Linux 系统缓冲区溢出攻击的机理分析

吴基科，凌 捷

（广东工业大学 计算机学院，广东 广州 510090）

摘要：缓冲区溢出是目前最常用的黑客攻击技术之一，本文分析了这种攻击技术的原理，指出了 Linux 系统中进行这种攻击时，本地攻击和远程攻击的差异，并总结和归纳了几种保护缓冲区免受缓冲区溢出攻击和影响的方法.

关键词：网络安全；黑客；缓冲区溢出

中图分类号：TP309 **文献标识码**：A **文章编号**：1007-7162(2003)02-0016-05

Internet 是一种开放的面向所有用户的技术，其资源通过网络共享，但资源共享与信息安全是一对矛盾，随着 Internet 覆盖范围的增广，使用人数的增多，再加上 TCP/IP 协议本身固有的安全缺陷，使得网络安全问题越来越严重，网络入侵次数和造成的损失急剧增长. 近十几年来，缓冲区溢出一直是引起许多严重的安全性问题的主要原因. 其中最著名的例子是：感染了因特网几十万台机器的蠕虫程序就是 Morris 利用缓冲区溢出编写的. 据统计[1,2]，通过缓冲区溢出进行的黑客攻击占所有系统攻击总数的 80% 以上，并且数据显示这一问题正在扩大，而不是在缩减. 本文详细分析了缓冲区溢出攻击 Linux 系统的机理，并在此基础上归纳了几种缓冲区保护方法.

1 缓冲区溢出的概念

缓冲区溢出是指通过程序的缓冲区写超出其长度的内容，造成缓冲区的溢出，从而破坏程序的堆栈，使程序转而执行其它的指令，以达到攻击的目的.

引起缓冲区溢出问题的根本原因是 C（与其后代 C++）本质就是非安全的，没有边界来检查数组和指针的引用[3,4]. 也就是说程序开发人员必须检查边界（而这一行为往往会被忽视），否则会冒遇到问题的风险. 例如下面的程序（为了便于说明，文中以 Intel x86 平台上的 Linux 操作系统为例，所用编译器为 gnu gcc）：

```
// ex1.c
void function(char * str)
{ char buffer[16];
    strcpy(buffer,str);
}
void main ()
{ int i;
```

收稿日期：2002-06-17

基金项目：广东省教育厅“千百十”优秀人才基金资助项目(200017)

作者简介：吴基科(1979-)，男，研究生，主要研究方向为网络安全.

```
char buffer[128];
for(i=0;i<127;i++)
        bufffer[i]='A';
function(buffer);
printf("buffer overflow, just a test!");
}
```

这是一个典型的存在缓冲区溢出错误的程序.上面的 strcpy()将直接把 str 内容复制到 buffer 中.因为 str 的长度远大于 16,所以在程序执行时就会造成 buffer 的溢出,使程序运行出错.标准 C 函数库还存在许多像 strcpy()这样问题的非安全字符串操作:strcat()、sprintf()、vsprintf()、gets()、scanf()、getchar()、fgetc()等等.

堆栈是一种在计算机中经常会用到的抽象数据类型,它的特点是后进先出(LIFO),函数调用就要用到堆栈的这种特点.当程序中发生函数调用时,计算机做如下操作:首先把参数压入堆栈;然后保存指令寄存器(IP)中的内容作为返回地址(RET);第三个放入堆栈的是基址寄存器(FP);然后把当前的栈指针(SP)拷贝到 FP,作为新的基地址;最后为本地变量留出一定的空间,把 SP 减去适当的数值.

当在主函数 main 调用 function 函数时,堆栈的情形如图 1 所示.

显然,程序的执行结果将显示"Segmentation fault(core dump)"或类似的出错信息.当程序执行到 function()时,原来子程序执行完毕,应返回到执行语句 printf("buffer overflow,just a test!").但是由于缓冲区的溢出,子程序的返回地址变成了 0x41414141(A 的十六进制为 0x41),一个显然还在进程地址空间,但已经不是程序正常流程的地址.我们无法预料在 0x41414141 地址,程序会执行什么指令,由于本程序很小,并没有引起什么后果.假如在主程序中对字符串数组赋值时,在一个刚好覆盖子程序返回地址的数组的位置,将一个有危险的指令序列的地址以字符串的方式填入,则当子程序执行完毕返回时,将执行一段危险的指令,其后果可能不堪设想.

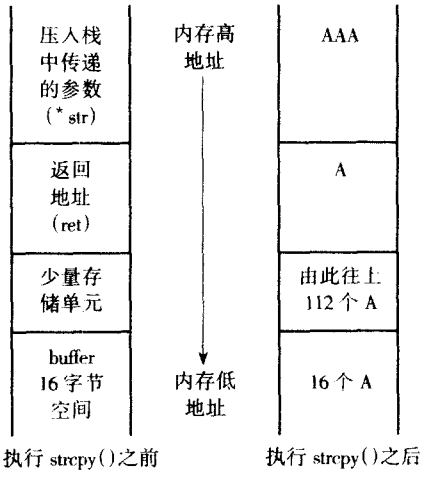

图 1　调用子过程 function()时堆栈情景

2　缓冲区溢出攻击的原理

随便往缓冲区中填东西造成它溢出一般是达不到攻击的目的.最常见的手段是通过缓冲区溢出使程序运行一个用户 shell,再通过 shell 执行其它命令.如果该程序属于 root 且有 suid 权限的话,攻击者就获得了一个有 root 权限的 shell,可以对系统进行任意操作了.

一个缓冲区溢出攻击程序通常由以下几个部分组成:

(1) 准备一段可以调出一个 shell 的机器码形成的字符串,即称之为 shellcode 的字符数组.

(2) 申请一个缓冲区,并将机器码填入缓冲区的低端.

(3) 估算机器码在堆栈中的起始位置,并将这个位置写入缓冲区的高端.这个起始位置也是我们在执行缓冲区溢出程序时,需要反复调整的一个参数.

(4) 将这个缓冲区作为系统一个有缓冲区溢出错误的程序的一个入口参数,并执行这个有错误的程序.

假定用来攻击的有缓冲区溢出错误的程序如下:

```
// ex2.c
void main(int argc, char * argv[]) {
char buffer[512];
if (argc > 1)
strcpy(buffer,argv[1]);
}
```

创建的缓冲区溢出程序可以接受两个参数,一是缓冲区大小,二是从其自身堆栈指针算起的偏移量(这个堆栈指针指明了想要使其溢出的缓冲区所在的位置).把溢出字符串放到一个环境变量 EGG 中,这样就容易操作一些.

```
// ex3.c
# define DEFAULT_OFFSET 0
# define DEFAULT_BUFFER_SIZE 512
# define NOP 0x90
char shellcode[ ] =
"\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";
      //shellcode 数组的作用就是开一个 shell,并且把它作为全局变量放在数据区
unsigned long get_sp(void) {
  __asm__("movl %esp,%eax");
                              }
void main(int argc, char * argv[ ])
{
char * buff, * ptr;
long * addr_ptr, addr;
int offset = DEFAULT_OFFSET, bsize = DEFAULT_BUFFER_SIZE;
int i;
if (argc > 1) bsize = atoi(argv[1]);
if (argc > 2) offset = atoi(argv[2]);
if (! (buff = malloc(bsize))) {
printf("Can't allocate memory.\n");
exit(0);
                              }
addr = get_sp() - offset;
printf("Using address: 0x%x\n", addr);
ptr = buff;
```

```
addr_ptr = ptr;
for (i = 0; i < bsize; i+ =4)
    *(addr-ptr+ +) = addr;
for (i = 0; i < bsize/2; i+ +)
    buff[i] = NOP;
ptr = buff + ((bsize/2) - (strlen(shellcode)/2));
for (i = 0; i < strlen(shellcode); i+ +)
    *(ptr+ +) = shellcode[i];
buff[bsize - 1] = '\0';
memcpy(buff,"EGG=",4);
putenv(buff);
system("/bin/bash");
}
```

现在我们使用这个程序来使被攻击的有缓冲区溢出错误的程序发生溢出，从而得到新的shell.

```
[root@sunrise/home] #  ./ex3 612
Using address: 0xbffffdb4
[root@sunrise/home] #  ./ex2  $ EGG
2.03 #  exit
exit
[root@sunrise/home]
```

从提示符2.03 # 来看，我们确实是得到了一个新的shell.至于在真实环境下的缓冲区溢出攻击实例，不同的操作系统存在不同的有缓冲区溢出错误的提示命令，如在Redhat linux下有mount和所有连接Xt库下的程序，Solaris for SPARC下fdformat命令等等.

3　防御缓冲区溢出的措施

基于上述讨论，我们给出几个防御缓冲区溢出攻击的措施：

(1) 安全代码编写

缓冲区溢出漏洞存在的根本原因是由于程序开发中使用了与字符串操作相关的函数，这些带有可变参数的函数，并且由于开发人员的懒惰或安全经验不足，没有很好地对这些存在危险的函数的输入参数进行控制.虽然缓冲区溢出漏洞已经存在有很长的时间，很多开发人员在开发程序之前经过了长时间的培训以提高编写安全程序的能力，人们还开发出一些工具和技术用于帮助经验不足的程序员来开发程序.可是，缓冲区溢出漏洞仍然没法彻底从程序中消除.编写安全的程序代码仍然是解决缓冲区溢出漏洞的根本办法，必须在开发中就已经详细考虑过安全问题，在编写程序过程中杜绝存在缓冲区溢出漏洞的一切可能，才使确保程序的最大化安全.

(2)安全工具防护

Libsafe是Arash Baratloo、Timothy Tsai和Navjot Singh(朗讯技术公司)开发出来的安全函数库，里面封装了若干已知的易受堆栈溢出方法攻击的库函数，使对于不安全函数调用转到替代的安全函数上，使用libsafe的真正收益在于保护尚未获知是否易受攻击的程序避免未来的攻

击.类似这样的安全工具有很多.

(3)安装安全补丁

即使是拥有最好的程序员,经过多年开发完善的软件,也没法彻底解决自身的安全问题.而且,大量的用户并非程序的开发者,不可能自己解决所有的安全问题,因此,用最新的程序修补有缺陷的程序是一个比较不错的解决缓冲区溢出问题的办法,但对系统管理员的要求要高一些.

(4)堆栈保护

堆栈保护是一种提供程序指针完整性检查的编译器技术,通过检查函数活动纪录中的返回地址来实现[5,6].堆栈保护作为 gcc 的一个补丁,在每个函数中,加入了函数建立和销毁的代码.加入的函数建立代码实际上在堆栈中函数返回地址后面加了一些附加的字节.而在函数返回时,首先检查这个附加的字节是否被改动过.如果发生过缓冲区溢出的攻击,那么这种攻击很容易在函数返回前被检测到.目前这种技术并不是很值得推荐,例如一些黑客杂志已经介绍如何绕过堆栈保护的方法.

(5)堆栈不可执行

缓冲区溢出攻击是利用缓冲区溢出修改程序的返回地址,使程序跳转到其他地址上执行预先放置的指令代码.如果能使堆栈不可执行,攻击者就很难利用缓冲区溢出进行攻击了.使缓存存区不可执行的技术叫非执行的缓冲区技术.很多操作系统(如 linux 和 Solaris)都有关于这项技术的内核补丁.但是一些操作系统(例如:Unix 和 Windows)需要实现的一些性能和功能,也需要在数据段中动态地放入可执行的代码,因此,为了保持程序的兼容性,不可能使得所有程序的数据段不可执行.

参考文献:

[1] 张小斌,严望佳. 黑客分析与防范技术[M].北京:清华大学出版社,1999.
[2] 王宏.黑客与防护[M].北京:中国青年出版社,2001.
[3] 星坤,堆栈溢出讲座[EB/OL]. www.521hacker.com,2000,12,3
[4] 孙海彬,徐良贤,杨怀银. 堆栈溢出攻击的原理及检测[J].计算机工程,2001,27(10):127-128
[5] W.Richard Stevens. Unix 环境高级编程[M].北京:机械工业出版社,2002.
[6] John Shapley Gray. Unix 进程间通信[M].北京:电子工业出版社,2001.

The Principle of Buffer Overflow Attack

WU Ji-ke, LING Jie
(Faculty of Computer, Guangdong University of Technology, Guangzhou 510090, China)

Abstract: Buffer overflow is a technology widely used by hackers to attack computer networks. In this paper the principle of such a kind of attack is analyzed, and the difference between local attacks and long-distance attacks is also disscussed. Then several methods for defending buffer overflow attack are present.
Key words: network security; hacker; buffer overflow