文章编号：2096-1472(2017)-09-30-04

Windows栈缓冲区溢出攻击原理及其防范

李云飞[1]，陈洪相[2]
(1.渭南师范学院，陕西 渭南 714099；
2.湖北信息工程学校，湖北 荆门 448000)

摘　要：计算机网络安全漏洞和网络攻击伴随着网络的存在会随时发生，栈缓冲区溢出漏洞攻击是网络攻击中最常见的一种攻击技术。文章剖析了Windows栈工作原理，以及栈溢出漏洞攻击技术方法，针对常见的栈溢出漏洞攻击提出了几种防御措施，能预防大部分针对栈溢出漏洞的攻击。

关键词：网络安全；内存；安全漏洞；堆栈；缓冲区溢出
中图分类号：TP309.1　　**文献标识码**：A

The Principle of Stack Buffer Overflow Attacks in Windows and the Prevention

LI Yunfei[1],CHEN Hongxiang[2]
(1.*Weinan Normal University*,*Weinan* 714099,*China*;
2.*Hubei Information Engineering College*,*Jingmen* 448000,*China*)

Abstract: Computer network security vulnerabilities and cyber attacks may occur at any time on the Internet,and the stack buffer overflow attack is the most common network attack technology.This paper analyzes the operating principle of the stack in Windows and the techniques of stack overflow attacks.Then,several prevention measures are proposed for common stack buffer overflow attacks,which can prevent most stack overflow attacks.

Keywords:network security;memory;security vulnerabilities;stack,buffer overflow

1 引言(Introduction)

缓冲区是已分配的一段大小确定的用于临时存放数据的内存存储区。当向一个已经分配了确定内存空间的缓冲区内写入超出该缓冲区处理能力的数据时，将发生缓冲区溢出[1]。

近十年，以缓冲区溢出为类型的安全漏洞攻击是最为常见的一种形式，在网络与分布式系统安全中，50%以上的漏洞攻击都是基于缓冲区溢出技术的，尤其在不进行边界检查的C/C++程序中仍然是软件可靠性和安全性的主要威胁之一[2]。

利用缓冲区溢出攻击，可以导致程序运行失败、重新启动、执行恶意代码等后果。缓冲区溢出中最危险的是栈溢出，因为入侵者可以利用堆栈溢出，在函数返回时改变返回程序的地址，让其跳转到任意地址，更为严重的是，它可被利用来执行非授权指令，甚至可以取得系统特权，进而进行各种非法操作[3]。

2 Windows栈缓冲区溢出原理(The principle of stack buffer overflow in Windows)

2.1 Windows内存程序结构

计算机运行时，必须首先把程序从外存装载到内存，然后由CPU从内存中依次读取执行指令。正在运行的程序叫进程，每个进程都有自己的独立内存空间，它被分成几个段(Segment)，分别是代码段(text)、数据段(data，bss)、堆(heap)、栈(stack)等，如图1所示。用户进程的内存空间，也是系统内核分配给该进程的虚拟内存。内存总是被进程占用，但并不表示这个进程占用了这么多的物理内存，Windows将虚拟内存地址映射到各进程的物理内存地址上，进程内存空间随着程序的执行会增大或者缩小[4]。

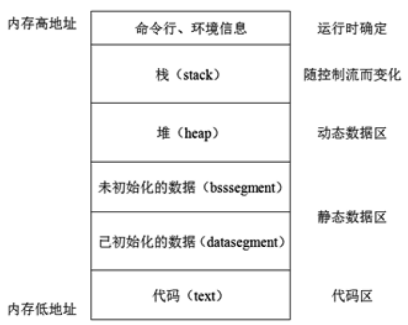


图1 Windows程序内存结构图

Fig.1 Windows memory structure

基金项目：渭南师范学院科研基金项目（16YKS001，09YKZ012）资助.

堆和栈都是一种数据项按序排列的数据结构。

栈是一种先进后出的数据结构，是自动开辟空间，用来分配局部变量、类的引用(指向堆空间段)，栈使用的是一级缓存，通常在被调用时处于存储空间中，调用完毕立即释放。

堆是一种经过排序的数据结构，每个结点都有一个值，可以被看成是一棵树，堆的存取是随意的，堆是存放在二级缓存中，生命周期由虚拟机的垃圾回收算法来决定。堆的特点是根结点的值最小(或最大)，且根结点的两个子树也是一个堆。由于堆的这个特性，常用来实现优先队列。

bss段(Block Started by Symbol segment)通常是指用来存放程序中未初始化的全局变量的一块内存区域，属于静态内存分配，程序一开始就将其清零了。

对于一个进程的内存空间而言，可以在逻辑上分成三个部分：代码区、静态数据区和动态数据区。动态数据区一般就是堆栈。进程的每个线程都有私有的栈，所以每个线程虽然代码一样，但本地变量的数据都是互不干扰。全局变量和静态变量分配在静态数据区，本地变量分配在动态数据区，即堆栈中。程序通过堆栈的基地址和偏移量来访问本地变量[5]。

32位系统中经典的内存布局是：程序起始1GB地址为内核空间，接下来是向下增长的栈空间和由0×40000000向上增长的内存映射地址。而堆地址是从底部开始，去除ELF(Executable and Linking Format)、代码段、数据段、常量段之后的地址并向上增长，这种布局导致了缓冲区容易遭受溢出攻击[6]。

2.2 Windows栈缓冲区溢出原理

Windows程序的执行流程由代码段ECS和指令指针EIP控制，EIP始终指向下一条要执行指令的地址。当发生中断或要调用子程序时，需要将当前断点信息入栈保存，然后转去执行中断子程序，执行完中断子程序后返回指令将栈顶内容出栈恢复断点ECS和EIP。

这种控制流程看似简单，只需将指令地址按序给ECS和EIP即可，但就是因为这简单的控制而不容许有任何地址计算差错。如果其他缓冲区溢出会导致多余的数据覆盖其他有用内存空间，其中如果将栈内数据覆盖就可能导致修改入栈保存的ECS和EIP，从而使得程序返回时跑飞。

当一个函数被调用时，函数参数、EIP、ECS(段间调用时)、EBP和函数局部变量会依次压栈保存，如图2所示。

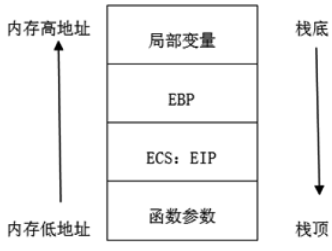


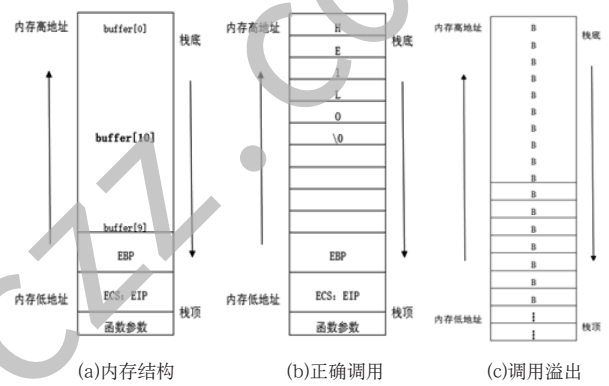
图2 栈结构

Fig.2 Windows stack structure

例如，在C语言中调用如下函数时，如果赋给函数的参数长度超过10个，则会导致程序崩溃发生缓冲区溢出：

```
voidbuf(charstr[])
{
char buffer[10];
strcpy(buffer,str);
printf("buffer:%s\n",buffer);
}
```

定义变量和正常调用时如图3(a)和图3(b)所示，但当把远大于10个字符的内容(如字符"B")拷贝到为缓冲区分配的10个字符空间时，多于10个字符的内容就会覆盖掉EBP和ECS:EIP，如图3(c)，函数执行完毕后返回的地址就已经不是原来保存的正确地址了。



(a)内存结构　(b)正确调用　(c)调用溢出

图3 调用函数内存示意图

Fig.3 Diagram of executing function in memory

如果溢出部分的数据量足够大或经过攻击者的精心设计，就可能覆盖返回地址，从而改变程序的执行流程，将程序的返回地址修改成其想执行的代码地址，达到攻击目的。

出现缓冲区溢出的情况主要包括三种：

(1)使用非类型安全的语言。缓冲区溢出主要出现在C和C++语言中，虽然C/C++语言可以允许程序员直接访问内存和CPU的寄存器，从而创建非常接近硬件运行的性能优异、运行速度快程序，但C/C++语言不执行数组边界检测和类型安全检查，所以在进行数组、字符串操作时容易造成缓冲区溢出[7]。

(2)以不安全的方式访问或操作缓冲区。如果应用程序需要获得数据，当用户将数据复制到应用程序所指定的缓冲区而未考虑目标缓冲区的大小时，就可能造成缓冲区溢出。

(3)编译器将缓冲区放在内存中关键数据结构旁边或邻近的位置。

3 Windows栈缓冲区溢出攻击防范(The prevention methods of stack buffer overflow attacks Windows)

在栈溢出的检查与防范方面，许多软硬件厂商已经做

了大量工作，如微软在Visual Studio中增加编译选项来检测栈的溢出，在Windows系列操作系统中增加了SEHOP(Structured Exception Handling Overwrite Protection)，阻止修改SEH增强系统的安全性，硬件方面，64位CPU引入了NX(No-eXecute)机制，在内存中区分数据区与代码区，当攻击者利用溢出使CPU跳转到数据区去执行时，就会异常终止等[8]。

但对大量现有软硬件资源，若要都升级或更新到最新的软硬系统是一件很困难的事，而且不断有绕过防御机制的新漏洞产生，攻击方法也在不断地发展变化。所以日常工作中的防御措施相当必要，针对不同的攻击原理和方法，也可以灵活采用各种技术进行针对性的防御。

缓冲区溢出攻击防范是和整个系统的安全性分不开的。除了系统管理上采用诸如关闭危险的特权程序，及时下载系统或软件的最新补丁，使用安全产品等措施之外，软件开发过程中的防范才是从根源上解决问题的渠道，常用的方法主要有几种[9]。

3.1 GS编译选项

Windows在Visual Studio 7.0(Visual Studio 2003)及以后版本中添加了一个针对函数的栈缓存溢出安全编译选项——GS，来增加栈溢出的难度。

GS编译选项的原理就是在堆栈上插入一个安全cookie，以测试堆栈上的返回地址是否被修改过。安全cookie为四个字节，在堆栈上的位置如图4和图2的传统内存结构相比，GS编译选项会增加四个字节的堆栈空间。

图4 GS编译选项内存结构

Fig.4 Memory structure of GS parameter

如果是堆栈的局部变量发生缓存溢出的错误而导致返回地址被覆盖的话，由于安全cookie所在的位置，它也一定会被覆盖。GS编译选项在函数的入口和出口添加了针对安全cookie操作的指令，如果发现安全cookie的值被改动就会转入异常处理终止程序运行。

函数的入口指令如下：

```
Push   ebp                    //保存上层函数堆栈基址
mov    ebp,esp
sub    esp,0x10               //设置当前函数堆栈基址
mov    eax,[security_cookie]
mov    [ebp-0x4],eax   //将security_cookie值放入堆栈的安全cookie位置
```

函数的出口指令为：

```
mov    ecx,[ebp-0x4]
call   security_check_cookie  //调用检查函数测试其值是否被修改过
mov    esp,ebp
pop    ebp
ret
```

如果堆栈上的安全cookie的值和security_cookie的值一致的话，那么函数正常退出，否则就会执行错误处理程序。

3.2 软件开发过程防范

发生栈缓冲区溢出的主要原因是软件程序中使用了不规范的数据操作或恶意代码攻击，所以在软件的编写过程中注意规范的代码审查，是杜绝缓冲区溢出的最直接因素。

(1)规范代码编写规则

C和C++开发工具不是为安全而设计的，属于非类型安全语言，为了保证编程的灵活性，C/C++的一些库函数(如strcpy(、gets()等))缺乏边界检测，如果调用时输入的参数过长，就会导致缓冲区溢出。所以在使用C/C++开发工具编程时，都应该有针对性地进行安全性测试和代码审查。

①数组边界检测。C语言不进行数组边界检测，容易产生超长数据操作植入代码，导致缓冲区溢出。当在编译时检查所有的数组读写操作，确保对数组的操作都在有效范围内。

目前的C程序编译调试检测提供了许多检测工具，主要对存储器存取检测、数组边界检查。譬如Purify使用目标插入代码技术检查可执行代码在执行时数组的所有应用来保障数组的合法使用，但程序的性能不可避免地要受到影响。

但由于所有的C数组在传送时是按指针传送的，所以传递给调用函数的数组不会被检查。例如库函数strcpy()、strcat()、gets()等函数，在编译时不会进行边界检查。

②指针完整性检查。程序的指针完整性检查在程序指针被引用之前检查其是否被改变，即使攻击者成功改变了程序指针，由于系统提前检测到了该改变而不执行该指针，所以该方法在防范缓冲区溢出方面性能比较好。

程序指针完整性检查是在函数返回地址或者其他的关键数据、指针之前插入防范值，或者存储一个返回地址、关键数据或指针的备份，在函数返回时进行比较。

③改进C库函数。C语言产生缓冲区溢出的根本是调用一些库函数时不对数据进行边界检测，比如strcpy()gets()、

strcat()、scanf()、printf()等，所以在用到该类型的库函数时，可以开发更安全的替代函数实现该部分功能，并对其进行安全检查调用，防范缓冲区溢出。

(2)栈的不可执行技术

Windows系统为了实现更好的性能和功能，往往在数据段中动态地插入可执行的代码，这样当缓冲区发生数据溢出时就会覆盖数据段，从而可能导致数据段中的可执行代码被修改。所以为了防止这种缓冲区溢出产生攻击，可以使被攻击程序的数据段地址空间不可执行，从而使得攻击者不可能执行被植入攻击程序的缓冲区代码。

为了保持程序的兼容性不可能将所有程序的数据段设为不可执行，但可以在必要的时候将堆栈数据段设为不可执行，因为几乎没有程序会在堆栈中存放代码，所以这样既可以最大限度地保证程序的兼容性，也可以有效地保证栈缓冲区溢出攻击。

(3)备份关键控制信息

栈溢出攻击程序最致命的攻击就是将程序流程的EIP和ECS内容修改，导致原程序流程不能正常执行。如果在调用程序或入栈时，将断点的EIP和ECS自动入栈的同时，用另外申请的静态或动态数组将程序流的关键信息如EIP和ECS备份保存，当出栈返回程序断点时，用备份的信息和栈中的信息进行比对，如果发现不一致则认为栈内容被修改，这时可以做出中断检测处理，防止进入攻击代码。

3.3 栈溢出检测防范技术

(1)漏洞特征检测

漏洞都有一定的触发条件，其攻击过程就是构造外部输入使之满足触发条件，针对栈溢出漏洞的攻击触发条件就是要注入超过系统逻辑计划存储的数据长度。根据漏洞的攻击特性系统会建立一个特征信息库并动态更新，当程序执行过程中系统会针对该信息库进行网络流量检测，当发现有类似特征代码就会及时采取有效措施或阻止，避免发生栈溢出漏洞攻击。

(2)攻击特征检测

漏洞需要被攻击者发现并利用才能被安全机构通过特征进行检测，这需要触发条件或可能经过很长时间才能被发现。为了及时防御该类溢出攻击，可以根据攻击的特点进行有针对性的防御。

栈溢出注入数据的长度与内容随着漏洞的不同而有所差异，但攻击者攻击时所使用的跳转地址却是常用或者比较固定易用的一些地址。由于不同平台下的shellcode一般都会重复利用。所以栈溢出攻击手法会具有一定的特征，通过提取这些频繁出现的特征对防御设备的攻击特征库及时更新，可以有效阻止栈溢出攻击发生。

(3)虚拟检测技术

防御设备最理想的情况是可以虚拟出一个除了数据不同其他均相同的虚拟机，通过在虚拟系统中插入检测点，检测系统某一进程完成后的现场跟踪进程的运行状态。如果进程的操作没有异常，后台再把操作重定向到真实的进程去处理，把真实数据返回给用户，否则可以阻止其对真实系统的访问。

4 结论(Conclusion)

计算机网络攻击几乎每时每刻都会发生，攻击的方式变化多样且会随着防御技术的变化而不断变化，所以对网络攻击的防御是个长期的持久战。本文提出的攻击原理和防御技术能有效地阻止大部分栈溢出攻击，但攻击技术在不断地发展，我们还需要根据新的攻击技术研究设计更有效的防御技术。

参考文献(References)

[1] Nashimoto S,et al.Buffer overflow attack with multiple fault injection and a proven countermeasure[J].Journal of Cryptographic Engineering,2016,7(1):1-12.

[2] Sui Y,et al.Eliminating Redundant Bounds Checks in Dynamic Buffer Overflow Detection Using Weakest Preconditions[J].IEEE Transactions on Reliability,2016,65(4):1682-1699.

[3] Wang X,et al.A Differential Approach to Undefined Behavior Detection[J].Acm Transactions on Computer Systems,2015,33(1):1.

[4] 肖蕾,刘克江.一种微型嵌入式系统动态内存分区管理机制的研究[J].软件工程,2016,19(4):59-60.

[5] 崔宝江,等.基于污点信息的函数内存模糊测试技术研究[J].清华大学学报(自然科学版),2016(1):7-13.

[6] 彭建山,等.基于指针时空分析的软件异常可利用性判定[J].计算机应用研究,2016,33(5):1504-1508.

[7] 谢汶兵,等.基于备份控制流信息的缓冲区溢出监测技术[J].计算机工程与应用,2016,52(11):101-107.

[8] Jaiswal S,Gupta D.Security engineering methods-in-depth analysis[J].International Journal of Information & Computer Security,2017,9(3):180.

[9] Howard M,Lipner S.The security development lifecycle:SDL,a process for developing demonstrably more secure software[J].Datenschutz und Datensicherheit-DuD,2015,34(3):135-137.

作者简介：

李云飞(1974-)，男，博士，副教授.研究领域：信息安全，嵌入式系统，模式识别.

陈洪相(1974-)，男，本科，高级教师.研究领域：计算机应用，软件开发.