第33卷 第17期 计 算 机 工 程 2007年9月
Vol.33 No.17 Computer Engineering September 2007
·安全技术· 文章编号：1000—3428(2007)17—0187—03 文献标识码：A 中图分类号：TP309

基于指针备份的随机化技术

王立民，曾凡平，李 琴
（中国科学技术大学计算机科学技术系，合肥 230026）

摘 要：以“缓冲区溢出”为代表的控制数据漏洞已成为最常见的安全隐患，这些漏洞是依靠修改目标进程的控制数据，使目标进程转向某一段事先注入的恶意代码，从而导致恶意代码以目标进程当前用户的权限而被执行。随机化技术是针对控制数据漏洞的有效手段，但仍无法阻止控制数据被恶意修改。基于指针备份的随机化技术可以对控制数据攻击做出及时、准确的处理，加强了原有模型的抗攻击强度。
关键词：漏洞；缓冲区溢出；内存随机化；控制数据随机化

Randomization Technology Based on Pointer Backups

WANG Li-min, ZENG Fan-ping, LI Qin
(Departmemt of Computer Seience & Technology, University of Science and Technology of China, Hefei 230026)

【Abstract】Control data vulnerability, characterized by buffer overflow, is the most common security problems. When exploited, the attacker tries to rewrite the value of some control data in the target process to redirect the control flow to the prepared malicious code. In this way, the malicious code is executed under the current user's rights of the target process. Randomization is an effective technique to defend against control data vulnerability, but it still can not protect the control data from being modified. This article proposes a randomization technology based on pointer backups. The model can act upon attacks in time and exactly, so it strengthens the original technique a lot.
【Key words】vulnerability; buffer overflow; memory layout randomization; control data randomization

“缓冲区溢出”使得网络上的某个匿名用户可以获得其他主机部分甚至全部的控制权。缓冲区溢出攻击已经成为安全威胁中最为严重的一类。从某种角度来说，近年来，C 语言的流行使得缓冲区溢出漏洞更加泛滥。因为在 C 语言中，编译器并不进行边界检查，使得缓冲区溢出漏洞大量存在。攻击者利用缓冲区溢出漏洞，通过插入恶意攻击代码使程序控制流转向恶意代码，使其运行在一定的特权下。在最坏的情况下，攻击者能够获得 root 权限，重新引导程序，甚至控制整个主机。针对缓冲区溢出漏洞，国内外许多学者提出了各种防范措施。这些方法大致可以划分为两大类：

(1)基于源代码检测的静态分析方法，通过检查程序的源代码找出可能存在的安全漏洞。通过静态检测找出所有的安全漏洞并不现实，但是静态检测确实可以减少被攻击的可能性。静态检测方法需要维护一个已知的程序漏洞数据库；

(2)动态保护方法，通过改变运行时环境或系统函数而使“漏洞程序不会造成危害”或仅能造成较小的危害。程序的漏洞仍然存在，只是它在这种新的运行环境下不能被利用。动态保护的方法要求漏洞的利用方式已知。

针对原有 MLR(memory layout randomization)和 CDR (control data randomization)方法存在的自身缺陷，本文提出了一种改进的方法，结合了这 2 种技术，从而有效地抵抗缓冲区溢出攻击。

1 随机化技术

Shuo Chen 和 Jun Xu 提出的 MLR 和 CDR 的动态保护方法可以有效地阻止漏洞被攻击者利用，而且仅仅引入了很小的运行开销。这 2 种技术并不需要修改程序的源代码，MLR 的实现需要修改动态程序加载器(dynamic program loader)，而 CDR 的实现需要扩展现有的 C 编译器。

1.1 一般攻击方式

为了有效地抵抗缓冲区溢出漏洞，必须了解针对缓冲区溢出漏洞的一般攻击方式。攻击者通过向有缓冲区溢出漏洞的系统发送恶意消息/数据来发起攻击。使用此恶意消息/数据，攻击者期望完成以下两件事情：注入恶意代码和改变当前控制信息(返回地址等)，从而指向注入的恶意代码。这里的消息/数据泛指应用程序可以接收的各种形式的外部输入，可以是网络消息、控制台输入、命令行参数或环境变量等。

设 m 是被注入的恶意代码的地址，p 是目标系统中控制信息的地址。为了发起一次成功的攻击，攻击者必须正确地找到 m 或 p 或二者全部的运行时的值。因为所有现代操作系统中进程的内存布局情形都是已知的，所以攻击者通过一些简单的分析便可以在远程站点上正确地获得这些值(即使在目标应用程序开始执行之前)。

通常获得 m 和 p 的值的方法为：(1)确定目标系统和应用程序的版本；(2)配置一个导航系统来模拟目标系统；(3)使用此导航系统测试攻击。一次成功的测试会获得 m 和/或 p 的值。

1.2 内存随机化 MLR

MLR 方法的基本思想是：每次程序启动时，通过将程序所占有的内存空间随机化，使攻击者无法正确地找到需要攻击的程序的进程空间，让攻击者无法确定恶意代码的植入位

作者简介：王立民(1982－)，男，硕士研究生，主研方向：信息安全；曾凡平，副教授；李 琴，硕士研究生
收稿日期：2006-09-10 **E-mail**：billzeng@ustc.edu.cn

置。MLR 的基本要求包括对用户的透明性和内存重定位的随机性。

Red Hat 公司从 RHEL v.3 开始引入了 ExecShield 技术，结合 CPU 的 NX(Not eXecutable)支持，利用内存随机化的方法，可以有效地阻止大多数的缓冲区溢出漏洞。

一个进程执行时所需要的内存空间主要包括堆(heap)、栈(stack)、共享库(shared library)和程序自身的主代码(main application)。其中，堆和栈都是程序使用的数据区；共享库由于其自身的用途而被编译成 PIC 格式(position independent code)；因此，它们容易被随机化，难点是程序自身主代码的随机化。为此，Red Hat 的技术人员进行了一系列的改进(包括 GCC,GLIBC,BINUTILS 等)而得到了 PIE(position independent executables)。PIE 有些像共享库和可执行代码的混合体，既可以被存放在任意内存位置，又可以是一段独立的应用程序主体。

1.3 控制数据随机化 CDR

控制数据是指用于指示程序控制流转换的程序数据。有 2 种控制数据：函数指针(用于间接函数调用或间接跳转)和返回地址(某个函数结束后用于继续程序流)。

在 CDR 中，当控制数据被定义时，它首先被使用随机密钥加密，然后才被存储。使用时，控制数据首先被解密然后被正常使用。内存中的控制数据始终以密文的形式存在。

1.4 原有模型的缺陷

CDR 机制并不能阻止攻击者改写控制数据。在原有 CDR 的实现中，当控制数据被改写时，被攻击的进程本身并不能及时地发现并处理已经出现的异常，而是需要底层内存保护机制的支持来检测出数据异常，这基于以下假设：攻击者写入的数据被解密后是一个随机地址，控制流转向这个随机地址后会引起内存访问异常(memory access violation)。当攻击者并未察觉到加密算法的存在时，大多数情况下，这样的实现是可用的，但并不是所有情况下都可以信赖；而当攻击者对使用的加密算法经过一定的分析后，情况会更糟糕，问题的症结在于：进程本身并不能自行判断返回地址是否被修改过，从而不能有效地判定是否被攻击。一个健壮的程序应该能尽早地发现可能存在的异常，并且能够针对发现的异常做出适当的处理。而在原有 CDR 实现中，仅仅依靠底层的内存保护机制来发现并处理异常的方法，其结果并不令人满意。

2 改进方案

2.1 函数调用的一般过程

发生函数调用时，堆栈状态见图 1。

图 1 堆栈状态

假设函数 A 调用函数 B，可以称 A 为调用函数，B 为被调用函数。从汇编层来看，函数调用的一般步骤表示如下：

(1)在 A 中，将调用 B 所需的参数压栈。

(2)调用函数 B，使用系统调用 call。这一步中，“ call 系统调用 ” 自动完成对运行状态和返回地址的保存。

(3)在 B 中，将堆栈基址指针 EBP 压栈。

(4)修改基址指针 EBP 使其指向当前栈顶，并使用它访问存在堆栈中的函数输入参数及堆栈中的其他数据。

(5)堆栈指针 ESP 向上移动一段距离，留出一个空间给该函数作为临时存储区。

(6)B 中的代码执行完毕后，将返回值存入 EAX 中，使用 ret 返回。

(7)控制流回到 A 中，从 EAX 中获得函数返回值，继续执行后续指令。

2.2 改进方案

针对实现方法的缺陷，作了相应的改进。基本思想是：将 MLR 和 CDR 技术有机结合，通过改写 GCC 编译器，使每次生成控制数据的时候，首先将其加密存储，然后自动在某个随机位置动态分配一个空间，来保存此控制数据的副本。在控制流将要返回之前，首先将原控制数据与其副本相对照：若二者相同，则继续执行，将其解密后作为返回地址使用；否则，说明原控制数据已经被改写，则终止程序，向用户报错。

如图 2 所示，通过改写 GCC 编译器，实现了对原有随机化技术的改进。控制数据有两种：函数指针和返回地址。对于函数指针，处理方法与 CDR 相同，即当函数指针被定义时，它首先被使用随机密钥加密，然后才被存储，使用时需要首先解密然后才能被正常使用；而对于返回地址，需要进行额外的改进。

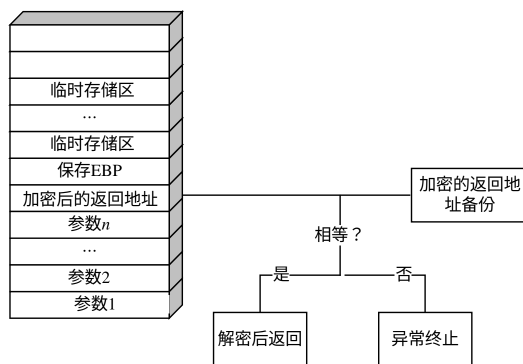

图 2 指针备份信息发现异常现象

在改进后的函数调用过程中，需要修改以下步骤：

(1)在步骤(3)中，在将堆栈基址指针 EBP 压栈之前，根据参数个数和类型，从堆栈中相应位置中取出返回地址，保存在某个临时寄存器中，并将其加密，然后保存到原来的堆栈中的适当位置，并动态分配一个随机地址空间，用于保存加密后的返回地址值。

(2)在步骤(6)中，B 中的代码执行完毕后，取出堆栈中的返回地址和预先保存的副本，比较二者是否相等。若不等，则调用 exit 退出程序，返回某个错误代号。若相等，则将此地址解密，保存回到堆栈中。将返回值存入 EAX 中，使用 ret 返回。

改进后的随机化技术能够尽早地发现程序运行中出现的异常，并在很短的时间内做出相应的处理。

2.3 性能分析

在原有随机化技术的实现中，进程本身并不能及时地发现并处理已出现的异常，而是需要底层内存保护机制的支持来检测出数据异常。这种实现，其抗攻击强度完全取决于所采用的加密算法的强度。加密算法一旦被攻破，所采用的随机化技术也就随之失效。

与原有随机化技术的实现方法相对比，可以增加了一层自我保护，而并不是像原有随机化技术的实现一样，仅仅依赖于底层的内存保护机制的支持。经过改进后的随机化技术，并不完全依赖于所采用的加密算法，即使加密算法被攻破，进程仍然可以自行检测出是否被攻击，并做出相应的处理，保存控制数据副本的位置是动态分配的，而由于 MLR 的支持，此副本的存在位置是随机的，因此攻击者并不能修改此控制数据的副本，这保证了改进后的随机化技术的有效性。

这种改进的方法简单、实用、易于实现；从效率上看，这种改进所引入的额外开销仅仅是备份一个数据而已，这样的开销几乎可以被忽略。改进后的主要时间消耗仍然是加密解密所消耗的时间。

3 实验结果及分析

为了验证改进后的随机化技术的有效性及其时间开销，笔者在本地运行了一系列比较简单的缓冲区溢出攻击代码，运行时间如图 3 所示。

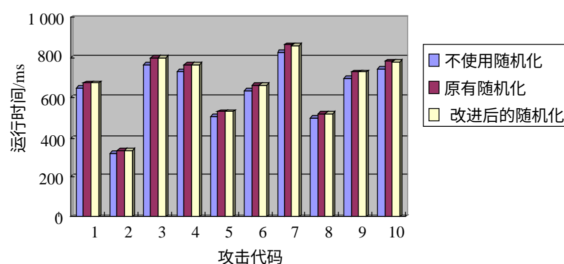

图 3 3 种不同模型运行时间的对比

为了模拟加密算法被破译情况下随机化技术的表现，可以使用简单的移位加密法，结果如下：

(1)不使用任何随机化技术，攻击者能够成功获得一个当前用户权限的 sh 进程；

(2)使用原有随机化技术，可以有效地阻止那些“并未察觉到加密算法存在”的攻击；通过对某些原攻击代码的简单修改，可以使其破译简单的移位加密算法，此时，原有随机化技术无法阻止“攻击”；

(3)使用改进后的随机化技术，可以有效地阻止原攻击代码以及简单修改后的攻击代码的攻击，进程被攻击时会异常终止，并返回特定的错误代码，而此错误代码正是在步骤(6)中所定义的；

(4)从时间上看，改进方法并不会显著增加时间开销，甚至在某些情况下还比原有方法更快地发现异常。因为改进后的方法比原有方法能更早地、主动地发现异常，所以不必被动地等待底层的内存保护机制产生的“内存访问异常”。

4 结论

Shuo Chen 和 Jun Xu 提出的随机化技术包括 MLR 和 CDR 2 个方面，它们既相互独立，又可以相互协作。通过二者的有机结合，能够有效地防止传统的缓冲区溢出漏洞。

本文针对原有随机化技术中存在的缺陷，提出了一种新的改进方法，在仅仅引入极小的额外开销的情况下，大大增强了原有随机化技术的保护强度。改进后的方法，不仅能够尽早地发现异常情况，而且能够在很短的时间内做出相应的处理。实验证明，改进后的随机化技术能够有效地发现缓冲区溢出攻击并向用户报告异常情况，而时间消耗并无明显的增加。

随机化技术的关键点是采用了加密算法。安全性较好的加密算法，时间消耗较大；相反的，时间消耗较小的加密算法，安全性可能较差。根据具体的应用，可以采用不同的加密算法，从而获得安全性和效率的权衡。下一步的主要工作可以从加密算法入手，并将其应用在一些较复杂的知名软件上，进一步验证本方法的有效性和时间复杂性。

参考文献

1 Chen Shuo, Xu Jun. Security Vulnerabilities: From Analysis to Detection and Masking Techniques[EB/OL]. (2006-06). http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/5/33381/01580509.pdf.

2 Xu Jun. Intrusion Prevention Using Control Data Randomization[EB/OL]. (2003-06). http://ase.csc.ncsu.edu/junxu/Papers/dsn03_control_data_randomization.pdf.

3 Arjan V D V. New Security Enhancements in Red Hat EnterpriseLinux v.3, Update 3[EB/OL]. (2004-08). https://www.redhat.com/f/pdf/rhel/WHP0006US_Execshield.pdf.

~~~~~~

（上接第 155 页）

2 Orman H. The Oakley Key Determination Protocol[S]. RFC 2412, 1998.

3 Krawczyk H. SKEME: A Versatile Secure Key Exchange Mechanism for Internet[C]//Proc. of IEEE Symposium on Network and Distributed System Security. 1996: 114.

4 Perlman R, Kaufman C. Analysis of the IPSec Key Exchange Standard[C]//Proc. of the 10th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises. 2001: 150-156.

5 Kaufman C. Internet Key Exchange (IKEv2) Protocol[S]. RFC 4306, 2005.

6 Aiello W, Bellovin S, Blaze M, et al. Just Fast Keying: Key Agreement in A Hostile Internet[J]. ACM Transactions on Information and System Security, 2004, 7(2): 242-273.

7 Qualum P K, Simpson W. Photuris: Session Key Management Protocol[S]. RFC 2522, 1999.
~~~~~~