thec00n fix link in chapter 2

1 contributor

854 lines (518 sloc) | 58.8 KB

0x Protocol v2 Audit

1 Introduction

From July 23, 2018 to September 7, 2018 ConsenSys Diligence conducted a security audit of the 0x project's protocol version v2 of their contract system. The findings and recommendations are presented here in this initial report.

NOTE: The 0x team's mitigation efforts of the audit findings are currently still in progress. The report will be subject to updates to reflect issues that can be considered closed.

1.1 Audit Dashboard

Audit Details

- **Project Name:** 0x 2.0
- **Client Name:** 0x
- **Client Contact:** Will Warren, Amir Bandeali
- **Auditors:** Suhabe Bugrara, Gerhard Wagner, John Mardlin, Steve Marx, Micah Dameron
- **GitHub :** https://github.com/ConsenSys/0x_audit_2018-07-23
- **Languages:** Solidity, TypeScript, JavaScript
- **Date:** 2018-07-23 to 2018-09-07

Number of issues per severity

	Minor	Medium	Major	Critical
Open	6	6	0	0
Closed	4	7	0	2

1.2 Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient and working according to its specifications. The audit activities can be grouped in the following three categories:

Security: Identifying security related issues within each contract and within the system of contracts.

Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:

- Correctness
- Readability
- Sections of code with high complexity
- Improving scalability
- Quantity and quality of test coverage

1.3 System Overview

Documentation

The following documentation was available to the audit team:

- The Wiki: high level information on the 0x system contract system.
- The Version 2 specifications: detailed specifications of the entire contract system.
- "Front-running, Griefing and the Perils of Virtual Settlement" parts 1 and 2: Technical challenges around DEX design and how 0x is designed to overcome them.

Scope

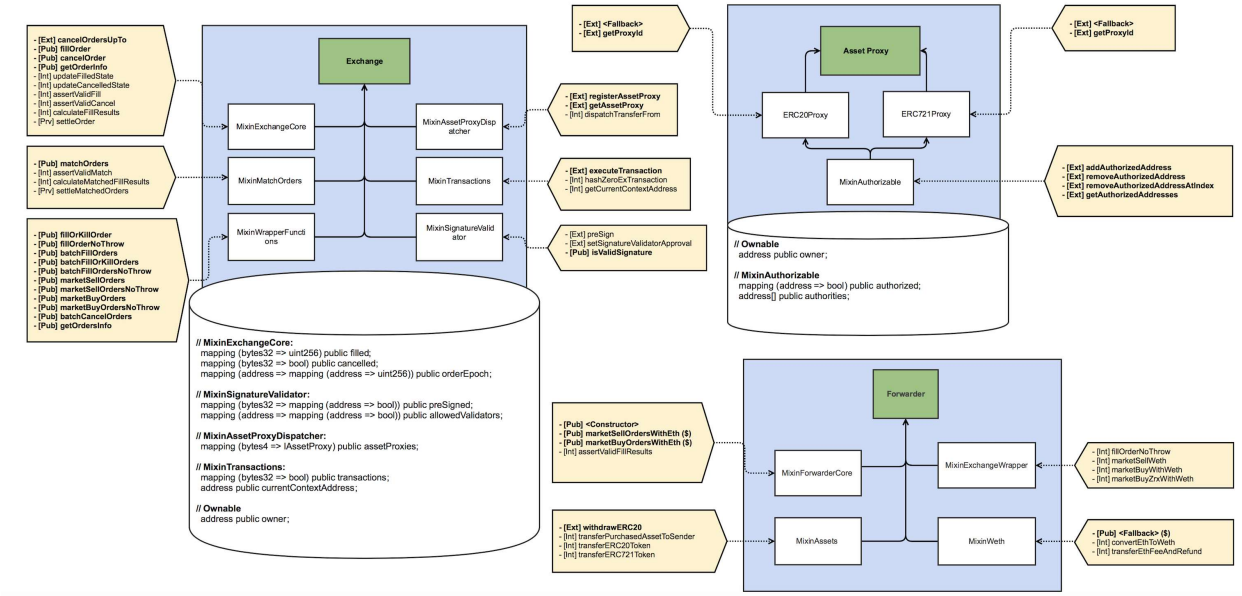
The audit focus was on the smart contract files, and test suites found in the following directories of the 0x-monorepo repository:

Directory	Commit hash
packages/contracts/src/2.0.0/forwarder packages/contracts/src/2.0.0/protocol packages/contracts/src/2.0.0/utils	a05b14e4d9659be1cc495ee33fd8962ce773

Architecture

The 0x audit scope does not include all of the components that can be found in a complete deployment. The token related contracts are out of scope. The in-scope items can be divided into the following three distinct parts:

- **Exchange** contains the bulk of the business logic within the 0x protocol. It is the entry point for filling orders, canceling orders, executing transactions, validating signatures and registering new ERC Proxy contracts into the system.
- **Asset Proxy** is responsible for decoding asset-specific metadata contained within an order, performing the actual asset transfer and authorizing/unauthorizing Exchange contract addresses from calling the transfer methods.
- **Forwarder** enables users to buy assets (ERC20 or ERC721 tokens) with ETH. It removes the required knowledge of WETH and allowances.



1.4 Key Observations

We found the quality of the code base to be high, which is particularly appreciated when approaching a complex protocol. In particular:

- The specification documents are thorough and well written. The diagrams of the system's interactions help to visualize the system.
- The code is well commented, particularly in sections where it is most needed to understand the developer's intent.
- The organization of the contract repository is thoughtful and consistent. For example, the names of Solidity contracts which are inherited but not deployed are differentiated with the prefix `Mixin_`. Interfaces are prefixed with `M_`.

Relative to the v1 version of the system, the v2 updates introduce features which significantly improve the user experience but also introduce many new edge cases, which directly resulted in some critical issues. These features include:

- Support for multiple `SignatureType` values, especially `Caller`, which was unique in that it did not check the signature. It rather assumed that if `msg.sender` is equal to the `order.makerAddress` then the order is valid and was in fact created by the `msg.sender`. See section 3.2 for more information on the resulting issue.
- Enabling a 3rd party to call Exchange functions on behalf of a user. See section 3.1 for the resulting issue, and remediations.

We believe that the system lacks a rigorous and systematic testing strategy that ensures comprehensive test coverage beyond mere line/branch coverage. We found that many important behaviors were untested during a spot check of the test suite. See issues #34, #43, #46, #49, and #54.

1.5 Recommendations

- **Test coverage is incomplete:** Any contract system that is used on the main network should achieve 100% test coverage.
- **Fix all issues:** We recommend addressing all the issues listed in the sections below, at the very least the ones with severity Critical, Major and Medium. All issues have also been created in a separate audit repository.

2 Issue Overview

The following table contains all the issues discovered during the audit. The issues are ordered based on their severity. More detailed description on the levels of severity can be found in Appendix 2. The table also contains the current GitHub status of the discovered issues.

Chapter	Issue Title	Issue Status	Severity
3.1	A malicious maker can empty a taker's account of all tokens.	Closed	Critical
3.2	MixinSignatureValidator: Insecure signature validator SignatureType.Caller	Closed	Critical
3.3	Use of outdated multisig wallet	Open	Medium
3.4	AssetProxyOwner: Insufficient Testing	Open	Medium
3.5	AssetProxyOwner: timelocked transactions affected by changing wallet parameters	Closed	Medium
3.6	AssetProxyOwner: removeAuthorizedAddressAtIndex requires multiple confirmations	Closed	Medium
3.7	LibBytes: Insufficient Testing	Closed	Medium
3.8	LibBytes: readBytes4 does not adhere to spec	Closed	Medium
3.9	AssetProxyOwner: readBytes4 does not adhere to spec	Closed	Medium
3.10	MixinAuthorizable: Insufficient Testing	Open	Medium
3.11	ERC721Proxy: Insufficient Testing	Open	Medium

Chapter	Issue Title	Issue Status	Severity
3.12	ERC721Proxy: fallback function silently fails	Closed	Medium
3.13	ERC20Proxy: fallback function silently fails	Closed	Medium
3.14	ERC20Proxy: Insufficient testing	Open	Medium
3.15	ERC721Token: inaccurate isContract function	Open	Medium
3.16	AssetProxyOwner: accepts ETH	Open	Minor
3.17	AssetProxyOwner duplicates code for readBytes4 function	Closed	Minor
3.18	Outdated compiler version	Closed	Minor
3.19	Use of this.balance in WETH9.sol	Open	Minor
3.20	ERC20Proxy: Reconsider use of inline assembly	Open	Minor
3.21	ERC20Proxy: Unclear comments	Closed	Minor
3.22	ERC20Proxy/ERC721Proxy: LibBytes imported but not used	Closed	Minor
3.23	LibBytes is imported but never used	Open	Minor
3.24	Optimization: refine function visibilities in the Exchange for gas efficiency	Open	Minor
3.25	LibConstants: dynamic constructor initialisation	Open	Minor

3 Issue Detail

3.1 A malicious maker can empty a taker's account of all tokens.

Severity	Status	Link	Remediation Comment

Severity	Status	Link	Remediation Comment
Critical	Closed	issues/53	PRs 0xProject/0x-monorepo#1012, 0xProject/0x-monorepo#1002 close the issue by using `staticcall`, which protects against reentrancy in calls to wallets and signature validator contract. Also all functions which access the `currentContextAddress` verifier are protected by the `nonReentrant` mutex modifier.

Description

A malicious maker can empty a taker's account of all tokens. The order does not need to involve a malicious token.

The attack uses reentry via `executeTransaction()`.

1. The attacker creates a maker order and signs it with `SignatureType.Wallet` signature type.
2. The taker signs a transaction, to take that order.
3. Some `sender` (or another account of the attacker) submits that transaction using `executeTransaction()`. This will write the taker's address to `currentContextAddress`.
4. The attacker's wallet contract re-enters to `setSignatureValidatorApproval()`, and sets a contract which always returns `true`.
5. The attacker can then submit any order for the victim with `SignatureType.Validator`.

The attack can be modified to execute any order filling function on the Exchange on behalf of the taker.

Note that, despite the comment, this check doesn't prevent calls to other functions in `Exchange.sol`.

packages/contracts/src/2.0.0/protocol/Exchange/MixinTransactions.sol:L60-L64

```
        // Prevent reentrancy
        require(
            currentContextAddress == address(0),
            "REENTRANCY_ILLEGAL"
        );
```

Recommendations

The developers have indicated plans for these mitigations, which we are evaluating:

1. use `pragma experimental 0.5.0`. This change would make all of the `view` functions use static calls, which automatically limits this specific attack vector to malicious tokens
2. Add a mutex to all variations of `fillOrder`. This will add 10k gas to each fill, but this will be significantly reduced after EIP1087/1283

3.2 MixinSignatureValidator: Insecure signature validator SignatureType.Caller

Severity	Status	Link	Remediation Comment
Critical	Closed	issues/44	the issue has been fixed with 0xProject/0x-monorepo#1015 by removing the SignatureType.Caller signature validation option from MixinSignatureValidator.

Description

MixinSignatureValidator has a signature validator called `SignatureType.Caller` that allows an order to be valid if `order.makerAddress == msg.sender`. If the `SignatureType.Caller` is set, no actual signature verification is performed for the order. An attacker could attack contracts that have:

1. ERC20/ERC721 tokens.
2. approved the Exchange proxy contracts to make transfers on their behalf.
3. a function that allows users to make calls to the exchange trading functions.

A contract that has the above listed properties is the Forwarder contract. It requires a ZRX token balance to function properly and it also approves the Exchange proxy contracts to make transfers on its behalf. An attacker can create an order to obtain ZRX (in exchange for, e.g., 1 wei), using SignatureType.Caller with the Forwarder contract as the maker. If such an order is used in a call to `Forwarder::marketSellOrderWithEthAsync`, the Forwarder address will be `msg.sender` when calling `fillOrder`, the signature will be seen as valid and the order will be filled.

The following order is a POC for the attack (full details in forwarder_malicious.ts.txt). The Forwarder will process the order and trade with itself as it is both the taker and the maker. The Forwarder will end up with exactly the same balance after the Exchange contract has settled the order and Forwarder will transfer the takerAmount to itself and makerAmount to the attacker.

```
Order {
  makerAssetData: ZRX_ASSET_DATA
  takerAssetData: WETH_ASSET_DATA
  makerAddress: address of Forwarder contract
```

```
    takerAddress: 0x0
    makerAmount: as many ZRX tokens as the Forwarder contract owns
    takerAmount: 1 (WETH)
    ...
  }
  Signature: SignatureType.Caller
```

Remediation

Removing `SignatureType.Caller` is the most important counter measure. Assuming that msg.sender has created the order because `order.makerAddress == msg.sender` is unsafe not just for the Forwarder but any contract that calls into the Exchange contract and that has a token balance. Proper signature recovery and validation needs to be performed by all variations of `SignatureType` .

Forwarder should check every order that `order.makerAddress` is not the Forwarder address itself.

Exchange check that the `order.makerAddress` id not `order.takerAddress` before processing an order.

3.3 Use of outdated multisig wallet

Severity	Status	Link	Remediation Comment
Medium	Open	issues/56	The issue is currently under review

Description

The repository includes a multisig wallet implementation, based on the well tested and widely used Gnosis Multisig. However, the wallet code contained in the project's contracts repository is outdated.

For example it contains the `if ... throw` pattern, which has been replace in the up to date code with `require()` .

Remediation

If possible, use the most recent wallet codebase. Ensure the additional features in `MultiSigWalletWithTimeLock.sol` pass testing with this updated code.

Caveat: if this code corresponds to a wallet which was deployed with v1 of the protocol, it would be safer to continue using that prior deployment.

3.4 AssetProxyOwner: Insufficient Testing

Severity	Status	Link	Remediation Comment
Medium	Open	issues/54	The issue is currently under review

Description

AssetProxyOwner is not thoroughly tested. Specifically, the following test cases are missing, among others not listed here:

1. The test cases only seem to check the scenario where `REQUIRED_APPROVALS == owners.length`, as opposed to the more likely and general scenario where `REQUIRED_APPROVALS < owners.length`.
2. `executionTransaction` should be callable be anyone, not just owners
3. `executionTransaction` should execute at most once for a transaction
4. `revokeConfirmation` not tested at all
5. `confirmTransaction` should throw if an owner tries to confirms multiple times.

3.5 AssetProxyOwner: timelocked transactions affected by changing wallet parameters

Severity	Status	Link	Remediation Comment
Medium	Closed	issues/52	Issue ignored because this functionality was intended.

Description

Increasing the `required` parameter of AssetProxyOwner by invoking the `changeRequirement` function prevents the execution of a transaction that was previously fully confirmed and is currently under timelock. Even if the transaction eventually receives the additional confirmations needed for the new `required` value, its timelock will be reset by `confirmTransaction` and it will need to wait another 2 weeks to get executed.

Similarly, increasing the `secondsTimeLocked` parameter by invoking `changeTimeLock` increases the timelock of unexecuted, fully confirmed transactions that have been scheduled to be executable at the 2 week mark after their last confirmation.

Evaluate whether this is the intended behavior, and if so, document it so the wallet members understand the consequences of changing these parameters on unexecuted, fully confirmed transactions.

3.6 AssetProxyOwner: removeAuthorizedAddressAtIndex requires multiple confirmations

Severity	Status	Link	Remediation Comment
Medium	Closed	issues/51	Issue ignored because this functionality was intended.

Description

AssetProxyOwner allows the `removeAuthorizedAddressAtIndex` function to be executed without being subject to the time lock. However, it is still subject to `required` number of confirmations, which may be too slow in the situation where a security bug is discovered. Consider adding an immediate, temporary pause functionality that requires fewer (or even a single) confirmation to remove an insecure authorized address.

3.7 LibBytes: Insufficient Testing

Severity	Status	Link	Remediation Comment
Medium	Closed	issues/49	Confirmed that the specific test cases listed in the issue have been mitigated in 0xProject/0x-monorepo#1039, however we did not review the entire file. This issue was created only as a spot check, and thus there are possible other missing test cases.

Description

LibBytes is not thoroughly tested which is concerning especially due to its extensive use of inline assembly. Specifically, the following test cases are missing, among others not listed here:

1. `readBytes4` when index > 0 and b.length >= index + 4
2. `readBytes4` when index > 0 and b.length < index + 4

3.8 LibBytes: readBytes4 does not adhere to spec

Severity	Status	Link	Remediation Comment
Medium	Closed	issues/48	Fixed with 0xProject/0x-monorepo#1039 by using the index parameter as the offset, rather than 0.

Description

The specification of the `readBytes4` function in LibBytes.sol is to "read an unpadded bytes4 value from a position in a byte array". However, the implementation ignores the `index` parameter and instead only ever returns the bytes4 at index 0, regardless of the value of the `index` parameter.

3.9 AssetProxyOwner: readBytes4 does not adhere to spec

Severity	Status	Link	Remediation Comment
Medium	Closed	issues/47	Fixed with 0xProject/0x-monorepo#1041 by removing the function from this file.

Description

The specification of the `readBytes4` function in AssetProxyOwner.sol is to "read an unpadded bytes4 value from a position in a byte array". However, the implementation ignores the `index` parameter and instead only ever returns the bytes4 at index 0, regardless of the value of the `index` parameter.

3.10 MixinAuthorizable: Insufficient Testing

Severity	Status	Link	Remediation Comment
Medium	Open	issues/46	The issue is currently under review

Description

The `authorizable.ts` file tests the MixinAuthorizable contract. The file uses the `address` variable as the parameter for the unit test cases of the contract functions. However, this `address` variable is always set to `owner` because of line 21. Thus the test file only covers scenarios where the owner is authorizing himself, and does not test the more common scenario where he is authorizing another, different address.

Remediation

Set the `address` variable to a fresh account, different from `owner` .

3.11 ERC721Proxy: Insufficient Testing

Severity	Status	Link	Remediation Comment
Medium	Open	issues/43	The issue is currently under review

Description

ERC721Proxy is not thoroughly tested which is concerning especially due to its extensive use of inline assembly. Specifically, the following test cases are missing, among others not listed here:

1. The `should throw if transferring 0 amount of a token` test case and the `should throw if transferring > 1 amount of a token` test case and the `should throw if allowances are too low` test case do not check that the NFT ownership is unchanged.
2. Fallback function should revert if a function selector is used besides the one for "transferFrom(bytes,address,address,uint256)".
3. No test case for when assetData has extra data (beyond the minimum 68 bytes), which, according to the specification, is possible: "The ERC721Proxy does not enforce strict length checks for assetData, which means that extra data may be appended to this field with any arbitrary encoding. Any extra data will be ignored by the ERC721Proxy but may be used in external contracts interacting with the Exchange contract. Relayers that do not desire this behavior should validate the length of all assetData fields contained in orders before acceptance."

3.12 ERC721Proxy: fallback function silently fails

Severity	Status	Link	Remediation Comment
Medium	Closed	issues/40	Fixed with 0xProject/0x-monorepo#1012 by adding a revert statement when a bad function selector is used.

Description

The ERC721Proxy fallback does not revert if an incorrect function selector is used. Instead, it silently fails, which is inconsistent with the rest of the fallback's behavior which is to revert on error, and also inconsistent with the IAssetProxy interface specification "Either succeeds or throws".

Based on the spec, a client of IAssetProxy would assume that, because a (malformed) transaction sent to the IAssetProxy did not throw, that the token transfer succeeded (when it didn't) and may incorrectly notify the user that his order was fulfilled.

Remediation

If the sender calls a function besides "transferFrom(bytes,address,address,uint256)", the fallback function should revert.

3.13 ERC20Proxy: fallback function silently fails

Severity	Status	Link	Remediation Comment
Medium	Closed	issues/39	Fixed with 0xProject/0x-monorepo#1012 by adding a revert statement when a bad function selector is used.

Description

The ERC20Proxy fallback does not revert if an incorrect function selector is used. Instead, it silently fails, which is inconsistent with the rest of the fallback's behavior which is to revert on error, and also inconsistent with the IAssetProxy interface specification "Either succeeds or throws".

Based on the spec, a client of IAssetProxy would assume that, because a (malformed) transaction sent to the IAssetProxy did not throw, that the token transfer succeeded (when it didn't) and may incorrectly notify the user that his order was fulfilled.

Remediation

If the sender calls a function besides "transferFrom(bytes,address,address,uint256)", the fallback function should revert.

3.14 ERC20Proxy: Insufficient testing

Severity	Status	Link	Remediation Comment
Medium	Open	issues/34	The issue is currently under review

Description

ERC20Proxy is not thoroughly tested which is concerning especially due to its extensive use of inline assembly. Specifically, the following test cases are missing, among others not listed here:

1. The `should throw if requesting address is not authorized` test case and the `should throw if allowances are too low` test case do not check that the balances are unchanged.
2. The `should successfully transfer tokens` test case and the `should do nothing if transferring 0 amount of a token` test case does not check that all addresses (besides maker and taker) have not changed.
3. The only token implementation tested is `DummyERC20Token` whose `transferFrom` method returns false on failure. The tests need to use other token implementations such as ones that revert on failure or ones that have the missing return type issue

described in https://medium.com/coinmonks/missing-return-value-bug-at-least-130-tokens-affected-d67bf08521ca.

4. No test case for "unlimited allowance".
5. No test case for when `iszero(returndatasize)` is true after the `token.transferFrom` call.
6. No test case for when `eq(returndatasize, 32)` is false after the `token.transferFrom` call.
7. Fallback function should revert if a function selector is used besides the one for "transferFrom(bytes,address,address,uint256)".
8. No test case for when assetData has extra data (beyond the minimum 36 bytes), which, according to the specification, is possible: "The ERC20Proxy does not enforce strict length checks for assetData, which means that extra data may be appended to this field with any arbitrary encoding. Any extra data will be ignored by the ERC20Proxy but may be used in external contracts interacting with the Exchange contract. Relayers that do not desire this behavior should validate the length of all assetData fields contained in orders before acceptance."

3.15 ERC721Token: inaccurate isContract function

Severity	Status	Link	Remediation Comment
Medium	Open	issues/23	The issue is currently under review

Description

The function `isContract` verifies if a calling address is a contract or a human actor. The current implementation if used as a security control could be compromised by a contract that calls the `checkAndCallSafeTransfer` from the constructor at initialisation.

packages/contracts/src/2.0.0/tokens/ERC721Token/ERC721Token.sol:L376-L406

```
    function checkAndCallSafeTransfer(
        address _from,
        address _to,
        uint256 _tokenId,
        bytes _data)
        internal
        returns (bool)
    {
        if (!isContract(_to)) {
            return true;
        }
        bytes4 retval = IERC721Receiver(_to).onERC721Received(_from, _tokenId, _c
        return (retval == ERC721_RECEIVED);
    }
```

```
    function isContract(address addr)
        internal
        view
        returns (bool)
    {
        uint256 size;
        // XXX Currently there is no better way to check if there is a contract i
        // than to check the size of the code at that address.
        // See https://ethereum.stackexchange.com/a/14016/36603
        // for more details about how this works.
        // TODO Check this again before the Serenity release, because all address
        // contracts then.
        assembly { size := extcodesize(addr) }  // solium-disable-line security/r
        return size > 0;
    }
}
```

Remediation

Do not use the `isContract` function as a security control.

3.16 AssetProxyOwner: accepts ETH

Severity	Status	Link	Remediation Comment
Minor	Open	issues/55	The issue is currently under review

Description

The fallback function in AssetProxyOwner inherited from MultiSigWallet is payable. Thus AssetProxyOwner accepts ETH. Evaluate whether having ETH possibly stored in AssetProxyOwner is acceptable.

3.17 AssetProxyOwner duplicates code for readBytes4 function

Severity	Status	Link	Remediation Comment
Minor	Closed	issues/50	Fixed with 0xProject/0x-monorepo#1041 by removing the function and importing LibBytes.

Description

The AssetProxyOwner contract has a `readBytes4` function implementation which already exists in LibBytes. Avoid unnecessary code duplication to minimize errors.

Remediation

Remove the `readBytes4` function from AssetProxyOwner and instead import LibBytes.

3.18 Outdated compiler version

Severity	Status	Link	Remediation Comment
Minor	Closed	issues/45	0xProject/0x-monorepo#1041 upgrades pragma version to 0.4.24.

Description

Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues (see also https://github.com/ethereum/solidity/releases) that affect the current compiler version.

packages/contracts/src/2.0.0/protocol/AssetProxyOwner/AssetProxyOwner.sol:L18

```
pragma solidity 0.4.10;
```

Remediation

It is recommended to use at least version 0.4.23 of the Solidity compiler.

3.19 Use of this.balance in WETH9.sol

Severity	Status	Link	Remediation Comment
Minor	Open	issues/41	The issue is currently under review

Description

The WETH9 contract, which is used as the system's EtherToken uses the contract's ether balance to track the totalSupply of the token.

packages/contracts/src/2.0.0/tokens/EtherToken/WETH9.sol:L46-L48

```
    function totalSupply() public view returns (uint) {
        return this.balance;
    }
```

Recommendation

This saves gas by not having to track a `totalSupply` storage value with each deposit and withdrawal. However this `totalSupply` value need not be equal to the sum of all account balances in the contract. This may lead to unforeseen issues with other contracts calling on the `totalSupply()` function. We leave it to the developer to consider if this is an acceptable trade off.

3.20 ERC20Proxy: Reconsider use of inline assembly

Severity	Status	Link	Remediation Comment
Minor	Open	issues/36	The issue is currently under review

Description

The ERC20Proxy fallback function is written entirely in inline assembly. The rationale given was to optimize gas costs caused by unnecessary ABI encoding/decoding. In general, inline assembly is concerning from a security perspective because it bypasses compiler checks and inhibits human code reasoning.

Remediation

Consider minimizing the amount of inline assembly used. For example, the function selector check and the `authorized` map check on lines 37-61 likely do not need to be written in assembly from a gas savings standpoint.

3.21 ERC20Proxy: Unclear comments

Severity	Status	Link	Remediation Comment
Minor	Closed	issues/35	Fixed with 0xProject/0x-monorepo#1039 by adding more detailed comments.

Description

The following comment in ERC20Proxy is confusing to follow and can be written more clearly. For example, what does "It" refer to and what does "Params" refer to?

```
// The token address is found as follows:
// * It is stored at offset 4 in `assetData` contents.
// * This is stored at offset 32 from `assetData`.
// * The offset to `assetData` from Params is stored at offset
//   4 in calldata.
// * The offset of Params in calldata is 4.
// So we read location 4 and add 32 + 4 + 4 to it.
```

3.22 ERC20Proxy/ERC721Proxy: LibBytes imported but not used

Severity	Status	Link	Remediation Comment
Minor	Closed	issues/33	Fixed with 0xProject/0x-monorepo#1041 by removing the import.

Description

LibBytes is imported by ERC20Proxy.sol but never used.

Remediation

Remove the import.

3.23 LibBytes is imported but never used

Severity	Status	Link	Remediation Comment
Minor	Open	issues/27	The issue is currently under review

Description

This line appears to be unnecessary. The contract compiles without it, and I cannot see where it would be used.

packages/contracts/src/2.0.0/protocol/Exchange/MixinAssetProxyDispatcher.sol:L30

```
    using LibBytes for bytes;
```

Recommendation

Remove it.

3.24 Optimization: refine function visibilities in the Exchange for gas efficiency

Severity	Status	Link	Remediation Comment
Minor	Open	issues/26	The issue is currently under review

Description

The `fillOrder()` function is labelled `public` because it is intended to be called either by external accounts, or from other 'wrapper' functions such as `fillOrKillOrder()`, and `batchFillOrders()`.

Relative to `external` and `internal` functions, the `public` keyword is gas inefficient, because it can be run either by a `JUMP` from a function within the same contract or by a `CALL` from another contract. If called by a `JUMP` the arguments will be passed in memory. If called by a `CALL`, the arguments are in the call data.

Functions labelled `public` require extra code to handle both cases.

Remediation

Given the sensitivity to gas costs in this system, it would be more efficient to move the code of `fillOrder()` to an internal function (ie. `internalFillOrder()` which is called by the wrapper functions as well as `fillOrder()`.

Additionally, it would be more efficient to change visibility of wrapper functions from `public` to `external`, unfortunately there is a compiler issue preventing this with the v2 encoder.

I have confirmed small gas savings in principle with very simple code samples, but have not yet tested in the Exchange itself. Given the number and size of arguments required to complete an order I expect they would be non-trivial.

3.25 LibConstants: dynamic constructor initialisation

Severity	Status	Link	Remediation Comment
Minor	**Open**	issues/24	The issue is currently under review

Description

According to the specs the dynamic constructor should be removed before deploying the contracts to the mainnet. Manually removing the code is not a good coding practice, it is recommended to build in a fail safe to ensure the code changes will not be missed.

packages/contracts/src/2.0.0/protocol/Exchange/libs/LibConstants.sol:L41-L49

```
    // @TODO: Remove when we deploy.
    constructor (bytes memory zrxAssetData)
        public
    {
        ZRX_ASSET_DATA = zrxAssetData;
    }
```

```
}
// solhint-enable max-line-length
```

Remediation

Build in a fail safe that throws if the contracts are deployed on the mainnet without changing them beforehand. This could be achieved by checking the `block.number` to identify on which network the contracts are deployed. The mainnet is ahead of all the test networks and the following code could be used to prevent a successful launch on the mainnet without the code changes.

```
require(block.number <  6019600)
```

4 Threat Model

The creation of a Threat Model is beneficial when building smart contract systems, as it helps to understand the potential security threats, assess risk, and identify appropriate mitigation strategies. This is especially useful during the design and development of a contract system, as it allows to create a more resilient design which is more difficult to change post-development.

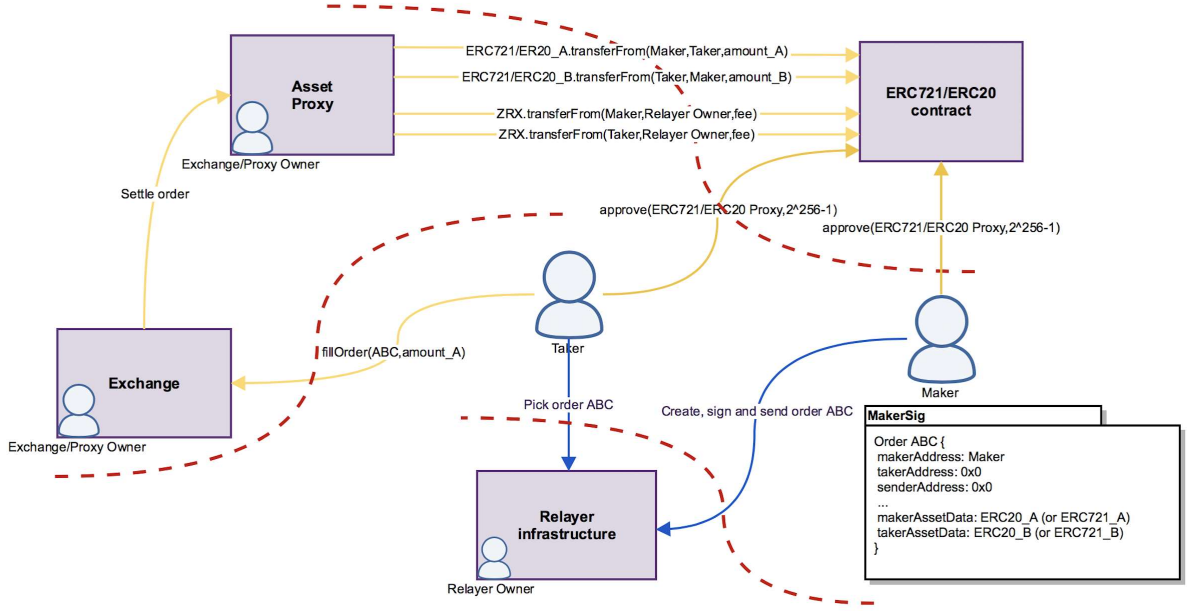
A Threat Model was created during the audit in order to analyze the attack surface of the contract system and to focus review and testing efforts on key areas that a malicious actor would likely attack. It consists of two parts: a high-level design diagram that visualizes the attack surface, and a list of threats that exist for the contract system.

The 0x contract system enables a fairly flexible decentralized exchange protocol that allows for several variants on how relayers and traders can interact with the system. The following Threat Model is an abstraction of the current system and has been compiled based on publicly available architecture and design specifications. It does not claim completeness and can be considered as a complementary analysis type to the manual code review and automated tool analysis.

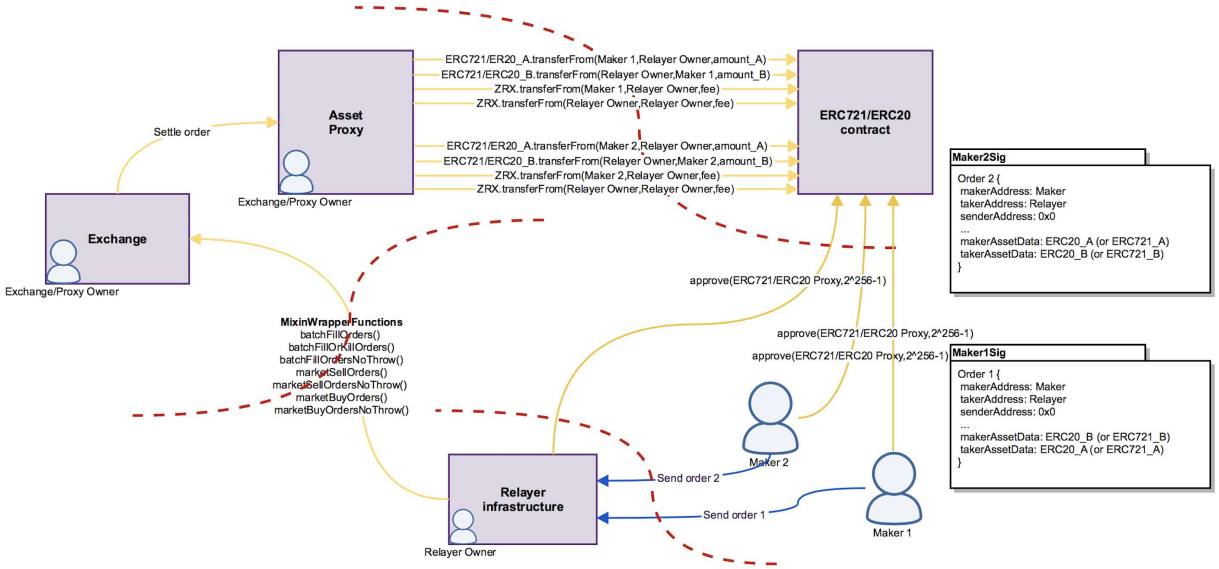
4.1 Overview

The scope of the Threat Model is to analyze the 0x protocol based on the two primary relayer strategies "Open Orderbook" and "Matcher". Other strategies such as the "Quote Provider" and the "Reserve Manager" were considered for analysis, but they do not have significantly different security properties than the primary relayer strategies.

Open Orderbook



Matcher



System components

The 0x system has multiple components that are either deployed and maintained by 0x or by a third party.

- **Exchange**: See chapter 1.3 System Overview.
- **Asset Proxy**: See chapter 1.3 System Overview.
- **ERC20/ERC721 contracts**: These contracts hold the token balances.
- **Relayer infrastructure**: This infrastructure can be composed of centralized and decentralized components.

Assets

Assets need to be protected, as potential threats could result in considerable loss for actors in the system. The following assets were identified:

- **ERC20/ERC721 tokens** are owned by Traders and Relayers.
- **Exchange/Proxy Owner accounts** are tied to a Gnosis MultiSig wallet.

Actors

The following actors take part in the 0x protocol:

- A **maker** creates and signs an order to trade token A for token B.
- A **taker** takes an order to trade token B for token A.
- A **trader** is either a maker or a taker.
- A **relayer owner** aggregates orders, provides liquidity, and may directly participate in trades as a maker or taker,
- An **exchange/proxy owner (or asset proxy owner)** adds and removes authorities to the Asset Proxy and registers and unregister an Asset Proxy in the Exchange.
- **Miners** include order processing and settlement transactions in new blocks.

4.2 Threat Analysis

The following table contains a list of identified threats that exist in the 0x protocol.

Threat	Relayer Strategy	Mitigation
The exchange/proxy owner is hacked, and their Ethereum private key is exposed. The traders and relayers could lose all ERC20/ERC721 tokens that they have approved.	Open Orderbook, Matcher	A newly added AssetProxyOwner has a two-week time-lock for approving new transactions. It gives Traders and Relayers the chance to remove their allowances in case an untrusted address is authorized for an AssetProxy.

Threat	Relayer Strategy	Mitigation
The exchange/proxy owner makes unauthorized transfers and misappropriates assets. The traders and relayers could lose all ERC20/ERC721 tokens that they have approved.	Open Orderbook, Matcher	At least the majority of the AssetProxyOwners need to collude in order to add a new AssetProxy. Currently 2 out of 3 AssetProxyOwners can authorize such an action. It is planned to increase the number of AssetProxyOwners to 6, at which time it will take at least 4 to authorize an action.
Miners and other traders could front-run any orders that have no taker specified.	Open Orderbook	Traders can specify a taker and choose a relayer with a Matcher strategy to prevent front-running attacks. Traders need to trust the chosen relayers to match orders fairly.
A relayer could get hacked and fake orders could get published through their trading interface.	Open Orderbook	Traders need to verify an order carefully before signing it.
Relayers could front-run orders that specify them as a taker.	Matcher	Traders can choose a relayer that uses an Open Orderbook strategy or choose other relayers that are more trusted.
Relayers could censor orders and prevent traders from participating	Matcher	Traders can switch to other relayers.

5 Test Coverage Measurement

Testing is implemented using the 0x project's own TypeScript-based tooling. **It has to be noted that the test coverage is only between 70-80% for some contracts.**

The Solidity-Coverage tool was used to measure the portion of the code base exercised by the test suite and identify areas with little or no coverage. Specific sections of the code where necessary test coverage is missing are included in Chapter 3 - Issue Detail.

It's important to note that "100% test coverage" is not a silver bullet. Our review also included an inspection of the test suite to ensure that testing included important edge cases.

The state of test coverage at the time of our review can be found in the coverage report directory.

Appendix 1 - File Hashes

Find the full list of files in the audit scope, including SHA1 hashes, in the Surya tool directory.

Appendix 2 - Severity

A.2.1 - Minor

Minor issues are generally subjective in nature or potentially deal with topics like "best practices" or "readability". Minor issues will in general not indicate an actual problem or bug in code.

The maintainers should use their own judgment as to whether addressing these issues improves the codebase.

A.2.2 - Medium

Medium issues are generally objective in nature but do not represent actual bugs or security problems.

These issues should be addressed unless there is a clear reason not to.

A.2.3 - Major

Major issues will be things like bugs or security vulnerabilities. These issues may not be directly exploitable or may require a certain condition to arise in order to be exploited.

Left unaddressed, these issues are highly likely to cause problems with the operation of the contract or to lead to a situation which allows the system to be exploited in some way.

A.2.4 - Critical

Critical issues are directly exploitable bugs or security vulnerabilities.

Left unaddressed, these issues are highly likely or guaranteed to cause major problems or potentially a full failure in the operations of the contract system.

Appendix 3 - Disclosure

ConsenSys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) -- on its GitHub account (https://github.com/ConsenSys). CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.