1. Reentrancy Attacks

Reentrancy attacks

In this assignment we'll create a *reentrancy attack* on a vulnerable smart contract.

Consensys has a good guide on reentrancy attacks.

There have been many reentrancy attacks in the real world.
The DAO Attack is the largest and most well-known example of a reentrancy attack, but there have been many others.

Assignment

In this assignment, you'll execute a reentrancy attack on a vulnerable contract.

We've prepared a contract "Bank.sol" which is vulnerable to a reentrancy attack through an ERC777 token withdrawal.

The Bank contract works a bit like wETH. Users can deposit ETH into the Bank contract, and in return they receive a new ERC777 token. At a later point, they can redeem the ERC777 token for the underlying ETH held in the contract.
The process goes like this:

1. Users call the "deposit()" function and send ETH to the contract. This increments the user's local balance inside the Bank contract.
2. Users who have a positive balance can "withdraw" ERC777 tokens by calling the claimAll() functions (this is the function that is vulnerable to a reentrancy attack).
3. Users can redeem their ERC777 tokens for ETH by calling the redeem() function

> If you were to build a contract like this in the real world, you would probably combine Steps 1 & 2, but that would eliminate the claimAll() function, and there is no simple opportunity for reentrancy in the deposit() function (since you must provide ETH with each call), thus this assignment would have to have a reentrancy vulnerability in the redeem() function.
> In that case, you would be stealing ETH from the contract (instead of stealing ERC777 tokens). This would mean that you