用远程线程挂接 API 的两种方法

Two Methods of Hook API in Remote Thread

李　家　（辽宁师范大学城市与环境学院 辽宁大连 116029）

摘　要： 挂接 API 是 Windows 编程中的一项重要技术。本文在远程线程插入 DLL[1] 和改动模块输入节挂接 API[1][2] 技术的基础上，提出了利用插入 DLL 的构造函数在目标进程中挂接 API 的方法和编写远程线程函数挂接 API 的方法，实现了挂接 API 记录打印文档。

关键词： 挂接 API　插入 DLL　远程线程　目标进程　模块输入节

1　远程线程挂接 API 方法的原理

1.1 建立远程线程

在 Windows 中，每个进程都有自己私有的地址空间。为了挂接 API，需要编写一个 DLL 插入到目标进程的地址空间，将目标进程中的某些 API 调用挂接到该 DLL 中的相应函数。插入 DLL 挂接 API 的一种方法是使用远程线程，即在目标进程中创建一个新线程，在这个线程中用 LoadLibrary 函数加载自己的 DLL，Windows 提供的 CreateRomoteThread 函数可以实现这一功能。

```
HANDLE CreateRemoteThread(
    HANDLE hProcess,
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    DWORD dwStackSize,
    LPTHREAD_START_ROUTINE lpStartAddress, LPVOID lpParameter,
    DWORD dwCreationFlags,
    LPDWORD lpThreadId
);
```

其中参数 hProcess 指定目标进程，参数 lpStartAddress 指定目标进程中线程函数的地址，参数 lpParameter 是传给远程线程的参数。

1.2 利用远程线程加载 DLL

一个线程函数的原型为：

```
DWORD WINAPI ThreadProc ( LPVOID lpParameter
// thread data);
```

LoadLibrary 函数的原型为：

```
HINSTANCE LoadLibrary( LPCTSTR lpLibFileName );
```

文献[1] 中利用这两个函数原型恰好相同以及 Windows 系统将 Kernel32. dll 映射到每个进程的同一地址这两点，在本地进程找到 Kernel32. dll 中的 LoadLibrary 函数的地址，将其作为 lpStartAddress 参数传入到 CreateRemoteThread 函数中，执行远程线程时用 LoadLibrary 函数加载自己的 DLL，在其自动执行的 DllMAi 中显示该进程的一些信息。

1.3 在远程线程中通过改动模块的输入节挂接 API

加载自己的 DLL 时，在 DllMAi 中可以通过改动模块的输入节挂接 API。主要包括以下步骤：

（1）用 GetProcAddress 函数获取要挂接的 API 的地址

（2）用 CreateToolhelp32Snapshot 函数取得当前进程的快照

（3）用 Module32First 函数取得当前进程第一个模块的地址

（4）对每一个模块用 ImageDirectoryEntryToData 函数找到其输入节

（5）查找要挂接的 API 在输入节中的地址，找到后用 VirtualProtect 函数改动该地址的页面属性为 PAGE_READWRITE，用 WriteProcessMemory 函数将 DLL 中替代 API 函数的地址写入该地址，这样以后在目标进程中调用该 API 时，就会自动挂接到这个函数。

2　在插入的 DLL 的构造函数中挂接 API

基于以上原理，本文实现了挂接 API 记录打印文

档，挂接的函数是 StartDoc，这是 Windows 的打印函数。下面以在 Word 中记录打印文档为例作以介绍。

因为使用者有计算机的控制权，所以采用了在起动 Word 时在其进程中插入 DLL 挂接 API 的方法。另外，由于程序中用到了 MFC 类库，要建立基于 MFC 的 DLL，无 DllMAi，所以选择在 DLL 的构造函数中改动模块的输入节挂接 API。Word 中进行打印时，有的模块调用了 StartDoc 函数的 Unicode 版本，有的模块调用了 StartDoc 函数的 ANSI 版本，所以对两者都进行挂接。

2.1 建立基于 MFC 用来插入到目标进程的 DLL

其中主要报括 UNICODE 版本和 ANSI 版本的挂接函数 MyStartDoc，在其中记录打印文档后继续打印工作。还包括 DLL 的构造函数，在其中实现 1.3 中的算法步骤，改动模块的输入节挂接 API。

```
CCaptureApp theApp; //实例化 DLL
extern " C" __declspec( dllexport) int WINAPI MyStart-
DocA ( HDC hdc,CONST DOCINFO * lpdi)
{ //ANSI
    … // 处理 lpdi 传来的打印信息，记录打印文档
    return TRUE;
}
    extern " C" __declspec( dllexport) int WINAPI MyS-
tartDocW ( HDC hdc,CONST DOCINFO * lpdi)
{//UNICODE
    … // 处理 lpdi 传来的打印信息，记录打印文档
    return TRUE;
}
CCaptureApp::CCaptureApp( ) //构造函数
{
    //记录挂接函数的地址
    PROC procMyFuncAddressA = ( PROC ) MyStart-
DocA;
    PROC procMyFuncAddressW = ( PROC ) MyStart-
DocW;
    HMODULE hmDllAddress = GetModuleHandle ( "
GDI32. dll" ) ;
    //记录被挂接函数的地址
    PROC procFuncAddressA = GetProcAddress ( hmDl-
lAddr
    ess , " StartDocA" ) ;
    PROC procFuncAddressW = GetProcAddress ( hmDl-
lAddr
    ess , " StartDocW" ) ;
    //对进程中所有模块循环，查找输入节中的 Start-
Doc 函数并挂接，具体代码参见文献[1] 546 – 564，文
献[2] 205 – 209。
    …
}
```

2.2 在启动进程中启动 Word

建立一个 WIN32 应用程序，在 WinMAi 中启动 Word，主要代码如下：

```
    …
    PROCESS_INFORMATION pi; //记录目标进程信息
    CreateProcess( " c: \programfiles \microsoftoffice \
office11 \winword. exe" ,NULL,NULL,NULL,FALSE, NOR-
MAL_PRI
    ORITY_CLASS | CREATE_NEW_CONSOLE | PROCESS_
VM_WRITE,NULL,NULL,&si,&pi) ;
    Sleep( 500) ; //等待目标进程调入所有模块，避
免过早执行远程线程。
```

2.3 执行远程线程在目标进程中插入 DLL

因为字符串常量在当前进程中分配内存，不能在目标进程直接使用字符串常量，所以对要插入的 DLL，首先要用 VirtualAllocEx 函数在目标进程中分配内存，用 WriteProcessMemory 函数将其路径写到目标进程中分配的内存地址中；然后将该地址作为 lpParameter 参数传入 CreateRemoteThread 函数，传给远程线程中执行的 LoadLibrary 函数。

```
    char dll[ 50 ] = " c: \windows \system1 \capture.
dll" ; //要插入的 DLL 的路径
    //在目标进程中分配内存
    char * p = ( char * ) : : VirtualAllocEx( pi. hProcess,
0,50, MEM_
    COMMIT,PAGE_READWRITE) ;
    ULONG n;
    WriteProcessMemory ( pi. hProcess,p,dll,50,&n) ;
//在目标进程中写入 DLL 的路径
    PTHREAD_START_ROUTINE pf = ( PTHREAD_STAR
    T_ ROUTINE ) GetProcAddress ( GetModuleHandle
( TEXT( " Kernel32" ) ) ," LoadLibraryA" ) ; //找到 Load-
```

LibraryA 函数的地址

HANDLE ht = CreateRemoteThread(hProcess,0,0, pf,p,0, NULL); //执行远程线程,在目标进程中执行 LoadLibrary,调入 capture. dll。调入后自动执行其构造函数,挂接 API。

WAiForSingleObject(ht, INFINITE); //等待远程线程结束

3 编写远程线程函数插入 DLL 并挂接 API

因为 CreateRomoteThread 函数可以在目标进程建立远程线程并执行,参数 lpStartAddress 指定了目标进程中线程函数的地址,线程函数的原型为:

DWORD WINAPI ThreadProc (LPVOID lpParameter // thread data);

所以可以考虑编写这个函数在远程线程中执行(而不是如上文中那样在远程线程中执行 LoadLibrary 函数),在这个线程函数中插入自己的 DLL,同时改动模块的输入节挂接 API。即将 2.1 构造函数中的代码改写到远程线程函数中,也就是说要为另一个进程写代码,这个过程中要解决以下问题。

3.1 建立向远程线程传参的结构体

从 ThreadProc 函数原型可以看出,只能向其中传入一个指针参数,为了传入多个数据项,要建立本地进程中用来向远程线程传参的结构体,其中主要是函数地址和字符串常量。

```
typedef struct _RemotePara{
    PROC procLoadLibrary; //LoadLibrary 函数的地址
    …
    char sMyDllName[50]; //插入的 DLL 名称
    char sMyFuncNameA[20]; //用来挂接的函数
    char sTargetFuncNameA[20]; //被挂接的函数
    …
} RemotePara;
```

3.2 远程线程函数中的 API 调用

Windows 调用外部 DLL 中的函数时,不是直接调用 DLL 的函数地址,而通过 CALL 指令转向可执行文件的. text 节中一个 JMP DWORD PRT[XXXXXXXX]或 CALL DWORD PTR[XXXXXXXX]处[2]。因为各进程的. text 节的地址是不一样的,所以编写远程线程函数时不能直接调用 API,需要在目标进程中找到各 API 函数的地址,用相应的函数指针调用。因为 Kernel32. dll 在本地进程和远程进程中映射在同一地址(实际上 GDI32. dll、USER32. dll 等 Windows 系统模块也是这样),所以可以在本地进程中找到 LoadLibrary、GetModuleHandle、GetProcAddress 等函数的地址,通过参数传入远程线程,进而找到其它模块和函数的地址;或者在本地进程中找到所有远程线程中要用到的 API 的地址,传入远程线程函数中,交与相应的函数指针调用。本文中采用了第一种方式。

```
DWORD __stdcall ThreadProc( RemotePara * lpPara)
{
    //声明 LoadLibrary 函数的指针类型
    typedef HINSTANCE (__stdcall * MLoadLibrary)
(LPCTSTR);
    //定义该类型的函数指针
    MLoadLibrary myLoadLibrary;
    //将传来的 LoadLibrary 函数地址赋给其指针
    myLoadLibrary = ( MLoadLibrary) lpPara -> pro-
cLoadLibrary;
    //调用 LoadLibrary 函数
    hmDllAddress = myLoadLibrary( lpPara -> sMyDll-
Name);
    …
}
```

3.3 远程线程函数中字符串常量的使用

如 2.3 中所述,因为字符串常量在当前进程中分配内存,目标进程中该地址处并没有存储字符串值,所以编写远程线程函数时不能在其中使用字符串常量,需要通过参数传过去。这就要在参数结构体中为用到的字符串常量建立成员,在本地进程赋值,再将参数结构体写入目标进程中,这样才能在远程线程函数中通过形参的结构体指针找到要用的字符串。例如对要插入的 capture. dll,首先在本地进程赋值,

strcpy(myRemotePara. sMyDllName," c: \windows \system1 \capture. dll");

再通过参数 lpPara 传入到远程线程中使用。

hmDllAddress = myLoadLibrary(lpPara -> sMyDll-Name);

3.4 在目标进程中为远程线程函数分配内存

在本地进程启动 Word 后,要在 Word 的进程中用

VirtualAllocEx 函数分配内存，用 WriteProcessMemory 函数写入要建立的远程线程函数。其中 THREADSIZE 值小于实际需要的值时运行远程线程函数时会出错，可通过试验给出。

```
PROCESS_INFORMATION pi; //记录目标进程信息
CreateProcess("c:\program files\microsoft office\offi
ce11\winword.exe", NULL, NULL, NULL, FALSE, NORMAL_PRIORITY_CLASS | CREATE_NEW_CONSOLE | PROCESS_VM_WRITE, NULL, NULL, &si, &pi);
const DWORD THREADSIZE = 1024 * 50;
void * pRemoteThread = ::VirtualAllocEx(pi.hProcess, 0, THREADSIZE, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE); //分配内存
::WriteProcessMemory(pi.hProcess, pRemoteThread, &ThreadProc, THREADSIZE, 0); //植入线程
```

3.5 在目标进程中为远程线程函数的参数结构体分配内存

在本地进程建立向远程线程传参的结构体并赋值后，用 VirtualAllocEx 函数在目标进程中为其分配内存，用 WriteProcessMemory 函数写入要传入的结构体参数（类似于 2.3 中为传入的 DLL 路径的字符串常量分配内存）。

```
RemotePara myRemotePara;
::ZeroMemory(&myRemotePara, sizeof(RemotePara));
myRemotePara.procLoadLibrary = GetProcAddress(hKernel32, "LoadLibraryA");
...
//在目标进程中为参数结构体分配内存
RemotePara * pRemotePara = (RemotePara *)::VirtualAllocEx(hProcess, 0, sizeof(RemotePara), MEM_COMMIT, PAGE_READWRITE);
//将结构体写入目标进程
::WriteProcessMemory(hProcess, pRemotePara, &myRemotePara, sizeof(myRemotePara), 0);
```

3.6 启动远程线程

在本地进程中调用 CreateRemoteThread 函数，传入远程线程地址和结构体参数地址，启动远程线程。

```
HANDLE hThread = ::CreateRemoteThread(hProcess, 0, 0, (DWORD(__stdcall *)(void *))pRemoteThread, pRemotePara, 0, &byte_write);
WAiForSingleObject(hThread, INFINITE); //等待远程线程结束
```

4 结束语

在 DllMAi 中或在 DLL 的构造函数中通过改动模块的输入节挂接 API 的方法因为是在本地进程中写代码，比较容易实现，编写远程线程函数挂接 API 难度较大，不仅包括 API 函数调用和字符串常量的使用，还包括程序的调试，一般只能利用 sprintf 函数和 MessageBox 函数报告程序状态，或者在出错时进入程序调试工具（如 Visual Studio 6.0、Visual Studio 2005 等）界面查错。但后一种方法不仅可用来挂接 API，也可用于研究、改造目标进程的其它方面。

上述程序在 Windows 2000/XP/2003 下用 Microsoft Visual C++ 6.0 编译调试通过。

参考文献

1 Jeffrey Richter 著，王建华等译. Windows 核心编程，第 1 版，北京：机械工业出版社，2000. 518－519，531－564.

2 梁肇新 编著. 编程高手箴言. 第 1 版，北京：电子工业出版社，2003. 119－132，205－213.