BlockChain Architecture and Design – CA3

Name: yagnesh yerra

Roll no: 30

Reg.No: 12107821

Section: K21CS

Course code: CSC403

Question.4

Imagine you are doing a manual audit and you come across above code. Write a comprehensive report explaining the issue and the fix for the issue.

```
function transfer(address to, uint amount) external {
  if (balances[msg.sender] >= amount) {
    balances[to] += amount;
    balances[msg.sender] -= amount;
  }
}

function withdraw() external {
  uint256 amount = balances[msg.sender];
  (bool success,) = msg.sender.call{value: balances[msg.sender]}("");
  require(success);
  balances[msg.sender] = 0;
}
```

Solution:

Issues of the given code:

Issue 1: Lack of Proper Input Validation

- The transfer function does not validate the recipient address (to). Specifically:
  - Transfers to the zero address (address(0)) are not explicitly prevented.
  - Transfers to msg.sender (self-transfers) will unnecessarily modify storage and consume gas.

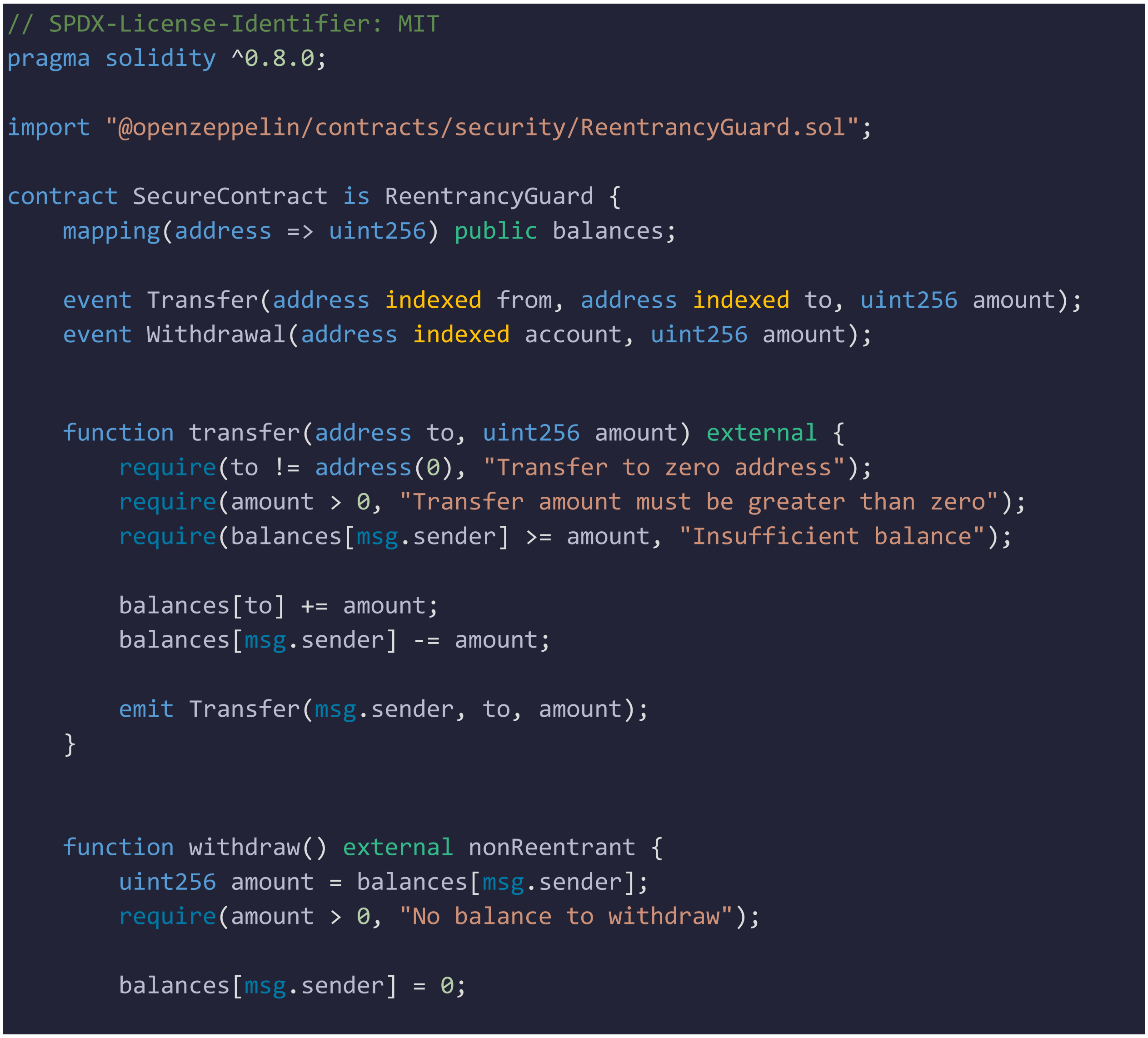
Issue 2: Reentrancy Vulnerability

- The function sends Ether to the caller before updating the balances[msg.sender] mapping. This sequence allows a malicious contract to re-enter the withdraw function and drain all Ether from the contract by repeatedly calling it before the balance is updated.

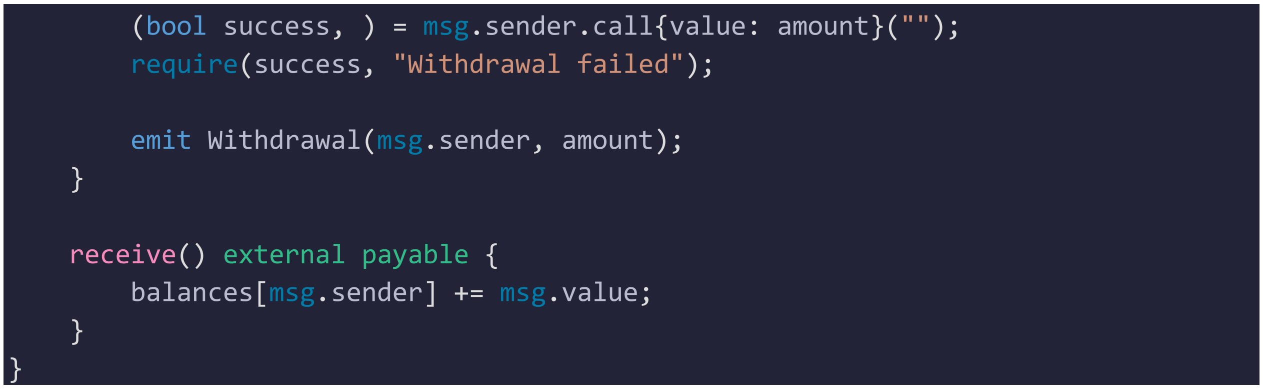
Issue 3: Lack of Validation for Successful Withdrawals

- The require(success) check is after the msg.sender.call{value: balances[msg.sender]}("") statement, meaning it does not prevent issues like reentrancy. The balance update should occur before the external call.

Code:

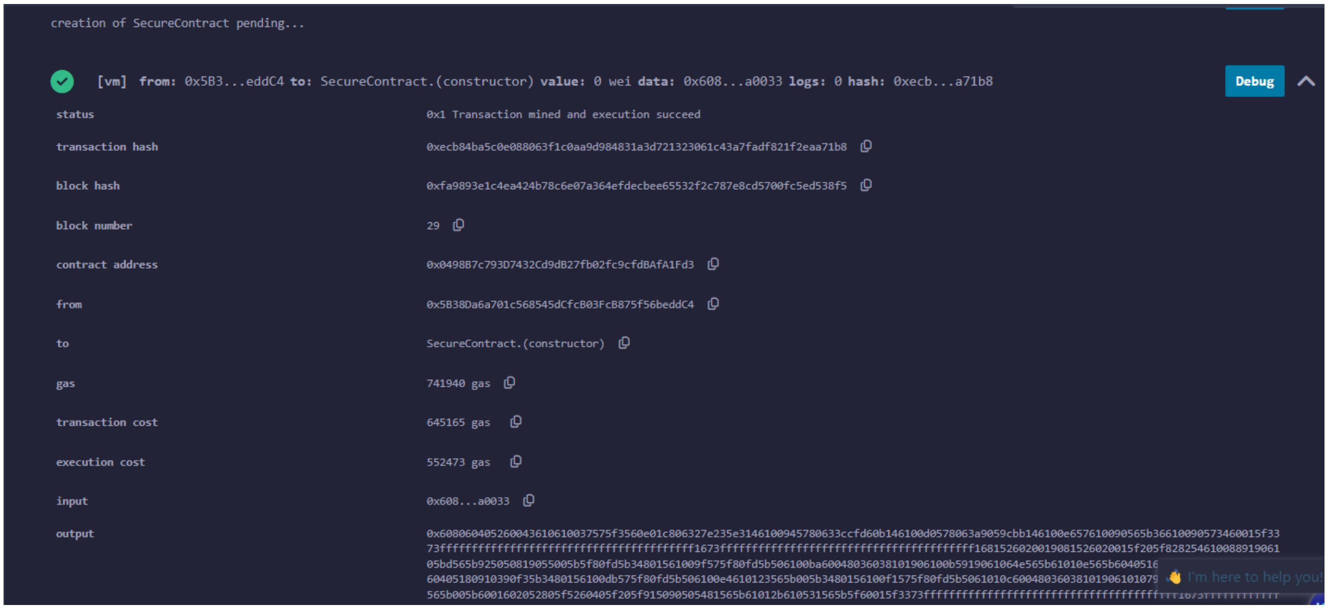


```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/security/ReentrancyGuard.sol";

contract SecureContract is ReentrancyGuard {
    mapping(address => uint256) public balances;

    event Transfer(address indexed from, address indexed to, uint256 amount);
    event Withdrawal(address indexed account, uint256 amount);


    function transfer(address to, uint256 amount) external {
        require(to != address(0), "Transfer to zero address");
        require(amount > 0, "Transfer amount must be greater than zero");
        require(balances[msg.sender] >= amount, "Insufficient balance");

        balances[to] += amount;
        balances[msg.sender] -= amount;

        emit Transfer(msg.sender, to, amount);
    }


    function withdraw() external nonReentrant {
        uint256 amount = balances[msg.sender];
        require(amount > 0, "No balance to withdraw");

        balances[msg.sender] = 0;
```



```
        (bool success, ) = msg.sender.call{value: amount}("");
        require(success, "Withdrawal failed");

        emit Withdrawal(msg.sender, amount);
    }

    receive() external payable {
        balances[msg.sender] += msg.value;
    }
}
```

Deployment:



```
creation of SecureContract pending...

[vm] from: 0x5B3...eddC4 to: SecureContract.(constructor) value: 0 wei data: 0x608...a0033 logs: 0 hash: 0xecb...a71b8
status                  0x1 Transaction mined and execution succeed
transaction hash        0xecb84ba5c0e088063f1c0aa9d984831a3d721323061c43a7fadf821f2eaa71b8
block hash              0xfa9893e1c4ea424b78c6e07a364efdecbee65532f2c787e8cd5700fc5ed538f5
block number            29
contract address        0x0498B7c793D7432Cd9dB27fb02fc9cfdBAfA1Fd3
from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                      SecureContract.(constructor)
gas                     741940 gas
transaction cost        645165 gas
execution cost          552473 gas
input                   0x608...a0033
output                  0x608060405260043610610037575f3560e01c806327e235e3146100945780633ccfd60b146100d0578063a9059cbb146100e657610090565b36610090573460015f33
                        73ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1681526020019081526020015f205f828254610088919061
                        05bd565b925050819055005b5f80fd5b34801561009f575f80fd5b506100ba6004803603810190610 0b5919061064e565b61010e565b6040516
                        60405180910390f35b3480156100db575f80fd5b506100e4610123565b005b3480156100f1575f80fd5b5061010c60048036038101906101079
                        565b005b6001602052805f5260405f205f915090505481565b61012b610531565b5f60015f3373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffff
```

Explanation of Code:

The `SecureContract` is a Solidity smart contract designed for securely handling Ether deposits, transfers, and withdrawals. It includes measures to protect against common vulnerabilities, such as reentrancy attacks, and provides transparency through event logging. Here's a breakdown of its main components:

Key Features:

Reentrancy Protection:

- The use of ReentrancyGuard ensures that the withdraw function is protected against reentrancy attacks, which is critical when sending Ether via .call.

Safe Ether Transfers:

- The call method is used for transferring Ether, which is the recommended approach to prevent issues with gas stipends in Solidity 0.8+.

Balance Update Before External Call:

- In the withdraw function, you update the user's balance to 0 before making the external call. This prevents reentrancy vulnerabilities, even without the ReentrancyGuard.

Input Validation:

- Proper require checks ensure that invalid operations, such as transferring to the zero address or withdrawing without a balance, are avoided.

Key Functions:

1. **State Variable: `balances`**:
   - A public mapping that tracks the Ether balance of each address interacting with the contract.
2. **Event Logging**:
   - **`Transfer`:** Logs details of Ether transfers between users.
   - **`Withdrawal`:** Logs Ether withdrawals made by users.
3. **`transfer` Function**:
   - Allows users to transfer Ether from their balance to another address.
   - Ensures the recipient address is valid, the transfer amount is positive, and the sender has sufficient balance.
   - Updates the balances of both the sender and recipient.
4. **`withdraw` Function**:
   - Enables users to withdraw their Ether balance from the contract to their wallet.
   - Uses the `nonReentrant` modifier from the `ReentrancyGuard` library to prevent reentrancy attacks.
   - Updates the user's balance to zero before transferring Ether.
5. **`receive` Function**:
   - A fallback function that allows the contract to accept Ether directly.
   - Updates the balance of the sender with the received Ether.
6. **Security Features**:

- **Reentrancy Protection:** The `nonReentrant` modifier prevents nested calls to the `withdraw` function.
- **Input Validation:** Ensures valid addresses and non-negative amounts for transactions.
- **Safe Ether Transfers:** Uses `.call` for transferring Ether to handle gas limitations safely.

This contract is designed to be simple yet secure, making it suitable for scenarios where users need to deposit Ether, transfer balances, and withdraw funds.