Exercise 4

Smart Contracts

Program Analysis for System Security and Reliability 2018
ETH Zurich

March 24, 2018

Problem 1. In this exercise, you will learn about the challenges of implementing a decentralized bank. We encourage you to code your contract in https://remix.ethereum.org, which enables you to compile and run your code.

Task 1 Complete the following contract that enables users to deposit ether (the Ethereum's currency) and withdraw all their ether.

```
contract SimpleBank {

  mapping(address => uint) balances;

function deposit(uint amount) payable {
  balances[msg.sender] += amount;
}

function withdraw() {
  msg.sender.call.value(balances[msg.sender])()
  balances[msg.sender] = 0;
}
```

Task 2 Your contract has been uploaded to the blockchain, and it became very popular. One day, your loyal user Alice reports that despite calling `withdraw`, she does not receive her ether. When you read the blockchain, you observe that Bob was able to withdraw all the users' ether that was stored in the contract. Describe a scenario that could lead to this situation.

Solution. Bob can exploit the `call` function, which passes the execution control to him, to call `withdraw` again, thereby forcing the contract to transfer ether to him (at the amount of ether he originally had), without executing the instruction that updates its balance to zero.

A scenario:

- Alice calls `deposit(50)`, which is executed on the blockchain.
- Bob calls `deposit(50)`, which is executed on the blockchain.
- Bob calls `withdraw`, which triggers `call` that invokes a function at Bob's wallet.
- In that function, Bob calls again `withdraw`.
- Since Bob's balance in SimpleBank has not been changed yet, this results in transferring another 50 ether to Bob. However, this time, the ether is actually Alice's.
- After the second transfer, the *nested* `withdraw` operation completes by updating Bob's balance to zero
- Finally, the *initial* `withdraw` operation completes by updating Bob's balance (again) to zero.

Task 3 Fix your contract to prevent this situation.

Solution. We will fix it by swapping the instructions order in `withdraw` and adding a variable to store the original amount of ether Bob has. Now, even if Bob follows the same scenario as before, the second transfer will transfer to him zero ether.

```
1
2 function withdraw() {
3   uint r = balances[msg.sender];
4   balances[msg.sender] = 0;
5   msg.sender.call.value(r)();
6 }
```

Task 4 After fixing your contract and deploying it on the blockchain, Alice still complains that despite calling `withdraw`, she does not receive her ether. This time when you read the contract, you see that Alice's ether is still in the contract, however your local data structure reports that her balance is zero. Describe a scenario that could lead to this situation.

Solution. If Alice's wallet throws an exception while it is being transferred ether from `withdraw`, then Alice will not see transferred ether, but her balance in `SimpleBank` nevertheless becomes zero.

A scenario:

- Alice calls `deposit(50)`, which is executed on the blockchain.
- Alice calls `withdraw`, which triggers `call` that invokes a function at Alice's wallet.
- In that function, due to some internal bug, an exception is thrown, which results in `call` returning 0.
- When `withdraw` resumes, it completes successfully, although Alice did not receive ether.

Task 5 Fix your contract to prevent this situation.

Solution. We will fix it checking the return value of `call` and if it threw an exception, we will throw an exception, too, thereby forcing the execution to revert without changing the balance.

```
1
2 function withdraw() {
3   uint r = balances[msg.sender];
4   balances[msg.sender] = 0;
5   if (msg.sender.call.value(r)() == 0)
6     throw;
7 }
```

Task 6 After the two unfortunate incidents, your contract has lost its popularity. To regain it, you post a puzzle online and write a contract with the function `submitSolution` that checks a solution and pays the first solver 10 ether. However, since you are low on ether, you do not want to pay 10 ether. Unfortunately, you do not know the solution to the puzzle you solved (you can only verify that a solution is correct). Describe how you can avoid paying the first solver 10 ether, without making it look suspicious.

Solution. We will be a miner, and when we observe a transaction that calls `submitSolution`, we will create a transaction from our address with the same solution and send it to ourselves (before executing the original transaction), thereby making ourselves the winner (this is known as *transaction reordering*).