0x ERC1155Proxy Audit

0 July 2019 Update

As of July 2019, there is a new Solidity-based version of the `ERC1155Proxy` contract. ConsenSys Diligence has conducted an audit of this new contract and found it to be functionality equivalent to the original.

The only security concern with the new code is the use of `LibBytes.sliceDestructive`, which is written in assembly and takes a hard dependency on the memory layout for `bytes` arrays. Inline assembly is inherently risky, but most of the risk here is that future versions of Solidity might change the way `bytes` arrays are represented in memory. This is a maintenance problem rather than a concern with the current code.

The Solidity alternative would require copying the data byte by byte, which would increase gas costs linearly with the size of the data. The assembly solution was deemed worth the risk because it reduces this to a constant cost in a relatively simple way. Due to a number of teams' dependencies on the current memory layout, we believe it is unlikely that Solidity will break this in the future in a minor release. Explicit tests for memory layout are being added by the 0x team to ensure that such changes would be immediately detected.

1 Summary

ConsenSys Diligence conducted a security audit on the new 0x `ERC1155Proxy` contract. 0x is a decentralized exchange protocol, and the `ERC1155Proxy` adds multi-token support in the form of the emerging ERC-1155 standard.

- **Project Name:** 0x ERC1155Proxy contract
- **Client Name:** 0x
- **Client Contact:** Amir Bandeali
- **Lead Auditor:** Steve Marx
- **Co-auditors:** Sergii Kravchenko
- **Date:** 2019-05-20

2 System Overview

A typical order in an exchange involves two transfers: one from the order maker to the order taker, and one in the opposite direction. Each transfer can be described by the following parameters:

- a *from* address
- a *to* address
- an *asset* to be transferred
- the *amount* to be transferred

In the 0x exchange, assets are described by *asset data* byte sequences, which allows for quite a bit of flexibility. In a simple token exchange, the asset data simply describes what token contract should be invoked, but more elaborate assets can be described.

An ERC-1155 trade is an example of a more complex asset. The asset data for an ERC-1155 transfer actually specifies multiple individual assets in arbitrary proportions that should be transferred. The asset data for an ERC-1155 transfer is as follows:

- the *address* of an ERC-1155 compatible contract
- an array of asset *ids*
- an array of *values*, representing the number of units of each asset to be transferred

The new 0x `ERC1155Proxy` contract is a translation layer that receives a transfer with the above asset data, multiplies each member of the `values` array by the `amount` specified in the transfer, and translates that data into a call to `safeBatchTransferFrom` on an ERC-1155 compatible contract.

2.1 Detailed Design

The `ERC1155Proxy` is a single contract, written largely in Solidity assembly for efficiency. The following tables, adapted from comments in the code, describe the translation that is performed from the contract's input to the `safeBatchTransferFrom` call it ultimately makes:

Input call data

Area	Offset (**)	Length	Contents
Header	0	4	function selector
Params		4 * 32	function parameters:
	4		1. offset to assetData (*)
	36		2. from
	68		3. to
	100		4. amount
Data			assetData:
	132	32	assetData Length
	164	(see below)	assetData Contents

Asset data

Area	Offset	Length	Contents

Area	Offset	Length	Contents
Header	0	4	assetProxyId
Params		4 * 32	function parameters:
	4		1. address of ERC1155 contract
	36		2. offset to ids (*)
	68		3. offset to values (*)
	100		4. offset to data (*)
Data			ids:
	132	32	1. ids Length
	164	a	2. ids Contents
			values:
	164 + a	32	1. values Length
	196 + a	b	2. values Contents
			data
	196 + a + b	32	1. data Length
	228 + a + b	c	2. data Contents

Call data for `safeBatchTransferFrom`

Area	Offset (**)	Length	Contents
Header	0	4	safeBatchTransferFrom selector
Params		5 * 32	function parameters:
	4		1. from address
	36		2. to address
	68		3. offset to ids (*)
	100		4. offset to values (*)
	132		5. offset to data (*)
Data			ids:
	164	32	1. ids Length

Area	Offset (**)	Length	Contents
	196	a	2. ids Contents
			values:
	196 + a	32	1. values Length
	228 + a	b	2. values Contents
			data
	228 + a + b	32	1. data Length
	260 + a + b	c	2. data Contents

(*): offset is computed from start of function parameters, so offset by an additional 4 bytes in the call data.

(**): the Offset column is computed assuming no call data compression; offsets in the data area are dynamic and should be evaluated in real-time.

3 Audit Scope

This audit covered just the `ERC1155Proxy` contract, located in the following file:

File	SHA-1 hash
ERC1155Proxy.sol	7c8cf8de642403ac884bf4bbaa7e20d4ea902e91

The audit team evaluated that the contract is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three broad categories:

1. **Security:** Identifying security related issues within the contract.
2. **Architecture:** Evaluating the system architecture through the lens of established smart contract best practices.
3. **Code quality:** A full review of the contract source code. The primary areas of focus include:
    - Correctness
    - Readability
    - Scalability
    - Code complexity
    - Quality of test coverage

4 Security Specification

This section describes, **from a security perspective**, the expected behavior of the system under audit. It is not a substitute for documentation. The purpose of this section is to identify specific security properties that were validated by the audit team.

4.1 Actors

The relevant actors are as follows:

- 0x exchange maintainers
- Traders participating in the exchange
- ERC-1155 contract maintainers

4.2 Trust Model

In any smart contract system, it's important to identify what trust is expected/required between various actors. For this audit, we established the following trust model:

- The 0x platform does not need to trust traders nor the assets they're trading.
- Traders do not need to trust each other.

4.3 Important Security Properties

The following is a non-exhaustive list of security properties that were verified in this audit:

- Only authorized callers can invoke the `ERC1155Proxy` contract.
- A malicious ERC-1155 contract cannot harm traders or the 0x platform itself.
- Malicious traders cannot harm the 0x platform.
- Traders are expected to determine that a trade is desirable before accepting it. This includes verifying that the ERC-1155 contract involved works as expected.
- That said, a malicious trader should not be able to force or trick another trader into an asset transfer they don't want.

5 Key Observations/Recommendations

- The code is written in assembly. This makes the contract harder to read, but it allows for significant efficiency gains.
- The contract has low complexity with very few branches.
- The code is straightforward and well commented.
- Test coverage is good, including edge cases and negative tests for error conditions.
- Only one major issue was identified and is described in the following section.

6 Issues

Each issue has an assigned severity:

- **Minor** issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- **Medium** issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- **Major** issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- **Critical** issues are directly exploitable security vulnerabilities that need to be fixed.

The following table contains all the issues discovered during the audit, ordered based on their severity.

Chapter	Issue Title	Issue Status	Severity
6.1	Asset data manipulation can lead to unexpected trade outcomes	Closed	Major
6.2	Check for invalid offsets and lengths	Closed	Minor
6.3	Consider using `safeTransferFrom` for single-asset transfers	Closed	Minor
6.4	Copying more data than necessary	Closed	Minor
6.5	For consistency and simplicity, `assetDataOffset` should account for the function selector	Closed	Minor

6.1 Asset data manipulation can lead to unexpected trade outcomes

Severity	Status	Remediation Comment
Major	Closed	Fixed in https://github.com/0xProject/0x-monorepo/pull/1837.

Description

The `ERC1155Proxy` doesn't perform any validation on the asset data. This is presumably because it's up to traders to decide whether they like a proposed trade or not. However, even if traders examine the ABI-encoded asset data carefully, they can end up performing an unexpected trade due to a combination of data manipulation by the contract and a lack of validation around ABI encoded values.

Example

Suppose the `ids` and `values` provided in the asset data are identical because their offsets in the ABI-encoded asset data are the same:

byte range	contents
0-4	assetProxyId
4-36	address of ERC1155 contract
36-68	128 (offset to ids)
68-100	128 (offset to values)

Even a trader who is diligent about checking the ABI-encoded asset data will see that this data properly decodes to a reasonable trade. E.g. token IDs [1, 2] and values [1, 2].

Now if they trade with an `amount` of 2, they might expect to trade token IDs [1, 2] with amounts [2, 4] because each value is multiplied by the specified amount.

However, due to the offsets pointing to the same data, they'll actually trade token IDs [2, 4] with amounts [2, 4]. The following loop is doing multiplication in place and thus affecting both arrays:

code/contracts/asset-proxy/contracts/src/ERC1155Proxy.sol:L187-L210

```
for { let tokenValueOffset := valuesBegin }
    lt(tokenValueOffset, valuesEnd)
    { tokenValueOffset := add(tokenValueOffset, 32) }
{
    // Load token value and generate scaled value
    let tokenValue := mload(tokenValueOffset)
    let scaledTokenValue := mul(tokenValue, amount)

    // Revert if `amount` != 0 and multiplication resulted in an overflow
    if iszero(or(
        iszero(amount),
        eq(div(scaledTokenValue, amount), tokenValue)
    )) {
        // Revert with `Error("UINT256_OVERFLOW")`
        mstore(0, 0x08c379a000000000000000000000000000000000000000000000000000000000)
        mstore(32, 0x0000002000000000000000000000000000000000000000000000000000000000
        mstore(64, 0x0000001055494e543235365f4f564552464c4f5700000000000000000000000000
        mstore(96, 0)
        revert(0, 100)
    }

    // There was no overflow, update `tokenValue` with its scaled counterpart
    mstore(tokenValueOffset, scaledTokenValue)
}
```

Note that it's also possible to only partially overlap the IDs and values and thus only multiply some of them. It's also possible to cause the `data` array to be multiplied instead.

This issue is partially mitigated by the fact that the function selector (hardcoded), the `ERC1155` contract address (stored safely on the stack early on), and the `from` and `to` parameters (copied directly from call data *after* the loop) cannot be manipulated.

Remediation

We recommend doing some validation of the ABI-encoded asset data. Specifically, each different field in the data should have to occupy non-overlapping regions in the call data. With such validation in place, simple ABI decoding of the asset data should reveal exactly what will be traded.

6.2 Check for invalid offsets and lengths

Severity	Status	Remediation Comment
Minor	Closed	Fixed in https://github.com/0xProject/0x-monorepo/pull/1837. No check was added for walking past the end of the passed-in call data, but this is handled at the EVM level by just returning zeros, which seems harmless.

Description

If the `values` offset is near the end of call data, you can end up reading some bogus values (e.g. the function selector as some asset value). In some sense, this is just what the data says, but note that Solidity has some sanity checks that don't let this sort of thing happen. E.g. although at the EVM level you can read call data beyond `calldatasize` and get zeros, Solidity when decoding an array checks the length and just refuses.

It would make sense for this code to perform similar checks.

Remediation

Perhaps there should be some sanity checks around offsets and lengths. E.g. offsets should be within `calldatasize`, array lengths shouldn't overflow when converted to byte lengths, and adding the byte length to the offset should also not exceed `calldatasize`.

6.3 Consider using `safeTransferFrom` for single-asset transfers

Severity	Status	Remediation Comment
Minor	Closed	From the client: We intend on continuing to use batch transfers for this implementation. Depending on how it gets used, we may create a separate single transfer version in the future.

Description

ERC-1155 contract has two different functions for transferring items/tokens: `safeBatchTransferFrom` for performing multiple transfers inside one transaction and `safeTransferFrom` for a single transfer. They emit different events and call different callback functions.

The `ERC1155Proxy` only uses `safeBatchTransferFrom`, even for single transfers, which we assume are the most common transfers. This function is less efficient for single transfers and emits *batch* events and calls *batch* callbacks.

Remediation

Consider differentiating between single and batch transfers and calling the corresponding functions of ERC-1155 contract.

6.4 Copying more data than necessary

Severity	Status	Remediation Comment
Minor	Closed	No longer an issue in https://github.com/0xProject/0x-monorepo/pull/1837.

Description

The following code copies some data to memory at bytes 32-68 that's not actually needed:

code/contracts/asset-proxy/contracts/src/ERC1155Proxy.sol:L156-L165

```
// This corresponds to the beginning of the Data Area for Table #3.
// Computed by:
// 4 (function selector)
// + assetDataOffset
// + 32 (length of assetData)
calldatacopy(
    32,                         // aligned such that "offset to ids" is at the correc
    add(36, assetDataOffset),   // beginning of asset data contents
    assetDataLength             // length of asset data
)
```

Later on, memory at bytes 0-68 is overwritten:

code/contracts/asset-proxy/contracts/src/ERC1155Proxy.sol:L218-L225

```
mstore(0, 0x2eb2c2d600000000000000000000000000000000000000000000000000000000)

// Copy `from` and `to` fields from Table #1 to Table #3
```

```
calldatacopy(
    4,          // aligned such that `from` and `to` are at the correct location for
    36,         // beginning of `from` field from Table #1
    64          // 32 bytes for `from` + 32 bytes for `to` field
)
```

There's no need to copy those extra 36 bytes. The intent of the code is clearer if those bytes are not being copied.

Remediation

In the initial copy, just start at offset 68 and copy 36 bytes less:

```
calldatacopy(
    68, // aligned such that "offset to ids" is at the correct location for Table htt
    add(72, assetDataOffset), // asset data starting at "offset to ids"
    sub(assetDataLength, 36) // length of asset data after the first 36 bytes
)
```

Note that the subtraction allows for an integer underflow that doesn't matter... the caller can specify an arbitrary `assetDataLength` anyway.

6.5 For consistency and simplicity, `assetDataOffset` should account for the function selector

Severity	Status	Remediation Comment
Minor	Closed	Fixed in https://github.com/0xProject/0x-monorepo/pull/1837.

Description

The following code has to deal with adding `4` bytes for the function selector twice:

code/contracts/asset-proxy/contracts/src/ERC1155Proxy.sol:L148-L165

```
// Load offset to `assetData`
let assetDataOffset := calldataload(4)

// Load length in bytes of `assetData`, computed by:
// 4 (function selector)
// + assetDataOffset
let assetDataLength := calldataload(add(4, assetDataOffset))

// This corresponds to the beginning of the Data Area for Table #3.
// Computed by:
// 4 (function selector)
```

```
    // + assetDataOffset
    // + 32 (length of assetData)
    calldatacopy(
        32,                         // aligned such that "offset to ids" is at the correc
        add(36, assetDataOffset),   // beginning of asset data contents
        assetDataLength             // length of asset data
    )
```

It's simpler to account for the function selector once at variable initialization:

```
let assetDataOffset := add(4, calldataload(4)) // account for function selector here
let assetDataLength := calldataload(assetDataOffset) // no add(4, ...)
calldatacopy(32, add(32, assetDataOffset), assetDataLength) // the more natural 32 in
```

This would also be more consistent with the way `valuesOffset` is handled:

code/contracts/asset-proxy/contracts/src/ERC1155Proxy.sol:L180

```
let valuesOffset := add(mload(100), 4) // add 4 for calldata offset
```

Remediation

Add 4 to `assetDataOffset` up front and simplify the subsequent calculations.

Appendix 1 - 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.

Releases

No releases published

Packages

No packages published