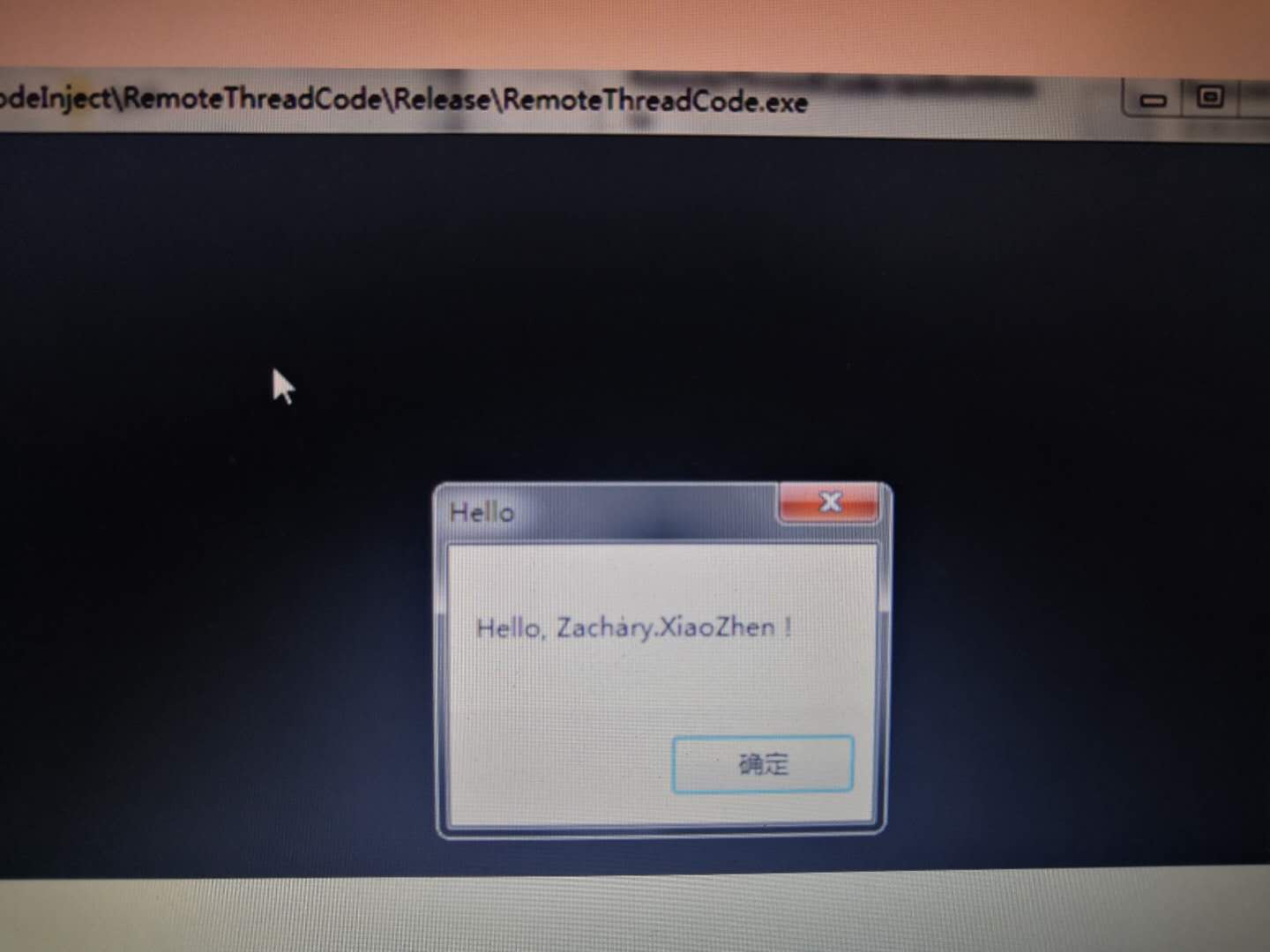
使用直接将所要执行的代码全部拷贝到宿主进程中,即代码远程注入技术

创建的项目为 RemoteThreadCode，即远程注入代码，其实现的功能是当运行 RemoteThreadCode.exe 时，

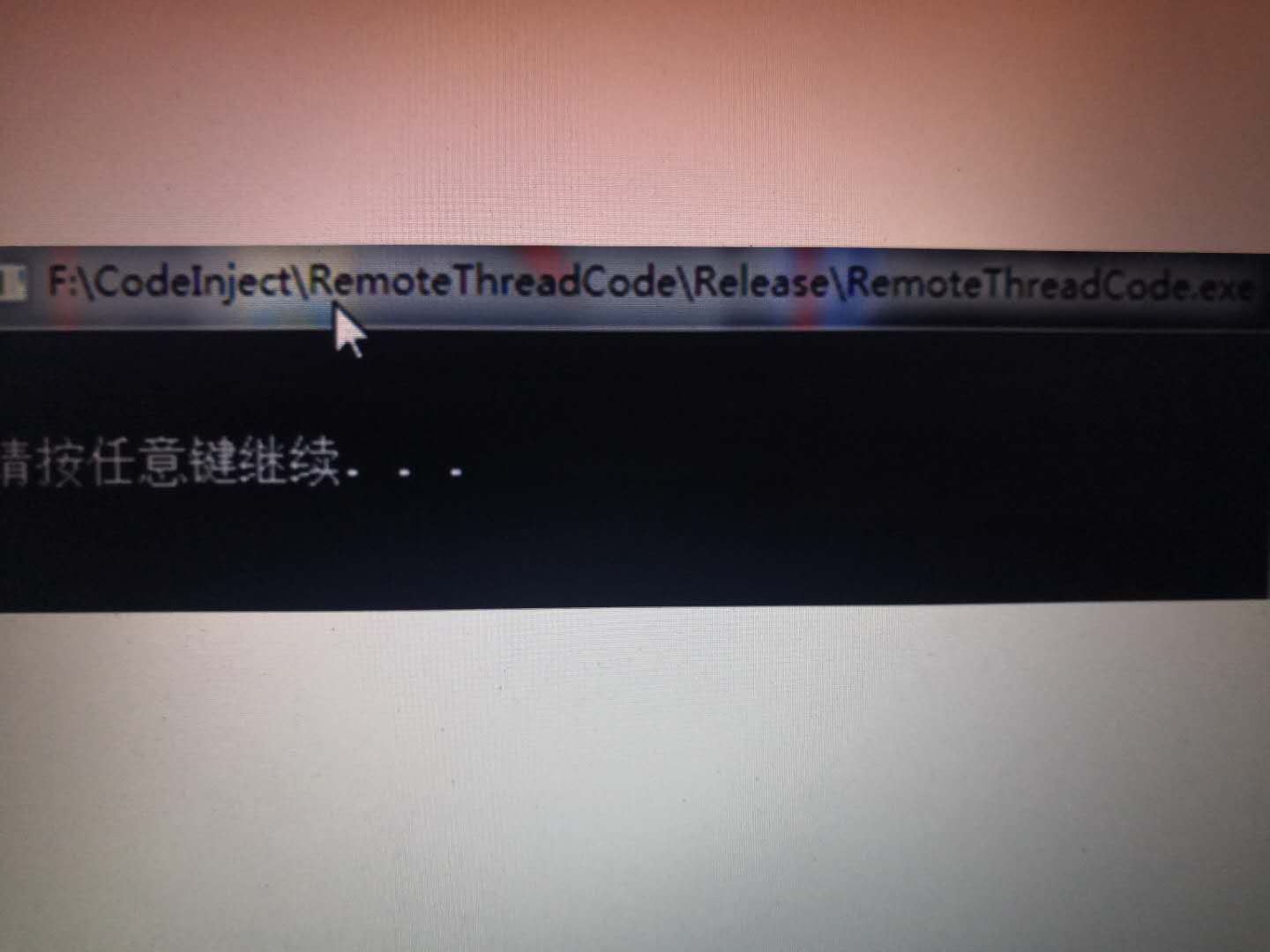
会在 Explorer.exe 进程中创建一个线程，而这个创建的线程功能实现很简单，

就是弹出一个消息框即 OK

当双击执行 RemoteThreadCode.exe 时，则会注入一个线程到 Explorer.exe 中（见图1）



当点击确定后，注入到 Explorer.exe 中的线程执行完毕，从而 WaitForSingleObject 等待成功 （见图2）



基本思路以及所对应的代码：

1. 提升进程权限，如果权限不够的话，很容易造成 OpenProcess 失败;

bool AdjustProcessTokenPrivilege()//提升当前进程权限

{

LUID luidTmp;

HANDLE hToken;

TOKEN\_PRIVILEGES tkp;

if(!OpenProcessToken(GetCurrentProcess(), TOKEN\_ADJUST\_PRIVILEGES | TOKEN\_QUERY, &hToken))

{

OutputDebugString("AdjustProcessTokenPrivilege OpenProcessToken Failed ! \n");

return false;

}

if(!LookupPrivilegeValue(NULL, SE\_DEBUG\_NAME, &luidTmp))

{

OutputDebugString("AdjustProcessTokenPrivilege LookupPrivilegeValue Failed ! \n");

CloseHandle(hToken);

return FALSE;

}

tkp.PrivilegeCount = 1;

tkp.Privileges[0].Luid = luidTmp;

tkp.Privileges[0].Attributes = SE\_PRIVILEGE\_ENABLED;

if(!AdjustTokenPrivileges(hToken, FALSE, &tkp, sizeof(tkp), NULL, NULL))

{

OutputDebugString("AdjustProcessTokenPrivilege AdjustTokenPrivileges Failed ! \n");

CloseHandle(hToken);

return FALSE;

}

return true;

}

2. 确定你的宿主进程，即你所要注入代码的进程

bool ProcessIsExplorer(DWORD dwProcessId)//判定一个进程是否为 Explorer 进程

{

HANDLE hProcess;

hProcess = NULL;

hProcess = OpenProcess(PROCESS\_VM\_READ | PROCESS\_QUERY\_INFORMATION, FALSE, dwProcessId);

if(NULL == hProcess)

{

OutputErrorMessage("ProcessIsExplorer - OpenProcess Failed , Error Code Is %d , Error Message Is %s !");

return FALSE;

}

DWORD dwNameLen;

TCHAR pathArray[MAX\_PATH];

ZeroMemory(pathArray, MAX\_PATH);

dwNameLen = 0;

dwNameLen = GetModuleFileNameEx(hProcess, NULL, pathArray, MAX\_PATH);

if(dwNameLen == 0)

{

CloseHandle(hProcess);

return FALSE;

}

TCHAR exeNameArray[MAX\_PATH];

ZeroMemory(exeNameArray, MAX\_PATH);

\_tsplitpath(pathArray, NULL, NULL, exeNameArray, NULL);

string str1 = exeNameArray;

if((str1.compare("Explorer") == 0) || (str1.compare("explorer") == 0))

{

CloseHandle(hProcess);

return TRUE;

}

return FALSE;

}

3. 打开宿主进程了(我这里打开的是 Explorer.exe 进程),思路是首先变量当前系统下运行的所有的进程,然后遍历获取到得所有的进程的 PID,再调用 ProcessIsExplorer 函数来判断这个进程是否为 Explorer.exe 进程，如果是则记录下这个进程的 PID 就可以了,这样就获得了 Explorer.exe 进程的 PID 了,

再通过 OpenProcess 来打开这个 Explorer.exe 进程

//提升当前进程的权限

AdjustProcessTokenPrivilege();

//第一个参数为用来保存所有的进程 ID

//第二个参数则是第一个参数的字节数

//第三个参数则是写入 dwProcess 数组的字节数

EnumProcesses(dwProcess, sizeof(dwProcess), &dwNeeded);

//找到 explorer.exe 进程的 ID

for(int i = 0; i < dwNeeded / sizeof(DWORD); i++)

{

if(0 != dwProcess[i])

{

if(ProcessIsExplorer(dwProcess[i]))

{

dwExplorerId = dwProcess[i];

break;

}

}

}

hProcess = NULL;

hProcess = OpenProcess(PROCESS\_ALL\_ACCESS, FALSE, dwExplorerId);

if(NULL == hProcess)

{

OutputErrorMessage("main - OpenProcess Failed , Error Code Is %d , Error Message Is %s !");

}

4. 在宿主进程中分配好存储空间，这个存储空间是用来存放我们将要创建的远程线程的线程处理例程的,这里需要注意的是：我们分配的内存必须标记必须带有 EXECUTE,因为分配的这块内存是用来存放线程处理例程的,而线程处理例程必须得执行，所以必须得带有 EXECUTE 标记。因为我们在后面的代码中还必须调用 WriteProcessMemory 来将线程处理例程写入到这块内存中，也需要WRITE 标记。

//在 hProcess 所代表的进程内部分配虚拟内存来容纳我们将要创建的远程线程

PVOID pRemoteThread = VirtualAllocEx(hProcess, NULL, THREAD\_SIZE, MEM\_COMMIT | MEM\_RESERVE, PAGE\_EXECUTE\_READWRITE);

if(NULL == pRemoteThread)

{

OutputErrorMessage("main - VirtualAllocEx Failed , Error Code Is %d , Error Message Is %s !");

//关闭进程句柄

CloseHandle(hProcess);

}

5. 将远程线程处理例程写入到 4 中在宿主进程中所分配的内存中,这个可以直接调用 WriteProcessMemory 来实现;

//往我们在 hProcess 进程中分配的虚拟内存里面写入数据，这里主要是将整个线程都写进去

if(WriteProcessMemory(hProcess, pRemoteThread, &RemoteThreadProc, THREAD\_SIZE, 0) == FALSE)

{

OutputErrorMessage("main - WriteProcessMemory Failed , Error Code Is %d , Error Message Is %s !");

//释放 VirtualAllocEx 分配的内存

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

CloseHandle(hProcess);

}

6. 在宿主进程中分配好存储空间，这个存储空间是用来存放我们将要传递给远程线程线程处理例程的参数,从下面的结构体中可以看出，其由三个参数组成，第一个参数代表要在对话框中显示的内容，第二个参数代表要在对话框中显示的标题，第三个参数则是 MessageBox 这个 API 的地址，因为在 Explorer.exe 中 MessageBox 的地址会发生重定向，所以需要将其地址通过参数传递给线程处理例程。

typedef struct \_REMOTE\_PARAMETER

{

CHAR m\_msgContent[MAX\_PATH];

CHAR m\_msgTitle[MAX\_PATH];

DWORD m\_dwMessageBoxAddr;

}RemotePara, \* PRemotePara;

void GetMessageBoxParameter(PRemotePara pRemotePara)//获得 MessageBox 这个 API 的地址以及填充的参数

{

HMODULE hUser32 = LoadLibrary("User32.dll");

pRemotePara->m\_dwMessageBoxAddr = (DWORD)GetProcAddress(hUser32, "MessageBoxA");

strcat(pRemotePara->m\_msgContent, "Hello, Zachary.XiaoZhen !\0");

strcat(pRemotePara->m\_msgTitle, "Hello\0");

//注意要释放掉 User32

FreeLibrary(hUser32);

}

RemotePara remotePara;

ZeroMemory(&remotePara, sizeof(RemotePara));

GetMessageBoxParameter(&remotePara);

//在 hProcess 所代表的进程中分配虚拟内存来容纳线程的参数部分

PRemotePara pRemotePara = (PRemotePara)VirtualAllocEx(hProcess, NULL, sizeof(RemotePara), MEM\_COMMIT, PAGE\_READWRITE);

if(NULL == pRemotePara)

{

OutputErrorMessage("main - VirtualAllocEx Failed , Error Code Is %d , Error Message Is %s !");

//释放 VirtualAllocEx 分配的内存

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

CloseHandle(hProcess);

}

7. 将参数写入到 6 中在宿主进程中所分配的内存中,同样是调用 WriteProcessMemory 来完成

//往在 hProcess 进程中分配的虚拟内存中写入参数数据

if(WriteProcessMemory(hProcess, pRemotePara, &remotePara, sizeof(RemotePara), 0) == FALSE)

{

OutputErrorMessage("main - WriteProcessMemory Failed , Error Code Is %d , Error Message Is %s !");

//释放 VirtualAllocEx 分配的内存

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

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

CloseHandle(hProcess);

}

8. 调用 CreateRemoteThread 在 Explorer.exe(宿主进程)中创建远程线程;

注意，当远程线程没有执行完时，不能够通过 VirtualFreeEx 来将远程进程中的内存释放掉

其执行完毕后再释放在 Explorer.exe 中所分配的存储空间

HANDLE hThread;

DWORD dwThreadId;

hThread = NULL;

dwThreadId = 0;

//将已经写入到 hProcess 进程中的线程以及线程的参数作为 CreateRemoteThread 的参数，从而创建远程线程

hThread = CreateRemoteThread(hProcess, NULL, 0, (DWORD (WINAPI \*)(LPVOID))pRemoteThread, pRemotePara, 0, &dwThreadId);

if(NULL == hThread)

{

OutputErrorMessage("main - CreateRemoteThread Failed , Error Code Is %d , Error Message Is %s !");

}

else

{

OutputSuccessMessage("Code Inject Success !");

}

//等待远程线程结束

WaitForSingleObject(hThread, INFINITE);

CloseHandle(hThread);

//必须等到远程线程结束后才能释放宿主进程中所分配的内存，否则宿主进程会直接崩溃

//释放 VirtualAllocEx 分配的内存

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

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

CloseHandle(hProcess);

9. 编写好远程线程的线程处理例程即可

DWORD WINAPI RemoteThreadProc(PRemotePara pRemotePara)// 远程线程处理例程

{

//这个 MessageBox 的地址必须由外部参数传入，因为在其他进程中需要重定向

typedef int (WINAPI \*MESSAGEBOXA)(HWND, LPCSTR, LPCSTR, UINT);

MESSAGEBOXA MessageBoxA;

MessageBoxA = (MESSAGEBOXA)pRemotePara->m\_dwMessageBoxAddr;

//调用 MessageBoxA 来打印消息

MessageBoxA(NULL, pRemotePara->m\_msgContent, pRemotePara->m\_msgTitle, MB\_OK);

return 0;

}