2018年全国大学生信息安全竞赛 CISCN Writeup

原创

aptx4869_li 于 2018-06-07 16:49:59 发布 10408 收藏 8
分类专栏： CTF 文章标签： 2018全国大学生信息安全竞赛 逆向 RE writeup CISCN

逆向：

RE:

题目文件：链接：点击打开链接 密码：hyzy

这题当时做的时候陷入了出题人的陷阱，上来找到了“关键部分”就开始逆，但是遇到了很多问题，因为有很多函数，而且篇幅都不小，而且每个函数还要调用其他一大堆函数，这就大大增加了工作量，后面实在看不下去了，就放弃了。

后面找到一些大佬的writeup看了看，原来是思路的问题：

总结一下：

逆向赛题一般工作量不会大到让你逆不了，一下锁定了十七八个函数肯定是有问题的，换思路

在这里就记录一下自己的学习历程，偏基础一点，适用于刚入门的，比如我。

看题目：

首先用IDA分析：

main函数：（代码分析见图中注释）

```
signed __int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  bool v3; // cf@1
  bool v4; // zf@1
  const char *v5; // rsi@1
  signed __int64 v6; // rcx@1
  char *v7; // rdi@1
  __int64 *v8; // rbx@5
  char *v9; // rbp@5
  void *v10; // rax@6
  char *v11; // rsi@8
  const char *v12; // r13@9
  signed int v13; // ebp@9
  char *v14; // rsi@10
  signed __int64 result; // rax@15
  __int64 v16; // rbx@23
  __int64 v17; // [sp+0h] [bp-B8h]@5
  void *dest; // [sp+8h] [bp-B0h]@9
  const char *v19; // [sp+10h] [bp-A8h]@13
  const char *v20; // [sp+18h] [bp-A0h]@14
  char buf; // [sp+20h] [bp-98h]@1
  char s; // [sp+26h] [bp-92h]@8
  char v23; // [sp+40h] [bp-78h]@1
  __int64 v24; // [sp+88h] [bp-30h]@1

  v24 = *MK_FP(__FS__, 40LL);
  read(0, &buf, 0x20uLL);
  v23 = 0;
  v5 = "CISCN{";
  v6 = 6LL;
  v7 = &buf;
  do
  {
    if ( !v6 )
      break;                                        // 判断前几个字符是否为CISCN{
    v3 = *v5 < (unsigned __int8)*v7;
    v4 = *v5++ == *v7++;
    --v6;
  }
  while ( v4 );
  if ( (!v3 && !v4) == v3 )
  {
    v8 = &v17;
    v9 = (char *)&v17;
    do
    {
      v10 = malloc(0x20uLL);
      *(_QWORD *)v9 = v10;
```

```
49      if ( !v10 )
50      {
51        result = 0xFFFFFFFFLL;
52        goto LABEL_23;
53      }
54      v9 += 8;
55    }
56    while ( &buf != v9 );
57    v11 = strtok(&s, "_");                    // 这里以“_”为分割符，分割flag中的第一个“_”之前的大括号之内的字符
58    if ( v11 )                                // v11指向第一部风字符串
59    {
60      v12 = (const char *)dest;
61      memcpy(dest, v11, strlen(v11));         // 将第一部分字符串复制到dest指向的内存
62      v13 = 1;
63      do
64      {
65        ++v13;
66        v14 = strtok(0LL, "_");               // v14指向第二个段字符串
67        if ( !v14 )
68        {
69          result = 0xFFFFFFFFLL;
70          goto LABEL_23;
71        }
72        memcpy((void *)v8[2], v14, strlen(v14));// 这里的(void *)v8[2]指向的就是v19
73                                              // 原因：
74                                              // 上面的v8等于v17的地址，相当于v8与v17指向同一个内存，
75                                              // v8数据类型为__int64是8个字节，
76                                              // v8[2]实质上就是v8+2*8=v17+16,而上面
77                                              // __int64 v17; // [sp+0h] [bp-B8h]@5
78                                              // const char *v19; // [sp+10h] [bp-A8h]@13
79                                              // 表明v17到v19偏移量变化了10h，即16个字符
80                                              // 所以v19指向第二个字符串
81                                              // 同理下面的v20指向v8++之后的v8[2],即sp+18
82                                              // v20存储第三个字符串
83        ++v8;
84      }
85      while ( v13 != 3 );                     // 最多循环两次，如果有一次不包含字符“_”则会直接结束
86                                              //
87                                              // 到此说面flag的格式为CISCN{XXXXX_XXXXX_XXXXXXXX}
88                                              // 并且三个函数每个检查一部分
89                                              //
90      if ( (unsigned int)sub_4012DE(v12) )    // "762306GI890905GKL6887E5D75776382"
91      {
92        result = 0xFFFFFFFFLL;
93      }
94      else if ( (unsigned int)sub_401411(v19) ) // "762306GI890905GKL6887E5D75776382"
95      {
96        result = 0xFFFFFFFFLL;
```

```
97        }
98        else if ( (unsigned int)sub_401562(v20) ) //  "1B2GH0J65NEG37F69158D9H193871222"
99        {
100         result = 0xFFFFFFFFLL;
101       }
102       else
103       {
104         puts("Congratulations!");
105         _IO_getc(stdin);
106         result = 0LL;
107       }
108     }
109     else
110     {
111       result = 0xFFFFFFFFLL;
112     }
113   }
114   else
115   {
116     result = 0xFFFFFFFFLL;
117   }
118 LABEL_23:
119   v16 = *MK_FP(__FS__, 40LL) ^ v24;
120   return result;
121 }
000018BB main:100
```

函数sub_4012DE(v12)：

```
__int64 __fastcall sub_4012DE(const char *a1)
{
  unsigned __int64 i; // rsi@1
  char v2; // cl@2
  unsigned __int64 v3; // kr18_8@5
  bool v4; // cf@5
  bool v5; // zf@5
  char *v6; // rdi@6
  const char *v7; // rsi@6
  signed __int64 v8; // rcx@6
  __int64 result; // rax@9
  __int64 v10; // rbx@9
  __int64 v11; // [sp+0h] [bp-A8h]@1
  __int64 v12; // [sp+60h] [bp-48h]@1
  __int64 v13; // [sp+68h] [bp-40h]@1
  char v14[40]; // [sp+70h] [bp-38h]@1
  __int64 v15; // [sp+98h] [bp-10h]@1

  v15 = *MK_FP(__FS__, 40LL);
  sub_400876((__int64)&v11);                    // v11 = "78FCDE14DE5412CE23D147552156D4EE"
  sub_4008A0((__int64)&v11);
  sub_40108B((__int64)&v11, a1, strlen(a1));
  sub_401170((unsigned int *)&v11, (__int64)&v12);
  sub_401285((__int64)v14, (unsigned __int8 *)&v12, 16);
  qword_6038B0 = v12;
  qword_6038B8 = v13;                           // for循环这部分完全可以直接逆向分析
                                                // 得到的原MD5值直接通过网上的在线工具破解即可
                                                // 为什么可以直接拿恢复出来的原MD5值作为原字符串的MD5
                                                // 理由：
                                                // for循环上面的函数有一大堆，而且里面还有很多调用，不好逆
                                                // 传过来的是a1为字符串，经过一系列函数之后变成了MD5值
                                                // 由此可以推断这一堆函数是实现MD5运算
                                                //
  for ( i = 0LL; ; ++i )
  {
    v3 = strlen(v14) + 1;
    v4 = i < v3 - 1;
    v5 = i == v3 - 1;
    if ( i >= v3 - 1 )
      break;
    v2 = v14[i];
    if ( (unsigned __int8)(v2 - 'A') <= 5u )
      v14[i] = (signed int)i % 10 + v2;
  }
  v6 = v14;
  v7 = "762306GI890905GKL6887E5D75776382";
  v8 = 32LL;
  do
  {
    if ( !v8 )
      break;
    v4 = *v7 < (unsigned __int8)*v6;
    v5 = *v7++ == *v6++;
    --v8;
  }
  while ( v5 );
  result = (unsigned int)-((!v4 && !v5) != v4); // 返回值应该为0
                                                // v4=0
                                                // v5=1
  v10 = *MK_FP(__FS__, 40LL) ^ v15;
  return result;
}
000012DE sub_4012DE:15
```

第一部分逆向脚本：

```
def check1(str0):
    flagmd5 = ''
    for i in range(len(str0)):
        temp = ord(str0[i])-i%10
        if temp <= ord('A') + 5 and temp >= ord('A'):
            flagmd5 += chr(temp)
        else:
            flagmd5 += str0[i]

    return flagmd5
```

得到的结果拿去在线解密： yubu

函数sub_401411(v19)：（与第一个检查函数同理）

关键部分如下：比上一个多了一点点

```
26  qword_6038A8 = v14;
27  v1 = 0LL;
28  do
29  {
30    *((_BYTE *)&v13 + v1) ^= byte_603880[v1];
31    ++v1;
32  }
33  while ( v1 != 16 );
34  sub_401285((__int64)v15, (unsigned __int8 *)&v13, 16);
35  for ( i = 0LL; ; ++i )
36  {
37    v4 = strlen(v15) + 1;
38    v5 = i < v4 - 1;
39    v6 = i == v4 - 1;
40    if ( i >= v4 - 1 )
41      break;
42    v3 = v15[i];
43    if ( (unsigned __int8)(v3 - 65) <= 5u )
44      v15[i] = (signed int)i % 10 + v3;
45  }
46  v7 = v15;
47  v8 = "762306GI890905GKL6887E5D75776382";
```

逆向脚本：

```
def change2(str0):
    byte_603860 = [0xd7,0xdb,0xb4,0x0e,0x1e,0x17,0x2b,0x76,0x34,0xcc,0x00,0x81,0xef,0x67,0x29,0xb3]
    flag_byte = ""
    for i in range(0,32,2):
        v5 = str0[i]
        v7 = str0[i+1]
        if ord(v5)>=0x41 and ord(v5)<=0x46:
            tmp = ord(v5)-0x41+10
        elif ord(v5)>=0x30 and ord(v5)<=0x39:
            tmp = ord(v5)-0x30
        else:
            print "Error"

        if ord(v7)>=0x41 and ord(v7)<=0x46:
            tmp1 = ord(v7)-0x41+10
        elif ord(v7)>=0x30 and ord(v7)<=0x39:
            tmp1 = ord(v7)-0x30
        else:
            print "Error"
        byte = str(bin(tmp)[2:]).rjust(4,"0")+str(bin(tmp1)[2:]).rjust(4,"0")
        flag_byte += hex(int(byte,2))[2:].rjust(2,"0")


    flagmd5 = ""
    for i in range(0,16,):
        flagmd5 += str( hex(int(flag_byte[2*i]+flag_byte[2*i+1],16) ^ byte_603860[i]).rjust(2,"0")[2:] )
    return flagmd5
```

得到的结果拿去在线解密： kulo

函数sub_401562(v20)：

发现最下面将最后的结果写入了一个文件“flag”,前面未知量较多，不适合逆向尝试爆破，脚本如下：

```
if ( (!v6 && !v7) == v6 )
{
  v12 = fopen("flag", "w+");
  if ( v12 )
  {
    v13 = &qword_6038A0;
    v14 = 0;
    do
    {
      v14 += *(_BYTE *)v13;
      v13 = (__int64 *)((char *)v13 + 1);
    }
    while ( &qword_6038B0 != v13 );
    v15 = v1[3] ^ (v14 >> 4);
    v16 = v1[4] ^ v14 & 0xF;
    v17 = 0LL;
    do
    {
      if ( v17 & 1 )
        byte_6020E0[v17] ^= v16;              // 奇数字节
      else
        byte_6020E0[v17] ^= v15;              // 偶数字节
      ++v17;
    }
    while ( v17 != 0x179F );
    fwrite(byte_6020E0, 0x179FuLL, 1uLL, v12);
    fclose(v12);
    result = 0LL;
```

这里我们从写入文件的一步开始倒着看，上面是一个循环，奇偶不同异或的值也不同，但只有v15，v16两个值。这里我们不管他之前的值是什么，只有两个字节我们可以爆破。

爆破脚本：

```
def get_data():
    f2 = open("result",'wb')
    for i in range(256):
        for j in range(256):
            f1 = open("data",'rb')
            for k in range(6047):
                ori_byte = f1.read(1)
                if k&1 ==0:
                    f2.write(chr(ord(ori_byte)^i))
                else:
                    f2.write(chr(ord(ori_byte)^j))

            f1.close()
            f2.write('\n')
    f2.close()

get_data()
```

写脚本的思路就是，因为v15、v16我们不知道，但是我们知道他们每个都是一个“char”类型的数，一个字节，那么范围就是从0到255，每一个v15，尝试256个v16，这时候每一种v15、v16的组合和一个大的数组byte_6020E0，做上述操作。一共会生成256*256*6047个字节的数据，这256*256次中一定有一次组合是正确的，是我们需要的数据。

我跑了大约有七八分钟的样子，得到了数据快400M

用HxD打开，这时候就需要我们脑洞了，这里出题人放的是一张 jpg，文件头为“FF D8 FF E0 00 10 4A 46 49 46 00 01”，搜索发现有头，再搜索“FF D9”文件尾，把这一部分数据截取出来单独保存成jpg后缀的文件。

打开图片就可以看到一个字符串，得到了第三部分数据。

我们将三部分的数据组合得到flag：

CISCN{yubu_kulo_JipCC%6&goFunMz}

前两段完整代码：

```
def change1(str0):

    str00 = ''
    for i in range(len(str0)):
        temp = ord(str0[i])-i%10
        if temp <= ord('A') + 5 and temp >= ord('A'):
            str00 += chr(temp)
        else:
            str00 += str0[i]

    return str00

def change2(str0):
    byte_603860 = [0xd7,0xdb,0xb4,0x0e,0x1e,0x17,0x2b,0x76,0x34,0xcc,0x00,0x81,0xef,0x67,0x29,0xb3]
    flag_byte = ""
    for i in range(0,32,2):
        v5 = str0[i]
        v7 = str0[i+1]
        if ord(v5)>=0x41 and ord(v5)<=0x46:
            tmp = ord(v5)-0x41+10
        elif ord(v5)>=0x30 and ord(v5)<=0x39:
            tmp = ord(v5)-0x30
        else:
            print "Error"

        if ord(v7)>=0x41 and ord(v7)<=0x46:
            tmp1 = ord(v7)-0x41+10
        elif ord(v7)>=0x30 and ord(v7)<=0x39:
            tmp1 = ord(v7)-0x30
        else:
            print "Error"
        byte = str(bin(tmp)[2:]).rjust(4,"0")+str(bin(tmp1)[2:]).rjust(4,"0")
        flag_byte += hex(int(byte,2))[2:].rjust(2,"0")

    flagmd5 = ""
    for i in range(0,16,):
        flagmd5 += str( hex(int(flag_byte[2*i]+flag_byte[2*i+1],16) ^ byte_603860[i]).rjust(2,"0")[2:] )

    return flagmd5

flag1 = ""
flag2 = ""
flag1_check = "762306GI890905GKL6887E5D75776382"
flag2_check = "762306GI890905GKL6887E5D75776382"
flag1 = change1(flag1_check)
print "flag1",flag1
flag2 = change2(change1(flag2_check))
print "flag2",flag2
```

```
flag1 762306AB890905CFF6887D5A75776382    在线解密之后为 ：  yubu
flag2 a1f8b2a5971e2eb9c2447ddb9a104a31    在线解密之后为 ：  kulo
```