# 使用远程线程来注入DLL技术

## 一、原理

使用远程线程来注入DLL，根本上说是要求目标进程使用LoadLibrary函数来加载DLL。为了实现这个目的，可以在我们的进程中远程让目标进程启动一个线程，这个线程的回调函数即为LoadLibrary函数。远程启动线程的函数是CreateRemoteThread。这里还有两个问题需要解决，一是不能将LoadLibrary函数名直接作为CreateRemoteThread，而应将LoadLibrary的实际地址赋给CreateRemoteThread。二是LoadLibrary函数的参数（字符串）也需要传递给CreateRemoteThread，但是字符串地址必须在目标进程的内存空间中，因此，我们需要在目标进程的空间中开辟空间，并写入字符串，然后将该空间地址传递给CreateRemoteThread函数。远程开辟空间的函数是VirtualAllocEx，远程读写的函数是ReadProcessMemory和WriteProcessMemory，这里只用到后者。

## 二、主要函数说明

### 2.1 OpenProcess

HANDLE OpenProcess(

DWORD fdwAccess,

BOOL fInherit,

DWORD IDProcess

);

打开已启动的目标进程，获得目标进程句柄。

### 2.2 CreateRemoteThread

HANDLE WINAPI CreateRemoteThread(

\_In\_   HANDLE hProcess,

\_In\_   LPSECURITY\_ATTRIBUTES lpThreadAttributes,

\_In\_   SIZE\_T dwStackSize,

\_In\_   LPTHREAD\_START\_ROUTINE lpStartAddress,

\_In\_   LPVOID lpParameter,

\_In\_   DWORD dwCreationFlags,

\_Out\_  LPDWORD lpThreadId

);

该函数除了第一个参数hProcess外，其他参数与CreateThread相同，第一个参数标明了远程进程的句柄。函数的回调函数格式如下：

DWORD WINAPI ThreadProc(

\_In\_  LPVOID lpParameter

);

该函数接收一个LPVOID的参数，返回一个DWORD类型返回值，调用约定为WINAPI。

而LoadLibrary函数的声明如下

HMODULE WINAPI LoadLibrary(

\_In\_  LPCTSTR lpFileName

);

该函数接收一个LPCTSTR的参数，返回一个HMODULE类型返回值，调用约定为WINAPI。

两个函数相似，在这里我们用LoadLibrary来替代回调函数，使线程执行时调用LoadLibrary加载DLL。

### 2.3 VirtualAllocEx

LPVOID VirtualAllocEx(

HANDLE hProcess,

LPVOID lpAddress,

DWORD dwSize,

DWORD flAllocationType,

DWORD flProtect

);

预定并调拨内存，具体使用方法见《Windows 核心编程》。

### 2.4 WriteProcessMemory

BOOL WriteProcessMemory(

HANDLE hProcess,

LPVOID lpBaseAddress,

LPVOID lpBuffer,

DWORD nSize,

LPDWORD lpNumberOfBytesWritten

);

在指定进程地址空间中写入数据。

### 2.5 GetProcAddress

FARPROC GetProcAddress(

HMODULE hModule,

LPCWSTR lpProcName

);

从DLL中导出指定函数的地址。

## 三、实现

### 3.1 主程序

实现代码如下：

BOOL InjectDll(LPCWSTR lpPath,DWORD dwProcessID)

{

HANDLE hProcess=NULL;

HANDLE hThread = NULL;

PTSTR pszLibFileRemote = NULL;

BOOL bOK = TRUE;

\_\_try{

hProcess = OpenProcess(PROCESS\_QUERY\_INFORMATION |

PROCESS\_CREATE\_THREAD |

PROCESS\_VM\_OPERATION |

PROCESS\_VM\_WRITE,

FALSE, dwProcessID);

if (hProcess == NULL)

\_\_leave;

int cch = 1 + lstrlenW(lpPath);

int cb = cch\*sizeof(TCHAR);

//在远程进程中分配内存空间用于保存DLL地址

pszLibFileRemote = (PTSTR)VirtualAllocEx(hProcess, NULL, cb, MEM\_COMMIT, PAGE\_READWRITE);

if (pszLibFileRemote == NULL)

\_\_leave;

//将DLL地址写入到远程进程中分配的空间中去

if (!WriteProcessMemory(hProcess, pszLibFileRemote, (LPCVOID)lpPath, cb, NULL))

\_\_leave;

//获得LoadLibraryW的地址

PTHREAD\_START\_ROUTINE pfnThreadRtn=(PTHREAD\_START\_ROUTINE)GetProcAddress(GetModuleHandle(\_T("Kernel32")),"LoadLibraryW");

if (pfnThreadRtn == NULL)

\_\_leave;

//起动线程，线程调用回调函数（LoadLibraryW）将指定DLL载入到目标进程中

hThread = CreateRemoteThread(hProcess, NULL, 0, pfnThreadRtn, pszLibFileRemote, 0, NULL);

DWORD ErrorCode = GetLastError();

if (hThread == NULL)

\_\_leave;

//等线程结束

WaitForSingleObject(hThread,INFINITE);

}

\_\_finally{

if (pszLibFileRemote != NULL)

VirtualFreeEx(hProcess, pszLibFileRemote, 0, MEM\_RELEASE);

if (hThread != NULL)

CloseHandle(hThread);

if (hProcess != NULL)

CloseHandle(hProcess);

bOK = FALSE;

}

return bOK;

}

int APIENTRY \_tWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPTSTR lpCmdLine,

\_In\_ int nCmdShow)

{

BOOL bFlag=InjectDll(\_T("G:\\其它\\VC驿站\\InjectDll\\Debug\\InjectDll.dll"), 1176);

return 0;

}

### 3.2 DLL代码

为了验证DLL是否被注入指定进程，在DLL的DllMain函数中：

#include "stdafx.h"

#include <Windows.h>

#include <tchar.h>

BOOL APIENTRY DllMain( HMODULE hModule,

DWORD ul\_reason\_for\_call,

LPVOID lpReserved

)

{

switch (ul\_reason\_for\_call)

{

case DLL\_PROCESS\_ATTACH:

MessageBox(NULL, \_TEXT("Attach!"), \_TEXT("Message"), MB\_OK);

break;

case DLL\_THREAD\_ATTACH:

break;

case DLL\_THREAD\_DETACH:

break;

case DLL\_PROCESS\_DETACH:

MessageBox(NULL, \_TEXT("DeAttach!"), \_TEXT("Message"), MB\_OK);

break;

}

return TRUE;

}

当DLL被载入的时候，会弹出消息对话框。

## 四、说明

在Window8系统中实验，目标进程只有是自己写的程序时才会注入成功，如果是系统的进程，注入将失败。