缓冲区溢出研究综述*)

林志强 夏 耐 茅 兵 谢 立
(南京大学软件新技术国家重点实验室 南京210093)
(南京大学计算机科学与技术系 南京210093)

摘 要 近年来非常流行的极具破坏力的系统攻击是基于缓冲区溢出漏洞的攻击,如今互联网上的绝大多数计算机系统都受着或潜在地受着缓冲区溢出漏洞的威胁。在本文中,我们研究了各种类型的缓冲区溢出漏洞和常见的攻击手段及防御措施,并且对这些研究成果进行了分析总结。最后阐述了我们对此问题的观点。

关键词 缓冲区溢出,计算机安全

A Survey of Research on Buffer Overflows

LIN Zhi-Qiang XIA Nai MAO Bing XIE Li
(State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing Jiangsu 210093)
(Department of Computer Science and Technology, Nanjing University, Nanjing Jiangsu 210093)

Abstract Buffer overflows are getting more and more popular in system security these years, which attack can inflict upon almost arbitrary programs. And as we have known, buffer overflow is one of the most common vulnerabilities that can seriously compromise the security of a network-attached computer system. This paper surveys all kinds of buffer overflows vulnerabilities, the most commonly used attacks, the respective defense methods related to their vulnerabilities, and the comparative summarizations up-to-date researches in this area. At last, we also present our point of view on buffer overflows.

Keywords Buffer overflows, Survey, Attacks, Defense, Computer security

1 引言

缓冲区是程序中分配的一段数据区域,缓冲区溢出就是通过往程序的缓冲区写超出其长度的内容,造成它的溢出,从而破坏程序的数据区域(堆或者栈、或者静态数据区等),使程序转而执行其它指令,以达到攻击的目的。

据有关资料记载,缓冲区溢出在上个世纪60年代的时候就被发现并利用来进行系统攻击[1]。不过那时最多也停留在实验阶段,仅为极少数人知道而已。直到1988年,Morris 的蠕虫事件(利用运行在 Free BSD 上的 fingerd 服务程序的缓冲区溢出漏洞),缓冲区溢出才得到广为关注。然后于1996年,Aleph One 在 Phrack 杂志发表了比较有影响的论文"Smashing The Stack For Fun And Profit",该文透彻地分析了利用缓冲区溢出进行攻击的原理和技巧并举了实例,由此许多攻击者受到启发后纷纷转到利用缓冲区溢出漏洞上来。我们通过图1缓冲区溢出所占 CERT 所建议的漏洞数[2]就能发现这点。而且前不久广为传播的 Blaster 病毒也正是利用 Windows 的 RPC 缓冲区溢出漏洞进行攻击的,所以缓冲区溢出问题由来已久并且仍未消除。

引起缓冲区溢出的实质原因是在大多数体系结构的计算机中,程序的代码区和数据区并不完全隔离,控制程序执行的关键数据结构易受数据区的影响,所以当某种高级语言(如

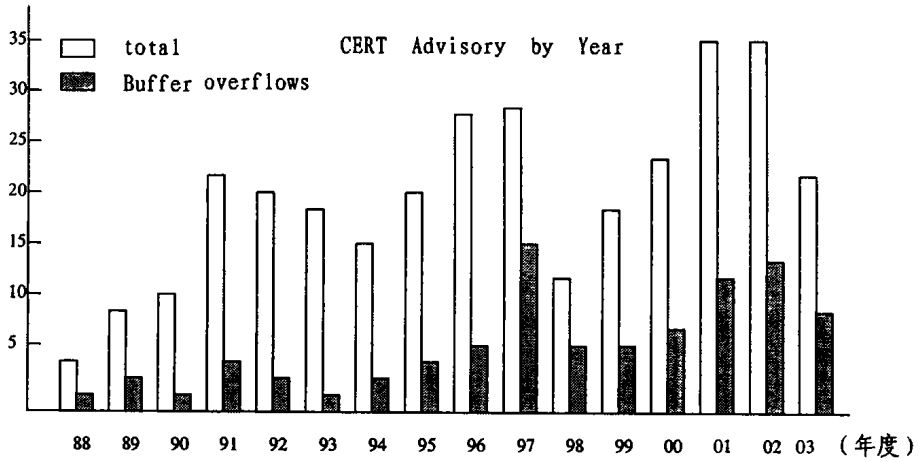

图1 CERT 漏洞建议报告(03年度截至8月底)

*)本课题得到国家"863"高技术(NO:2003AA144010)经费资助。林志强 硕士生,研究方向:安全操作系统,系统安全;夏 耐 博士生,研究方向:信息与网络安全,系统安全;茅 兵 教授,研究方向:系统安全;谢 立 教授,博导,研究方向:安全操作系统,分布式操作系统,信息安全。

C/C++),如果其对数组和指针的引用不进行边界检查,就有可能导致缓冲区溢出漏洞的产生。如果这些漏洞再被攻击者加以利用,轻则导致程序崩溃,产生拒绝服务攻击;重则导致操作系统的权限泄漏,使入侵者控制整个计算机系统等。缓冲区溢出攻击使任何人都有可能取得主机的控制权,所以它代表了一类极其严重的安全威胁。

当前许多重要的系统都是由C/C++开发的,由于C/C++是一种效率优先的语言,对数据类型检查较弱,程序风格自由,并且数组的越界在编译器中不进行检查,再加之程序员撰写安全代码有一定难度性或者说程序员缺乏安全观念,致使缓冲区溢出漏洞比较普遍地存在于这些系统中。另外利用缓冲区溢出漏洞进行攻击相当的简单,带来的危害巨大,直到现在,缓冲区溢出问题都没有完全的解决,所以如何有效地抵御缓冲区溢出攻击得到了不少研究机构的重视,我们目前也正在进行有关这方面的工作。

本文接下来回顾了国内外近年来在缓冲区溢出方面研究的重要成果,通过对它们的分析总结来找出问题所在,以期对我们的研究有所帮助。文章的第2部分,分析了缓冲区溢出攻击的原理,第3部分列举了常见的缓冲区溢出攻击手段,第4部分是当前针对缓冲区溢出的研究进展,最后是总结。

2 缓冲区溢出基本原理

一般可执行文件在装入内存之后都有固定的格式(如PE、ELF等),与存在磁盘时的格式不同。我们以ELF为例,ELF对一个可执行程序分了许多段区[3],通常地如只读的代码区.text,初始化的静态数据区.data,未初始化的数据区.bss,程序运行时动态分配的数据区.heap,保存程序运行轨迹和临时局部变量的栈区.stack,还有其他的如过程连接表PLT和全局偏移表GOT区,包含程序初始化时执行的指令init区,包含程序终止时要执行的指令fini区,以及包含一些全局构造指令和析构指令的ctors和dtors区等。典型的一个程序在内存中的执行映像如图2所示。

在C程序中,缓冲区可以分配在.data、.bss、.heap以及.stack区,而与程序执行流有关的数据结构,如函数指针、函数返回地址等也可以在这些区域,所以当控制程序执行流的敏感数据结构受到缓冲区溢出攻击的时候就有可能改变,从而使正在执行的程序转向,以运行非法代码。这就是我们常说的缓冲区溢出攻击。

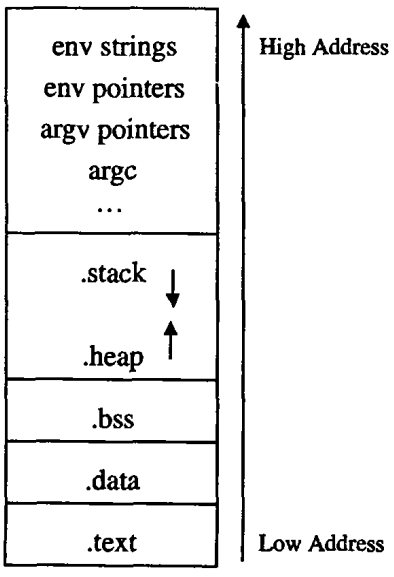

图2 可执行程序内存映像

缓冲区溢出根据程序数据在内存中的增长方式分两种情况,一种是栈溢出,其数据分配是从高地址向低地址方向增长(这是栈比较特别的地方),典型的攻击如覆盖函数返回地址;另一种是堆溢出,程序数据分配按照从低地址向高地址方向增长,溢出点可以发生在.data、.bss、.heap区,攻击通常如覆盖分配在这些区域的函数指针等。

当然,并不是随便往缓冲区中填代码就能达到攻击的目的,这与程序具体的运行环境、程序代码本身和数据的分配情况有关。相比而言,栈溢出比堆溢出更容易奏效,因为堆溢出与程序的代码关联紧密,有堆溢出漏洞的程序比覆盖函数返回地址这种栈溢出程序少得多,而且堆溢出也需要攻击者进行精心设计,比栈溢出难度更大。

3 攻击手段

在C/C++程序中,常见的可能会遭受攻击的与程序执行有关的数据结构有:函数返回地址,栈帧指针,函数指针,其位置可能在.data、.bss、.heap、.stack段区,以及ANSI C提供的setjmp/longjmp函数用到的保存程序执行时栈帧情况的buf,还有可能是malloc库用到的在.heap区的管理数据(即通常所说的malloc库溢出),C++中的虚函数指针VPTR(virtual pointer)[4]等。

缓冲区溢出攻击的目的在于扰乱某些具有特权功能的运行程序,以使得攻击者取得程序的某些控制权限,甚至控制整个主机。要使缓冲区溢出攻击成功,通常有两个关键点。第一,要改变程序执行的前文所述的关键数据结构;第二,要有被执行到的攻击代码。攻击代码可以是攻击者通过参数传递植入程序中的,也可以是直接利用系统已经存在的函数(如libc库中的exec,system等),前提是需要改变传递给这些函数的参数,当然还是离不开溢出这种手段。最常见的具有极大威胁的攻击代码是通过制造缓冲区溢出创建shell,再通过shell执行其它命令,如果该shell具有root权限的话,攻击者就可以对系统进行任意操作。

同时缓冲区溢出攻击是有条件的。对于覆盖函数返回地址和栈帧指针这种类型的攻击,要求函数体内有局部变量buf,有对buf进行操作的且不进行边界检查的处理函数,典型的情况如图3所示。攻击者可以在图中所示的函数的局部缓冲区中植入攻击代码,并且通过如strcpy()这种不进行边界检查的函数来覆盖程序的返回地址或者前一个栈帧指针,当然这时可能要攻击者精心地构造适当的覆盖地址攻击才能成功。这种情况还包括覆盖分配在堆栈中的函数指针的攻击。

对于覆盖函数指针攻击,典型地也要求程序的变量具有如图4所示的逻辑结构。攻击者通过精心构造溢出数据,使得函数指针被覆盖而指向恶意代码。

对于ANSI C提供的setjmp/longjmp函数用到的保存程序执行时栈帧情况的缓冲区溢出,条件是该函数用到的buf也毗邻与其他缓冲区,或者能通过其他手段(比如其他指针指向到该buf区域)修改其中保存的程序栈帧情况。因为setjmp/longjmp函数能进行代码深层的跳转,被跳转到的当时的程序执行环境(如pc、ip等寄存器)是被保存到buf中的,所以当buf被溢出致使控制程序执行情况的敏感数据被修改,从而也能达到攻击目的。

对于malloc库溢出以及VPTR溢出等,比前几种条件更强,也更复杂。malloc函数分配的空间在heap区,管理这些被分配的数据也在heap区,而且有可能与被分配的数据紧邻,那么分配到的heap中的数据被溢出后就会影响到对堆数据

管理的库函数的工作(比如 free 等),如果攻击者再精心构造的话也可能造成极大破坏。VPTR 是 C++特有的情况,我们不再赘述,其核心思想就是通过溢出手段改变虚函数指针,以达到攻击目的。

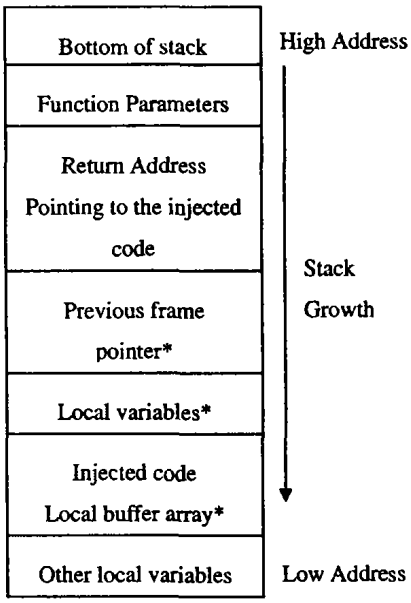

图3 覆盖 Return Address

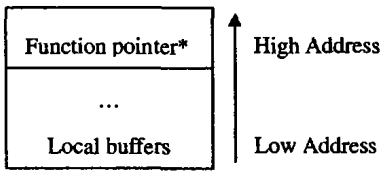

图4 覆盖函数指针情况

4 研究进展

由于缓冲区溢出存在的普遍性和破坏的巨大性,如何有效地抵御缓冲区溢出攻击得到了许多研究机构的重视。截至目前已有很多研究成果。

我们知道,要使缓冲区溢出攻击奏效,攻击者通常需要先通过溢出植入攻击代码,然后改变程序执行流程,最后让攻击代码执行,所以自然而然地,研究者就可以围绕着如何阻止这三步中的任何一步失效来达到抵御缓冲区溢出攻击的能力。比如 Jones & Kelly 针对 gcc 的数组边界检查[11]以及 Compaq C 等通过对编译器进行改进而杜绝溢出产生;通过 StackGuard[8]对返回地址区域附加的 canary 标记是否改变来检测溢出是否发生,RAD(Return Address Defender)[6]的对返回地址保存到其他区域等措施,以允许溢出但不让程序改变执行流等;通过使堆栈不可执行(如 PaX)以及入侵检测手段等,使得即使改变了程序执行流但可以不让攻击代码执行。

程序开发通常是利用编译器将高级语言代码转换成机器可执行的二进制文件,然后将可执行文件通过操作系统的支持载入到内存中运行。那么对缓冲区溢出的防御也可以从程序开发时,通过静态地检查和扫描技术来发现程序中是否有缓冲区溢出漏洞的角度,还可以从操作系统和编译系统角度,通过编译器自动地在程序中添加额外的代码以及让操作系统提供一些辅助的手段,来动态地发现运行的程序是否有缓冲区溢出。我们也是主要基于这两方面来对这些比较有代表性的研究成果进行分析总结,概述其研究进展的,我们重点关注的是动态防御技术。

4.1 动态防御技术

4.1.1 StackGuard StackGuard 可能是提出最早的、目前被引用率最高的缓冲区防御技术[8]。该方法是基于覆盖函数返回地址,则与返回地址相邻区域的数据也会受到改变的这一事实实现的。它通过对编译器 gcc 加补丁,使得在函数入口处能自动地在栈中生成 canary 标记,在函数调用结束检测 canary 标记是否改变来发现并且阻止缓冲区溢出。StackGuard 是一种对函数返回地址进行完整性检查的基于编译器的技术。思想如图5所示。

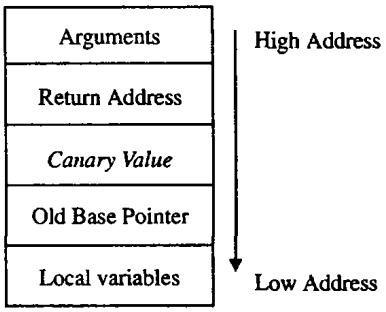

图5 StackGuard 实现机理

但是在上述的方法中,StackGuard 有其局限。比如攻击者预见到了 canary 标记的存在,并且能在溢出过程中同样地制造它们,那么他就能成功地跳过栈保护的检测。基于此,StackGuard 又有如下的两种方案对付这种欺骗,一种是利用在 C 语言中的如0(null),CR,LF,-1(EOF)等不能在常用的字符串函数中使用的终止符号作 canary,因为这些有潜在漏洞的字符串处理函数一旦遇到这些终止符号,函数就结束了;另一种是使用随机符号,利用一个在函数调用时产生的一个32位的随机数来实现,使得攻击者不可能猜测到 canary 中的内容,而且每次调用,canary 中的内容都在改变,也无法预测,这就使得攻击者不能成功构造 canary 标记。如果攻击者直接绕过 canary 来更改返回地址[12],对于这种情况 StackGuard 又采用了双 canary 技术,将其中一个按正常的作法保存在堆栈中,另一个保存 canary 与 Return Address 的 XOR 值,所以 StackGuard 是在不断发展与改进中的。

StackGuard 虽然仅能有效抵御现在和将来的基于栈溢出的对函数返回值的攻击,并且需要对源程序进行重新编译,但事实证明它已经能防御大多数的缓冲区溢出了[5,10],而且能保持较好的兼容性和系统性能,所以 StackGuard 现在已经被集成到安全产品 Immunix™中,用于提供对系统的栈的保护。

4.1.2 PointGuard 由于 StackGuard 仅能保护栈中函数活动记录的返回地址,对其他诸如函数指针类型的攻击不能有效防御,所以在推出 StackGuard 后,Crispin Cowan 等在今年的 USENIX 会议上又提出针对指针型缓冲区溢出攻击的 PointGuard[13],其也是通过对编译器 gcc 的改进来实现的,但目前尚未正式发布。

因为存储在内存中的指针数据等容易受到缓冲区溢出的攻击,而存储在寄存器中的数据则不可能被类似的方法修改。PointGuard 就基于这种思想,在程序装入内存的时候,先将指针型数据加密,当程序访问到这些数据的时候,再在寄存器中进行解密。这样如果攻击者修改了内存中的指针数据,那么经过解密过程后必然使得指针不会指向攻击者期望的地方。当然前提是攻击者不应该知道具体的加解密算法[13]。

为了在实现上的方便和减小开销,PointGuard 用到的加

密过程其实很简单，就是将原来的指针数据与一个32位的随机数XOR(如果和一个固定的值XOR，那保护措施将相当脆弱)，这个随机数可以根据当时程序的执行时间在/dev/random目录中动态产生，并且该随机数不和其他程序共享。这样攻击者将很难绕过PointGuard这道障碍。

经实验数据表明，PointGuard对程序性能大概有0-20%左右[13]的影响，并且被攻击者绕过的可能性非常小，所以不失为好的缓冲区溢出防御办法，当然它也需要重新编译源程序。

4.1.3 ProPolice ProPolice是IBM日本研究院实现的，可能是目前对编译器gcc改动最大的缓冲区溢出防御技术[15]。其主要思想来源于StackGuard，不同之处在于ProPolice将函数中用到的局部变量重新排列，如图6所示，使得函数中用到的缓冲区紧邻Guard Value，这样栈中的其他变量(如函数指针等)就不会受到缓冲区溢出的影响，而且这时候如果缓冲区发生溢出，也会被Guard Value发现。

所以ProPolice比较出色地解决了基于堆栈的缓冲区溢出攻击，不仅能保护函数的返回地址，而且也保护了函数中用到的局部变量，缺陷就是不能防御堆溢出攻击。

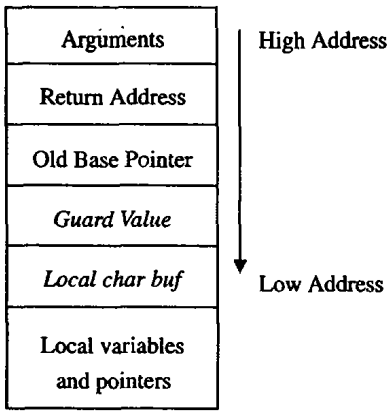

图6

4.1.4 Stack Shield Stack Shield是比StackGuard安全性更好的使系统免受缓冲区溢出攻击的工具[5,14]，它提供对堆栈中的函数返回地址和函数指针的保护。

Stack Shield对函数返回地址的保护是通过全局返回地址栈或者返回地址范围检查实现的[5]，这两种方法在编译的时候可选。在Stack Shield中，全局返回地址栈是系统实现的一个全局性的数组，里面保存着被调用到的函数的返回地址，这样当函数返回时，系统根本不管通常的堆栈中的函数返回地址，而直接用全局返回地址栈中的数据来代替。

Stack Shield对函数指针的保护思想也比较简单。我们知道，在正常情况下，函数指针应当指向代码区.text，而攻击者植入进来的代码通常在.bbs、.data以及.heap区，所以如果函数指针指向了这些区域，那我们就可以假定发生了缓冲区溢出攻击。它在实现的时候是通过在程序的.data开始处(在图2中我们能看出这点)设置一个变量，其地址就作为程序的函数指针访问地址范围的参考，如果超过了该变量的范围，就中止该进程。这一思想的前提是程序员没有使用动态的分配函数指针技术(比如在某些情况下，函数指针可能会访问在堆或者栈中的程序动态创建的代码)，如果出现这一情况就会出现误报。

Stack Shield在2000年的时候发布，但至今尚未有其改进版。

4.1.5 PaX PaX(Page Exec)[4,16]是通过使堆栈不可执行来阻止缓冲区溢出攻击的。我们知道，程序导入内存运行的时候是经过页面映射机制，实现从逻辑地址到物理地址的转换，每一个逻辑地址都对应到具体的某一个物理页面，而每一个物理页面的性质是由属性位进行标识的。PaX通过扩展熟悉位的定义，将.stack/.heap定义为不可执行，当然.text区的代码还是可执行的，这样在程序运行过程中，进程所访问到的每个页面都会先被页面处理函数检查是否可执行的，从而就能比较顺利地发现基于堆或者栈的缓冲区溢出攻击。

如果攻击者利用已加载在内存中的系统库函数进行攻击(比如通过传递恶意参数，触发libc中的exec)，而不是植入代码使其在堆或者栈中运行。基于此，PaX使用了一种ASLR(Address Space Layout Randomization，随机分布地址空间)技术来进行保护。也就是说，系统的库函数在内存中的分布是随机的，在每一台系统上都不一样，这样就杜绝了Return into Libc攻击，因为在那样的系统里，系统库函数通常在某一个固定的地址空间，攻击者经过几次试验就能知道。

PaX是Linux平台安全增强GrSecurity[9]的一个子项目，目前仍有改进版出现。测试数据表明PaX对系统性能的影响是3%－20%。因为PaX是通过对操作系统的修改来实现的，所以现有的程序无需重新编译，不足之处是如果原有程序需要堆栈可执行，就会在该系统上运行不起来。

4.1.6 LibSafe 缓冲区溢出通常就是利用libc中的字符串处理函数没进行边界检查进行攻击的，所以LibSafe就针对C库中的潜在有缓冲区溢出漏洞的函数重新包装[5]，在函数调用前，先计算目标地址有否被缓冲区溢出的可能，即要先进行边界检查，然后才调用以完成该函数正常的功能。

LibSafe在实现时，其边界检查只是计算目标缓冲区是否会溢出到函数返回地址，对函数内部用到的局部变量(如函数指针)就检查不到，所以该方法不是万能的。LibSafe的优点是不需要重新编译源程序，只需更改系统的库函数即可。

4.2 静态发现技术

静态发现技术主要用来在程序设计过程中，根据一定的规则发现源码里潜在的有漏洞之处，以便程序员发现并改进。完全找出程序中所有的错误是不可能的，所以静态发现技术只是一种建议性的不完整的解决方案，它只是降低了程序被攻击的可能性，而且静态发现工具还需要维护一个与漏洞有关的不断变化的规则库。

4.2.1 ITS4 ITS4(It's the Software Stupid Source Scanner)是Cigital公司2000年发布的一种命令模式的交互式的程序漏洞扫描工具[4]，它能对C/C++程序的每一个函数进行代码分析，根据所建立的模式库匹配，然后依据危险级别给用户一个修改提示报告[17]。

ITS4采用模块化的结构开发，可以集成到IDE中，作为辅助软件开发的一个手段。它的模式库要随着新漏洞的发现而不断地更新。ITS4不是开放源码的软件。

4.2.2 Flawfinder Flawfinder是2001年发布的基于Python语言开发的用来辅助进行安全审查C/C++程序的工具[18]。它内嵌了一些常见的类似于ITS4的程序漏洞数据库，比如缓冲区溢出、格式化串漏洞等，并且扫描速度快，是程序静态检查中极有竞争力的工具。

Flawfinder是遵循GPL2协议开发的，运行于类Unix平台。Flawfinder源程序公开，是一款比较好的在程序发布前找出其潜在漏洞的工具。

4.2.3 RATS RATS(Rough Auditing Tool for Security)作为程序安全性粗略的审计工具，几乎和Flawfinder同

(下转第160页)

式的挖掘和满足约束的挖掘结合得更为紧密是我们今后所要研究的问题。

参 考 文 献

1 Agrawal R, Imielinski T, Swami A. Mining Association Rules between Sets of Items in Large Databases. In: Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'93), Washington, DC, MAY 1993. 207～216

2 Agrawal R, Srikant R. Fast Algorithms for Mining Association Rules, VLDB'94, 487～499

3 Park J S, Chen M S, Yu P S. An Effective Hash-based Algorithm for Mining Association Rules. In SIGMOD'95, 175～186

4 Han J, Pei J, Yin Y. Mining Frequent Patterns without Candidate Generation. In: Proc. ACM SIGMOD, 2000. 1～12

5 Pei J, Han J. Can We Push More Constraints into Frequent Pattern Mining? In: Proc. 2000 Int. Conf. Knowledge Discovery and Data Mining (KDD'00), Boston, MA, AUG. 2000. 350～354

6 Pei J, Han J, Lakshmanan L V S. Mining Frequent Itemsets with Convertible Constraints. In: Proc. 2001 Int. Conf. on Data Engineering (ICDE'01), Heidelberg, Germany, April 2001

7 Pasquier N, Bastide Y, Taouil R, Lakhal L. Discovering Frequent Closed Itemsets for Association Rules. In ICDT'99, Jan. 1999

8 Pei J, Han J, Mao R. CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets. In: Proc. 2000 ACM-SIGMOD Int. Workshop on Data Mining and Knowledge Discovery (DMKD'00), Dallas, TX, May 2000

9 Burdick D, Calimlim M, Gehrke J. MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases. In ICDE'01, April 2001

10 Zaki M, Hsiao C. CHARM: An Efficient Algorithm for Closed Itemset Mining. In SDM'02, April 2002

11 Wang J, Han J, Pei J. CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets. In: Proc. 2003 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'03), Washington, D. C., Aug. 2003

(上接第113页)

时发布[18]，事实上两款工具的开发者经常探讨交流他们的技术，期望最后两个工具能融合一体。

RATS 比 Flawfinder 支持更多的语言，提供了对 C、C++、Perl、PHP 以及 Python 语言的漏洞扫描。RATS 也是遵循 GPL 协议开发的开放源码的工具。

4.2.4 BOON BOON(Buffer Overrun detectiON)是加州伯克利 David Wagner 博士论文的实现原型，是2002年发布一款专门针对 C 程序的缓冲区溢出漏洞检测工具。

BOON 在实现内部借鉴面向对象思想，对每个字符串变量设置了两个属性，一个描述该字符串被分配的大小，另一个描述字符串实际被使用的大小。所有的字符串处理函数都将参考这两个属性对字符串进行操作。BOON 通过扫描计算后得出源程序中的安全漏洞级别，程序员根据 BOON 的报告然后手动改正源程序的缺陷。

BOON 不检查有关格式化串(Format String)漏洞的扫描，而且现在作者也明确表示不再有更新版本出现。

结论 尽管缓冲区溢出存在了多年，并且业界也对缓冲区溢出展开了不少研究，但问题还是没有得到根本的解决，缓冲区溢出仍然是计算机安全所面临的重大课题。

在我们的总结中发现，无论是动态检测技术，还是静态发现技术，都或多或少地减少了缓冲区溢出攻击的可能。我们尚未发现哪一种技术是完美的，当然我们分析的也是其中的一部分比较有代表意义的解决方案。在动态检测技术中，还有其他的如针对格式化串的攻击检测 FormatGuard、硬件技术和编译技术结合的 StackGhost、增强 C 的类型和边界检查的 Ccured 和 CyClone 等，以及在静态发现技术中的，对程序安全属性进行建模分析的 MOPS、对 C 类型增强的 CQUAL、对 Unix 平台 Lint 进行安全扫描改进的 Splint 等，限于篇幅我们没有分析。不过解决问题的角度基本都从编译器或者操作系统方面进行探索。

假设每一个对缓冲区的操作都得到严格的检查，假设根本就没有溢出发生，那就不会有缓冲区溢出的攻击。如果我们的高级语言能保证不会发生越界情况，那这个问题也就能得到解决。但计算机已经有长达半个多世纪的历史，现在遗留下来的软件是宝贵的财富，我们不可能做到所有的程序都重新编译，所以解决的办法都尽量地从兼容性、从最小化改动原有程序考虑。同时，各种各样针对缓冲区溢出的新的攻击手段可能也会出现，这些都是缓冲区研究的难点。

今后研究的重点可能会集中在如入侵检测技术、程序自保护技术、系统体系结构研究等方面，我们也正是从这些方面着手的。完全地走出缓冲区溢出阴影，可能还需要研究人员作出更多努力。

参 考 文 献

1 Cowan C. Buffer Overflow and OS/390. http://cert.uni-stuttgart.de/archive/bugtraq/1999/02/msg00081.html

2 CERT, the Computer Emergency Response Team Coordination Center. http://www.cert.org/advisories/

3 Executable and Linkable Format (ELF). Portable Formats Specification, Version 1.1

4 Fayolle P A, Glaume V. A Buffer Overflow Study Attacks & Defenses. ENSEIRB, Networks and Distributed Systems, 2002

5 Wilander J, Kamkar M. Dept. of Computer and Information Science, Link¨opings universitet. A Comparison of Publicly Available Tools for Dynamic Buffer Overflow Prevention. 2002

6 Chiueh T, Hsu F H. Computer Science Department, State University of New York at Stony Brook. RAD: A Compile-Time Solution to Buffer Overflow Attacks. In: Proc. of The 21st IEEE Intl. Conf. on DISTRIBUTED COMPUTING SYSTEM, 2001. 409

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

8 Cowan C, et al. StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks. In: 7th USENIX Security Conf. San Antonio, Texas, USA, 1998

9 GrSecurity. http://www.grsecurity.net/

10 Cowan C, et al. Buffer overflows: Attacks and defenses for the vulnerability of the decade. In: Proc. of the DARPA Information Survivability Conf. and Expo (DISCEX), Hilton Head, South Carolina, Jan. 2000. 119～129

11 Jones R, Kelly P. Backwards-compatible bounds checking for arrays and pointers in C programs. In: Proc. of the Third Intl. Workshop on Automatic Debugging AADEBUG'97, Linkoping, Sweden, May 1997

12 Bulba and Kil3r. Bypassig StackGuard and Stack- Shield. Phrack Magazine 56 http://www.phrack.org/phrack/56/p56-0x05, May 2000

13 Cowan C, et al. PointGuard: Protecting Pointers From Buffer Overflow Vulnerabilities. In: 12th USENIX Security Symposium, Washington DC, 2003

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

15 Etoh H. GCC extension for protecting applications from stack-smashing attacks. http://www.trl.ibm.com/projects/security/ssp/, June 2000

16 PaX http://pageexec.virtualave.net

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

18 Wheeler D A. Flawfinder, Web page http://www.dwheeler.com/flawfinder/, May 2001