第33卷 第16期 计 算 机 工 程 2007年8月
Vol.33 No.16 Computer Engineering August 2007
·安全技术· 文章编号：1000—3428(2007)16—0142—02 文献标识码：A 中图分类号：TP309.2

缓冲区溢出攻击检测技术的分析和研究

徐启杰，薛 质
(上海交通大学信息安全工程学院，上海 200030)

摘 要：缓冲区溢出攻击是目前最具威胁的攻击方式之一，对信息安全造成了极大的危害。该文通过分析缓冲区溢出攻击的原理，归纳出攻击所必需的3个步骤，根据检测这3个攻击步骤，将当前最常用的缓冲区溢出攻击检测技术分为3种类型，并对其进行了分析和研究。
关键词：缓冲区溢出；ShellCode；攻击检测

Analysis and Study of Buffer Overflow Attack Detection Technology

XU Qi-jie, XUE Zhi
(School of Information Security Engineering, Shanghai Jiaotong University, Shanghai 200030)

【Abstract】 Buffer overflow attack is one of the most threatening attack types and it jeopardizes information security a lot. According to the principle of the attack, this paper generalizes three necessary steps of a buffer overflow attack. It divides the most popular technologies of buffer overflow attack detection into three types in light of detecting the three attack steps, and also analyzes and studies those technologies.
【Key words】 buffer overflow; ShellCode; attack detection

在信息安全日益被人们所关注的今天，缓冲区溢出毫无疑问是最大的安全威胁之一。Internet上的第1例蠕虫(Morris)攻击，就是利用了fingerd的缓冲区溢出漏洞。SANS评选出的2005年威胁最大的20个漏洞中，有8个跟缓冲区溢出有关[1]。根据CERT的统计数据，近几年与缓冲区溢出有关的安全事件在50%以上[2]。

在缓冲区中写入大量的数据将会导致缓冲区溢出。缓冲区溢出如此普遍是因为C语言固有的不安全性，它对数组和指针的操作没有自动的边界检查机制，许多标准C函数库所支持的字符串操作：strcpy()，strcat()，sprintf()，gets()等都是不安全的[3]。程序员要负责检查这些操作不会造成缓冲区溢出，但往往会出现忘查或误查的情况。

利用缓冲区溢出漏洞进行攻击的日益普遍是因为缓冲区溢出漏洞具有极大的破坏力和隐蔽性。它可以导致程序运行失败、系统死机或重启。更为严重的是，可以利用它执行非授权指令，甚至可以取得系统的超级特权，进行各种非法操作。它的隐蔽性主要表现在：(1)一般程序员很难发觉自己编写的程序中存在缓冲区溢出漏洞，从而疏忽监测；(2)攻击者所发送的溢出字符串在形式上跟普通的字符串几乎无区别，传统的防御工具如防火墙不会认为其为非法请求，从而不会进行阻拦；(3)ShellCode与执行时间都很短，在执行中系统不一定报告错误，并可能不影响正常程序的运行；(4)攻击者通过缓冲区溢出改变程序执行流程使ShellCode能够进行本来不被允许或没有权限的操作，而防火墙认为其是合法的；(5)攻击的随机性和不可预测性使得防御攻击变得异常艰难，在没有攻击时，存在漏洞的程序并不会有什么变化(这和木马有着本质的区别)，这也是缓冲区溢出最难被发现的原因；(6)缓冲区溢出漏洞的普遍存在，使得针对这种漏洞的攻击防不胜防(各种补丁程序也可能存在着这种漏洞)[4]。

1 缓冲区溢出攻击特征

利用缓冲区溢出漏洞是攻击者最常用的获得本地或远程系统控制权的方法。为控制被攻击系统，攻击者需要取得足够的权限。然而多数情况下，攻击者的权限都不足以使其能够控制被攻击系统。因此，攻击者希望利用一个以高权限在运行的程序，注入攻击代码，使得攻击代码在高权限下执行。

缓冲区溢出主要有以下3种：(1)基于堆栈的缓冲区溢出；(2)基于堆的缓冲区溢出；(3)基于数据段的缓冲区溢出。一般情况下，静态存储区和堆上的缓冲区溢出漏洞很难被攻击者利用，而堆栈上的漏洞较为容易被利用，具有极大的危险性。

由于函数里局部变量的内存分配是发生在栈帧里的，因此如果在某一个函数里定义了缓冲区变量，则这个缓冲区变量所占用的内存空间是在该函数被调用时所建立的栈帧里。对缓冲区的操作(比如字符串的复制)都是从内存低址到高址的，而内存中所保存的函数调用返回地址恰好处于该缓冲区的上方(高地址处)，如图1所示。

图1 函数调用时的栈帧

当攻击者使用大于缓冲区空间的溢出字符串向缓冲区进行填充时，就可以改写函数栈帧中的返回地址，使其成为攻击代码的起始地址；当函数返回时，攻击代码得以执行。如果被攻击程序是以管理员权限运行，就意味着攻击代码是以管理员的特权在运行。

综上分析，为完成一次缓冲区溢出攻击，攻击者需要完成以下3个步骤：

作者简介：徐启杰(1982－)，男，硕士研究生，主研方向：网络与信息安全；薛 质，副教授
收稿日期：2006-08-29 **E-mail：**xuqijie@sjtu.edu.cn

(1)注入攻击代码或找到已有的适合攻击的代码；

(2)改变特权程序的执行流程，使得攻击代码可以以足够的权限运行；

(3)执行攻击代码。

以上步骤缺一不可，其中，第 2 个步骤是发动一次缓冲区溢出攻击的最大难点和关键所在。

2 常用缓冲区溢出攻击检测方法

根据缓冲区溢出攻击的步骤，可将常用的缓冲区溢出攻击检测技术分为以下 3 种类型 :基于输入字符串的检测方法，基于保护堆栈中的返回地址的检测方法和基于监视系统调用的检测方法。

2.1 基于输入字符串的检测方法

对输入的字符串进行检测，确定其为溢出攻击字符串时采取阻拦措施，使攻击者无法注入攻击代码。

一般有以下 3 种方法构建溢出攻击字符串。分别如图 2~图 4 所示。

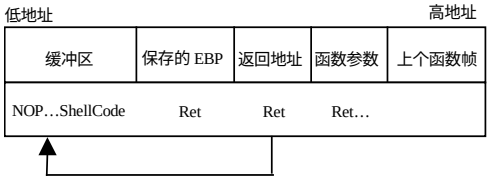

图 2 溢出攻击字符串 1

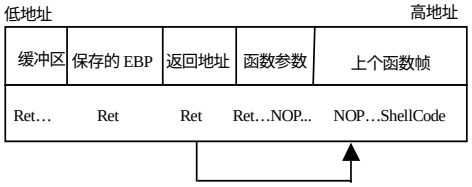

图 3 溢出攻击字符串 2

图 4 溢出攻击字符串 3

第 1 种溢出攻击字符串适用于缓冲区大于 ShellCode 长度的情况；第 2 种溢出攻击字符串一般用于缓冲区小于 ShellCode 长度的情况；第 3 种方法是将 ShellCode 放在环境变量里，是目前较为常用的方法。

在第 1 种和第 2 种类型的溢出攻击字符串中 ShellCode 前都加了若干的 NOP 指令，因为这 2 种情况下 ShellCode 的地址无法确定，但只要返回地址指向 ShellCode 前的任一条 NOP 指令，ShellCode 就可以执行，大大增加了 ShellCode 执行的可能性。这些 NOP 指令称为 sledge。其他单字节指令如 AAA 等也可构成 sledge。因此缓冲区溢出攻击检测系统可以通过检查输入的字符串中是否含有大量 NOP 等可构成 sledge 的指令来判断此字符串是否是溢出攻击字符串。不过这种方法并不适用于检测第 3 种类型的攻击。但这 3 种类型的攻击字符串中都含有 ShellCode。因此，确定出 ShellCode 的基本特征,如不含有“ 0x00 ”，含有某些特殊的系统调用等，然后利用人工智能、模式匹配、规则匹配等方法检查输入字符串中是否包含 ShellCode 也可检测出是否有缓冲区溢出攻击发生。这些检测都可以在入侵检测等外围防御系统中实现，优点是实现较为简单，不会增加被保护系统的开销；缺点是漏报率较高，无法检测出无明显特征的溢出攻击字符串。

2.2 基于保护堆栈中返回地址的检测方法

缓冲区溢出攻击最关键的步骤是要通过修改函数返回地址来改变程序的流程，因此，在函数调用返回前，通过检查返回地址是否被修改可以判断是否有缓冲区溢出攻击发生。该检测的实现可以通过在源码中插入一些约束和判断的模块，然后在编译后的程序运行期间对有关变量和堆栈区域进行监控，检测是否有攻击发生。StackGuard 和 StackShield 就是这一类型的工具，它们都是 gcc 编译器的扩展工具，用于监控调用的函数返回地址是否正常。StackGuard 主要是在内存中的返回地址及缓冲区之间插入一个“ Canary ”字，如图 5 所示。在函数调用返回前通过检查“ Canary ”字来判断返回地址是否已经被修改，如果这个 Canary 的值被改变了，说明可能有人正进行缓冲区溢出攻击，程序会立刻响应，发送一则入侵警告消息，然后停止工作。为防止攻击者构造“ Canary ”字，StackGuard 选用“ 终止符 ”和“ 随机数 ”作为“ Canary ”字的值。但由于“ Canary ”字所在的位置是固定的，因此也可能被绕过。

图 5 添加 Canary 后的栈帧

StackShield 对此作了改进，创建了一个新的堆栈用于备份被保护函数的返回地址。它在被保护函数开始处增加一段代码，用来将函数返回地址拷贝到一张特殊的表中；同样在被保护函数的结尾处也增加一段代码，用来将函数返回地址从表中拷贝回堆栈。从而保证函数正确返回。

除这 2 种工具外，还有其他一些类似的编译器扩展方法和工具，如 PointGuard、Snarskii 等。这些工具提高了被保护系统的安全性，不过增加了系统开销，降低了系统性能。

2.3 基于监视系统调用的检测方法

如果攻击者成功注入攻击代码，并改变了程序的执行流程使指令的执行指针指向了 ShellCode 的入口地址。按照一次缓冲区攻击的 3 个步骤，还须执行 ShellCode 来完成攻击目的。因此，通过检测是否有 ShellCode 运行可以检测是否发生缓冲区溢出攻击。

攻击者既希望 ShellCode 利用获得的权限启动一个交互式的 shell 进程来完成尽量多的事情，又希望 ShellCode 尽量短小从而更加隐蔽，所以绝大多数 ShellCode 都会调用系统函数。由于监视所有系统调用会耗费大量系统资源，因此只对 ShellCode 常用的系统调用进行监视，根据某些特征判断受监视的系统调用是否为非法调用就可确定被保护系统是否遭到缓冲区溢出攻击。例如，如果发现系统调用的返回地址为堆栈，则可认为其为非法调用，因为很少有程序在堆栈上运行代码。

对于 Linux 系统，ShellCode 中常用的系统调用如表 1 所示。由于在 Linux 系统中系统调用在汇编级上都是通过 int 0x80H 语句来进行的，因此可根据这一特征来对系统调用进行拦截。在它们执行之前判断其返回地址的合法性。从而确定是否有缓冲区溢出攻击发生。 (下转第 152 页)