

# 智能合约逆向心法2（案例篇）——34C3\_CTF题目分析续篇

转载

FLy\_鹏程万里 于 2018-11-23 09:50:45 发布 294 收藏

分类专栏：[【区块链】#智能合约安全](#)



[【区块链】](#) 同时被 2 个专栏收录

333 篇文章 15 订阅

订阅专栏



[智能合约安全](#)

35 篇文章 0 订阅

订阅专栏



## 前言

上周34C3\_CTF的文章分析得到了很好的反响，后台收到白帽子的反馈，说想了解整个完整的分析过程是怎样的。

因此这篇文章，主要关于该题从零到有的思路，顺带解释一些solidity的特性与语法。

## 题目解读

```
send 1505 szabo 457282 babbage 649604 wei
0x949a6ac29b9347b3eb9a420272a9dd7890b787a3
```

1. eth: 1505 szabo + 457282 babbage + 649604 Wei

从题目的提示中，我们可以看到一些以太坊的单位

#### Notice

eth使用了对社会做出伟大贡献的大牛名字作为单位，以下单位从小到大排序

Wei: eth的最小单位，戴伟密码学先驱，B-Money的提出者

Babbage: 查尔斯·巴贝奇，通用计算机之父。1 Babbage =  $10^3$  Wei = 1 KWei

Lovelace: 洛夫莱斯，计算机程序创始人。1 Lovelace =  $10^6$  Wei = 1 MWei

Shannon: 香农，信息论之父。1 Shannon =  $10^9$  Wei = 1 GWei

Szabo: 尼克·萨博，比特币概念的提出者。1 Szabo =  $10^{12}$  Wei = 1 Microether

Finney: 哈尔·芬尼，比特币最早支持者，当时中本聪第一笔转账的那个牛。1 Finney =  $10^{15}$  Wei = 1 Milliether

Ether: 最大的单位，就是平时说的eth所指的面值大小。1 ether =  $10^8$  Wei

```
PS C:\Users\doctor> python
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16)
Type "help", "copyright", "credits" or "license" for more
>>> 1505 * 10**12 + 457282 * 10**3 + 649604
1505000457931604
>>> (1505 * 10**12 + 457282 * 10**3 + 649604) / 10**18
0.001505000457931604
```

2. contract address:

0x949a6ac29b9347b3eb9a420272a9dd7890b787a3

给了 ETH 单位，又给了地址，那么题目的意思就可以理解成：send much money to this contract

#### 合约逆向

在上一篇文章中已经讲述了，怎么从合约字节码获得初步可读的伪代码，这里直接进入正题。

合约中总共有3个函数，分别是func\_00cc, Withdraw, Receive。

在上一篇中，讲过怎么根据交易的input data来找到对应调用的函数，这里就不再多说。因此本次的重点在于函数的功能解读。如果任何不解之处，可以提出疑惑，然后统一予以解答。

#### func\_00cc解读

首先看一下逆向出来的伪代码

```

if (var0 == 0x2a0f7696) {
  // part-I
  if (msg.value) {revert(memory[0x00:0x00]); }

  var var1 = 0x0081;

  // part-II
  var var2 = msg.data[0x04:0x24] & 0xffff;

  // part-III
  var1 = func_00CC(var2);
  var temp0 = memory[0x40:0x60];
  memory[temp0:temp0 + 0x20] = var1;
  var temp1 = memory[0x40:0x60];
  return memory[temp1:temp1 + (temp0 + 0x20) - temp1];
}

```

这个函数的流程如下所示：

1. no payable 判断
2. 获取用户输入的数据，做and操作，取最后的两个字节。
3. 使用用户参数调用func\_00cc，然后将返回值返回给用户，做一下变量代换就可以看出来了。

下面分析func\_00cc函数体：

```

function func_00CC(var arg0) returns (var r0) {
  var var0 = 0x00;

  // part-I
  if (arg0 & 0xffff != storage[0x01] & 0xffff) { return 0x00; }

  // part-II
  memory[0x00:0x20] = msg.sender;
  memory[0x20:0x40] = 0x02;
  return storage[keccak256(memory[0x00:0x40])];
}

```

1. 判断用户调用参数的最后两个字节的值是否等于slot[1]最后两个字节的值
2. 3行代码实际上就是做了一个，return var\_map[msg.sender]

#### Notice

storage可以理解成一个个连续的槽位(数组)，称为 slot[]，每个槽位可以存放32字节的数据

关于storage的进一步介绍包括变量寻址等内存布局，可以参考官方文档，或者之后的系列连载文章

综合上面代码，以及

```
function Withdraw(var arg0) {
    if (msg.sender != storage[0x00] & 0xffffffffffffffffffffffffffffffff) { revert(memory[0x00:
    //...省略不看
}
```

可以得出一个初步的信息，即合约的状态变量的声明情况

```
address var_addr;
bytes32 var_bytes;
mapping var_map;
```

可以看出，func\_cc 是用于返回flag的一个函数，返回flag的条件是输入数据的最后两个字节与 slot[1]的最后两个字节一致。通过

eth.getStorageAt(0x949a6ac29b9347b3eb9a420272a9dd7890b787a3, 1)，即可看到 slot1 的返回值。



### Receive解读

```
function Receive() {
    var var0 = 0x00;
    var var1 = var0;
    var var2 = 0x02;
    memory[memory[0x40:0x60] + 0x20:memory[0x40:0x60] + 0x20 + 0x20] = 0x00;
    var temp0 = memory[0x40:0x60];
    memory[temp0:temp0 + 0x20] = msg.value;
    var var3 = temp0 + 0x20;
    var temp1 = memory[0x40:0x60];
    var temp2;

    //part-I
    temp2, memory[temp1:temp1 + 0x20] = address(var2).call.gas(msg.gas - 0x646e)(memory[temp1:temp1 + v

    // part-II
    if (!temp2) { revert(memory[0x00:0x00]); }

    // part-III
    var temp3 = memory[memory[0x40:0x60]:memory[0x40:0x60] + 0x20] ~ storage[0x01];
    memory[0x00:0x20] = msg.sender;
    memory[0x20:0x40] = 0x02;
    storage[keccak256(memory[0x00:0x40])] = temp3;
}
```

函数流程如下，

1. part-I 之上的所有代码主要是为了调用call而做的事情。
2. 如果调用不成功，则回滚交易并结束。那么要转入的eth就是题目中所给出的那个值，将其换算好，发起一次交易就好了。

3. part-III 就是将flag（上一步调用函数的时候，返回的值）放入var\_map[msg.sender]中。

## 总结

综上所述，整个思路是：

1. 分析题干，获取信息
2. 逆向合约，获取伪代码
3. 分析伪代码，获得合约逻辑
4. 调用Receive函数，转入0.001505000457931604eth
5. 查看storage的slot[1]的值，获取其最后两个字节的值0xc1cb
6. 用0x2a0f7696c1cb作为input data向合约发起交易
7. 获得0x333443335f6772616e646d615f626f756768745f736f6d655f626974636f696e
8. 将十六进制转换成字节数组，并且打印出来，得34C3\_grandma\_bought\_some\_bitcoin

## 资料

### 题目地址：

<https://archive.aachen.ccc.de/34c3ctf.ccc.ac/challenges/index.html>

### 合约地址：

<https://etherscan.io/address/0x949a6ac29b9347b3eb9a420272a9dd7890b787a3>

### 反编译地址：

<https://ethervm.io/decompile?address=0x949A6aC29B9347B3eB9a420272A9DD7890B787A3>

### writeup：

<https://github.com/kuqadk3/CTF-and-Learning/blob/master/34c3ctf/crypto/chaingang/readme.md>

## 关于玄猫安全实验室

玄猫区块链安全实验室专注区块链安全领域，致力于提供区块链行业最专业的安全解决方案，团队成员来自于百度、阿里、360等国际顶尖安全团队，已为数十家交易所、电子钱包、智能合约等提供基础安全建设、渗透测试、漏洞挖掘、应急响应等安全服务。

玄猫安全实验室提供专业权威的智能合约审计服务、区块链专项应用评估、区块链平台安全评估等多项服务。