缓冲区溢出攻击及防御*)

詹 川 卢显良 袁连海
(电子科技大学计算机科学与工程学院 成都610054)

摘 要 缓冲区溢出在CERT报道的安全事件中占到一半。缓冲区溢出攻击是当前十分常见的一种攻击。本文首先介绍了C程序的存储空间布局,然后具体说明了缓冲区溢出攻击的原理、攻击类型和方法,接着对目前主要的几种防御工具软件的原理、优缺点进行了分析,最后,总结归纳防御缓冲区溢出攻击的基本方法。

关键词 缓冲区溢出,堆栈,黑客攻击,动态防御,静态防御

Buffer Overflow Attack and Prevention

ZHAN Chuan LU Xian-Liang YUAN Lian-Hai
(College of Computer Science and Engineering, UESTC of China Chengdu 610054)

Abstract It is reported that buffer overflows stand for about 50% of the vulnerabilities reported by CERT. Buffer overflow attack has become a rather popular form of attacks. This paper, above all, introduces memory layout of program; second, describes theory of overflow, attack types as well as attack methods; third, analyzes theory, advantages and disadvantages of present main prevention tools, at last, summarizes basic prevention methods of buffer overflow.

Keywords Buffer overflow, Stack, Heap, Hacker attack, Dynamic prevention, Static prevention

1 引言

在当今信息时代,越来越多的应用软件给我们的生活、工作带来极大的方便,然而,由于一些软件存在着安全隐患,甚至有时会给我们带来意想不到的巨大损失。根据卡内基梅隆大学合作中心(CERT)的统计[1],近五年中每年发现的软件漏洞数以近乎一倍的速度递增,见图1。美国加尼福利亚大学的David Wagner等在分析CERT报道的软件漏洞事件中,发现缓冲区溢出大约占50%[2]。早在1988年11月,世界上第一种缓冲区溢出攻击的病毒Morris蠕虫就造成全世界6000多台网络服务器瘫痪[3],它就是利用两个普通的UNIX程序sendmail和finger的缺陷,成功进行了缓冲区溢出攻击。这种缓冲区溢出的缺陷相继在许多操作系统和程序中发现。如BSD上的打印守护进程缓冲区溢出、SunOS上Sloaris中的ToolTalk缓冲区溢出、Linux中的Linuxconf缓冲区溢出、Windows上的Outlook溢出和IIS4.0/5.0服务器缓冲区溢出[12]以及Oracle上的缓冲区溢出。可见缓冲区溢出攻击广泛存在,并且危害性极大。

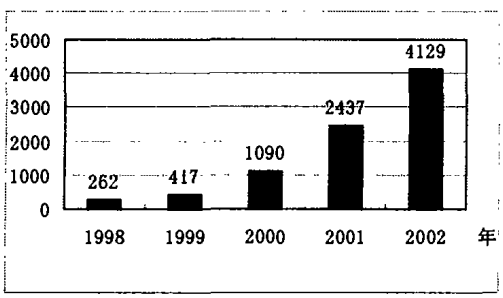

图1 1998—2002年CERT统计的软件漏洞数

攻击者通过缓冲区溢出攻击可以更改缓冲区的数据、注入恶意代码、改变程序的控制权、使未授权的用户获得管理员权限,以致可以非法执行任意代码。从缓冲区溢出攻击第一次出现到现在,大量信息安全的研究者致力于如何尽量避免缓冲区溢出的产生,及时发现软件中缓冲区溢出的漏洞,有效防御缓冲溢出攻击的研究,产生了不少有用、有效的缓冲区溢出防御的方法和技术。

本文分析了缓冲区溢出攻击的机理、攻击类型和方法,指出了几种主要防御工具软件的优缺点,总结归纳出防御缓冲区溢出攻击的基本方法和对策。第2节简介C程序空间布局,缓冲区溢出攻击的原理和方法;第3节介绍目前主要几种缓冲区溢出防御技术,分为动态防御和静态防御;第4节归纳总结缓冲区溢出防御的基本方法;最后是全文的结束语。

2 缓冲区溢出攻击

2.1 C程序的存储空间布局

为更好地理解缓冲区溢出攻击是如何实现的,先让我们来了解C程序执行时在内存中的分布状况。程序运行时,程序是分别加载到内存中几个分区中,即栈、堆、BSS段、数据段和代码段。它们各自有相应的功能,由编译器负责把程序装入相应的区域。图2显示在UNIX操作系统中C程序的存储空间布局,在MS Windows系统中也具有基本相同的模型[4]。

1. 栈:它是在运行时分配的,函数参数、局部变量和调用者的环境信息存储在此区,栈存储模式是先进后出,通过压栈和弹栈来操作数据,同一个程序中,所有的函数调用使用的同一个栈。通过基址指针和堆栈指针来对栈进行寻址。栈的增长是从高址向低址发展的,但对栈的处理却是由低端到高端。

2. 堆:跟栈不同,它的增长由低端到高端,处理也是由低

*)本文受国家95重点攻关项目支持。詹 川 博士研究生,主要研究方向:计算机网络,信息安全,人工智能。卢显良 教授,博士生导师,主要研究方向:计算机网络,分布式系统,信息安全。袁连海 博士研究生,主要研究方向:计算机网络,信息安全。

端到高端。运行时动态申请内存(如通过 malloc 函数)的局部变量存储在此区。

3. BBS 段:所有没初始化的全局变量或静态数据在编译时被存储在此区,运行时被初始化为0。

4. 数据段:所有已初始化的全局变量、已初始化的静态变量和常数在编译时被存储在此区。

5. 代码段:可执行代码存储在此区,该区一般为可读不可写,防止代码意外被修改。

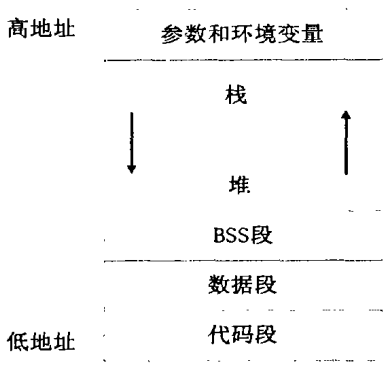

图2 C 程序的存储空间布局

2.2 缓冲区溢出攻击的原理

缓冲区溢出攻击是利用程序不对输入的数据进行边界检查的缺陷,向一个给定缓冲区中写入过量的数据,当数据量超过给定缓冲区的大小时,输入数据溢出并覆盖缓冲区邻近的数据。黑客利用这种方法,精心设计写入的数据,可导致程序去执行恶意代码、让系统死机或非法获得系统访问权。这种攻击可以在本地发起,也可以通过网络远程发起。

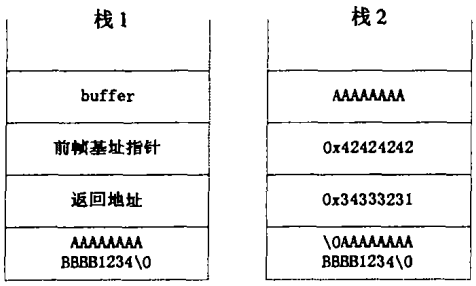

图3 一种简单的溢出攻击

缓冲区溢出在大多数操作系统中都存在,下面的介绍主要以 UNIX 系统为基础,Axelsson[13]比较了 Windows NT 和 UNIX 系统,对于已知的攻击它们存在大致类似的缺陷。下面是个缓冲区溢出的简单例子。图3为该程序执行过程中内存中的分布情况。

```
int main(int argc, char**argv)
{
    char buffer[8];
    strcpy(buffer, argv[1]);
    return 0;
}
```

图3中的栈1为调用 main 函数时的状态,其传入的参数 argv[1]为字符串"AAAAAAAABBBB1234",栈2为把参数 argv[1]拷入 buffer 后的状态,由于 buffer 只有8个字节,于是发生了溢出,覆盖了 buffer 下面的前帧基址指针和返回地址。0x42424242和0x34333231分别为字符 BBBB 和1234的 ASCII 码的值。

缓冲区溢出攻击一般有以下三个步骤:

1. 植入攻击代码或参数到正在运行的程序中;

2. 对给定的缓冲区溢出,篡改缓冲区邻近的数据;

3. 改变程序的控制流,让其执行攻击代码。

在攻击的过程中,主要对以下四个目标进行攻击:

1. 栈中的返回地址。攻击者利用缓冲溢出,写入精心设计的数据,改变返回地址,使它指向攻击代码。

2. 栈中的前帧基址指针。攻击者首先伪造一个帧,使它的返回地址指向攻击代码,然后用缓冲区溢出改写前帧基址指针,使之指向伪造帧的地址,当其返回后,程序执行控制权转移给伪造帧,接着伪造帧再把控制权传给攻击代码。

3. 函数指针。它可存储在堆、BSS 段或数据段,也可能以局部变量或参数的形式存在于栈中。攻击者通过篡改让函数指针指向攻击代码,当此函数指针被使用时则执行攻击代码。

4. longjmp 缓冲区。longjmp 和 setjmp 函数是成对使用的,用于直接跳出多层函数调用。Longjmp 缓冲区纪录了用于恢复 setjmp 设置点运行环境的参数,包括一个基址指针和计数器。如果改变计数值让其指向攻击程序,当调用 longjmp 函数时就会发生攻击。

根据溢出攻击的不同地方,缓冲区溢出攻击大致分为两类:

1. 栈溢出攻击。这是最常见、最主要的缓冲区溢出攻击方式[14],通过这种方式可以改写栈中的返回地址、前帧基址指针、函数指针以及在栈中植入攻击代码等等。

2. 堆/BSS 的溢出攻击。这种攻击相对较少而且相对较困难些。w00w00是一种堆溢出攻击,Matt Conover 等人对其作了深入的分析和研究[5]。通过溢出改写存在堆或 BSS 中的函数指针来获得控制。

3 缓冲区溢出攻击的防御

缓冲区溢出攻击防御分为两大类:静态防御和动态防御。

静态防御是从源代码中找出程序的漏洞并进行修改,虽然找出所有的漏洞是不可能的,但是找出已知攻击的漏洞并修改,可以大大降低被攻击的可能性。静态防御不足的是需要程序的源代码,但对用户来说,有时是不可能的,并且需要不断升级已知攻击的数据库,一旦找到漏洞后,用户要知道如何去修补漏洞。

动态防御则是在有漏洞的程序运行时,生成一个保护环境,让这些利用漏洞攻击的方法不能奏效。但是它并没有去除程序中的漏洞,在平常环境下漏洞依然存在。动态防御不需要程序的源代码。可以防御新的恶意代码对已受保护的目标的攻击,但是对未保护的新目标攻击则无能为力。

下面介绍目前主要几种动态防御和静态防御软件及其原理。

3.1 动态防御

3.1.1 StackGuard 该软件[6]由 Crispin Cowan 等人开发,是一种简单的编译器技术(gcc 补丁),它不需要改变程序的源代码,能有效地防止针对栈中返回地址的溢出攻击。

在对栈中返回地址的攻击中,攻击者一般首先写栈中局域变量区,然后从高地址到低地址依次重写前帧的基址指针区和返回地址区。StackGuard 采取的策略是在前帧基址指针与返回地址之间加入一个虚构字段,称为侦探字段(canary value)。在返回地址之前,检查侦探字段是否与原先一致,否则就报警并终止执行。为避免攻击者在改写返回地址时能正确重写侦探字段的情况,StackGuard 采取了两种策略:一种是运行时随机生成侦探字段;另一种是由换行符、-1、Null、回

车符组成的侦探字段。对于通过指针来改写返回地址，而不改动侦探字段的攻击，StackGuard 通过增加侦探字段与返回地址的异或值的比较来诊断。

3.1.2 Stack Shield 其也是一个由 Vendictor[7]开发的 gcc 编译器的补丁，它能防止针对返回地址和函数指针的溢出攻击。

对于返回地址的攻击，它采用如下的策略：当函数被调用时，返回地址不仅被压入栈，同时也被拷贝到一个受保护的全局数组中，在函数返回时，通过比较栈中的返回地址是否与全局数组中的一致，来判断是否受到攻击。

在防止函数指针方面，因为函数指针一般指向程序内存的文本区，而注入的攻击代码只可能存储在内存的数据段、BSS 段以及堆栈中。如果能保证函数指针的指向的是文本段，就能防止攻击者利用函数指针的攻击。因此，Stack Shield 利用这个原理，首先声明一个全局变量，存储在数据段，以它作为判断是在文本段还是在数据段的边界地址。在程序使用函数指针时，检查其所指地址是否低于边界地址，否则终止程序。但是对于使用动态分配的函数指针，则将产生误报。

3.1.3 Libsafe 其是由 Arash Baratloo 等人写的一个应用在 Linux 上的动态链接库[8]，它不需要改变源代码和重新编译。它对 C 语言中存在漏洞的函数，如 strcpy、strcat，进行了封装。在调用这些存在漏洞的函数时，先进行范围检查。由于压入栈的局部变量地址不应低于前帧的基址指针地址，Libsafe 就把前帧的基址指针地址作为边界地址，比较局部变量的输入长度与缓冲区允许的大小，如果输入的长度大于缓冲区的允许范围，则推断将发生缓冲区溢出攻击，于是发出报警并且程序终止运行。

3.1.4 Fault Containment Wrappers 其是由 Christof FETZER 等人开发的[9]，类似于 Libsafe 的 Linux 动态链接库，但它主要防御的目标是通过堆溢出改写函数指针的攻击。它对 C 语言中对内存操作的函数进行了封装，比如在调用内存动态分配 malloc()函数时，同时把动态分配的内存大小和位置记录在一个单独的内部表中。当对堆进行写操作时，如 strcpy(dst,src)，先进行边界检验，确保操作在相应的动态分配内存块上，防止溢出覆盖堆中别的区域，以致攻击者不能通过堆来进行溢出攻击。

3.2 静态防御

3.2.1 ITS4 2000年，Cigital 公司的研发人员开发了一种用来诊断 C 和 C++代码安全缺陷的静态分析工具，他们把它称之为 ITS4[10]，它能提供词法分析，但不提供语法分析。在分析源代码时，它会查找源代码中是否存在数据库中已知的、存在漏洞的函数，如果存在，它会给出对存在问题的描述、解决意见、危险等级等等，帮助编程人员迅速修改程序中的漏洞。

3.2.2 Splint David Evans 等人开发了 Splint 静态分析工具[11]，它是利用有开发者提供的语义注释，对源代码执行语法级的静态分析。它比工作在词法级的分析工具具有更好地区分是否正确调用函数的能力。语义注释给出了函数正确使用的限制，以确保函数能被正确地调用。

4 缓冲区溢出防御方法

从现有的缓冲溢出防御软件来看，基本的防御方法大致归纳为以下五类：

4.1 堆栈不可执行

大多数操作系统并不需要堆栈是可执行的，然而大部分通常的缓冲区溢出攻击是把攻击代码写入堆栈中，然后让其执行攻击代码。所以一种简单、可行、相对有效的方法是让堆栈区不可执行。如在 Linux 和 Solaris 上都提供了类似的补丁[15,16]。这种方法程序不会有额外的开销，不需要重新编译源代码。

然而，此方式存在严重的缺陷。首先，有些程序虽然少，但确实需要堆栈是可执行的；更重要的是，攻击者可以把攻击代码注入到内存中别的区域，只要知道写入的地址，然后设法把程序执行权转交给它，攻击一样可以发生。Wojtczuk 就成功地在具有不可执行堆栈的 Solaris 系统上实施了缓冲区攻击[17]。

4.2 对源代码进行安全漏洞分析

利用专用安全分析工具软件，如 ITS4和 Splint 等，对源代码进行词法、语法上的分析，根据已知缓冲区溢出漏洞的数据库，找出程序中可能存在的安全隐患，并给予相应的解决提示，帮助编程人员迅速地修改程序。但对于一般用户来说，这种方法不可行。因为，第一，对于用户，有时源代码是不可得的；其次，即使用此方法分析出程序的许多安全隐患，但用户并不知道如何去修改这些缺陷。

4.3 改善编译器

StackGuard 和 StackShield 都采用该方法。通过改进编译器，增加缓冲区完好检验、输入数据的边界检查等，防止缓冲区溢出，让存在隐患的程序编译不能通过。这种方法为防止缓冲区溢出攻击提供了较好的解决方法，但是它需要程序的源代码，需要对其重新编译，增加了程序运行时的额外开销，存在一定的误报率，并且没有真正除去程序中的安全漏洞，当程序在普通环境运行时，漏洞依然存在。

4.4 使用安全的编程语言

用 C 语言写的程序通常易受到缓冲区溢出攻击，因为，其一，在 C 语言中没有提供对数组、指针和引用的边界的自动检测；更重要的是，标准 C 函数库中提供的有些函数是不安全的，如 strcpy，scanf 等。程序员在调用这些函数时，如果没有进行仔细的检查，就很容易留下受到缓冲区溢出攻击的漏洞。

使用 Java 语言可以避免上述情况，因为 Java 在对缓冲区操作时自动提供了边界检查。但是 Java 也不一定绝对安全，因为 Java 虚拟机是用 C 语言写的，Java 虚拟机也会受到缓冲区溢出攻击[18]。

4.5 使用更安全的函数库

如上面提到的 Libsafe，Fault Containment Wrappers 以及应用在 Windows NT 系统上的 BOWall[19]都是对标准函数库中有漏洞的函数进行封装，加入安全检查。它们都是以动态链接库的形式存在于系统中，对于用户是透明的。当程序调用有缺陷的函数时，它们会自动加载安全检查，通过检查后才真正去调用该函数。这种方法给用户带来了极大的便利，不需要源代码，用户不需要对程序做任何的改动。但是，在没安装此安全动态链接库的环境下运行程序，程序的缓冲溢出缺陷依然存在；并且对于程序调用的不是标准函数库的函数时，它们也是无能为力的。

结束语 本文详细介绍了缓冲区溢出攻击的原理、攻击方法，分析了当前几种主要的缓冲区溢出攻击防御工具，归纳总结了缓冲区溢出防御的基本方法。但是，到目前为止还没有

（下转第75页）

情况下 XMLGenerator 和 XMLSaver 与 XSU 的性能比较数据，测试平台为 Windows2000，JDK 版本为1.4.1，测试机配置为 P41.6G，内存256M，数据库为 Oracle 8i。

XSU 和 XMLGenerator 的性能比较

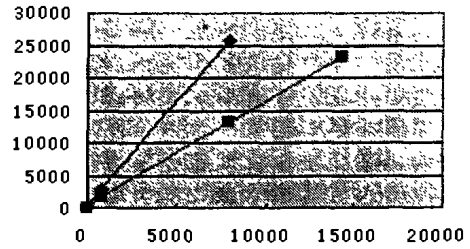

图7 flat XML→关系数据库

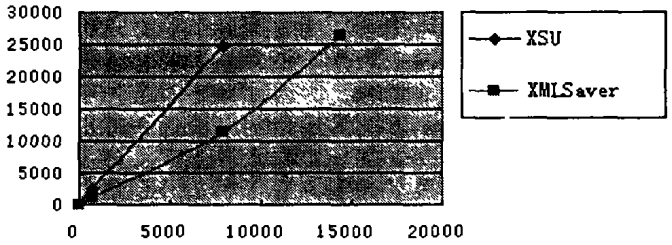

图8 复杂结构的 XML→关系数据库

以上图5～图8的横轴为记录的个数，纵轴为运行时间(ms)，这个数据是多次测试的平均值。从图中可以看出：

(1) 从关系模型向 XML 模型转换时，如果只是简单的 flat XML，不涉及到复杂的映射机制，这种情况下二者的性能相当；如果生成的 XML 中有重复结构，XMLGenerator 的处理时间并没有大幅的提高，而 XSU 中由于 XSLT 处理复杂结构效率的低下，使得整体效率与 XMLGenerator 相比显得差距很大。

(2)从 XML 模型向关系模型转换时，对于 flat XML 和复杂结构的 XML，XMLSaver 的性能优势基本相同，因为不清楚 XSU 内部的处理机制，所以这里不能从底层处理机制的角度进行解释，但是从测试结果看，XMLSaver 的性能优势还是很明显的。需要说明的是，XSU 在处理数据量大的时候，在图中反映为数据记录为10000多行时，就发生了溢出错误，而 XMLSaver 表现得更为稳定。

为了保证测试结果的正确性及通用性，我们使用了不同的数据库，例如 MySQL、Access、PostGreSQL，并且在 Solaris 及 Linux 等平台下做了同样的测试，都得到了相同的结果。

结语 本文介绍了一种 XML 与关系数据库信息交换的方法，利用自身定义的映射规则集和映射方法，实现关系模型和 XML 之间双向的转换。与目前通用的方法相比，本方法的优点是没有采用书写复杂的 XSLT 转换 XML 的结构，而是采用了自身定义的映射规则，比 XSLT 的编写要简单得多；并且将从数据库生成 flat XML 以及从 flat XML 到目标 XML 的转换过程合并为一个过程，因此简化了处理流程，从而达到提高性能的目的。整个系统的核心是映射规则集的设计，对于特别复杂的结构，映射机制仍然不是很完善，今后工作的重点仍然在映射机制的研究和规则的扩展等方面，设计出一个完善的转换系统。

参 考 文 献

1 Zhang Xin, et al. The XQuery! --- An XML Algebra Optimization. Approach Workshop on Web Information and Data Management. ACM Press New York, NY, USA, Nov. 2002

2 XSL Transformations (XSLT) Version 1.0. W3C Recommendation, 16 November, 1999. http://www.w3.org/TR/xslt

3 蔡飞，等. 基于关系数据库的 XQuery 查询的实现. 计算机科学，2004(5)

4 陶列骏，等. XML 与数据库的存取技术：[硕士学位论文]. 南京：南京大学计算机系，2003

5 XML Path Language (XPath) Version 1.0. W3C Recommendation. 16 November 1999. http://www.w3.org/TR/xpath

(上接第60页)

一个完全的解决方法，作为编程人员应该清晰地认识到缓冲区溢出攻击的普遍性和危害性，增强安全意识，养成安全编程的思想，对软件进行全面、充分的测试，写出高质量的软件，切除缓冲区溢出的根源。

参 考 文 献

1 http://www.cert.org/stats/cert_stats.html

2 Wagner D, Foster J S, Brewer E A, Aiken A. A first step towards automated detection of buffer overrun vulnerabilities. In: Proc. of Network and Distributed System Security Symposium, Catamaran Resort Hotel, San Diego, California, Feb. 2000. 3～17

3 Spaford E. The Interner Worm Program : Analysis Computer Communication Review, Jan. 1989. 3～17

4 DilDog. The tao of Windows buffer overflow. http: //www.cultdeadcow.com/cDc_files/cDc-351/, April 1998

5 Conover C, w00w00 Security Team. w00w00 on heap overflows. http://www.w00w00.org/files/articles/heaptut.txt, Jan. 1999

6 Cowan C, et al. StackGuard: Automatic adaptive detection and prevention of buffer overflow attacks. In : Proc. of the 7th USENIX Security Conf. San Antonio, Texas, Jan. 1998. 63～78

7 Vendicator. Stack Shield technical info file v0.7 http://www.angelfire.com/sk/stackshiedl/, Jan. 2001

8 Baratloo A, Singh N, Tsai T. Libsafe: Protecting critical elements of stacks. White paper http://www.research.avayalabs.com/project/libsafe/, Dec. 1999

9 Fetzer C, Xiao Z. Detecting heap smashing attacks through fault containment wrappers. In: the Proc. of the 20th symposium on Reliable Distributed Systems, Oct. 2001

10 Viega J, Bloch J T, et al. ITS4: A static vlnerability scanner for C and C++ code. In: Proc. of the 16th Annual Computer Security Applications Conf. Dec. 2000

11 Larochelle D, Evans D. Statically detecting likely buffer overflow vulnerabilities. In: Proc. of the 2001 USENIX Security Symposium, Washington DC, USA, Aug. 2001

12 Cole E 著，苏雷，等译. 黑客一攻击透析与防范. 北京：电子工业出版社，2002

13 Axelsson S. A comparison of the security of windows NT and UNIX, 1998. http://www.securityfocus.com/data/library/nt-vs-unix.pdf

14 Levy E. Smashing the stack for fun and profit. Phrack Magazine, 1996, 49(14)

15 Dik C. Non_executable stack for solaris, posted to comp.secuity.unix Jan. 1997. http://x10.dejanews.ocm/

16 the Linux OpenWall Project. Nonexecutable stack patch for Linux. http://www.openwall.com/linux

17 Wojtczuk R. Defeating solar designer's nonexecutable stack patch. in Bugtraq, Jan. 1998

18 Dean D, Felten E W, Wallach D S. Java Security: from hotjava to netscape and beyond. In: Proc. of the IEEE Symposium on Security and Privacy, Oakland, CA, 1996

19 Kolishak A, BOWall. buffer overflow protection for Windows NT, 2000. http://developer.nizhny.ru/bo/eng/BOWall