# 498.https://stackoverflow.com/questions/70705439/burning-deployed-erc-tokens-in-an-nft-mint-function-compiles-but-transaction

**T:**Burning Deployed ERC Tokens In an NFT Mint Function - Compiles, but Transaction Fails

**Q:**I am very new to Solidity, and have recently been working on trying to learn the ropes. For reference, I have been using code from this video (https://www.youtube.com/watch?v=tBMk1iZa85Y) as a primer after having gone through the basic crypto zombies tutorial series.  
  
I have been attempting to adapt the Solidity contract code presented in this video (which I had functioning just fine!) to require a Burn of a specified amount of an ERC-20 token before minting an NFT as an exercise for myself. I thought I had what should be a valid implementation which compiled in Remix, and then deployed to Rinkeby. I call the allowAccess function in Remix after deploying to Rinkeby, and that succeeds. But, when I call the mint function with the two parameters, I get: "gas estimation errored with the following message (see below). The transaction execution will likely fail. Do you want to force sending? execution reverted."  
  
If I still send the transaction, metamask yields "Transaction xx failed! Transaction encountered an error.".  
  
I'm positive it has to do with "require(paymentToken.transfer(burnwallet, amounttopay),"transfer Failed");", though I'm not sure what's wrong. Below is my entire contract code. I'm currently just interacting with the Chainlink contract on Rinkeby as my example, since they have a convenient token faucet.  
  
pragma solidity ^0.8.0;import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/ERC721.sol";import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/IERC20.sol";import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol";import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/Counters.sol";contract myNFTwithBurn is ERC721, Ownable { address externalTokenAddress = 0x01BE23585060835E02B77ef475b0Cc51aA1e0709; //Token Type to burn on minting uint256 amounttopay = 5; //number of these tokens to burn IERC20 paymentToken = IERC20(externalTokenAddress); //my code: create an interface of the external token address burnwallet = 0x000000000000000000000000000000000000dEaD; //burn wallet using Counters for Counters.Counter; Counters.Counter private \_tokenIds; using Strings for uint256; // Optional mapping for token URIs mapping (uint256 => string) private \_tokenURIs; // Base URI string private \_baseURIextended; constructor() ERC721("NFTsWithBurn","NWB") { } function setBaseURI(string memory baseURI\_) external onlyOwner() { \_baseURIextended = baseURI\_; } function \_setTokenURI(uint256 tokenId, string memory \_tokenURI) internal virtual { require(\_exists(tokenId), "ERC721Metadata: URI set of nonexistent token"); \_tokenURIs[tokenId] = \_tokenURI; } function \_baseURI() internal view virtual override returns (string memory) { return \_baseURIextended; } function tokenURI(uint256 tokenId) public view virtual override returns (string memory) { require(\_exists(tokenId), "ERC721Metadata: URI query for nonexistent token"); string memory \_tokenURI = \_tokenURIs[tokenId]; string memory base = \_baseURI(); // If there is no base URI, return the token URI. if (bytes(base).length == 0) { return \_tokenURI; } // If both are set, concatenate the baseURI and tokenURI (via abi.encodePacked). if (bytes(\_tokenURI).length > 0) { return string(abi.encodePacked(base, \_tokenURI)); } // If there is a baseURI but no tokenURI, concatenate the tokenID to the baseURI. return string(abi.encodePacked(base, tokenId.toString())); }function allowAccess() public{ paymentToken.approve(address(this), 5000000); //This is my attempt to allow the contract access to the user's external tokens, in this case Chainlink (paymentToken)}function mintItem(address to, string memory tokenURI) public onlyOwner returns (uint256) { require(paymentToken.transfer(burnwallet, amounttopay),"transfer Failed"); //Try to transfer 5 chainlink to the burn wallet \_tokenIds.increment(); uint256 id = \_tokenIds.current(); \_mint(to, id); \_setTokenURI(id, tokenURI); return id; }}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
If anybody can at least point me to what I'm doing completely wrong in the code that I've added, please do! TIA!

1 **Answer**

**A1:**I'm not sure why are you trying to burn link in order to mint and nft but first check if the link code does not have a require that check if the destination address is the burn address if it has then burn the link is not possible and you should use any other erc20 maybe your own erc20, also your contract probably does not have any link and if you want to transfer the link from the user you should do this in the contract paymentToken.transferFrom(msg.sender,destinationAddress,amount) and if the user previously approve your contract you will able to send the tokens, and i suppose that the purpose of the allowAccess function is to make the user approve the contract to move the tokens that will never work, the approve function let's anyone that call it approve any address to move an amount of tokens, the thing is that to know who is approving to let other to move the tokens the function use msg.sender to explain how this work take a look at this example  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
let's say that your contract is the contract A and the link contract is the contract B  
  
now a user call allowAccess in the contract A, so here the msg.sender is the user because they call the function  
  
now internally this function call approve on contract B, here the contract A is the msg.sender, because the contract is who call the function  
  
so what allowAccess is really doing is making the contract approving itself to move their own tokens that I assume it doesn't have

**C1:**Thank jhonny! You were right on both parts. AllowAccess was allowing my CONTRACT access to its OWN funds, which were 0. I don't know why I was thinking the call to that function would just magically know that I meant the user's funds. I get it now, on the front-end code to interact with this, there would need to be an "Approve XYZ Token Transfers" button to approve the contract to spend the user's link, that reaches out direct to the chainlink contract. And, same thing goes for that paymentToken.transfer() function. It thought I was accessing the contract's tokens! Your fix worked perfect!

**C2:**i'm glad that help you, good luck