广西科学院学报
Journal of Guangxi Academy of Sciences

2007,23(4):266~269
Vol. 23,No. 4 November 2007

一个 Windows 进程抓包器的 C++实现*

Implementation of the Windows Process Packetcapture Based on C++

胡小春[1],陈 燕[2],何潜航[2],李陶深[2]
HU Xiao-chun[1],CHEN Yan[2],HE Qian-hang[2],LI Tao-shen[2]

(1.广西财经学院计算机与信息管理系,广西南宁 530003;2.广西大学计算机与电子信息学院,广西南宁 530004)
(1. Department of Computer and Information Management, Guangxi University of Finance & Economics, Nanning, Guangxi, 530003, China; 2. School of Computer, Electronics and Information, Guangxi University, Nanning, Guangxi, 530004, China)

摘要:对 Windows 进程中的模块 WS2_32.dll 和 wsock32.dll 代码段进行修改,通过 C++的 API 函数编程实现进程抓包器,并以实现 IEXPLORE 浏览器抓包的过程为例,给出了进程抓包器的具体编程方法。

关键词:Windows 进程 抓包 数据包
中图法分类号:TP393.2 **文献标识码**:A **文章编号**:1002-7378(2007)04-0266-04

Abstract: By modifying the codes of WS_32.dll and wsock32.dll windows models, this paper describes how to implement the process packetcapture based on the C++ API function programming and shows how to program the process packetcapture by taking the example of how to implement the IEXPLORE packetcapture.

Key words: Windows process, packetcapture, datapacket

Windows 是一个开放的系统,一个进程可以对其它进程进行控制,包括内存的访问和读取、虚拟内存的控制、系统权限的控制等,甚至可以调试其它进程控制目标进程的运行[1,2]。要实现这些功能的控制首先要分析、研究 Windows 系统的内核对象管理,然后研究利用 Win32 ASM 对 Windows 进程的具体实现进行分析、研究。通过对进程运行的机器码反编译来分析程序,了解其逻辑结构并加以修改,就可以让目标进程按照设定的程序来运行[3,4]。对于一个经过加密的进程数据包,目前互联网上的软件并不能截取到进程的明文数据包。本文通过对 Windows 进程中的模块 WS2_32.dll 和 wsock32.dll 代码段进行修改,编程实现进程抓包器,并以 VC++.NET 实现 IEXPLORE 浏览器抓包的过程为例,介绍进程抓包器的具体编程实现。

收稿日期:2007-09-18
作者简介:胡小春(1974-),男,讲师,主要从事计算机网络与并行分布式计算研究。
*广西大学院校共管项目(X061002)和广西大学创新学分项目联合资助。

1 进程抓包器的实现过程[3]

抓包器实现的目的为了查看和了解进程中数据包的内容和结构,它的实现主要是通过 C++的 API 函数实现。实现抓包器的主要过程如下。

(1)在目标进程的窗口中再生成一个窗口,实现"注入"目标进程的内部,窗口用于显示截取到的数据包内容。"注入"的操作就是把一个模块或者自己编写功能代码段潜入到目标进程中去。"注入"的方法有远端线程注入、系统内核级注入等,最常用的是在系统上安装"钩子(HOOK)"。API 的函数 SetWindowsHookEx 可以实施一个钩子的安装,并且可以把相应的消息传送到指定的回调函数处理。系统发送给各种程序窗口的消息都要先经过钩子处理之后再送到目的窗口,而在钩子处理到来的消息之前 Windows 已经自动把"钩子""钩"在了消息目的进程窗口上了,也就是说此时"钩子"已经"混入"了目的窗口的内部。利用 HOOK 可以把一个或几个模块加载进入目标进程,并截取目标进程的窗口消息。钩子设定成功后,当系统有键盘消息要传给目标

进程窗口时,键盘消息就会先经过回调函数 KeyboardProc 进行处理;而模块 g _ hInstance 就会在钩子设定成功后,系统传给目标进程窗口第一个键盘消息的时候被注入到目标进程。

(2)修改目标进程的模块 WS2 _ 32. dll 和 wsock32. dll 数据包传输函数的代码段,使得程序跳转到设计好的数据包处理函数里执行,完成之后再跳转回原来的代码执行。设计的函数主要完成数据包过滤,并把过滤的信息在数据包窗口中显示出来。模块 WS2 _ 32. dll 中数据包的发送和接收函数分别是 send 和 recv,模块 wsock32. dll 中数据包的接收函数是 recv。Windows socket 中有两个接收函数,由于不可能预先知道目标进程将用哪个模块中的接收函数,所以在设计时同时处理。

(3)最后,在用自己定义的 mysend()函数时要先把 send 函数的代码还原,然后再调用,之后如果想继续截取 send 函数的数据,就再次修改 send 函数的代码。

通过以上 3 个过程,可以截取到进程发送的数据包,也可以截取该进程任何函数的数据。以下是用到的两个关键函数。

①回调函数:

```
LRESULT CALLBACK KeyboardProc(int nCode, WPARAM wParam, LPARAM lParam){
BOOL bKeyUp = (BOOL) lParam & (1 << 31);
if (bKeyUp && wParam = VK _ HOME && nCode = HC _ ACTION) //截取到按下 HOME 键的消息
{ ……;
 //按下 HOME 键后的处理代码,生成数据包显示窗口 }
}
```

②SetWindowsHookEx 函数:

```
SetWindowsHookEx(WH _ KEYBOARD, (HOOKPROC) KeyboardProc, g _ hInstance, wThreadID);//KEYBOARD 是指定是键盘消息;(HOOKPROC)KeyboardProc 是消息处理的回调函数;g _ hInstance 指定要注入的模块基址;dwThreadID 是目标进程窗口线程 ID。
```

2 进程抓包器的具体实现

2.1 抓包器注入模块的实现

抓包器的注入模块通过以下 7 个步骤实现。

(1)用 VC++. NET 新建一个解决方案,在方案里添加一个 MFC DLL 工程,命名为 Packet。在 Packet 中添加窗口 ID 为 IDD _ DIALOG _ MAIN。然后为这个窗口添加类 CMainWnd 继承 CDialog。窗口中三个勾选框 Send,Recv2,Recvs 分别表示是否打开对 WS2 _ 32. dll 中 send,WS2 _ 32. dll 中 recv,wsock32. dll 中 recv 行数的拦截。

(2)新建类 CJmpHookApi,对应的文件为 JmpHookApi. cpp 和 JmpHookApi. h。CJmpHookApi 类目的是获得 IE 的句柄、得到 IE 的控制权,并在其中实现抓包的最终目的。

①JmpHookApi. h 文件的主要代码:

```
public:
  FARPROC m _ lpOldFunc;//要钩的目标的函数
  FARPROC m _ lpNewFunc;//新的函数
  CJmpHookApi();
  virtual ~CJmpHookApi();
  BOOL InitMemory();//初始化得到内存数据(模块名,函数名,新函数 FARPROC)
  void SetHookOn();//启用 新函数
  void SetHookOff();//不启用 新函数
  void UnLock();
  void Lock();
protected:
  HANDLE m _ hProc;//进程句柄
  BYTE m _ bOldData[5];//内存旧数据
  BYTE m _ bNewData[5];//内存新数据
  CRITICAL _ SECTION m _ cs;};//CRITICAL _ SECTION 属于轻量级的线程同步对象
```

②JmpHookApi. cpp 里几个重要的函数实现代码:

```
CJmpHookApi::CJmpHookApi()
//得到当前进程的伪句柄,并初始化 CRITICAL _ SECTION
BOOL CJmpHookApi::InitMemory()
 { DWORD dwOldFlag;//修改旧函数内存前 5 字节为 PAGE _ READWRITE
  if(VirtualProtectEx(m _ hProc,m _ lpOldFunc,5, PAGE _
  READWRITE,&dwOldFlag))
   { //调入旧函数内存前 5 字节
      if(ReadProcessMemory(m _ hProc,m _ lpOldFunc,
        m _ bOldData,5,0))
      { //取得旧函数内存前 5 字节内存数据
      if (VirtualProtectEx (m _ hProc, m _ lpOldFunc,
        5,dwOldFlag,&dwOldFlag))
```

```
{  m_bNewData[0]=0xE9;//JMP 指令
    DWORD * pNewFuncAddress;
        //得到新函数的地址
    ……}
void CJmpHookApi::SetHookOn()
    //修改旧函数内存前5字节为新的内容
void CJmpHookApi::SetHookOff()
    //修改旧函数内存前5字节为原来内容
void CJmpHookApi::Lock() //多线程下使用  进入临界区
void CJmpHookApi::UnLock()//多线程下使用离开临界区
```

(3)新建文件 Global.h 和 Global.cpp,主要是定义全局变量和自定义的函数实现,例如 mysend、myrecv、myrecvs 函数的实现。Global.h 的几个主要变量:

```
typedef int (WSAAPI * FUNWS2_32)(IN SOCKET,IN const char FAR *,IN int,IN int);
extern FUNWS2_32 funRecv;//WS2_32.dll 原 recv 函数基址
extern FUNWS2_32 funRecvs;//wsock32.dll 原 recv 函数基址
extern FUNWS2_32 funSend;//WS2_32.dll 原 send 函数基址
extern SOCKET g_Socket; //上次 send 函数的 SOCKET
extern CMainWnd  *g_pMainWnd;//抓包器窗体指针
  //新的数据包处理函数声明
int __stdcall FAR mysend (SOCKET s,const char FAR * buf,int len,int flags);
int __stdcall FAR myrecv (SOCKET s, const char FAR * buf, int len, int flags);
int __stdcall FAR myrecvs(SOCKET s, const char FAR * buf, int len, int flags);
BOOL HookInitMemory(void); //内存初始化
void ShowPack(char *pData, int len, int type);//显示数据包
BOOL CheckFilter(const char FAR * pData, int nFlag);//判断数据包第一个字节的过滤器
BOOL CheckFilterLen(int nLen,int nFlag);  //判断数据包长度的过滤器
int StringToHex(char * strIn, char * strOut);
//字符串转换为数据
void Char2Cstring(char * buf, int len, CString * Str);//数据转换为字符串
CString GetConfigFileName();//获得保存的配置文件名
BOOL GetCListCtrl(CListCtrl * pList, CString strListKey,int nColCount,CString strFileName)
    //读取并设定 List 控件内容
BOOL SaveCListCtrl(CListCtrl * pList, CString strListKey,int nColCount,CString strFileName)
    //保存 List 控件内容
```

(4)新建文件 Hook.h 和 Hook.cpp,主要实现 Hook 成功后的回调函数的处理,代码如下:

```
//Hook.cpp
extern HINSTANCE g_hInstance;//当前模块的基址
BOOL __declspec(dllexport) WINAPI InstallHook(DWORD dwThreadId);//安装勾子
BOOL __declspec(dllexport) WINAPI UninstallHook(void);//取消勾子
LRESULT CALLBACK KeyboardProc(int nCode, WPARAM wParam, LPARAM lParam);//消息处理回调函数
//Hook.cpp
#include "stdafx.h"
#include "Global.h"
#include "Hook.h"
HHOOK g_hhook=NULL;//勾子
HINSTANCE  g_hInstance = NULL;//当前模块的基址
//HOOK 成功后 处理 WH_KEYBOARD 消息的回调函数
LRESULT CALLBACK KeyboardProc(int nCode, WPARAM wParam, LPARAM lParam)
{  if (bKeyUp && wParam= VK_HOME && nCode=HC_ACTION)//截取到按下 HOME 键的消息
  {if (g_pMainWnd == NULL) //窗口生成代码
    { CWnd::GetForegroundWindow(); //找到当前窗口
     if(pCWnd! =NULL)
      {g_pMainWnd = new CMainWnd(); g_pMainWnd -> Create (CMainWnd:: IDD, pCWnd);
```

```
……//在当前窗口创建抓包器窗口
}
```

(5)在 Packet.cpp 中包括文件 Hook.h,添加函数 InitInstance()和 ExitInstance()。

```
#include "Hook.h"
BOOL CPacketApp::InitInstance()
{  AfxInitRichEdit();
   CWinApp::InitInstance();
   g_hInstance = theApp.m_hInstance;
        //得到当前模块的基址
   if (! AfxSocketInit())
    { AfxMessageBox (IDP_SOCKETS_INIT_FAILED);
      return FALSE; }
   return TRUE;
}
BOOL CPacketApp::ExitInstance()
{  JmpSend.SetHookOff();  //还原代码段
   JmpRecv.SetHookOff();  //还原代码段
   UninstallHook();  //清理钩子
}
```

(6)在解决方案的配置处选择 Release,并把工程 Packet 的输出目录改为 out。

(7)用 Rclease 生成工程 Packet,生成后,在../out 目录可以看到 Packet.dll 和 Packet.lib 文件。

2.2 抓包器注入程序的实现

抓包器的注入程序通过以下 3 个步骤实现。

(1)在解决方案添加一个 MFC 应用程序工程,作为简单的 CDialog 应用程序,命名为 Begin。在窗体中添加开始按钮,按钮执行的代码段如下:

```
#include "../Packet/Hook.h"//勾子函数定义文件
void CBeginDlg::OnBnClickedButtonBegin()//开始按钮
{ DWORD dwThreadID = 0x0;
   CWnd * pCWnd = FindWindow(_T("IEFrame"),NULL);//打开 IE 浏览器
if(dwThreadID ! = 0x0)
{InstallHook(dwThreadID);//安装钩子
OutputDebugString("InstallHook(dwThreadID)");
CString outMsg;
outMsg.Format("目标:%08x",dwThreadID);
SetWindowText(outMsg);
GetDlgItem(IDC_BUTTON_BEGIN) -> EnableWindow(FALSE)
}
  else  MessageBox("没有找到目标","提示",MB_OK);
}
```

(2)把 Begin 的输出目录也改为../out,并输入,附加依赖../out/Packet.lib。

(3)用 Release 生成 Begin。

2.3 抓包器应用测试

按以下步骤对抓包器进行测试。

(1)运行 2.2 中生成的 Begin.exe 程序,点击开始,如果没有打开 IE,则提示找不到目标。如果 IE 已经打开,则可以看到 Begin 的窗口标题变为:目标:0X000xxxx。找到目标之后,在 IE 里按下 HOME 键,就可以看到抓包器的窗口了。

(2)用调试器 OllyICE 调试。重新打开 IE,然后打开调试器,动态加载 IE 进程。加载完成后察看 OllyICE 的模块列表。

(3)运行 Begin 程序,点击开始,然后在被调试的 IE 里按下 HOME 键,可以看到调试器 OllyICE 的模块列表里面增加了 Packet.dll 模块,证明已经成功注入抓包器模块了。

(4)在调试器 OllyICE 中查找所有模块中的名称,找到 WS2_32.dll 中的 send 函数,双击 send 函数转到 CPU 视图的 send 函数的基址。然后钩选抓包器中的 send,可以在 OllyICE 里看到 WS2_32.dll 中的 send 函数的前 5 个字节的指令被改为:JMP Packet.xxxx,取消抓包器的 send 的钩选,这条指令又会被还原。可以用同样的方法测试其它两个 recv 函数。

3 结束语

利用 HOOK 来截取 Windows 的消息是 Windows 核心编程的基础知识,可以利用 HOOK 实现丰富的 Windows 程序效果。更重要的是,通过 HOOK 可以实现 Windows 进程对象的控制,进一步了解 Windows 对内核对象的管理原理。

参考文献:

[1] 孟庆倩,李清宝.基于 Windows 环境进程监控的设计与实现[J].信息工程大学学报,2007,8(1):26-29.

[2] 王峰,董亮卫.Windows(2000/XP)下隐藏进程的检测机制[J].计算机工程,2006,32(20):95-96,99.

[3] 周炎涛.Windows 中的多线程编程技术和实现[J].计算技术与自动化,2002,21(3):109-116.

[4] 梁骥.防火墙与入侵检测系统(IDS)互动模型的构建[J].广西科学院学报,2007,23(2):80-81,84.

(责任编辑:韦廷宗)