The Challenge

Welcome to the Smart Contract challenge. See below for a smart contract that is fairly short but has some fundamental flaws that could result in a significant loss. We want you to put your contact auditor hat on and find all the flaws you can find.

Here is what we want in your response to this challenge.

1. Create a report as if we were your client and we have hired you for this work.
2. Save your report as a PDF and attach to the email sent to the address below.
3. Provide details about the problems you find in the contract below.
4. Provide an explanation of what you would do differently. Your solution.
5. ***Optional:*** Provide an example of your proposed solution.

Format Example:

Problem: Race Conditions / Front Running. Two loops attempting to access the same variable simultaneously. The following two loops are poorly formed.

Solution: Prevent the 2nd loop from running until the first loop has finished.

Example: <provide example>

Here is the contract we would like you to evaluate. Good luck!

```
///////////////////////////////////////////////////////

// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.10;

import "@openzeppelin/contracts/utils/math/SafeMath.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

contract WalletSplitter is ReentrancyGuard {

   address payable private owner0; // gets 2/5 of fees + dust
   address payable private owner1; // gets 1/5 of fees
   address payable private owner2; // gets 1/5 of fees
   address payable private owner3; // gets 1/5 of fees
```

```
    constructor(address payable _owner0, address payable _owner1, address payable _owner2,
address payable _owner3) {
        owner0 = _owner0;
        owner1 = _owner1;
        owner2 = _owner2;
        owner3 = _owner3;
    }

    // to recieve native token
    receive() external payable {}

    function withdraw() public payable nonReentrant {
        // get balance of native token
        uint256 balance = address(this).balance;
        // calculate fees
        uint256 oneFifth = SafeMath.div(balance, 5);
        uint256 twoFifth = SafeMath.mul(2, oneFifth);
        uint256 dust     = SafeMath.mod(balance, 5);
        // owner0 gets 2/5 of fees + dust
        owner0.transfer(twoFifth + dust);
        // owner1,2,3 gets 1/5 fees
        owner1.transfer(oneFifth);
        owner2.transfer(oneFifth);
        owner3.transfer(oneFifth);
    }

    function withdrawToken(address _contract) public nonReentrant{
        IERC20 erc20 = IERC20(_contract);
        // get balance of tokens
        uint256 balance = erc20.balanceOf(address(this));
        // calculate fees
        uint256 oneFifth = SafeMath.div(balance, 5);
        uint256 twoFifth = SafeMath.mul(2, oneFifth);
        uint256 dust     = SafeMath.mod(balance, 5);
        // owner0 gets 2/5 of fees + dust
        erc20.transfer(owner0, (twoFifth + dust));
        // owner1,2,3 gets 1/5 fees
        erc20.transfer(owner1, oneFifth);
        erc20.transfer(owner2, oneFifth);
        erc20.transfer(owner3, oneFifth);
    }

    function updateOwner(address payable _newOwner) public returns(bool) {
        if (msg.sender == owner0) {
            owner0 = _newOwner;
            return true;
        }
        if (msg.sender == owner1) {
            owner1 = _newOwner;
            return true;
        }
        if (msg.sender == owner2) {
            owner2 = _newOwner;
            return true;
        }
        if (msg.sender == owner3) {
            owner3 = _newOwner;
            return true;
        }
        return false;
    }

}


/////////////////////////////////////////////////////
```