维普资讯 http://www.cqvip.com

防止缓冲区溢出攻击的增强编译技术分析

潘 亦 吴春梅 武港山
(南京大学计算机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 通过增强传统的编译处理技术来防止缓冲区溢出攻击是一种常用有效的方法。本文比较了几个典型的应用这一技术来防卫缓冲区溢出攻击的常用工具的原理与性能,比较的结果可以指导软件开发者根据自己的安全需求来选择合适的工具来防止缓冲区溢出攻击,以达到提高软件安全性能的目的,也有助于正在研究缓冲区溢出攻击防范技术的工作者提出更加有效而安全的防止缓冲区溢出攻击工具。

关键词 缓冲区溢出攻击,C程序边界检查,StackGuard,PointGuard,RAD

Analysis of Compiler Technology Enhancement for Buffer Overflow Prevention

PAN Yi WU Chun-Mei WU Gang-Shan
(State Key Laboratory for Novel Software Technology,Nanjing University,Nanjing 210093)
(Department of Computer Science & Technology,Nanjing University,Nanjing 210093)

Abstract Enhancing compiler technology is one of the publicly available methods for buffer overflow prevention. This paper compares four typical and publicly available tools for buffer overflow prevention by enhancing compiler technology. Our motivation for this is to guide software developers to choose an appropriate tool to increase software quality from a security point of view,and also guide the researchers to bring forward their own more effective and safer method on buffer overflow prevention.

Keywords Prevent buffer overflow,Bounds checking for C,StackGuard,Pointguard,RAD

1 引言

近年来,随着信息技术的快速发展,网络互连和资源共享在方便了人们的工作和生活的同时也造成了各种安全隐患。各种利用软件的漏洞、安全弱点的恶意攻击手段层出不穷,以缓冲区溢出为代表的安全漏洞则是最为常见的一种形式,简单来说,它是由于编程机制而导致的在软件中出现的内存错误。这样的内存错误使得黑客可以运行一段恶意代码来破坏系统正常地运行,甚至获得整个系统的控制权。如果能有效地消除缓冲区溢出的漏洞,则很大一部分的安全威胁可以得到缓解。国外这方面的研究已有很多,而国内相应的研究还不多见。

防止缓冲区溢出攻击有很多种形式,增强编译技术是其中一种常用有效的方法,目前有许多工具都是通过增强编译技术来防止缓冲区溢出攻击的,它们都修改了编译器,修改后的编译器对用户程序进行编译后,会自动在程序中添加新的指令,这些指令有效地防止了缓冲区溢出,从而提高了程序的安全性,并为程序员编写程序提供了透明性。本文比较了几个典型的应用这一技术来防卫缓冲区溢出攻击的常用工具的原理与性能,比较的结果可以指导软件开发者根据自己的安全需求来选择合适的工具来防止缓冲区溢出攻击,以达到提高软件安全性能的目的[1],也有助于正在研究缓冲区溢出攻击防范技术的工作者提出更加有效而安全地防止缓冲区溢出攻击的工具。

2 缓冲区溢出攻击

缓冲区溢出攻击的目的在于扰乱具有某些特权运行的程序的功能,这样可以使得攻击者取得程序的控制权,如果该程序具有足够的权限,那么整个主机就被攻击者控制了[2]。

2.1 缓冲区溢出攻击原理

如果程序中没有仔细检查用户输入的变量,通过往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指令,以达到攻击的目的[3]。图1显示了最常见的缓冲区溢出攻击实例,利用重复的'A'字符串使得一个本地缓冲区发生溢出,并且改写了返回地址,在这个例子中返回地址被修改为0xbffff740[1]。

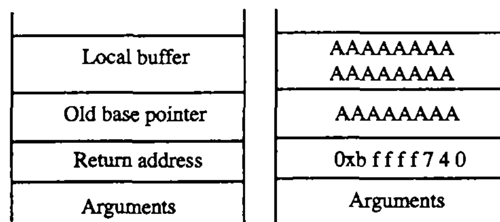

图1 利用缓冲区溢出改写返回地址

2.2 缓冲区溢出攻击的防止方法

由于缓冲区溢出攻击是目前常见的攻击手段,所以进行这个方面的研究工作是有意义和成效的。目前有编写正确的

潘 亦 硕士研究生,主要研究方向:智能信息服务;吴春梅 硕士研究生,主要研究方向:系统安全;武港山 教授,博士,主要研究方向:智能信息检索。

维普资讯 http://www.cqvip.com

代码、非执行的缓冲区、数组边界检查和程序指针完整性检查这四种基本的方法来保护缓冲区免受缓冲区溢出的攻击和影响,详细介绍请参考文[1]。

3 典型防止缓冲区溢出攻击的增强编译技术解析

3.1 C程序边界检查

Richard Jones 和 Paul Kelly 通过检查数组和指针边界来实现攻击防止[4]。我们很容易通过检查数组索引值是否超过数组大小来进行数组边界检查,但是在检查指针边界时,由于无法直接知道指针的大小,C程序边界检查就为每个指针表达式定义了一个基指针,然后通过检查这个基指针来确定表达式的结果是否在容许的范围之内。这种方法解决了一些诸如函数指针溢出的缓冲区溢出攻击问题,但是也付出了巨大的效率代价,对于一个频繁使用指针的程序,如向量乘法,将由于指针的频繁使用而使速度慢30倍[2]。

3.2 StackGuard

StackGuard[5]是一种提供程序指针完整性检查的编译技术,它通过检查函数活动记录中的返回地址来防止缓冲区溢出攻击。StackGuard 作为编译器 GCC 的一个补丁,在每个调用的函数的开始和结尾分别加入了定位附加字节 canary 和检查 canary 的代码。在调用函数的开始处将一些附加的字节 canary 插入到进程堆栈中,canary 处于函数返回地址的后面,如图2所示。

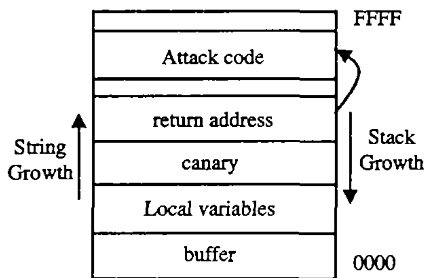

图2 利用 StackGuard 防止栈溢出攻击

在函数返回时,首先检查 canary 是否被改动过,如果发生改动,则说明缓冲区被攻击。

StackGuard 对程序中所有调用的函数里都加入了定位附加字节 canary 和检查 canary 的代码,这会导致系统的性能略有下降,但对于安全级别要求高的特定应用来讲 StackGuard 仍然是一个十分有效的防止缓冲区溢出攻击的方法。

然而 StackGuard 不能防止所有的溢出攻击,攻击者可以利用 canary 正确值重写 canary 而保证不改变 canary,或者通过指针来修改返回地址而不改变 canary,即在保证 canary 完好的情况下改变返回地址,从而成功绕过 StackGuard 防止。C. Cowan 等人提出了解决第一种问题的两种方案[6]:

·随机化 canary:利用一个在函数调用时产生的一个32位的随机数来实现保密,使得攻击者不可能猜测到附加字节的内容。而且,每次函数里添加的附加字节的内容都在改变,所以无法预测 canary 值。

·利用 MemGuard[7]保护返回地址:MemGuard 将内存中每个字都设计为只读,程序员只能使用 MemGuard API 来写内存。MemGuard 为 StackGuard 提供了更好的内存保护。

这些技术只能阻止那些重写栈周围所有数据的攻击,而不能阻止针对返回地址的攻击,攻击者可以利用指针来修改返回地址而不修改 canary[1]。

3.3 PointGuard

很多攻击者开始利用指针修改返回地址这种更一般的方法攻击缓冲区溢出,如堆溢出、printf 格式串弱点、大量 free 错误等,它们成功避开了像上面提到的 C 程序边界检查和 StackGuard 等现存的溢出保护。

PointGuard[8]是对 StackGuard 的改进版本,它可以解决指针溢出攻击问题。PointGuard 可以防止绝大多类型缓冲区溢出,它加密存储在内存中的指针,并仅当指针被装载到 CPU 寄存器里时解密指针。攻击者可以破坏一个指针值,但不能伪造指针值,因为他们没有解密密钥。PointGuard 可以防止函数指针的溢出以及 jmpbuf 的溢出。图3显示了 PointGuard 防止溢出攻击的过程。攻击者修改指针时所注入的值经过 PointGuard 解密过程,生成一个随机地址引用,而进程的地址空间范围很小,这个随机地址引用很有可能指向一个不在进程地址空间范围,从而很有可能导致受害程序崩溃。我们认为使得受害程序崩溃好于将控制权交给攻击者。

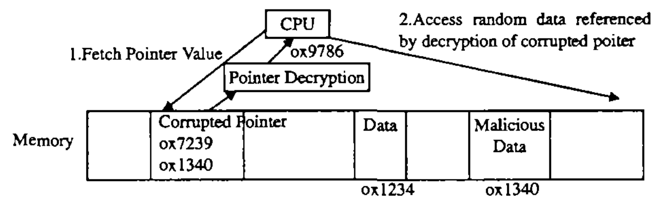

图3 利用 PointGuard 防止溢出攻击

PointGuard 通过在所有的代码指针之后放置附加字节 canary 来检验指针在被调用之前的合法性。如果检验失败,会发出报警信号并退出程序的执行。

3.4 RAD

RAD(返回地址保护)通过阻止攻击者修改返回地址来防止缓冲区溢出攻击[9]。它将函数返回地址保存到数据段中被称为返回地址仓库(RAR)的特定区域。MineZone RAD 和 Read-Only RAD 是 RAD 的两个版本,它们是保护 RAR 中返回地址的两种方法,它们阻止攻击者试图修改 RAR。每次函数调用返回时,函数返回地址就和 RAR 中的副本比较,如果相同则说明返回地址未被攻击者修改。

RAD 使用内核空间来保存 RAR,内核显然是保护 RAR 很安全的地方。内核中大约有7kB 的自由空间可以被用来保存 RAR,一般这么大的空间已足够了。

MineZone RAD 通过将 RAR 两个邻接区域设置为只读来保护 RAR 中返回地址,它可以防止绝大多数缓冲区溢出攻击,但是当攻击者只修改 RAR 中返回地址而不修改 RAR 两个邻接区域时,MineZone RAD 就不能防卫溢出攻击。Read-Only RAD 并没有定义两个只读邻接区域,而是将 RAR 本身设置为只读,这种方法可以防卫曾在上面提到的 MineZone RAD 不能防卫的缓冲区溢出攻击。但是由于 RAR 被设置为只读,因此在每个调用函数的头部添加更新 RAR 的代码时,需要增加两个系统调用,这会严重地影响程序效率。概括来讲 MineZone RAD 效率更高而 Read-Only RAD 更安全[9]。

4 工具比较

C 程序边界检查、StackGuard、PointGuard 和 RAD 都是通过增强编译技术来防止缓冲区溢出攻击的几个典型工具,

维普资讯 http://www.cqvip.com

它们都通过修改编译器来为用户程序自动添加保护代码，有效地防止了一些缓冲区溢出攻击，提高了程序的安全性能，但是这些技术并不成熟，还存在不少的缺点。下面列出了它们的一些共同缺点：

·不能防止所有的缓冲区溢出攻击。

·只提供了攻击检测而未提供攻击预防，它们都试图解决已知的安全问题，而不是解决程序本身的安全隐患问题。

·当发生缓冲区溢出攻击时，它们要么强迫程序中止，要么执行错误流程，而不能保证程序继续正常运行，尽管这样比将控制权交给攻击者好。

·需要大量额外的内存空间保存返回地址，并需要额外的系统开销来执行插入的代码。

·需要对所有代码进行重新编译，这对于正在运行的操作系统和已经出售的系统来讲，是一个非常严重的缺陷[1]。

理论上边界检查可以防止所有的缓冲区溢出，但实际上在C中不可能进行完全的边界检查，因为存在无法估计长度的指针变量。比如编译器在编译 strcpy(char * dest,char * src)时，无法知道两个指针变量的长度，因此编译器无法在函数内进行指针边界检查。

StackGuard 并没有像边界检查一样阻止攻击者修改返回地址，而是阻止攻击者获取系统控制权。它是一种简单有效的方法，可以防卫缓冲区和返回地址之间整个内存空间被修改的攻击，而且因为C程序边界检查需要为每个数组添加检查代码，因此 StackGuard 系统开销比C程序边界小得多。但 StackGuard 是以缓冲区溢出攻击会修改附加字节 canary 为前提，只能阻止那些重写栈周围所有数据的攻击，而不能阻止那些未修改 canary 而仅针对返回地址的攻击，目前 Bulba 和 Kil3r 已经发现成功绕过 StackGuard 的对策[10]。

尽管 StackGuard 没有C边界检查安全，但是C程序边界检查的系统开销大，并且程序兼容性差[1]，这些缺点导致了 StackGuard 比C程序边界检查更受程序开发者欢迎。

RAD 利用 RAR 来保存返回地址并利用两种策略来保护返回地址免受攻击，这两种策略分别对应于 RAD 的两个版本 MineZone RAD 和 Read-Only RAD。MineZone RAD 无法防止只修改 RAR 中返回地址而不改变 RAR 两个邻接区域的攻击，而 Read-Only RAD 可以对其防止，但是 Read-Only RAD 使用了两个系统调用来修改 RAR，时间开销远远大于 MineZone RAD。而且 RAD 为了保存 RAR，需要为每个进程分配昂贵的内核内存空间。目前 RAD 只能保护栈返回地址，而和C程序边界检查、StackGuard 一样不能保护函数指针变量。

PointGuard 可以防止所有对指针溢出的攻击，它加密存储在内存中的指针，并且只有当指针被装载到 CPU 寄存器里时解密指针。PointGuard 不仅可以防止缓冲区和返回地址之间整个内存空间被修改的溢出攻击，而且可以防止C程序边界检查、StackGuard 和 RAD 未解决的仅修改指向返回地址的指针的溢出攻击。但是目前 Crispin Cowan 等人已开发了一个 PointGuard 初始化模型，PointGuard 的实现还没有完成。它的安全性能是建立在安全密码基础上的，PointGuard 是否提供安全的密钥算法关系到 PointGuard 的安全性能。

结束语 本文分析了几个典型的通过增强编译技术来防止缓冲区溢出攻击的常用工具，但是它们都不能防止目前所有的缓冲区溢出攻击。但我们可以考虑将它们结合起来使用，比如由于 MineZone RAD 比 Read-Only 效率高而没有 Read-Only RAD 安全，如果我们能够利用编译器分析每个函数，选择一个足够安全的 RAD 版本来保护每个函数，这样可以充分利用 MineZone RAD 的效率高和 Read-Only RAD 安全的优点。通过结合使用各种技术，可以扬长补短，充分利用它们各自的优点，从而更有效而安全地防止缓冲区溢出攻击。

参考文献

1 Wilander J,Kamkar M. A Comparison of Publicly Available Tools for Dynamic Buffer Overflow Prevention[EB/OL]. http://www.mcs. csuhayward. edu/?simon/security/boflo. html,2003

2 Cowan C,Wagle P,Pu C,Beattie S,Walpole J. Buffer overflows: Attacks and defenses for the vulnerability of the decade[C]. In: Proc. of the DARPA Information Survivability Conf. and Expo (DISCEX),Hilton Head,South Carolina,2000. 119～129

3 Fayolle P A,Glaume V. A Buffer Overflow Study Attacks & Defenses [EB/OL]. http://www. enseirb. fr/? glaume/indexen. html,2002

4 Jones R W M,Kelly P H J. Backwards-compatible bounds checking for arrays and pointers in C programs [C]. In: Third Intl. Workshop on Automated Debugging,1997

5 Cowan C,Beattie S,Day R,Pu C,Wagle P,Walthinsen E. Protecting systems from stack smashing attacks with StackGuard [EB/OL]. http://www. cse. ogi. edu/?crispin/,1999

6 Cowan C,Pu C,Maier D,Walpole J,Bakke P,Beattie S,Grier A,Wagle P,Zhang Q,StackGuard H H. Automatic adaptive detection and prevention of buffer-overflow attacks [C]. In:Proc. of the 7th USENIX Security Conf. San Antonio,Texas,1998. 63～78

7 Cowan C,McNamee D,Black A,Pu C,Walpole J,Krasic C,Marlet R,Zhang Q. A Toolkit for Specializing Production Operating System Code [R]:[Technical Report CSE-97-004]. Dept. of Computer Science and Engineering,Oregon Graduate Institute,1997

8 Cowan C,Beattie S,Johansen J,Wagle P. PointGuard: Protecting Pointers From Buffer Overflow [C]. In: the 12th USENIX Security Symposium,Washington DC,2003

9 Chiueh T C,Hsu F H. RAD. A compile-time solution to buffer overflow attacks [C]. In:Proc. of the 21th Intl. Conf. on Distributed Computing Systems (ICDCS). Phoenix,Arizona,USA,2001

10 Bulba,Kil3r. Bypassing StackGuard and Stackshield [J]. Phrack Magazine,1999(5): 56