# 237.https://stackoverflow.com/questions/72093394/error-vm-exception-while-processing-transaction-reverted-with-reason-string-e

**T:**Error: VM Exception while processing transaction: reverted with reason string 'ERC721: transfer caller is not owner nor approved'

**Q:**i try to write NFT marketplace and i have problem with reselling items from user that buy item from owner and new, third user.I use next.js + openzeppelin + hardhat + infura and metamask.Creating and buying NFT is successful, all params on sell-item.js page getting correctly.  
  
This is my function in NFTMarket.sol:  
  
function resellToken(address nftContract, uint256 tokenId, uint256 price) public payable { require(idToMarketItem[tokenId].owner == msg.sender, "Only item owner can perform this operation"); require(msg.value == listingPrice, "Price must be equal to listing price"); idToMarketItem[tokenId].sold = false; idToMarketItem[tokenId].price = price; idToMarketItem[tokenId].seller = payable(msg.sender); idToMarketItem[tokenId].owner = payable(address(this)); \_itemsSold.decrement(); IERC721(nftContract).transferFrom(msg.sender, address(this), tokenId); }  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
This is my web page:  
  
import { useEffect, useState } from 'react'import { ethers } from 'ethers'import { useRouter } from 'next/router'import axios from 'axios'import Web3Modal from 'web3modal'import { nftaddress, nftmarketaddress} from '../config'// import NFT from '../artifacts/contracts/NFT.sol/NFT.json'import NFTMarket from '../artifacts/contracts/NFTMarket.sol/NFTMarket.json'export default function ResellNFT() { const [formInput, updateFormInput] = useState({ price: '', image: '' }) const router = useRouter() const { id, tokenUri } = router.query const { image, price } = formInput useEffect(() => { fetchNFT() }, [id]) async function fetchNFT() { if (!tokenUri) return const meta = await axios.get(tokenUri) updateFormInput(state => ({ ...state, image: meta.data.image })) } async function listNFTForSale() { if (!price) return const web3Modal = new Web3Modal() const connection = await web3Modal.connect() const provider = new ethers.providers.Web3Provider(connection) const signer = provider.getSigner() const priceFormatted = ethers.utils.parseUnits(formInput.price, 'ether') const marketContract = new ethers.Contract(nftmarketaddress, NFTMarket.abi, signer) //const tokenContract = new ethers.Contract(nftaddress, NFT.abi, provider) let listingPrice = await marketContract.getListingPrice() listingPrice = listingPrice.toString() let transaction = await marketContract.resellToken(nftaddress, id, priceFormatted, { value: listingPrice }) await transaction.wait() router.push('/') } return ( <div className="flex justify-center"> <div className="w-1/2 flex flex-col pb-12"> <input placeholder="Asset Price in Eth" className="mt-2 border rounded p-4" onChange={e => updateFormInput({ ...formInput, price: e.target.value })} /> { image && ( <img className="rounded mt-4" width="350" src={image} /> ) } <button onClick={listNFTForSale} className="font-bold mt-4 bg-pink-500 text-white rounded p-4 shadow-lg"> List NFT </button> </div> </div> )}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Error code:  
  
 eth\_estimateGas Contract call: NFTMarket#resellToken From: 0x1cbd3b2770909d4e10f157cabc84c7264073c9ec To: 0x5fbdb2315678afecb367f032d93f642f64180aa3 Value: 0.025 ETH Error: VM Exception while processing transaction: reverted with reason string 'ERC721: transfer caller is not owner nor approved' at NFT.transferFrom (@openzeppelin/contracts/token/ERC721/ERC721.sol:156) at NFTMarket.resellToken (contracts/NFTMarket.sol:89) at EthModule.\_estimateGasAction (E:\маркетплейс\polygon-ethereum\node\_modules\hardhat\src\internal\hardhat-network\provider\modules\eth.ts:425:7) at HardhatNetworkProvider.\_sendWithLogging (E:\маркетплейс\polygon-ethereum\node\_modules\hardhat\src\internal\hardhat-network\provider\provider.ts:139:22) at HardhatNetworkProvider.request (E:\маркетплейс\polygon-ethereum\node\_modules\hardhat\src\internal\hardhat-network\provider\provider.ts:116:18) at JsonRpcHandler.\_handleRequest (E:\маркетплейс\polygon-ethereum\node\_modules\hardhat\src\internal\hardhat-network\jsonrpc\handler.ts:188:20) at JsonRpcHandler.\_handleSingleRequest (E:\маркетплейс\polygon-ethereum\node\_modules\hardhat\src\internal\hardhat-network\jsonrpc\handler.ts:167:17) at Server.JsonRpcHandler.handleHttp (E:\маркетплейс\polygon-ethereum\node\_modules\hardhat\src\internal\hardhat-network\jsonrpc\handler.ts:52:21)  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
UPDATE:This is my smart contract of the marketplace  
  
// SPDX-License-Identifier: MITpragma solidity 0.8.2;import "@openzeppelin/contracts/utils/Counters.sol";import "@openzeppelin/contracts/token/ERC721/ERC721.sol";import "@openzeppelin/contracts/security/ReentrancyGuard.sol";contract NFTMarket is ReentrancyGuard { using Counters for Counters.Counter; Counters.Counter private \_itemIds; Counters.Counter private \_itemsSold; address payable owner; uint256 listingPrice = 0.025 ether; constructor() { owner = payable(msg.sender); } struct MarketItem { uint256 itemId; address nftContract; uint tokenId; address payable seller; address payable owner; uint256 price; bool sold; } mapping(uint256 => MarketItem) private idToMarketItem; event MarketItemCreated ( uint256 indexed itemId, address indexed nftContract, uint256 indexed tokenId, address seller, address owner, uint256 price, bool sold ); function getListingPrice() public view returns (uint256) { return listingPrice; } function createMarketItem( address nftContract, uint256 tokenId, uint256 price ) public payable nonReentrant { require(price > 0, "Price must be at least 1 wei"); require(msg.value == listingPrice, "Price must be equal to listing price"); \_itemIds.increment(); uint256 itemId = \_itemIds.current(); idToMarketItem[tokenId] = MarketItem( itemId, nftContract, tokenId, payable(msg.sender), payable(address(0)), price, false ); IERC721(nftContract).transferFrom(msg.sender, address(this), tokenId); emit MarketItemCreated( itemId, nftContract, tokenId, msg.sender, address(0), price, false ); } function resellToken(address nftContract, uint256 tokenId, uint256 price) public payable { require(idToMarketItem[tokenId].owner == msg.sender, "Only item owner can perform this operation"); require(msg.value == listingPrice, "Price must be equal to listing price"); idToMarketItem[tokenId].sold = false; idToMarketItem[tokenId].price = price; idToMarketItem[tokenId].seller = payable(msg.sender); idToMarketItem[tokenId].owner = payable(address(this)); \_itemsSold.decrement(); IERC721(nftContract).transferFrom(msg.sender, address(this), tokenId); } function createMarketSale( address nftContract, uint256 itemId ) public payable nonReentrant { uint price = idToMarketItem[itemId].price; uint tokenId = idToMarketItem[itemId].tokenId; require(msg.value == price, "Please sumbit the asking price in order to complete the purchase"); idToMarketItem[itemId].seller.transfer(msg.value); IERC721 (nftContract).transferFrom(address(this), msg.sender, tokenId); idToMarketItem[itemId].owner = payable(msg.sender); idToMarketItem[itemId].sold = true; \_itemsSold.increment(); payable(owner).transfer(listingPrice); } function fetchMarketItems() public view returns (MarketItem[] memory) { uint ItemCount = \_itemIds.current(); uint unsoldItemCount = \_itemIds.current() - \_itemsSold.current(); uint currentIndex = 0; MarketItem[] memory items = new MarketItem[](unsoldItemCount); for (uint i = 0; i < ItemCount; i++) { if (idToMarketItem[i + 1].owner == address(0)) { uint currentId = idToMarketItem[i + 1].itemId; MarketItem storage currentItem = idToMarketItem[currentId]; items[currentIndex] = currentItem; currentIndex += 1; } } return items; } function fetchMyNFTs() public view returns (MarketItem[] memory) { uint totalItemCount = \_itemIds.current(); uint itemCount = 0; uint currentIndex = 0; for (uint i = 0; i < totalItemCount; i++) { if (idToMarketItem[i + 1].owner == msg.sender) { itemCount += 1; } } MarketItem[] memory items = new MarketItem[](itemCount); for (uint i = 0; i < totalItemCount; i++) { if (idToMarketItem[i + 1].owner == msg.sender) { uint currentId = idToMarketItem[i + 1].itemId; MarketItem storage currentItem = idToMarketItem[currentId]; items[currentIndex] = currentItem; currentIndex += 1; } } return items; } function fetchItemsCreated() public view returns (MarketItem[] memory) { uint totalItemCount = \_itemIds.current(); uint itemCount = 0; uint currentIndex = 0; for (uint i = 0; i < totalItemCount; i++) { if (idToMarketItem[i + 1].seller == msg.sender) { itemCount += 1; } } MarketItem[] memory items = new MarketItem[](itemCount); for (uint i = 0; i < totalItemCount; i++) { if (idToMarketItem[i + 1].seller == msg.sender) { uint currentId = idToMarketItem[i + 1].itemId; MarketItem storage currentItem = idToMarketItem[currentId]; items[currentIndex] = currentItem; currentIndex += 1; } } return items; }}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
This is the nfc creation code  
  
import { useState } from 'react'import { ethers } from 'ethers'import { create as ipfsHttpClient } from 'ipfs-http-client'import { useRouter } from 'next/router'import Web3Modal from 'web3modal'import {MPLayout} from '../../components/MPLayout'const client = ipfsHttpClient('https://ipfs.infura.io:5001/api/v0')import { nftaddress, nftmarketaddress} from '../../config'import NFT from '../../artifacts/contracts/NFT.sol/NFT.json'import NFTMarket from '../../artifacts/contracts/NFTMarket.sol/NFTMarket.json'export default function CreateItem () { const [fileUrl, setFileUrl] = useState(null) const [formInput, updateFormInput] = useState({ price: '', name: '', description: '' }) const router = useRouter() async function onChange(e) { const file = e.target.files[0] try { const added = await client.add( file, { progress: (prog) => console.log('received: ${prog}') } ) const url = `https://ipfs.infura.io/ipfs/${added.path}` setFileUrl(url) } catch (e) { console.log(e) } } async function createItem() { const { name, description, price } = formInput if (!name || !description || !price || !fileUrl) return const data = JSON.stringify({ name, description, image: fileUrl }) try { const added = await client.add(data) const url = `https://ipfs.infura.io/ipfs/${added.path}` /\* after file is uploaded to IPFS, return the URL to use it in the transaction \*/ createSale(url) } catch (error) { console.log('Error uploading file: ', error) } } async function createSale(url) { const web3Modal = new Web3Modal() const connection = await web3Modal.connect() const provider = new ethers.providers.Web3Provider(connection) const signer = provider.getSigner() let contract = new ethers.Contract(nftaddress, NFT.abi, signer) let transaction = await contract.createToken(url) let tx = await transaction.wait() let event = tx.events [0] let value = event.args[2] let tokenId = value.toNumber() const price = ethers.utils.parseUnits(formInput.price, 'ether') contract = new ethers.Contract(nftmarketaddress, NFTMarket.abi, signer) let listingPrice = await contract.getListingPrice() listingPrice = listingPrice.toString() transaction = await contract.createMarketItem(nftaddress, tokenId, price, { value: listingPrice}) await transaction.wait() router.push('/') } return ( <MPLayout title={'Create Item'}> <div className="flex justify-center"> <div className="w-1/2 flex flex-col pb-12"> <input placeholder="Asset Name" className="mt-8 border rounded p-4" onChange={e => updateFormInput({ ...formInput, name: e.target.value })} /> <textarea placeholder="Asset Description" className="mt-2 border rounded p-4" onChange={e => updateFormInput({ ...formInput, description: e.target.value })} /> <input placeholder="Asset Price in Eth" className="mt-2 border rounded p-4" onChange={e => updateFormInput({ ...formInput, price: e.target.value })} /> <input type="file" name="Asset" className="my-4" onChange={onChange} /> { fileUrl && ( <img className="rounded mt-4" width="350" src={fileUrl} /> ) } <button onClick={createItem} className="font-bold mt-4 bg-pink-500 text-white rounded p-4 shadow-lg" > Create Digital Asset </button> </div> </div> </MPLayout> ) }  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]

**C1:**Why is createMarketSale() never called? I expect it to be called by the first buyer who is going to resell it.

**C2:**Hello @Ilya. Any updates on this question? I have exactly the same issue (and I am using the same sample code from the certification :-) ) setApprovalForAll() is here, the market address is properly set as an argument in the migration file #2... Your help will be appreciated !!

3 **Answer**

**A1:**It seems that somewhere else in your code after initially creating the nft, the nft is not actually owned by idToMarketItem[tokenId].owner but by someone else. If you're creating the nft with \_mint(...), your first parameter to \_mint is probably wrong. If this doesn't help, can you share the code where you create the nft and also the code where you initially set idToMarketItem[tokenId].owner?  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]

**C1:**I added a smart contract and the year of creation of the nft

**A2:**If you look at ERC721.sol you have this function  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
function transferFrom(address from, address to, uint256 tokenId ) public virtual override { require(\_isApprovedOrOwner(\_msgSender(), tokenId), "ERC721: transfer caller is not owner nor approved"); \_transfer(from, to, tokenId); }  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
You are calling this function but you are not satisfying the require stament:  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
require(\_isApprovedOrOwner(\_msgSender(), tokenId), "ERC721: transfer caller is not owner nor approved");  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
In smart contracts, if you are transfering from another address, you have to be authorized to do it. You cannot just go transfer something from any account. So before you call transferFrom, you have to call this first  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
function \_isApprovedOrOwner(address spender, uint256 tokenId) internal view virtual returns (bool) { require(\_exists(tokenId), "ERC721: operator query for nonexistent token"); address owner = ERC721.ownerOf(tokenId); return (spender == owner || isApprovedForAll(owner, spender) || getApproved(tokenId) == spender); }  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]

**A3:**As explained in this thread, the owner has to re-approve the NFT-market contract to change the ownership on his behalf. Add the code below in your NFT contract and call it from your FE before calling your resellToken() and voila !  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
 function giveResaleApproval(uint256 tokenId) public { require( ownerOf(tokenId) == msg.sender, "You must own this NFT in order to resell it" ); setApprovalForAll(contractAddress, true); return; }  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]