1 Reentrancy Attack

You are given the following smart contract:

```solidity
contract Token {
    mapping(address => uint) balances;
    uint tokenPrice = 10**18; // 1 ETH = 1 token = 10^18 Wei

    // The sender receives tokens based on the amount of ether send by the transaction
    function () public payable {
        // TODO1
    }

    // Returns the amount of tokens owned by the sender
    function balanceOf(address tokenHolder) public returns(uint){
        // TODO2
    }

    // Sets the sender’s balance to 0
    // Refunds the sender based on the sender’s balance
    function sell() public {
        // refund sender via call.value()
        uint tokens = balances[msg.sender];
        require(msg.sender.call.value(tokens * tokenPrice)());
        // update balance
        balances[msg.sender] = 0;
    }

    // Returns the amount of ether owned by the contract
```
The smart contract allows users to buy tokens (for the price of 1 ETH per token) by sending ether to the default fallback function.

Any user can check the balance of a user $U$ by calling the `balanceOf` function and passing the address of $U$ as an argument.

A user can sell all his/her tokens by calling the function `sell`. Tokens are sold for 1 ETH per token.

A user can check the amount of ether owner by the token contract by calling the function `eth`.

**Task 1**  Complete the three TODOs in the smart contract.

**Task 2**  Deploy your contract in Remix:

`https://remix.ethereum.org`

You can use the JavaScript VM environment to simulate the smart contract in the browser. Deploy the Token smart contract. Let three different users (i.e., addresses) buy tokens for 10 ETH each.

**Task 3**  Write down the source code of an attacker’s smart contract, who aims to steal all ETH owned by the Token smart contract:

```solidity
import "browser/Token.sol";

contract Attack {
    Token token; // address of the token contract
    ...
}
```

Perform the attack step-by-step on Remix and show the attacker’s balance.
2 Delegatecall Attack

You are given the following two smart contracts:

```solidity
class WalletLibrary {
    address owner;

    event Deposit(address sender, uint value);
    event Withdraw(uint value);

    function initWallet(address _owner) {
        owner = _owner;
    }

    function withdraw() public {
        if (msg.sender == owner) {
            owner.send(this.balance);
            Withdraw(this.balance);
        }
    }

    function () payable {
        if (msg.value > 0) {
            Deposit(msg.sender, msg.value);
        }
    }
}

class Wallet {
    address owner;
    address walletLibrary;

    event Deposit(address sender, uint value);

    function Wallet(address _owner, address _walletLibrary) {
        walletLibrary = _walletLibrary;
        walletLibrary.delegatecall(bytes4(sha3("initWallet(address)")), _owner);
    }

    function withdraw() public {
        walletLibrary.delegatecall(bytes4(sha3("withdraw()")));
    }

    function () public payable {
        if (msg.value > 0) {
            Deposit(msg.sender, msg.value);
        }
    }
}
```
else if (msg.data.length > 0)
    walletLibrary.delegatecall(msg.data);
}

The wallet library implements common functionality of a wallet. When the wallet contract is created, the user passes as arguments his/her address and the address of the wallet library. A user may deposit ether to a wallet by calling the default fallback function. A user can withdraw all ether owned by the wallet by calling the withdraw function.

**Task 1** Implement an attacker smart contract that allows an attacker to steal the ether owned by the wallet. The attacker should steal ether from a wallet that is owned by a different user.

contract AttackerWallet {
    ...
}

**Task 2** Perform the attack step-by-step on Remix and show the attacker’s balance.