentsDecoder.sol

**Description** This can read the memory after the end of the "data" array.
**Recommendation** Consider adding an explicit length check to ensure that "data" is at least 1 byte long.
**Client Comment** We do that check before all calls.

Listing 24:

```
+value := mload(add(data, 0x20))
```

### 3.25 CVF-25

- **Severity** Major
- **Category** Flaw
- **Status** Fixed
- **Source** ArgumentsDecoder.sol

**Description** This can read the memory after the end of the "data" array.
**Recommendation** Consider adding an explicit length check to ensure that "data" is at least 20 bytes long.

Listing 25:

```
+target := mload(add(data, 0x14))
```

## 3.26 CVF-26

- **Severity** Moderate
- **Status** Fixed
- **Category** Flaw
- **Source** ArgumentsDecoder.sol

**Description** This may write the memory after the end of the "data" array.
**Recommendation** Consider adding an explicit check that "data" is at least 20 bytes long.

Listing 26:

```
36 +mstore(args, sub(mload(data), 0x14))
```

## 3.27 CVF-27

- **Severity** Minor
- **Status** Fixed
- **Category** Flaw
- **Source** ArgumentsDecoder.sol

**Description** This may read the calldata after the end of the "data" array. In such situation, the slicing attempt below will fail, however it still would be better to do an explicit length check.

Listing 27:

```
45 +target := shr(96, calldataload(data.offset))
```