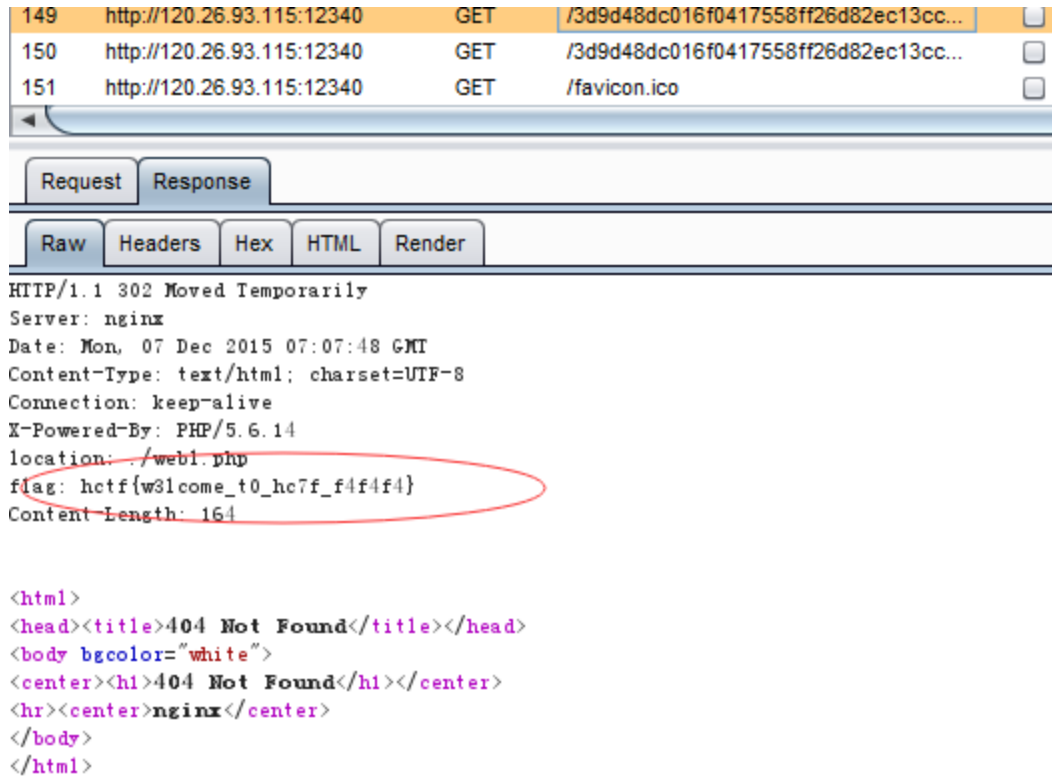


HCTF RS-Team writeup 解题报告

一、WEB 题

1、 404

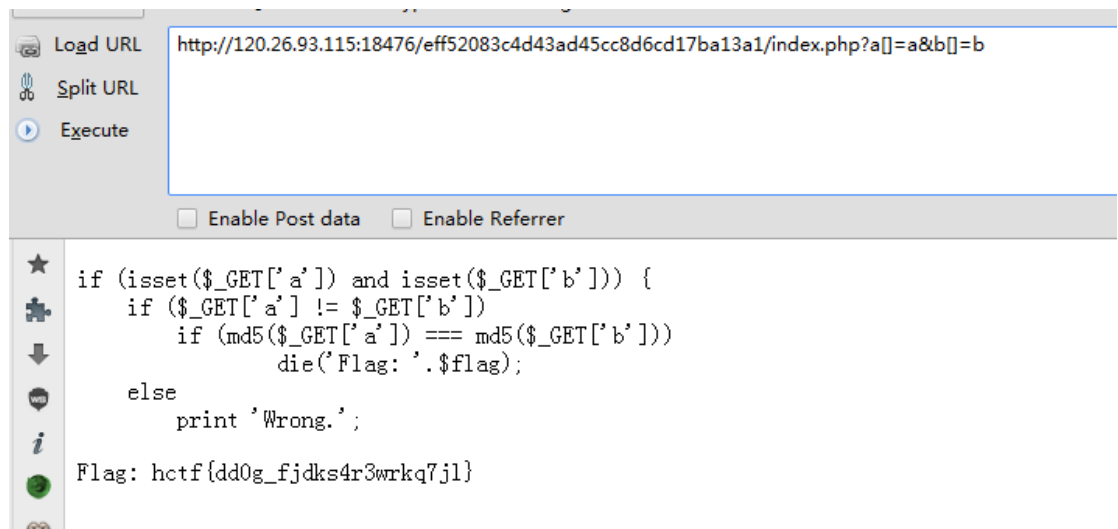
burp 抓包，flag 在返回包的头中



2、 fuck ===

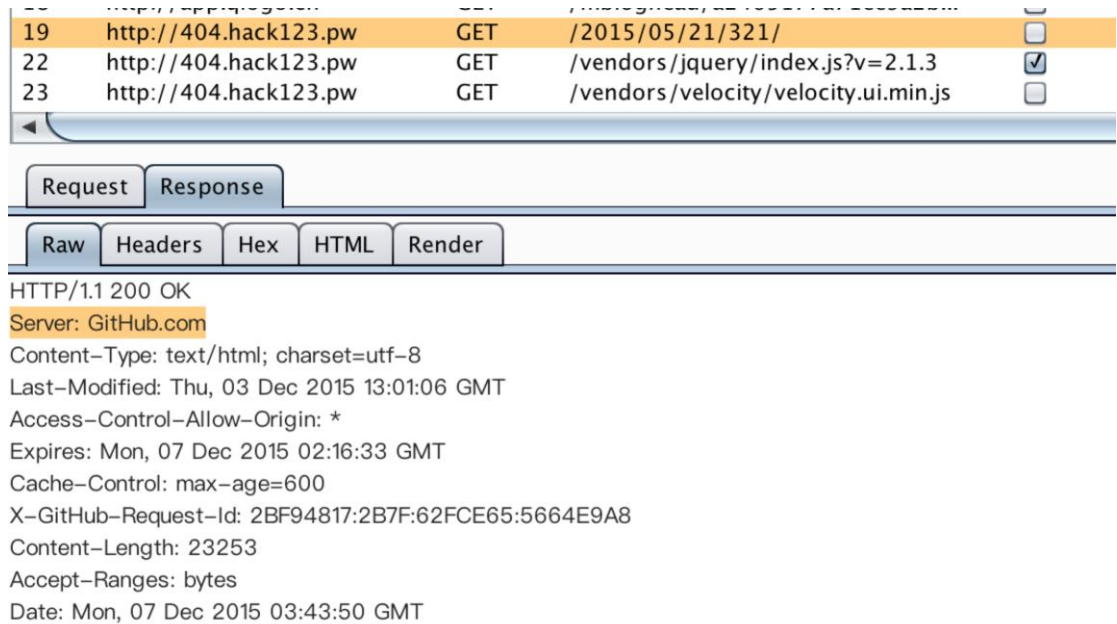
乍一看感觉不可能解决，仔细一想就发现 md5(array[])会返回 false

False===false, php 弱类型



3、 Personal Blog

打开 blog, 提示“flag is in the page source”; 用 burp 抓包, 在返回的 HTTP 头部中, 有一个 Server: GitHub.com;



于是在 github 上搜索“LoRexxar”, 在 LoRexxar.github.io/here is f10g.html, 得到 base64 编码后的 flag, 解码后提正确的 flag。

4、COMA WHITE

查看网页源码, 将其中 eval() 那一段, 改为 alert, 得到 javascript 代码, 分析其算法。

```
(function($, coveredFlag){
    eval(function(p,a,c,k,e,d){e=function(c){return(c<a?"":e(parseInt(c/a)))+(c=c%a)>35?
String.fromCharCode(c+29):c.toString(36)};if(!''.replace(/^/,String)){while(c-
-)d[e(c)]=k[c]||e(c);k=[function(e){return d[e]};e=function()
{return'\w+'};c=1;};while(c--)if(k[c])p=p.replace(new
RegExp('\\b'+e(c)+'\\b','g'),k[c]);return p;}('$.$q("j",8(e,a){3 5=a.5;3 d=
[1,2,1,1,1,2,1,1,2,1,2,2,2,1,2,1,2,1,1,2,1,2];3 4=[];3 9=0;$h(d,8(f,7)){3
n=5.w(9,9+7);9+=7;4.m(n)};$i("r",{4:4}));$.q("r",8(e,a){3 4=a.4;3 c=[];$h(4,8(f,7)
{3 6=v(7);6=y(6);6=x(6);c.m(6)});3 l=c.K();L(I(l)==J){g("s t b k o p, b M P N O
H.")}B{g("s t b k o C p.")}};$("#z").A('\\F\\',8(u){u.G();3 5=D.5.E;$i("j",
{5:5}));',52,52,'||var|davidFincher|flag|bpPart|val|function|nortonPointer|data|YOU|b
radPitt|edwardNorton||index|showAlertBox|each|publish|step_0|GIVEN|MarilynManson|push|d
fPart|IS|CORRECT|subscribe|step_1|THE|FLAG|event|FFBA94F946CC5B3B3879FBEC8C8560AC|slice
|E3AA318831FEAD07BA1FB034128C7D76|AD9539C3B4B28AABF6F6AF8CB85AEB53|blood|on|else|NOT|th
is|value|submit|preventDefault|IT|BF5B983FF029B3BE9B060FD0E080C41A|coveredFlag|join|if|
ARE|TO|SUBMIT|SUPPOSED'.split('|'),0,{}))
```

```

d.js
1 $.subscribe("step_0",
2   function(e,data){
3     var flag=data.flag;
4     var edwardNorton=[1,2,1,1,1,2,1,1,2,1,2,2,2,1,2,1,2,1,2];
5     var davidFincher=[];
6     var nortonPointer=0;
7     $.each(edwardNorton,function(index,val){
8       var dfPart=flag.slice(nortonPointer,nortonPointer+val);
9       nortonPointer+=val;
10      davidFincher.push(dfPart));
11    });
12    $.publish("step_1",{davidFincher:davidFincher});
13
14    $.subscribe("step_1",
15      function(e,data){
16        var davidFincher=data.davidFincher;
17        var bradPitt=[];
18        $.each(davidFincher,function(index,val){
19          var bpPart=FFBA94F946CC5B3B3879FBEC8C8560AC(val);
20          bpPart=AD9539C3B4B28AABF6F6AF8C8B85AEB53(bpPart);
21          bpPart=E3AA318831FEAD07BA1FB034128C7D76(bpPart);
22          bradPitt.push(bpPart));
23          var MarilynManson=bradPitt.join();
24          if(BF5B983FF029B3BE9B060FD0E080C41A(MarilynManson)===coveredFlag)
25            {showAlertBox("THE FLAG YOU GIVEN IS CORRECT, YOU ARE SUPPOSED TO SUBMIT IT.");}
26          else{showAlertBox("THE FLAG YOU GIVEN IS NOT CORRECT.");}});
27
28    $("#blood").on('submit',function(event){
29      event.preventDefault();
30      var flag=this.flag.value;
31      $.publish("step_0",{flag:flag});
32    });

```

其算法是：先将 flag 按 edwardNorton 长度，分成 22 组，进行 base64 编码，去掉=号及,号后生成 md5 值，合并成 22*32=704 位长的值与 result 进行比较。

获取 flag 方法：将 result 分为 22 组 md5 值，通过 cmd5.com 得到 22 组字符串后进行 base64 解码。

5、 Injection

http://120.26.93.115:24317/0311d4a262979e312e1d4d25565815

09/index.php

根据 hint 提示 user=user1 和 xpath 注入

尝试

http://120.26.93.115:24317/0311d4a262979e312e1d4d25565815

09/index.php?user=user1

发现找对切入点

开始注入

http://120.26.93.115:24317/0311d4a262979e312e1d4d25565815

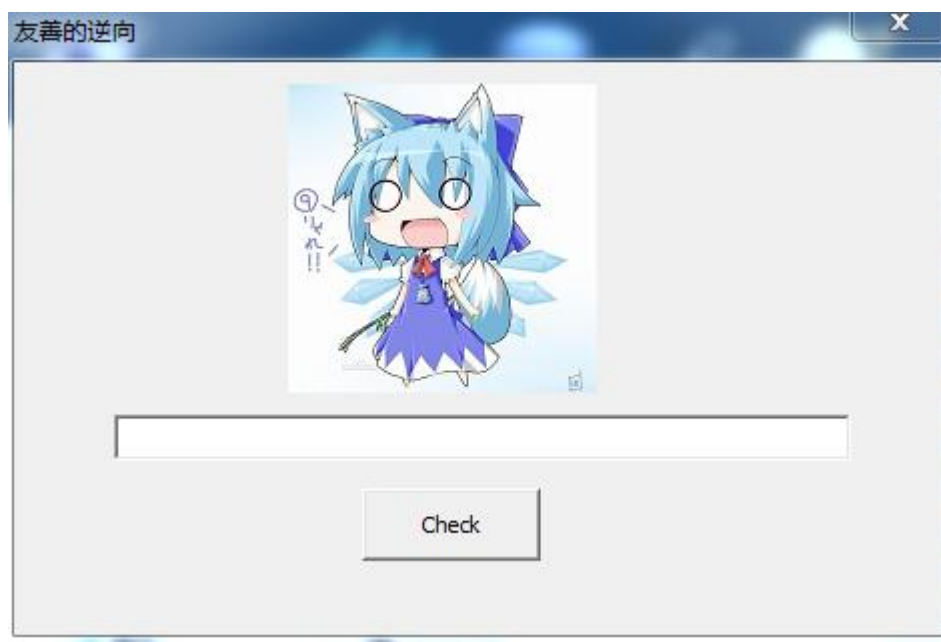
09/index.php?user=user1'] | count(//*) or ['1'=1

列出当前路径下的所有节点

出现 flag

二、RE 题

6、真的很友善的逆向题（福利）



“友善的逆向”并不友善，点击 Check 按钮的时候，按钮会跑掉。ida 看到如表，看到了 MoveWindow，查看其引用。干掉按钮乱跑的那段代码。我的做法是把 0x00401871 处的 jnz 改成 jz。

```
if ( !hObject )
{
    sub_4029F6(&unk_415584, a1);
    return 0;
}
else
{
    if ( [EAX] == 16 )
    {
        CloseHandle(hObject);
        EndDialog(hDlg, 0);
        DestroyWindow(hDlg);
        return 0;
    }
    if ( [EAX] == 0x20 && hWnd == a4 )
    {
        if ( dword_4181F8 == 34 )
        {
            MoveWindow(hWnd, 10, 10, 80, 35, 1);
            dword_4181F8 = 21;
            return 0;
        }
        if ( dword_4181F8 == 21 )
        {
            MoveWindow(hWnd, 20, 230, 80, 35, 1);
            dword_4181F8 = 24;
            return 0;
        }
        if ( dword_4181F8 == 24 )
        {
            MoveWindow(hWnd, 350, 230, 80, 35, 1);
            dword_4181F8 = 34;
            return 0;
        }
    }
}
```

干掉这个判断，让其进不来

```

-- .text:0040185F      jz      loc_401964
-- .text:00401865      cmp     eax, 10h
-- .text:00401868      jz      loc_401934
-- .text:0040186E      cmp     eax, 20h
-- .text:00401871      jnz     loc_401AC1
-- .text:00401877      mov     eax, hWnd
-- .text:0040187C      cmp     eax, [ebp+arg_8]
-- .text:0040187F      jnz     loc_401AC1
-- .text:00401885      mov     ecx, dword_4181F8
-- .text:00401888      cmp     ecx, 22h
-- .text:0040188E      jnz     short loc_4018BF
-- .text:00401890      push    1                ; bRepaint
-- .text:00401892      push    23h             ; nHeight
-- .text:00401894      push    50h             ; nWidth
-- .text:00401896      push    0Ah             ; y
-- .text:00401898      push    0Ah             ; x
-- .text:0040189A      push    eax              ; hWnd
-- .text:0040189B      call    ds:MoveWindow
-- .text:004018A1      pop     edi
-- .text:004018A2      mov     dword_4181F8, 15h
-- .text:004018AC      xor     eax, eax
-- .text:004018AE      pop     esi
-- .text:004018AF      mov     ecx, [ebp+var_4]
-- .text:004018B2      xor     ecx, ebp
```

图 1 按钮乱跳的代码

需要逆向的功能代码就在 0x00401871 上面不远处。

```

if ( a3 == 273 && !((unsigned int)a4 >> 16) && (_WORD)a4 == 1004 )
{
    GetWindowTextA(dword_4191F8, &String, 30);
    v6 = &String;
    do
    {
        v7 = *v6++;
        while ( v7 );
        if ( v6 - (CHAR *)v6 == 22 && sub_401DA0((int)&String, SBYTE4(v7)) && sub_401BB0(&String) )// sub_401DA0(str,最后一个字符)
        {
            v8 = 0;
            while ( 1 )
            {
                v9 = dword_4191B0 ^ byte_418217;
                if ( !((dword_4191B0 ^ byte_418217) < 0 | dword_4191B0 == byte_418217) )
                {
                    if ( (v9 ^ (char)v7) == byte_418218// 3431634A
                        && (v9 ^ SBYTE1(v7)) == byte_418219
                        && (v9 ^ SBYTE2(v7)) == byte_41821A
                        && (v9 ^ SBYTE3(v7)) == byte_41821B )
                        break;
                    // 未知部分后4位
                }
                Sleep(0x14u);
                ++v8;
                if ( v8 >= 100 )
                    goto LABEL_30;
            }
            v10 = dword_4191D8;
            dword_4191D8 = dword_4191C0[0]; // 未知部分前12位续
            dword_4191C0[0] = v10;
            v11 = dword_4191E0;
            dword_4191E0 = dword_4191CC;
            dword_4191CC = v11;
            v12 = dword_4191D4;
            dword_4191D4 = dword_4191C8;
            dword_4191C8 = v12;
            v13 = dword_4191D0;
            dword_4191D0 = dword_4191EC;
            v14 = 0;
            dword_4191EC = v13;
            do
            {
                if ( dword_415600[v14] != dword_4191C0[v14] )
                {
                    MessageBoxW(NULL, L"Try Again", L"Fail", 0);
                    exit(-1);
                }
                ++v14;
            }
            while ( v14 < 12 );
            if ( v9 == 2 )
            {
                MessageBoxW(NULL, L"YOU GOT IT", L"OK", 0);
                exit(0);
            }
        }
    }
    EL_30:
    MessageBoxW(NULL, L"Try Again", L"Fail", 0);
}

```

图 2 需要逆向的代码

首先看一看 if 判断。输入的字符串长度要为 22；sub_401DA0 的功能是取字符串的前 5 个字节和最后一个字节，前 5 个字节要为“HCTF{”，最后一个字符要为“}”；sub_401BB0 的功能是读取“HCTF{”后面的 12 个字节，并依次对每一个字节分类处理（大概是大写字符、小写字符、数字 3 类），转成一个数字存到 dword_4191C0 数组中。

if 判断之后，进入了 while 循环。这里是这题另一个不友好的地方，刚开始调试的时候一直在这里循环，后来才发现有坑。dword_4191B0 时而为 1，时而为 2，byte_418217 也是多变。可以在 od 里下硬件断点查看哪里动了这两个地址里的值，也可以在 ida 里面按 x 查看引用位置。dword_4191B0 ^ byte_418217 所有结果如下图。在这些结果中只能选为 2 的那个，因为后面“YOU GOT IT”上面的 if 判断有指定。知道了异或结果，就能够很容易的知道未知部分的后 4 位。

```

00000001^00000003=00000002
00000001^0000000c=0000000d
00000001^00000030=00000031
00000001^ffffffffc0=fffffffc1
00000002^00000003=00000001
00000002^0000000c=0000000e
00000002^00000030=00000032
00000002^ffffffffc0=fffffffc2

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.

```

图 3 dword_4191B0 ^ byte_418217 所有结果

while 循环之后，有一段代码在交换 dword_4191C0 数组里的值。具体怎么交换的其实不用管，在交互前后分别下个断点看下，然后比较下就知道了。

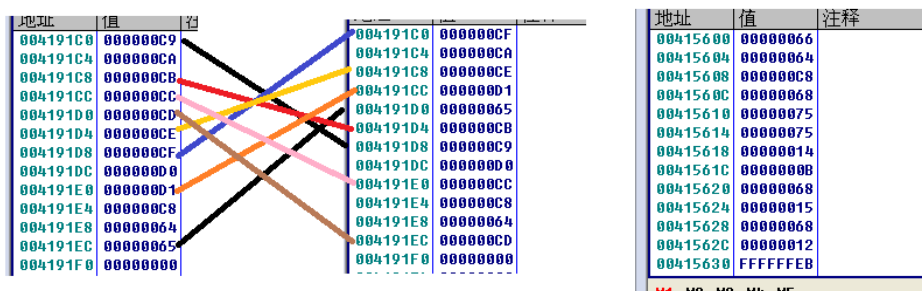


图 4 dword_4191C0 数组数值交换和目标数组 dword_415600

最终算的 flag: HCTF{UareS0cLeVerHa36}

7、 复古的程序

emu8086 上调试逆向，程序对输入字符串进行了三次变换

1) 输入为一个 24 字节长的字符串，进行 4 组函数替换，是的输出变成 32 字节长度的字符串，4 组函数如下：

- AL = (a[i] & 0xFC) >> 2;
- AL = (a[i] & 0x03) << 4; AH = ((a[i+1]) & 0xF0) >> 4; AL = AL+AH;
- AL = ((a[i+1]) & 0x0F) << 2; AH = ((a[i+2]) & 0xC0) >> 6; AL = AL+AH;
- AL = (a[i+2]) & 0x3F;

2) 基于以上四组函数，得到数值 AL 进行第二次变换，见附件 check 函数中

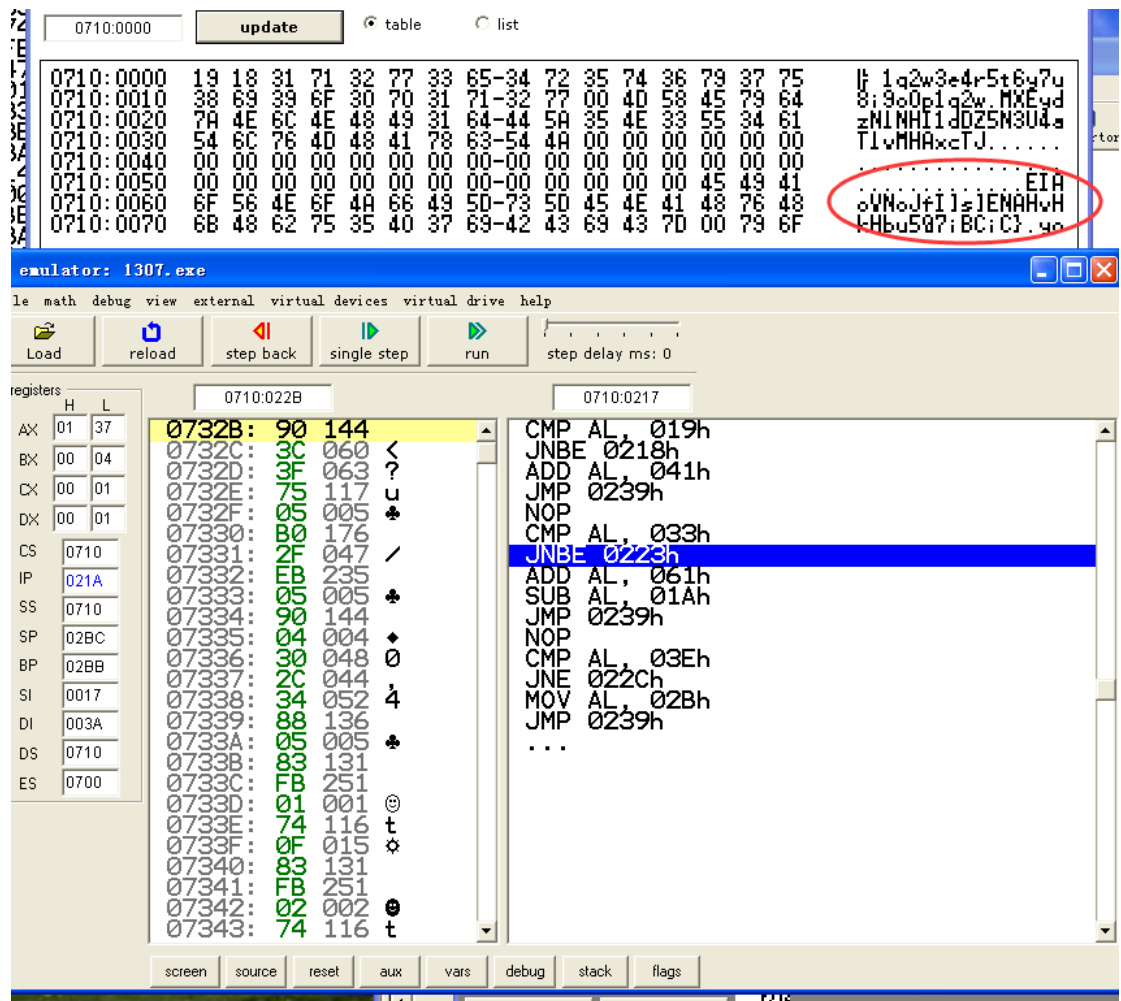
3) 进行异或变换和位置替换

AL = b[i] ^ 0x7;

$AH = b[i+16] \wedge 0xC;$

$b[i] = AH;$

$b[i+16] = AL;$



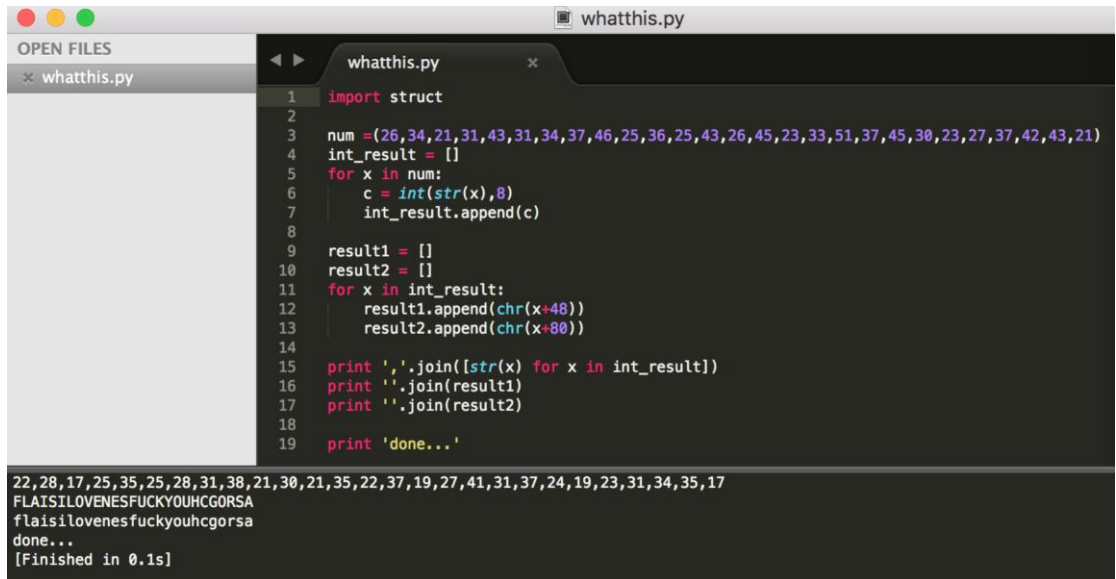
htcf{Dd0g 1s 1307 d0gs!}

见附件 old.cpp

三、MISC 题

8、 What's this

用 file 命令查看下载的附件，得知是 iNES ROM Dump，于是下载一个 NES 模拟器，加载后是 NES 的 Jackal 游戏。从网上下载到一个原始的 jackal.nes 的 ROM Dump，用 cmp 比较二者的差异。发现从 13294—13324 字节处存在差异，用凯撒密码的解法，得到字符串中包含有 ILOVENES，多次尝试提交得到正确的 flag。



```
1 import struct
2
3 num = (26, 34, 21, 31, 43, 31, 34, 37, 46, 25, 36, 25, 43, 26, 45, 23, 33, 51, 37, 45, 30, 23, 27, 37, 42, 43, 21)
4 int_result = []
5 for x in num:
6     c = int(str(x), 8)
7     int_result.append(c)
8
9 result1 = []
10 result2 = []
11 for x in int_result:
12     result1.append(chr(x+48))
13     result2.append(chr(x+80))
14
15 print ','.join([str(x) for x in int_result])
16 print ''.join(result1)
17 print ''.join(result2)
18
19 print 'done...'
```

22,28,17,25,35,25,28,31,38,21,30,21,35,22,37,19,27,41,31,37,24,19,23,31,34,35,17
FLAISILOVENESFUCKYOUHCGORSA
flaisilovenesfuckyouhcgorsa
done...
[Finished in 0.1s]

9、 Andy

将 andy.apk 用 dex2jar 得到 jar 包,用 jd-gui,可以反编译得到 java 代码。

在 MainActivity.class 得到检查 flag 的入口:

```
public void onClick(View paramAnonymousView)
{
    MainActivity.access$002(MainActivity.this, MainActivity.this.etInput.getText().toString());
    MainActivity.this.make = new Make(MainActivity.this.input);
    if (MainActivity.this.make.andy().equals("SRLhb70YZHKvLTrNrt08F=DX3cdD3txmg"))
    {
        Toast.makeText(MainActivity.this, "You get the flag!", 1).show();
        return;
    }
    Toast.makeText(MainActivity.this, "Sorry", 0).show();
}
});
}
```

跟进 make.class, 得加密 flag 的算法: 将 flag 加上 hdu1s8 后反转, 进行 base64, 再进行查表变换加密。

```
public Make(String paramString)
{
    this.input = paramString;
}

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();
}
```

获得 flag 算法: 将 “SRLhb70YZHKvLTrNrt08F=DX3cdD3txmg” 串反向查表变换, 进行 base64 解码, 反转并去掉 hdu1s8 即是 flag。


```

1  #!/usr/bin/env python
2  -*- coding:utf-8 -*-
3  import base64
4
5  def classical(s):
6      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 = /"
7      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,"
8      a1 = array1.split(' ')
9      a2 = array2.split(',')
10     result = []
11     for a in s:
12         index = classical_find(a,a2)
13         result.append(a1[index])
14     return ''.join(result)
15
16 def classical_find(c,a):
17     for index,ch in enumerate(a):
18         if ch == c:
19             return index
20
21 def base64_decode(s):
22     return base64.b64decode(s)
23
24 def main():
25     s = 'SRlhb70YZHKvLrNr08F=DX3cdD3txmg'
26     cs = classical(s)
27     print cs
28     b64cs = base64_decode(cs)
29     print b64cs
30
31     input_str = b64cs[:-1]
32     print input_str
33     result = input_str.replace('hdu1s8','')
34     print result
35
36 if __name__ == '__main__':
37     main()

```

OHMxdwRoZDBpMnczcmRuYXk2bjhkbmEE=

8s1udhd0i2w3rdnay6n8dna[0]

[0]and8n6yandr3w2i0dhdu1s8

[0]and8n6yandr3w2i0d

[Finished in 0.1s]

见附件，andy.py

10、送分要不要

在文件里面发现一串字符

0	03	D1	01	50	4B	05	06	00	00	00	01	00	01	00	5A	. 7PK.....Z. 9	
0	00	00	00	49	A3	00	00	00	00	52	31	6B	30	52	45	31	...I?...R1k0RE1?
0	4E	57	6C	68	48	55	54	4E	45	54	55	34	79	51	30	64	NWlhHUTNETU4yQ0d?
0	61	51	31	52	4E	55	6B	70	55	52	30	55	7A	56	45	64	aQ1RNUkpUR0UzVEd?
0	4F	55	6C	52	48	56	6B	52	45	54	56	46	61	57	45	64	OUIRHVkrETVFaWEd?
0	4E	4D	6C	56	4E	54	6C	70	55	52	30	31	5A	52	45	74	NM1VNTlpUR01ZREt?
0	53	55	6C	52	48	54	56	70	55	53	55	35	61	56	45	63	SUIRHTVpUSU5aVEc?
0	30	4E	46	52	46	54	55	70	59	53	56	45	39	50	54	30	ONFRFTUpYSVE9PTOF
0	39	50	54	30	89	50	4E	47	0D	0A	1A	0A	00	00	00	0D	9PTO時NG.....?
0	49	48	44	52	00	00	05	56	00	00	03	00	08	06	00	00	IHDR...V.....
0	00	CF	3E	3C	C2	00	00	00	09	70	48	59	73	00	00	0E	.?<...pHYs.....
0	04	00	00	0E	C4	01	05	2B	0E	1B	00	00	00	00	00	40	...

R1k0RE1NWlhHUTNETU4yQ0daQ1RNUkpUR0UzVEdOUIRHVkrETVFaWEd

NM1VNTlpUR01ZREtSUIRHTVpUSU5aVEcONFRFTUpYSVE9PT09PT0

Base64 解码

Base64:	kpUROUzVEdOULRHVkrETVFawEdNM1VNT1pURO1ZREtSULRHTVpUSU5aVEcONFRFTUpYSVE9PTO9PTO
解密Base64:	GY4DMMZXGQ3DMN2CGZCTMRJTGE3TGNRTGVDDMQZXGM2UMN2TGM YDKRRTGMZTINZTG44TEMJXIQ====

Base32 解码

```
>>> base64.b32decode(<'GY4DMMZXGQ3DMN2CGZCTMRJTGE3TGNRTGVDDMQZXGM2UMN2TGM YDKRRTGMZTINZTG44TEMJXIQ===='>
'686374667B6E6E3173635F6C735F73305F33347379217D')
```

Base16 解码

```
>>> base64.b16decode(<'686374667B6E6E3173635F6C735F73305F33347379217D'>
'httf{nn1sc_ls_s0_34sy!}')
```