Report

v. 1.0

Customer

0x

Smart Contract Audit

# MetaTransaction

30th March 2023

Report prepared by

ABDK Consulting

# Contents

# 1 Changelog

| # | Date | Author | Description |
|---|---|---|---|
| 0.1 | 30.03.23 | A. Zveryanskaya | Initial Draft |
| 0.2 | 30.03.23 | A. Zveryanskaya | Minor revision |
| 1.0 | 30.03.23 | 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.

0x is important infrastructure for the emerging crypto economy and enables markets to be created that couldn't have existed before. As more assets become tokenized, public blockchains provide the opportunity to establish a new financial stack that is more efficient, transparent, and equitable than any system in the past.

# 3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

| | | |
|---|---|---|
| **contracts/zero-ex/contracts/** | | |
| IZeroEx.sol | | |
| **contracts/zero-ex/contracts/src/features/** | | |
| MetaTransactions FeatureV2.sol | UniswapV3Feature.sol | |
| **contracts/zero-ex/contracts/src/features/interfaces/** | | |
| IMetaTransactions FeatureV2.sol | IMultiplexFeature.sol | IUniswapV3Feature.sol |
| **contracts/zero-ex/contracts/src/features/multiplex/** | | |
| MultiplexFeature.sol | Multiplex LiquidityProvider.sol | MultiplexOtc.sol |
| MultiplexRfq.sol | Multiplex TransformERC20.sol | MultiplexUniswapV2.sol |
| MultiplexUniswapV3.sol | | |
| **contracts/zero-ex/contracts/src/storage/** | | |
| LibMetaTransactions V2Storage.sol | LibStorage.sol | |
| **contracts/zero-ex/tests/** | | |
| MetaTransaction V2Test.t.sol | MultiplexMeta TransactionsV2.t.sol | |
| **contracts/zero-ex/tests/forked/** | | |
| MultiplexRfqTest.t.sol | | |
| **contracts/zero-ex/tests/utils/** | | |
| DeployZeroEx.sol | LocalTest.sol | TestUtils.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 1 critical, 1 major, and a few less important issues. All identified Critical and Major issues have been fixed.

| Issues | Active | Fixed |
|---|---|---|
| | 0 | |
| Severity: Critical | | 1 |

| Major | Active | Fixed |
|---|---|---|
| | 0 | 1 |

| Moderate | Active | Fixed |
|---|---|---|
| | 0 | 1 |

| Minor | Active | Fixed |
|---|---|---|
| | 15 | 2 |

Fixed 5 out of 20 issues

# 6 Critical Issues

## CVF-1. FIXED

- **Category** Flaw
- **Source** MetaTransactionsFeatureV2.sol

**Description** These two modifiers may conflict with each other.

**Recommendation** Consider the following scenario: 1. The contract initially have balance of 1 ether 2. A meta-transaction is execution with msg.value of 2 ether, so the contract's balance becomes 3 ether 3. The meta-transaction consumes 1 ether, leaving 2 ether at the contract's balance 4. The "refundsAttachedEth" modifier will repay 2 ether to the sender, leaving zero ether at the contract's balance 5. The "doesNotReduceEthBalance" modifier will revert the transaction as the contract's balance actually was reduced. Consider merging these two modifiers into one that refunds ether above the initial balance.

**Client Comment** *We will address this issue by removing the refundsAttachedEth() modifier and make executeMetatransaction nonpayable.*

```
152 +doesNotReduceEthBalance
153 +refundsAttachedEth
```

```
177 +doesNotReduceEthBalance
178 +refundsAttachedEth
```

# 7 Major Issues

## CVF-2. FIXED

- **Category** Suboptimal
- **Source** MetaTransactionsFeatureV2.sol

**Description** Usage of ”transfer” is discouraged, as gas costs of operations could change in the future and it is hard to guarantee that stipend will be enough going forward.

**Recommendation** Consider using the ”call” function instead.

**Client Comment** *Same resolution as #1*

```
112 +msg.sender.transfer(remainingBalance);
```

# 8 Moderate Issues

## CVF-3. FIXED

- **Category** Unclear behavior
- **Source** MetaTransactionsFeatureV2.sol

**Description** This doesn't guarantee that the caller won't take other people's money, as "msg.value" ether could be consumed in transaction and "this.balance" could consume some extra ether that were there before the transaction.

**Recommendation** Consider allowing to refund only the actual leftover ether from the current transaction.

**Client Comment** *Same resolution as #1*

```
110 +uint256 remainingBalance = LibSafeMathV06.min256(msg.value, address
    ↪ (this).balance);
```

# 9 Minor Issues

## CVF-4. INFO

- **Category** Procedural
- **Source** MetaTransactionsFeatureV2.sol

**Description** This compiler version is quite outdated.

**Recommendation** Consider migrating to a newer version. Migration to 0.7.x should be quite simple.

**Client Comment** *We are aware of this issue, but won't address it at this time.*

```
15 +pragma solidity ^0.6.5;
```

## CVF-5. INFO

- **Category** Procedural
- **Source** MetaTransactionsFeatureV2.sol

**Description** Consider specifying as "^0.6.0" unless there is something special about this particular version.

**Recommendation** Also relevant for: LibMetaTransactionsV2Storage.sol, IMetaTransactionsFeatureV2.sol.

**Client Comment** *We are aware of this issue, but won't address it at this time.*

```
15 +pragma solidity ^0.6.5;
```

## CVF-6. INFO

- **Category** Procedural
- **Source** MetaTransactionsFeatureV2.sol

**Description** We didn't review these files.

**Client Comment** *Ok*

```
18 +import "@0x/contracts-erc20/src/IEtherToken.sol";
19 +import "@0x/contracts-utils/contracts/src/v06/errors/
      ↪ LibRichErrorsV06.sol";
20 +import "@0x/contracts-utils/contracts/src/v06/LibBytesV06.sol";
21 +import "@0x/contracts-utils/contracts/src/v06/LibSafeMathV06.sol";
22 +import "../errors/LibMetaTransactionsRichErrors.sol";
23 +import "../fixins/FixinCommon.sol";
24 +import "../fixins/FixinReentrancyGuard.sol";
25 +import "../fixins/FixinTokenSpender.sol";
26 +import "../fixins/FixinEIP712.sol";
27 +import "../migrations/LibMigrate.sol";
```

```
29 +import "./interfaces/IFeature.sol";
```

```
32 +import "./interfaces/INativeOrdersFeature.sol";
33 +import "./interfaces/ITransformERC20Feature.sol";
34 +import "./libs/LibSignature.sol";
```

## CVF-7. INFO

- **Category** Bad datatype
- **Source** MetaTransactionsFeatureV2.sol

**Recommendation** The type of the "zeroExAddress" argument should be more specific.

**Client Comment** *This is the way the code is written across the codebase, so we aren't going to fix it.*

```
124 +constructor(address zeroExAddress, IEtherToken weth) public
      ↪ FixinCommon() FixinEIP712(zeroExAddress) {
```

## CVF-8. INFO

- **Category** Suboptimal
- **Source** MetaTransactionsFeatureV2.sol

**Description** The storage position of ”LibMetaTransactionsV2Storage.getStorage().mtxHashToExecutedBlockNumber[state.hash]” is calculated twice: once here and another time inside the ”_validateMetaTransaction” function.

**Recommendation** Consider refactoring to avoid double calculation.

**Client Comment** *Fixing this issue would be more trouble than it is worth right now.*

```
245 +_validateMetaTransaction(state);
```

```
250 +LibMetaTransactionsV2Storage.getStorage().
    ↪ mtxHashToExecutedBlockNumber[state.hash] = block.number;
```

## CVF-9. INFO

- **Category** Procedural
- **Source** LibMetaTransactionsV2Storage.sol

**Description** This compiler version is quite outdated.

**Recommendation** Consider migrating to a newer version. Migration to 0.7.x should be quite simple.

**Client Comment** *We are aware of this issue, but won’t address it at this time.*

```
15 +pragma solidity ^0.6.5;
```

## CVF-10. FIXED

- **Category** Suboptimal
- **Source** MultiplexOtc.sol

**Recommendation** This variable is used only once and thus is probably redundant.

**Client Comment** *We will fix this issue.*

```
85 +uint256 sellAmount = state.outputTokenAmount;
```

## CVF-11. INFO

- **Category** Documentation
- **Source** MultiplexFeature.sol

**Recommendation** It is unclear what does ”Internal variant” mean here. Consider clarifying.

**Client Comment** *Fixing this would create an inconsistency with documentation for other functions in the contracts. So we will leave it as is.*

```
189 +///      `minBuyAmount` of `outputToken` was bought. Internal
       ↪ variant.
```

```
333 +/// @dev Executes a multi-hop sell. Internal variant.
```

## CVF-12. INFO

- **Category** Documentation
- **Source** IUniswapV3Feature.sol

**Recommendation** It is unclear, what does ”Internal variant” mean here. Consider clarifying.

**Client Comment** *Fixing this would create an inconsistency with documentation for other functions in the contracts. So we will leave it as is.*

```
57 +/// @dev Sell a token for another token directly against uniswap v3
      ↪ . Internal variant.
```

## CVF-13. INFO

- **Category** Procedural
- **Source** IUniswapV3Feature.sol

**Description** Underscore (”_”) prefix in function names is commonly used for internal or private functions, while here it is used for an external function.

**Recommendation** Consider not using the prefix.

**Client Comment** *Fixing this would create an inconsistency with naming conventions for other functions in the contracts. So we will leave it as is.*

```
64 +function _sellTokenForTokenToUniswapV3(
```

## CVF-14. INFO

- **Category** Procedural
- **Source** IMultiplexFeature.sol

**Description** Underscore ("_") prefix in function names is commonly used for internal or private functions, while here it is used for an external function.

**Recommendation** Consider not using the prefix.

**Client Comment** *Fixing this would create an inconsistency with naming conventions for other functions in the contracts. So we will leave it as is.*

```
166 +function _multiplexBatchSell(
```

```
229 +function _multiplexMultiHopSell(
```

## CVF-15. INFO

- **Category** Procedural
- **Source** IMetaTransactionsFeatureV2.sol

**Recommendation** This compiler version is quite outdated. Consider migrating to a newer version. Migration to 0.7.x should be quite simple.

**Client Comment** *We are aware of this issue, but won't address it at this time.*

```
15 +pragma solidity ^0.6.5;
```

## CVF-16. INFO

- **Category** Procedural
- **Source** IMetaTransactionsFeatureV2.sol

**Description** We didn't review these files.

**Client Comment** *Ok*

```
18 +import "@0x/contracts-erc20/src/IERC20Token.sol";
19 +import "../libs/LibSignature.sol";
```

## CVF-17. FIXED

- **Category** Documentation
- **Source** IMetaTransactionsFeatureV2.sol

**Description** It is unclear what properties are hashed and how.

**Recommendation** Consider documenting.

**Client Comment** *We will fix this issue.*

```
50 +/// @param hash The meta-transaction hash.
```

## CVF-18. INFO

- **Category** Bad naming
- **Source** IMetaTransactionsFeatureV2.sol

**Recommendation** Events are usually named via nouns, such as ”Execution”.

**Client Comment** *Changing this would create an inconsistency with event naming across the contracts, so we will leave it as is.*

```
54 +event MetaTransactionExecuted(bytes32 hash, bytes4 indexed selector
   ↪ , address signer, address sender);
```

## CVF-19. INFO

- **Category** Procedural
- **Source** IMetaTransactionsFeatureV2.sol

**Recommendation** The ”signed” and ”sender” parameters should be indexed.

**Client Comment** *We can’t think of a use case where it’d be helpful to have these indexed as the application logic currently doesn’t require these to be index. So we don’t want to pay the additional gas to index them if it isn’t needed.*

```
54 +event MetaTransactionExecuted(bytes32 hash, bytes4 indexed selector
   ↪ , address signer, address sender);
```

## CVF-20. INFO

- **Category** Suboptimal
- **Source** IMetaTransactionsFeatureV2.sol

**Recommendation** It would be more efficient to accept a single array of structs with two fields, rather than two parallel arrays. This would also make the length check unnecessary.

**Client Comment** *The fix would be more efficient, but it isn't worth the effort at this time. Changing this would also break consistency across the contracts.*

```
70 +MetaTransactionDataV2[] calldata mtxs,
71 +LibSignature.Signature[] calldata signatures
```

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