SOLIDIFIED

Audit Report for Transak - November 24, 2022

Summary

Audit Report prepared by Solidified covering Transak's Decentraland integration.

Process and Delivery

Two (2) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on November 1, 2022, and the results are presented here.

Audited Files

The source code has been supplied in the following source code repository:

Repo: https://github.com/Transak/nft-connector
Commit hash: `2bcb794cc838768762a689e712498afde6a48e4f`

```
src
├── Interactor.sol
├── adaptors
│   └── Decentraland.sol
├── interfaces
│   ├── IDecentralandCollectionStore.sol
│   ├── IERC721BaseCollectionV2.sol
│   └── IMarketPlaceV2.sol
└── libraries
    ├── DecentralandDataTypes.sol
    └── GetterTypes.sol
```

Intended Behavior

The code base implements a Transak adaptor for Decentraland.

Findings

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Criteria	Status	Comment
Code complexity	Low	-
Code readability and clarity	Medium	The contract mixes up the terminology from Decentraland. What is referred to as asset ID is the token ID there and vice-versa. This can be confusing and is error-prone (see issue #1).
Level of Documentation	Low	-
Test Coverage	Medium	-

Issues Found

Solidified found that the Transak contracts contain no critical issues, 1 major issue, 1 minor issue, and 0 informational notes.

We recommend issues are amended, while informational notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	Decentraland.sol: Function getNFTData returns wrong values when itemId > 0	Major	Fixed
2	Decentraland.sol: encodeAssetId returns invalid IDs for large numbers	Minor	Fixed

Critical Issues

No critical issues have been found.

Major Issues

1. Decentraland.sol: Function getNFTData returns wrong values when itemId > 0

The function `getNFTData` first calls `decodeAssetId(assetId)` to get the `itemId` and `tokenId`. Then, `IERC721BaseCollectionV2(nftAddress).decodeTokenId(tokenId)` is called on this retrieved `tokenId` and the result is used to overwrite the `itemId`. This step is wrong, because what is referred to in Transak's codebase as `assetId`, is the `tokenId` in the `ERC721BaseCollectionV2` contract. This can be seen in the corresponding contract. `encodeTokenid` / `decodeTokenId` contain exactly the same logic as Transak's `encodeAssetId` / `decodeAssetId`. However, what is referred to as the `tokenId` in Transak's codebase, is the issued ID in `ERC721BaseCollectionV2`.
Therefore, the issued ID is currently passed to `IERC721BaseCollectionV2(nftAddress).decodeTokenId` in order to get the `itemId`. Because this is a 216 bit number, the result (returned `itemId`) will always be 0. Therefore, the result happens to be correct when the `itemId` of the requested token is 0 (which is always the case in the current test cases). But the following test for instance fails because the item ID is not 0:

```
    function testGetNFTDifferentItemID() public {
        uint256 assetId = decentralandAdapter.encodeAssetId(67, 2);
        assertEq(assetId,
21062458333711437339583605536734086463779019080109822250862195513 9);
        address alternativeNft = 0x00E0EFaec8a56918104e88C4839C1d7A656a4355;
```

```
        NFTDetails memory details = decentralandAdapter.getNFTData(alternativeNft,
assetId);
        assertEq(details.priceDetails.price, uint256(0), "Incorrect price fetch");
        assertEq(details.priceDetails.token,
address(decentralandAdapter.acceptedToken()), "Incorrect token address");
        assertEq(details.name, "PEA  COLLECTION", "Incorrect name");
        assertEq(details.symbol, "DCL-PEACOLLECTION", "Incorrect symbol");
        assertEq(details.tokenURI,
"https://peer.decentraland.org/lambdas/collections/standard/erc721/137/0x00e0efaec8a56
918104e88c4839c1d7a656a4355/2/67", "Incorrect tokenURI");
        assertEq(details.metadata, "1:w:T-SHIRT::upper_body:BaseFemale,BaseMale",
"Incorrect metadata");
        assertEq(details.contentHash, "QmYHkUccrK1e1j6iJ76zH97jMQpv15WTuV7y6AnFRWdatv",
"Incorrect contentHash");
    }
```

Recommendation

Pass the correct ID to the functions that expect the token ID and do not extract the item ID a second time:

```
@@ -203,10 +203,9 @@ contract Decentraland is Ownable, ReentrancyGuard, IERC721Receiver {
             // There will be no `tokenURI` before the mint.
             tokenURI = "";
         } else { // Secondary market or maybe asset is free floated.
-            (itemId,) = IERC721BaseCollectionV2(nftAddress).decodeTokenId(tokenId);
             // It will return 0 if there is no order exists
-            priceOfNFT = secondarySaleContract.orderByAssetId(nftAddress, tokenId).price;
-            tokenURI = IERC721Metadata(nftAddress).tokenURI(tokenId);
+            priceOfNFT = secondarySaleContract.orderByAssetId(nftAddress, assetId).price;
+            tokenURI = IERC721Metadata(nftAddress).tokenURI(assetId);
             orderStatus = priceOfNFT > 0 ? OrderStatus.Live : OrderStatus.NotLive;
         }
```

After these changes, all tests (the original ones with `itemId` 0 and the newly added one) will pass.

It is also recommended to use the same terminology as `ERC721BaseCollectionV2` for these IDs, because the contracts become much easier to read and extend.

Minor Issues

2. Decentraland.sol: encodeAssetId returns invalid IDs for large numbers

When the `tokenId` that is passed to `encodeAssetId` is larger than `type(uint216).max`, the returned ID will be invalid because the `tokenId` will set some of the upper 40 bits, which are intended for the `itemId`. Similarly, an `itemId` that is larger than `type(uint40).max` will overflow, resulting in undesired results.

Recommendation
Consider restricting the size of these parameters to `type(uint216).max` & `type(uint40).max`. Note that this is also done in `ERC721BaseCollectionV2` such that the functions do not produce invalid IDs.

Informational Notes

No informational notes have been found.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Transak or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Oak Security GmbH