



ABDK CONSULTING

SMART CONTRACT
AUDIT

1inch

LimitOrderProtocol

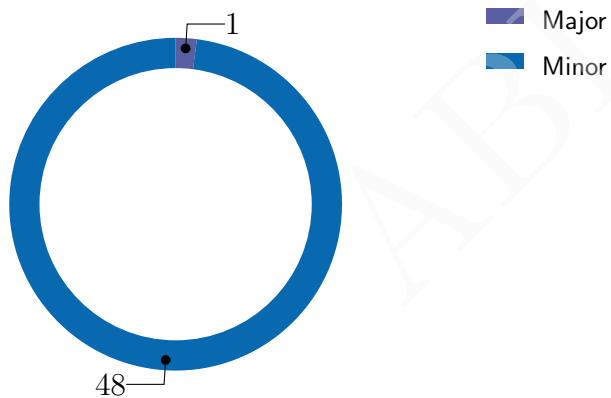


abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
31st May 2021

We've been asked to review the Limit Order Protocol smart contract as part of 1inch ecosystem. We have found only one major issue and a few minor ones.



Findings

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

ID	Severity	Category	Status
CVF-28	Minor	Suboptimal	Opened
CVF-29	Minor	Flaw	Opened
CVF-30	Minor	Suboptimal	Opened
CVF-31	Minor	Suboptimal	Opened
CVF-32	Minor	Procedural	Opened
CVF-33	Minor	Flaw	Opened
CVF-34	Minor	Bad naming	Opened
CVF-35	Minor	Suboptimal	Opened
CVF-36	Minor	Unclear behavior	Opened
CVF-37	Minor	Suboptimal	Opened
CVF-38	Minor	Suboptimal	Opened
CVF-39	Minor	Suboptimal	Opened
CVF-40	Minor	Suboptimal	Opened
CVF-41	Minor	Unclear behavior	Opened
CVF-42	Minor	Bad naming	Opened
CVF-43	Minor	Suboptimal	Opened
CVF-44	Minor	Suboptimal	Opened
CVF-45	Minor	Unclear behavior	Opened
CVF-46	Minor	Flaw	Opened
CVF-47	Minor	Suboptimal	Opened
CVF-48	Minor	Suboptimal	Opened
CVF-49	Minor	Unclear behavior	Opened

Contents

1	Document properties	7
2	Introduction	8
2.1	About ABDK	8
2.2	Disclaimer	8
2.3	Methodology	8
3	Detailed Results	10
3.1	CVF-1	10
3.2	CVF-2	10
3.3	CVF-3	10
3.4	CVF-4	11
3.5	CVF-5	11
3.6	CVF-6	11
3.7	CVF-7	12
3.8	CVF-8	12
3.9	CVF-9	12
3.10	CVF-10	13
3.11	CVF-11	13
3.12	CVF-12	13
3.13	CVF-13	14
3.14	CVF-14	15
3.15	CVF-15	16
3.16	CVF-16	16
3.17	CVF-17	16
3.18	CVF-18	17
3.19	CVF-19	17
3.20	CVF-20	17
3.21	CVF-21	18
3.22	CVF-22	18
3.23	CVF-23	18
3.24	CVF-24	19
3.25	CVF-25	19
3.26	CVF-26	19
3.27	CVF-27	20
3.28	CVF-28	20
3.29	CVF-29	20
3.30	CVF-30	21
3.31	CVF-31	21
3.32	CVF-32	22
3.33	CVF-33	22
3.34	CVF-34	22
3.35	CVF-35	23
3.36	CVF-36	23
3.37	CVF-37	23

3.38	CVF-38	24
3.39	CVF-39	24
3.40	CVF-40	24
3.41	CVF-41	25
3.42	CVF-42	25
3.43	CVF-43	25
3.44	CVF-44	26
3.45	CVF-45	26
3.46	CVF-46	26
3.47	CVF-47	27
3.48	CVF-48	27
3.49	CVF-49	27

ABDK

1 Document properties

Version

Version	Date	Author	Description
0.1	May 28, 2021	D. Khovratovich	Initial Draft
0.2	May 28, 2021	D. Khovratovich	Minor revision
1.0	May 31, 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 were given access to the [1inch repo](#), [release abdk-audit](#). The files are written in Solidity language and are located [here](#):

- helpers/AmountCalculator.sol;
- helpers/ERC1155Proxy.sol;
- helpers/ERC20Proxy.sol;
- helpers/ERC721Proxy.sol;
- helpers/ImmutableOwner.sol;
- helpers/NonceManager.sol;
- helpers/PredicateHelper.sol;
- interfaces/InteractiveMaker.sol;
- libraries/ArgumentsDecoder.sol;
- libraries/UncheckedAddress.sol;
- LimitOrderProtocol.sol.

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 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** Opened
- **Source** AmountCalculator.sol

Recommendation Since this function calls an arbitrary external contract, it probably should not be specified as view.

Listing 1:

```
21 function arbitraryStaticCall(address target, bytes memory data)
    ↪ external view returns(uint256) {
```

3.2 CVF-2

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **Source** LimitOrderProtocol.sol

Description EIP712 is not imported explicitly.

Recommendation Consider doing this.

Listing 2:

```
21 EIP712("1inch Limit Order Protocol", "1"),
```

3.3 CVF-3

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **Source** LimitOrderProtocol.sol

Recommendation This is not needed since Solidity 0.8.

Listing 3:

```
6 "@openzeppelin/contracts/utils/math/SafeMath.sol";
30 using SafeMath for uint256;
```

3.4 CVF-4

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **Source** LimitOrderProtocol.sol

Description The commented code should probably be removed.

Listing 4:

```
35 // Partial Fill:
//   getMakerAmount := GetMakerAmountHelper.disablePartialFill(
//     ↪ makerAmount, ...)
//
// Expiration Mask:
//   predicate := PredicateHelper.timestampBelow(deadline)
40 //
// Maker Nonce:
//   predicate := this.nonceEquals(makerAddress, makerNonce)
```

3.5 CVF-5

- **Severity** Minor
- **Category** Bad naming
- **Status** Opened
- **Source** LimitOrderProtocol.sol

Recommendation Events are usually named via nouns, such as "OrderFill", "OrderFillRFQ", or just "Fill", "FillRFQ".

Listing 5:

```
44 event OrderFilled(
50 event OrderFilledRFQ(
```

3.6 CVF-6

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** LimitOrderProtocol.sol

Recommendation These parameters should be indexed.

Listing 6:

```
46 bytes32 orderHash ,
51 bytes32 orderHash ,
```

3.7 CVF-7

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** LimitOrderProtocol.sol

Recommendation This value could be made a compile-time constant rather than an immutable variable.

Listing 7:

```
84 bytes4 immutable private _maxSelector = bytes4(uint32(IERC20.  
    ↪ transferFrom.selector) + 10);
```

3.8 CVF-8

- **Severity** Minor
- **Category** Documentation
- **Status** Opened
- **Source** LimitOrderProtocol.sol

Description Semantics of keys in these mappings is unclear.

Recommendation Consider adding documentation comments.

Listing 8:

```
86 mapping(bytes32 => uint256) private _remaining;  
mapping(address => mapping(uint256 => uint256)) private  
    ↪ _invalidator;
```

3.9 CVF-9

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** LimitOrderProtocol.sol

Recommendation This function would not be necessary if the '`_remaining`' mapping would be public.

Listing 9:

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

3.10 CVF-10

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** LimitOrderProtocol.sol

Recommendation These functions are quite generic and should probably be moved to some helper or library.

Listing 10:

```
113 function checkPredicate(Order memory order) public view returns(  
    ↪ bool) {  
  
119 function simulateTransferFroms(IERC20[] calldata tokens, bytes[]  
    ↪ calldata data) external {
```

3.11 CVF-11

- **Severity** Minor
- **Category** Flaw
- **Status** Opened
- **Source** LimitOrderProtocol.sol

Recommendation There should be a check that 'tokens' and 'data' have the same length.

Listing 11:

```
119 function simulateTransferFroms(IERC20[] calldata tokens, bytes[]  
    ↪ calldata data) external {
```

3.12 CVF-12

- **Severity** Minor
- **Category** Bad naming
- **Status** Opened
- **Source** LimitOrderProtocol.sol

Recommendation The name is confusing, as the functionality of this method is quite generic as it just passes arbitrary calls to other contracts, and is not restricted to 'transferFroms'.

Listing 12:

```
119 function simulateTransferFroms(IERC20[] calldata tokens, bytes[]  
    ↪ calldata data) external {
```

3.13 CVF-13

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** LimitOrderProtocol.sol

Recommendation A single array of structs with two fields would be more efficient.

Listing 13:

```
119 function simulateTransferFroms(IERC20[] calldata tokens, bytes[]  
    ↪ calldata data) external {
```

3.14 CVF-14

- **Severity** Minor
- **Category** Bad datatype
- **Status** Opened
- **Source** LimitOrderProtocol.sol

Recommendation The indexes of various parameters should be made named constants.

Listing 14:

```
135 require(order.makerAssetData.decodeAddress(0) == msg.sender, "  
    ↳ LOP: Access denied");  
  
152 address maker = order.makerAssetData.decodeAddress(0);  
  
158 uint256 orderMakerAmount = order.makerAssetData.decodeUint256(2)  
    ↳ ;  
    uint256 orderTakerAmount = order.takerAssetData.decodeUint256(2)  
    ↳ ;  
  
202         remainingMakerAmount = order.makerAssetData.  
    ↳ decodeUint256(2);  
  
223         takingAmount = (makingAmount == order.makerAssetData.  
    ↳ decodeUint256(2))  
        ? order.takerAssetData.decodeUint256(2)  
  
228         makingAmount = (takingAmount == order.takerAssetData.  
    ↳ decodeUint256(2))  
        ? order.makerAssetData.decodeUint256(2)  
  
249         InteractiveMaker(order.makerAssetData.decodeAddress(0))  
  
310 address maker = address(makerAssetData.decodeAddress(0));  
  
319 address orderTakerAddress = makerAssetData.decodeAddress(1);  
  
321         makerAssetData.patchAddress(1, taker);  
  
328         makerAssetData.patchUint256(2, makingAmount);  
  
340 takerAssetData.patchAddress(0, taker);  
  
344         takerAssetData.patchUint256(2, takingAmount);
```

3.15 CVF-15

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Opened
- **Source** LimitOrderProtocol.sol

Description The function does not check that an order was created. Probably it is ok.

Listing 15:

```
138 _remaining[orderHash] = 1;
```

3.16 CVF-16

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** LimitOrderProtocol.sol

Description Emitting the 'OrderFilled' event on order cancellation is confusing.

Recommendation Consider emitting a separate cancellation event.

Listing 16:

```
139 emit OrderFilled(msg.sender, orderHash, 0);
```

3.17 CVF-17

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** LimitOrderProtocol.sol

Recommendation Right shift and bitwise 'and' would be more efficient.

Listing 17:

```
143 _invalidator[msg.sender][uint64(orderInfo) / 256] |= (1 << (
    ↪ orderInfo % 256));

153 uint256 invalidator = _invalidator[maker][uint64(order.info) /
    ↪ 256];
    require(invalidator & (1 << (order.info % 256)) == 0, "LOP:
    ↪ already filled");
    _invalidator[maker][uint64(order.info) / 256] = invalidator | (1
    ↪ << (order.info % 256));
```


3.18 CVF-18

- **Severity** Minor
- **Category** Flaw
- **Status** Opened
- **Source** LimitOrderProtocol.sol

Recommendation The expression " $1 \ll (\text{order.info} \% 256)$ " is calculated twice.

Listing 18:

```
154 require(validator & (1 << (order.info % 256)) == 0, "LOP:
    ↪ already filled");
    _validator[maker][uint64(order.info) / 256] = validator | (1
    ↪ << (order.info % 256));
```

3.19 CVF-19

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** LimitOrderProtocol.sol

Recommendation There are functions for this in 'AmountCalculator'.

Listing 19:

```
170 takingAmount = (makingAmount * orderTakerAmount +
    ↪ orderMakerAmount - 1) / orderMakerAmount;

173 makingAmount = takingAmount * orderMakerAmount /
    ↪ orderTakerAmount;
```

3.20 CVF-20

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** LimitOrderProtocol.sol

Recommendation The function emits the remaining amount, which can be acquired from the contract directly, but does not emit the actual making/taking amounts, which might be more relevant.

Listing 20:

```
242 emit OrderFilled(msg.sender, orderHash, remainingMakerAmount);

256 return (makingAmount, takingAmount);
```

3.21 CVF-21

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **Source** LimitOrderProtocol.sol

Description We didn't review this function.

Listing 21:

```
260 return _hashTypedDataV4(  
280 return _hashTypedDataV4(
```

3.22 CVF-22

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** LimitOrderProtocol.sol

Recommendation The return statement is redundant as the callee does not return anything.

Listing 22:

```
299 return _validate(order.makerAssetData, order.takerAssetData ,  
    ↪ signature, orderHash);
```

3.23 CVF-23

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **Source** LimitOrderProtocol.sol

Description ECDSA is not imported explicitly.

Recommendation Consider doing this.

Listing 23:

```
311 if ((signature.length != 65 && signature.length != 64) || ECDSA.  
    ↪ recover(orderHash, signature) != maker) {
```

3.24 CVF-24

- **Severity** Major
- **Category** Flaw
- **Status** Opened
- **Source** LimitOrderProtocol.sol

Description ECDSA.recover' will revert in case the signature is invalid, thus preventing 'is-ValidSignature' on the maker to be called. Common way is to check the code size of the potential signer, and if it is non-zero, call isValidSignature on it, otherwise try validating the ECDSA signature.

Listing 24:

```
311 if ((signature.length != 65 && signature.length != 64) || ECDSA.  
    ↪ recover(orderHash, signature) != maker) {
```

3.25 CVF-25

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** UncheckedAddress.sol

Recommendation The " returndata_size" variable is redundant as it is used only once.

Listing 25:

```
40 let returndata_size := mload(returndata)  
   revert(add(32, returndata), returndata_size)
```

3.26 CVF-26

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **Source** ArgumentsDecoder.sol

Recommendation It is not checked that the data length is at least 4 bytes.

Listing 26:

```
7 function decodeSelector(bytes memory data) internal pure returns  
  ↪ (bytes4 selector) {
```

3.27 CVF-27

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **Source** ArgumentsDecoder.sol

Recommendation It is not checked that the argument position is within the array length.

Listing 27:

```
13 function decodeAddress(bytes memory data, uint256 argumentIndex)
    ↪ internal pure returns(address account) {
19 function decodeUint256(bytes memory data, uint256 argumentIndex)
    ↪ internal pure returns(uint256 value) {
25 function patchAddress(bytes memory data, uint256 argumentIndex,
    ↪ address account) internal pure {
31 function patchUint256(bytes memory data, uint256 argumentIndex,
    ↪ uint256 value) internal pure {
```

3.28 CVF-28

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** ArgumentsDecoder.sol

Recommendation Left shift would be more efficient, than multiplication.

Listing 28:

```
15 account := mload(add(add(data, 0x24), mul(argumentIndex, 0x20)))
```

3.29 CVF-29

- **Severity** Minor
- **Category** Flaw
- **Status** Opened
- **Source** PredicateHelper.sol

Recommendation The data array length is not checked.

Listing 29:

```
11 function or(address[] calldata targets, bytes[] calldata data)
    ↪ external view returns(bool) {
22 function and(address[] calldata targets, bytes[] calldata data)
    ↪ external view returns(bool) {
```

3.30 CVF-30

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** PredicateHelper.sol

Recommendation A single array of structures with two fields would be more efficient than two parallel arrays.

Listing 30:

```
11 function or(address[] calldata targets, bytes[] calldata data)
    ↪ external view returns(bool) {
```

3.31 CVF-31

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** PredicateHelper.sol

Recommendation These functions should be declared as public rather than external, to make internal calls possible.

Listing 31:

```
11 function or(address[] calldata targets, bytes[] calldata data)
    ↪ external view returns(bool) {
22 function and(address[] calldata targets, bytes[] calldata data)
    ↪ external view returns(bool) {
33 function eq(uint256 value, address target, bytes memory data)
    ↪ external view returns(bool) {
38 function lt(uint256 value, address target, bytes memory data)
    ↪ external view returns(bool) {
43 function gt(uint256 value, address target, bytes memory data)
    ↪ external view returns(bool) {
48 function timestampBelow(uint256 time) external view returns(bool
    ↪ ) {
```

3.32 CVF-32

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **Source** PredicateHelper.sol

Recommendation The argument order is counterintuitive: usually the lefthand operand is the first input.

Listing 32:

```
33 function eq(uint256 value, address target, bytes memory data)
    ↪ external view returns(bool) {
38 function lt(uint256 value, address target, bytes memory data)
    ↪ external view returns(bool) {
43 function gt(uint256 value, address target, bytes memory data)
    ↪ external view returns(bool) {
```

3.33 CVF-33

- **Severity** Minor
- **Category** Flaw
- **Status** Opened
- **Source** PredicateHelper.sol

Recommendation The result length is not checked.

Listing 33:

```
34 bytes memory result = target.uncheckedFunctionStaticCall(data, "
    ↪ PH: eq");
39 bytes memory result = target.uncheckedFunctionStaticCall(data, "
    ↪ PH: lt");
44 bytes memory result = target.uncheckedFunctionStaticCall(data, "
    ↪ PH: gt");
```

3.34 CVF-34

- **Severity** Minor
- **Category** Bad naming
- **Status** Opened
- **Source** NonceManager.sol

Recommendation Events are usually named via nouns, such as 'NonceIncrease'.

Listing 34:

```
7 event NonceIncreased(address indexed maker, uint256 newNonce);
```

3.35 CVF-35

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** NonceManager.sol

Recommendation These functions should probably return the updated nonce for convenience.

Listing 35:

```
11 function increaseNonce() external {  
15 function advanceNonce(uint8 amount) public {
```

3.36 CVF-36

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Opened
- **Source** NonceManager.sol

Description What is the reason to allow increases by more than 1, if intermediate values can not be used?

Listing 36:

```
15 function advanceNonce(uint8 amount) public {
```

3.37 CVF-37

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** NonceManager.sol

Recommendation This could be rewritten in one sentence: 'emit NonceIncreased(msg.sender, nonce[msg.sender] += amount);'

Listing 37:

```
16 uint256 newNonce = nonce[msg.sender] + amount;  
   nonce[msg.sender] = newNonce;  
   emit NonceIncreased(msg.sender, newNonce);
```

3.38 CVF-38

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** NonceManager.sol

Recommendation This function is probably redundant as the nonce mapping is public.

Listing 38:

```
21 function nonceEquals(address makerAddress, uint256 makerNonce)
    ↪ external view returns(bool) {
```

3.39 CVF-39

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** ERC20Proxy.sol

Recommendation This inheritance is not used and probably should be removed as the descendants inherit 'ImmutableOwner' anyway.

Listing 39:

```
12 contract ERC20Proxy is ImmutableOwner {
```

3.40 CVF-40

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** ERC20Proxy.sol

Description The constructor of this contract doesn't explicitly call the constructor of a base contract. This makes it less convenient to use this contract.

Recommendation Consider explicitly accepting the owner address as a constructor argument and forwarding it to the constructor of the "ImmutableOwner" contract.

Listing 40:

```
15 constructor() {
```


3.41 CVF-41

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Opened
- **Source** ERC20Proxy.sol

Description This function looks like a dirty hack. Why should it have the selector close to the normal 'transferFrom' function?

Listing 41:

```
19 // keccak256("func_50BkM4K(address,address,uint256,address)") =  
    ↪ 0x23b872de  
20 function func_50BkM4K(address from, address to, uint256 amount,  
    ↪ IERC20 token) external onlyImmutableOwner {
```

3.42 CVF-42

- **Severity** Minor
- **Category** Bad naming
- **Status** Opened
- **Source** ImmutableOwner.sol

Description The word "immutable" is redundant in the variable name. Note, that this variable is public and thus its name is a part of the contract's public API.

Listing 42:

```
7 address public immutable immutableOwner;
```

3.43 CVF-43

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** ERC1155Proxy.sol

Description This inheritance is not used and probably should be removed as the descendants inherit 'ImmutableOwner' anyway.

Listing 43:

```
13 contract ERC1155Proxy is ImmutableOwner {
```

3.44 Cvf-44

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** ERC1155Proxy.sol

Description The constructor of this contract doesn't explicitly call the constructor of a base contract. This makes it less convenient to use this contract.

Recommendation Consider explicitly accepting the owner address as a constructor argument and forwarding it to the contractor of the 'ImmutableOwner' contract.

Listing 44:

```
14 constructor () {
```

3.45 Cvf-45

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Opened
- **Source** ERC1155Proxy.sol

Description These functions look like dirty hacks. Why should they have these specific selectors?

Listing 45:

```
18 // keccak256("func_00cMjE8(address,address,uint256,address,
    ↪ uint256)") == 0x23b872e1
function func_00cMjE8(address from, address to, uint256 amount,
    ↪ IERC1155 token, uint256 tokenId) external
    ↪ onlyImmutableOwner {
```

3.46 Cvf-46

- **Severity** Minor
- **Category** Flaw
- **Status** Opened
- **Source** ERC1155Proxy.sol

Recommendation The function always sends empty data to the token recipient. Probably it should allow the data to be passed by the caller.

Listing 46:

```
20 token.safeTransferFrom(from, to, tokenId, amount, "");
```

3.47 CVF-47

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** ERC721Proxy.sol

Recommendation This inheritance is not used and probably should be removed as the descendants inherit 'ImmutableOwner' anyway.

Listing 47:

```
13 contract ERC721Proxy is ImmutableOwner {
```

3.48 CVF-48

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** ERC721Proxy.sol

Description The constructor of this contract doesn't explicitly call the constructor of a base contract. This makes it less convenient to use this contract.

Recommendation Consider explicitly accepting the owner address as a constructor argument and forwarding it to the constructor of the 'ImmutableOwner' contract.

Listing 48:

```
14 constructor() {
```

3.49 CVF-49

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Opened
- **Source** ERC721Proxy.sol

Description These functions look like dirty hacks. Why should they have these specific selectors?

Listing 49:

```
19 // keccak256("func_40aVqeY(address,address,uint256,address)") ==
    ↪ 0x23b872df
20 function func_40aVqeY(address from, address to, uint256 tokenId,
    ↪ IERC721 token) external onlyImmutableOwner {

24 // keccak256("func_20xtkDI(address,address,uint256,address)") ==
    ↪ 0x23b872e0
    function func_20xtkDI(address from, address to, uint256 tokenId,
    ↪ IERC721 token) external onlyImmutableOwner {
```