缓冲区溢出攻击原理与防范的研究*

王业君，倪惜珍，文伟平，蒋建春
(中国科学院 软件研究所 信息安全技术工程研究中心，北京 100080)

摘　要: 缓冲区溢出攻击是网络攻击事件中最常用的一种攻击方式,成为系统和网络安全中亟待解决的重要问题。在分析缓冲区溢出攻击原理的基础上,说明了攻击的成因,然后描述了目前常见的攻击类型,最后给出了现有的防范措施,进而提出了一种新的通过对编译器及相关系统级函数进行修改来解决该问题的方案。

关键词: 缓冲区溢出攻击; 系统安全; 边界检查
中图法分类号: TP393.08　　**文献标识码**: A　　**文章编号**: 1001-3695(2005)10-0101-04

Research on Principle and Defense of Buffer Overflow Attacks

WANG Ye-jun, NI Xi-zhen, WEN Wei-ping, JIANG Jian-chun
(Engineering Research Center for Information Security Technology, Institute of Software, Chinese Academy of Sciences, Beijing 100080, China)

Abstract: The buffer overflow attacks have been the most common form in the network attacks and become a predominant problem in the system and network security area. After the principle of the buffer overflow attack is explained, this paper analyzes the causes leading to the attacks. Then the usual attack types the attackers often exploit are presented. Last, the common measures to defend these attacks and my own idea about the solution of this problem by modifying the compiler and the relevant functions of the system are given.

Key words: Buffer Overflow Attack; System Security; Edge Checking

1　引言

自从缓冲区溢出漏洞被发现以来,缓冲区溢出攻击一直是网络攻击事件中用得最多的一种攻击方式,成为系统和网络安全中亟待解决的重要问题。早在 1988 年,美国康奈尔大学计算机科学系研究生莫里斯就利用 UNIX Fingered 不限输入长度等漏洞感染了互联网中的数万台机器[1];2003 年 8 月席卷全球的“冲击波(Worm.MSBlast)”[2]和 2004 年 5 月出现的“震荡波(Worm.Sasser)”[3]分别利用了 Windows 系统 RPC DCOM 和 LSASS 服务的缓冲区溢出漏洞进行攻击。

在 ICAT 2003-9-12 号公布的 10 大漏洞排名中位居榜首的仍是缓冲区溢出漏洞[4];从 CERT 公布的到去年为止的统计数据(图 1)中可以看出,无论是在绝对数量上还是相对比例上缓冲区溢出攻击的问题正在扩大。

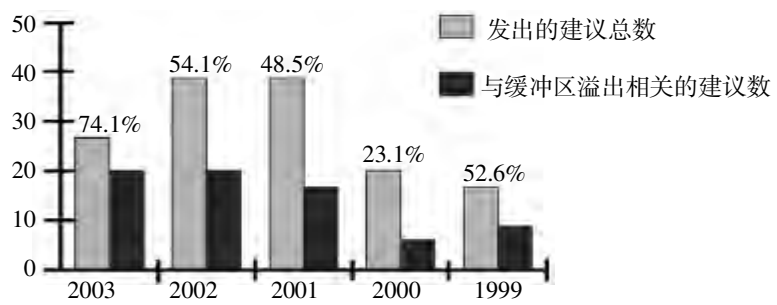

图 1　CERT 近五年发出的建议统计

由于缓冲区溢出攻击危害的严重性,使它至今仍是人们研究的重要课题,文章在总结当前已有研究成果的基础上提出了解决该问题的方案。

收稿日期: 2004-09-26; 修返日期: 2004-10-28
基金项目: 国家自然科学基金资助项目(60083007);国家“973”计划资助项目(G1999035810)

2　攻击原理及危害

要对缓冲区溢出攻击的原理有个清楚认识,就必须先对可执行文件执行时的内存结构有个总体了解。一般程序从逻辑上可分为两大部分:代码区和数据区。数据区从逻辑上可划分为静态数据、堆栈和堆三部分,它们之间的区别依赖于各部分对应内存分配的时间和方式以及数据存储的位置。总体来说,Windows 平台下程序的内存结构如图 2 所示,Linux 或 UNIX 平台下程序的内存结构如图 3 所示。

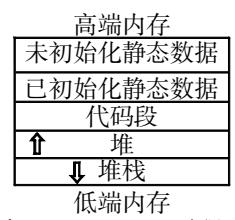

图 2　Windows 下程序在内存中的结构

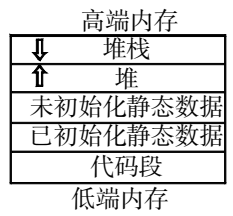

图 3　UNIX 下程序在内存中的结构

通常所说的缓冲区溢出指的就是“堆栈和堆”这两部分空间中产生了溢出。

正如微软给缓冲区溢出所下定义[5]中所说的,缓冲区溢出是因为人们向程序中提交的数据超出了数据接收区所能容纳的最大长度,从而使提交的数据超过相应的边界而进入了其他区域。如果是人为蓄意提交超长数据且对系统正常运行造成了不良影响,那么我们就说发生了缓冲区溢出攻击。

缓冲区溢出攻击通常会带来以下后果:①过长的字符串覆盖了相邻的存储单元而造成程序异常,严重的会造成死机、系统或进程重启等;②可让攻击者执行恶意代码或特定指令,甚至获得超级权限等,从而引发其他的攻击。

3 造成攻击的技术原因

3.1 缺少必要的边界检查

这方面最明显的例子是 C 和 C ++ 语言。由于一开始设计的时候 C 和 C ++就没有为数组和字符指针的引用提供任何的边界检查机制,从而导致了它们的不安全性。而它们是日常开发中使用最广的语言,甚至连 Windows/UNIX 这两个广泛使用的操作系统的很多模块也是用它们开发的,所以造成现在缓冲区溢出漏洞频频暴露。并且用它们开发的大部分软件在今后很长一段时间里仍会被使用,这也就给缓冲区溢出攻击的存在提供了有利环境。

3.2 操作系统设计策略上的隐患

这方面主要是指堆栈和堆数据区的可执行性属性。UNIX 和 MS Windows 系统为实现更好的性能和功能,往往在数据段中动态地放入可执行的代码[6],以保持程序的兼容性,使堆栈和堆具有可执行的属性。但从两者用途本质——存储数据来看,赋予堆栈和堆可执行的属性是不必要的。而且堆栈和堆作为程序与用户交互的窗口,赋予其可执行的权力对系统安全构成了威胁。

4 攻击类型

任何类型的缓冲区溢出攻击要达到预定的攻击目的通常要完成两个任务:在程序的地址空间里安排适当的代码;通过适当初始化寄存器和存储器,让程序跳转到安排好的地址空间执行[7]。根据完成这两个任务时着眼点不同,可以将缓冲区溢出攻击分为以下三种主要类型。

4.1 基于堆栈的缓冲区溢出攻击

当一个函数被调用的时候,系统总是先将被调用函数所需的参数以逆序方式入栈,然后将调用指令后面那条指令的地址(即返回地址)入栈。随后控制转入被调用的函数去执行。程序一般在将需要保存的寄存器的值入栈后开始为被调用函数内的局部变量分配所需的存储空间,从而形成如图 4 所示的堆栈的结构,然后接下去执行其他指令。

基于堆栈的缓冲区溢出攻击[8]之所以会发生主要与堆栈对字符串的处理方式有关(因为大部分溢出都是通过精心构造的字符串来触发的,所以我们这里只考虑字符串的处理),虽然堆栈在为字符数组分配空间时是按其出现的先后顺序从高端内存向低端内存依次分配,但在实际为字符数组赋值时却是按从低端内存向高端内存的方向来操作。由于程序缺少必要的边界检查,所以从图 4 中我们不难发现,如果局部变量中有字符数组存在,只要赋予该数组的字符串足够长,我们就能将上面的返回地址给覆盖掉。这样,缓冲区溢出也就发生了。堆栈中的返回地址决定了函数执行完后的去向,所以只要精心构造溢出用的字符串,将对应返回地址的那几个字节替换成想让程序完成后转向的地址(记作 A),那么在被调用函数执行完后,程序就会转到 A 所指定的地方。如果地址 A 所指定的内存空间事先存放了设计好的攻击代码,那么攻击也就随之发生了。现实中的很多此类攻击正是通过精心设计溢出所用的字符串,通过该字符串将攻击代码和所需的跳转地址植入有此漏洞的程序中的。

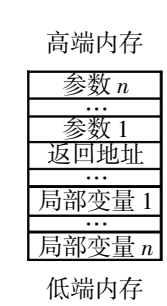

图 4 堆栈结构

图 5 堆的结构

4.2 基于堆的缓冲区溢出攻击

由于程序所需的数据和空间并不是在一开始或程序编写的过程中就能完全确定的,这就要求有一种机制使我们能根据具体情况来分配所需的空间,堆正是应此需求而被提出的。

系统在创建新进程的时候为其分配相应的内存空间作为该进程的默认堆,当然在进程执行过程中用户可以根据需要申请新的堆空间,但不论是默认堆,还是用户后来申请的堆,它们的组织结构是一样的,均如图 5 所示[9]。其中,通过多次实验证实 Windows 平台下双指针区中重复存储了放有下次分配的起始地址的内存的地址。

由于堆在分配和释放时的动态性,使得利用堆进行缓冲区溢出攻击要比利用堆栈难得多,并且基于堆的缓冲区溢出攻击[10]也不像基于堆栈的缓冲区溢出攻击那样有固定模式可用,这就是基于堆的溢出攻击在现实中比较少的原因。基于堆的缓冲区溢出攻击会因为堆溢出后产生的后果不同而有不同的利用方式。常见的利用方式有:

(1) 改写内存参数。通过改写程序中重要变量的值,比如某个字符串的长度等,使程序在运行中再产生其他溢出。这种方法一般只能改写数据段里面的东西,因为该位置相对固定并且可写。

(2) 改变程序中函数指针值。通过溢出将该函数指针的值改为攻击者想让程序跳转到的位置[11],当程序中调用该函数指针指向的函数时,系统转去执行那里的指令,从而达到攻击的目的。

(3) 利用图 5 中所示的双指针区。通过跟踪调试我们发现,Windows 下该双指针区中的指针实际上是下次堆分配时起始地址的存储地址。在堆分配的过程中有类似如下形式的两条写内存的语句:

```
mov[ ecx] , eax
mov [ eax + 4] , ecx
```

其中的 eax 与 ecx 正好存储的是双指针区中的两个指针,设想如果 eax 中存储的是攻击代码的入口地址,而 ecx 中存储另一个合适的值(如存储线程默认异常处理指针的位置),那么在程序随后出现异常而调用异常处理函数时(因为更改了其双指针区)攻击就发生了。所以只要合理设计溢出串,将双指针区中的两个指针用合适的地址覆盖,就会发生攻击。

从上面所述的三种利用方式也可以看出,基于堆的缓冲区溢出攻击并不像基于堆栈的那样简单,它没有较为通用的模式,所需的条件有时也比较苛刻。

4.3 基于 LIB 库的缓冲区溢出攻击

上面两类攻击都是利用自己开发的攻击代码来执行操作,需要解决攻击代码的定位和跳转问题,而基于 LIB 库的缓冲区溢出攻击方法[12]避免了这两个问题并躲过了堆栈和堆不可执

行的保护措施。该方法利用的是系统中现成的 LIB 库函数,不用手动植入攻击代码。这种方法之所以能成功,是因为有些系统函数(如 WinExec( ) , system( ) 等) 是根据参数中给出的函数名称(cmd. exe) 来调用相应的函数。利用这种方法需要解决的问题是,将完成特定目的所需数据存放合适的位置,并想办法使程序跳转到特定库函数的入口。这种方法会用到上面两类攻击的一些技巧。

5　防范措施

面对缓冲区溢出攻击的挑战,根据各种攻击的机制,提出了不同的防范措施。现在常见的防范措施主要有如下几种。

5.1　安全编码

现在缓冲区溢出攻击之所以成为主要攻击手段,除了开发语言本身和操作系统设计策略上的缺陷外,人为因素也占了很大比重。由于以前人们的安全编码意识不强或者急于完成任务而忽略了对所编代码安全性的检查,所以造成隐患。提高编码者的安全意识,消除造成缓冲区溢出攻击的人为因素,可以大大减少这类攻击的发生。这一点可以从已发生过的攻击事件中得以证实。

安全编码首先尽量选用自带边界检查的语言(如 Perl, Python 和 Java 等) 来进行程序开发; 其次在使用像 C 这类的开发语言时要做到尽量调用安全的库函数,对不安全的调用进行必要的边界检查。

编写正确代码,除了人为注意外,还可以借助现有的一些工具。例如,使用源代码扫描工具 PurifyPlus 可以帮助发现程序中可能导致缓冲区溢出的部分。

这种措施的缺点是:对人的依赖性太强,现实中程序员要受多方面因素的影响,很难取得满意的效果。

5.2　非可执行的缓冲区

由第 4 节,我们会发现除了基于 LIB 库的缓冲区溢出攻击外,所有其他的攻击方式之所以得逞正是由于操作系统赋予了缓冲区可执行的属性。取消缓冲区的可执行性属性,虽然会引起操作系统作相应的变化,但对遏制缓冲区溢出攻击的作用却非常显著。

这种方法对基于堆栈和堆的缓冲溢出的攻击效果较好,但对基于 LIB 库的缓冲区溢出攻击却无能为力。

5.3　数组边界检查

通过对所输入的或将传给函数的字符串的长度做严格的边界检查,就可杜绝溢出的发生,也就不可能再发生缓冲区溢出攻击。这种措施的缺点同上述,即对人的依赖性太强。

为了减轻编程者的操作量,人们开发了很多的工具来进行数组边界的检查,常见的有: Compaq C 编译器, Richard Jones 和 Paul Kelly 开发的 gcc 补丁等。

5.4　加强对返回地址的保护

返回地址在缓冲区溢出攻击中扮演了极其重要的角色,攻击者需要借助对它的修改来使程序转向选定的库函数或者向着预先植入的恶意代码方向去执行。通过在系统的其他地方对正确的返回地址进行备份[13],在返回前对两者进行核对,可以成功地阻断攻击。

这种措施仅局限环境遏制通过修改返回地址来实现跳转的攻击方式。

5.5　了解系统常见进程

这要求用户对系统的各种正常进程有个大体的了解。用户通过了解系统中正在运行的进程,可以及时地发现可疑进程,及时终止其运行,从而降低遭受攻击的风险。

5.6　及时打补丁或升级

经常关注网上公布的补丁和软件升级信息,这对那些经常上网的用户来说是一种简单有效的防范措施。

以上六种防范措施是目前最为常见的方法,但它们对人的依赖性太强,增加了相应的人员负担。通过对缓冲区溢出攻击机制的研究,本文提出了第七种防范措施。

5.7　从系统级和编译器两个方面来加强对内存空间的管理

该措施也是从加强边界检查的角度来防止缓冲区溢出攻击的发生,但又不同于 5.3 节,因为 5.3 节把边检查的任务留给了编程者,受其他因素的影响太大,而这儿所说的措施是从系统和编译器两个方面来加强对边界的检查。

在系统级,所要做的改进是为系统中所有可能引起缓冲区溢出的函数增加用于进行溢出检查的代码,并且将其置于该函数的开始位置,以尽可能早地发现和防止溢出。

在编译器方面,当我们编译含有字符串数组的程序时会发现编译器在程序开头预留出足够的空间后,接着就将这些空间进行初始化。如下面的程序:

```
void main( )
{
char buf[ 8]
int num[ 512]
stcpy( buf, " abcdefghijkl" ) ; //此处产生溢出
num[ 0] = 1;
}
```

其汇编代码如下:

```
00401010    push    ebp
00401011    mov     ebp, esp
00401013    sub     esp, 848h
00401019    push    ebx
0040101A    push    esi
0040101B    push    edi
0040101C    lea     edi, [ ebp - 848h]
00401022    mov     ecx, 212h
00401027    mov     eax, 0CCCCCCCCh
0040102C    rep stos    dword ptr [ edi]
```

从这段汇编代码我们能够看出,虽然编译器为函数中的字符串数组分配了空间,但并没有标志数组的末尾位置,所以根本无法正确测量字符串数组的确切大小。为了使前面在系统级所做的改进正常工作,我们应该修改现有编译器的初始化操作,使它在字符串数组的末尾加上结束标志“\0”。这样,每当进行可能会引起溢出的操作时,总是先对源串和目的串的大小进行测量,然后将两者进行比较,如果源串的大小大于目的串的大小,那么就对源串进行截尾操作; 或者直接报错,退出。下面是对 strcpy( ) 函数所做的修改:

```
char *  strcpy( char *  strDestination, const char *  strSource)
{  int lenSource, lenDestination;
lenSource = strlen( strSource) ;
lenDestination = strlen( strDestination) ;
if( lenDestination < = lenSource)
//截尾或者报错退出
…
}
```

上面是以堆栈为例进行说明,对于堆应对其相应的分配函

数进行修改,使其在所分配空间最后加上结束标志“\0”。

该方案的优点在于:所有的防范工作均由系统和编译器来完成,减轻了程序员的编程负担,与常见的防范措施相比它有更普遍的应用性,而且更为有效。

6 结论

本文首先对缓冲区溢出攻击的原理、技术原因和攻击类型进行了探讨,然后给出了现在常见的几种防范措施,最后在对造成攻击的技术原因和现有防范措施研究的基础上,提出了自己的“从系统级和编译器两个方面加强对内存空间分配的管理来防止缓冲区溢出攻击”的方法。该方法通过对编译器及系统级相关函数作相应的修改来加强边界检查,从而防范缓冲区溢出攻击的发生。

参考文献:

[1] ark W Eichin, *et al*. An Analysis of the Internet Virus of November 1988[C]. IEEE Computer Society Press, 1989. 326-344.

[2] 微软网站. What You Should Know about the Blaster Worm[EB/OL]. http://www.microsoft.com/security/incident/blast.asp, 2003.

[3] 中国网络信息安全. “震荡波”E 变种病毒全面分析报告[EB/OL]. http://www.china-infosec.org.cn/virus/index.php?id=7484&page=7484, 2004.

[4] ICAT. Top Ten List[EB/OL]. http://icat.nist.gov/icat.cfm?function=topten, 2003.

[5] Mark Donaldson. Inside the Buffer Overflow Attack: Mechanism, Method, & Prevention[EB/OL]. http://rr.sans.org/code/inside_buffer.php, 2002.

[6] 瑞星网站. 对于缓冲区溢出的保护方法[EB/OL]. http://it.rising.com.cn/newSite/Channels/Safety/SysSafety/Safe_Other/200210/31-153601950.htm, 2002.

[7] Crispin Cowan, *et al*. Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade[C]. DARPA Information Survivability Conference and Exposition, 2000.

[8] Aleph One. Smashing the Stack for Fun and Profit[EB/OL]. http://phrack.org/show.php?p=49&a=14, 2003.

[9] Isno. Windows 下的 HEAP 溢出及其利用[EB/OL]. http://www.xfocus.net/articles/200205/397.html, 2002.

[10] Conover, Matt. w00w00 on Heap Overflows[EB/OL]. http://www.w00w00.org/files/articles/heaptut.txt, 1999.

[11] Kyung-Suk Lhee, Steve J Chapin. Buffer Overflow and Format String Overflow Vulnerabilities[J]. Software: Practice & Experience, 2003, 33(5).

[12] Litchfield, David. Non-stack Based Exploitation of Buffer Overrun Vulnerabilities on Windows NT/2000/XP[EB/OL]. http://www.nextgenss.com/papers/non-stack-bo-windows.pdf, 2002.

[13] Jun Xu, *et al*. Architecture Support for Defending Against Buffer Overflow Attacks[EB/OL]. http://www.crhc.uiuc.edu/EASY/Papers02/EASY02-xu.pdf, 2002.

作者简介:

王业君(1979-),男,硕士研究生,主要研究方向为网络安全、恶意代码;倪惜珍(1947-),女,研究员,主要研究方向为大型计算机网络安全;文伟平(1976-)男,博士研究生,主要研究方向为网络安全、恶意代码;蒋建春(1971-),男,博士,主要研究方向为信息安全对抗技术。

(上接第 92 页)

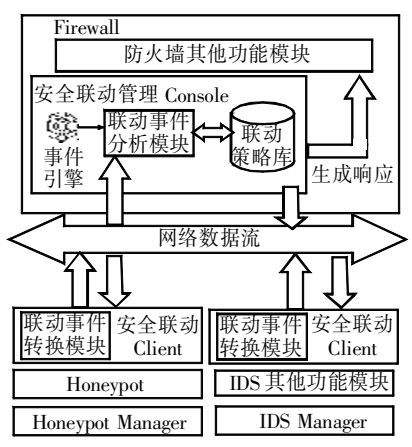

图 3 基于主机的安全组件之间的联动图

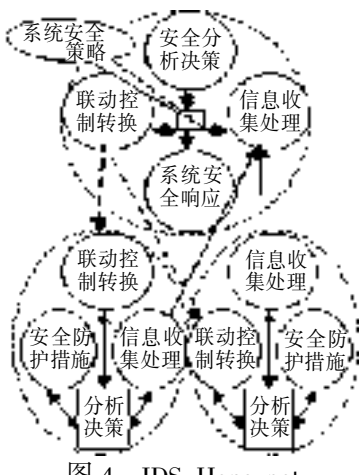

图 4 IDS,Honeypot,Firewall 之间的联动原理

安全防护措施模块指 IDS, Honeypot 自己的独立的安全防护功能。IDS 主要是入侵检测功能, Honeypot 主要是协助诱骗的功能。安全分析与决策模块主要是针对 IDS, Honeypot, Firewall 各自的模块单独的分析与决策。联动控制与转换模块负责三大部分联动控制,职责主要是防火墙根据系统安全策略作出的决策对 Honeypot 和 Firewall 协调控制。安全信息收集与处理模块主要是使不同信息格式的 IDS, Honeypot, Firewall 之间能够进行通信,最终将所有的信息收集到 Firewall 部分,以便进行集中的决策与处理。系统安全响应模块主要是负责主机的响应与动作,根据决策对整个主机进行保护,并负责将审计信息传递给网络化集成防卫系统的审计中心。

3 结论

本文提出的动态 Honey 的设计方案及其控制方法,在“863”计划(网络协同安全技术)中得到了很好的验证,充分利用了最新的主动式动态防御的思想,使网络的安全防卫由被动转为主动,系统的防御能力得到动态的提升。模型内各安全组件实现了协同与联动,提高了对抗黑客的能力以及减轻了管理员的负担。主要创新之处如下:

(1) 在集成化网络防卫系统中,引入了 Honeypot 和 Honeynet 的概念,构成了多层次、立体的主动监控体系。在此两种诱骗技术的基础上,发展出了基于嵌入式 Honeynet 的欺骗空间技术(Embedded-honeynet Based Deception Space Technology),提高了欺骗质量。

(2) 在网络协同安全系统的安全组件的控制中,采用两层控制方法,大大地提升了分布式网络中的系统监测和控制能力。

(3) 各安全部件之间的协同和联动控制,有效地增强了系统的综合防御能力和入侵响应能力,为集成化解决网络安全问题提出了新的思路,并在实际的网络协同安全系统当中得到了良好的应用。

参考文献:

[1] ance Spitzner. Dynamic Honeypots[EB/OL]. http://www.securityfocus.com/infocus/1731, 2003.

[2] 史美林,向勇,等. 计算机支持的协同工作理论与应用[M]. 北京:电子工业出版社, 2000.

[3] Jed Pickel. Enabling Automated Detection of Security Events that Affect Multiple Administrative Domains[Z]. Carnegie Mellon University Information Networking Institute, 2000.

作者简介:

张博(1977-),男,陕西富平人,博士研究生,主要从事网络安全、电子商务方面的研究;王万诚(1960-),男,辽宁沈阳人,副教授,博士研究生,主要从事网络安全、管理决策、软件工程方面的研究;潘炜(1979-),男,湖北武汉人,博士研究生,主要从事计算机网络安全、多媒体通信技术方面的研究;李伟华(1951-),男,湖北黄冈人,教授,博士生导师,主要从事网络安全、多媒体通信方面的研究。