

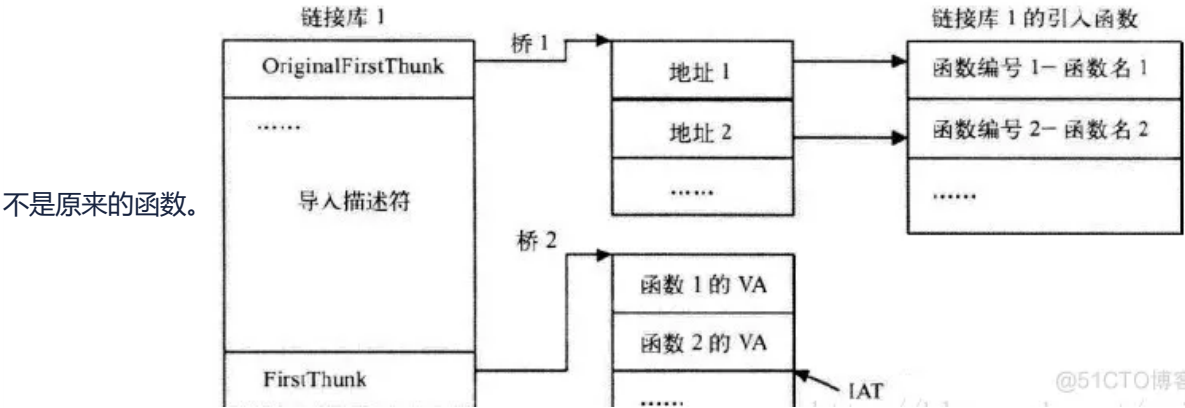
[windows平台] IATHook&原理与实现 (4)

作者 | 榴莲

编辑 | 栢燃

IAT(Import Address Table) HOOK与我们以往学习过的HOOK有着较大的区别。无论是InlineHook还是HotFixHook，都是基于指令的修改，劫持流程来实现HOOK的。而IATHOOK则是采取另一种劫持思路。这里就不得不提到一个Windows系统的文件格式，PE（Portable Executable）文件格式。

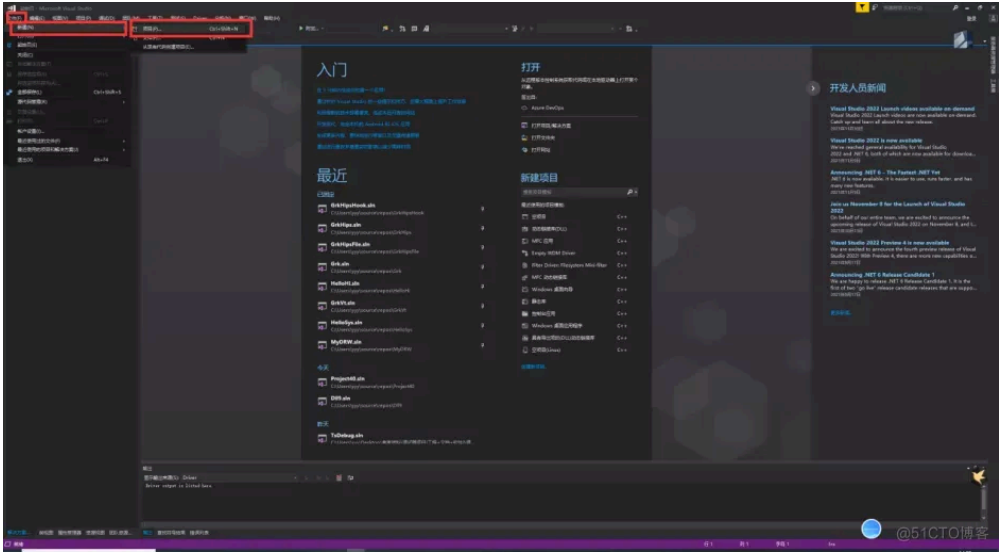
在我们编写一个Windows程序的时候，通常需要调用许多系统或第三方的函数。而需要调用这些函数，就必须加载导出这些函数的动态链接库。对于动态链接库来说，这些函数是导出函数。而对于调用者，也就是我们而言，这些函数就是导入函数。而导入函数的相关信息就存储在PE文件格式的数据目录表中的导入表中。导入表内有三个表，分别是序号，地址以及函数名，进行分别存储。当调用者调用一个函数的时候，就会去导入表中寻找对应函数名的函数地址。那么我们在这个位置就可以做一些事情了。只要将导入地址表（Import Address Table）中我们想要HOOK的函数的地址，替换成我们自己的函数的地址。那么当调用者再次调用函数的时候，因为地址被我们替换。被调用的将是我们提供的HOOK函数，而



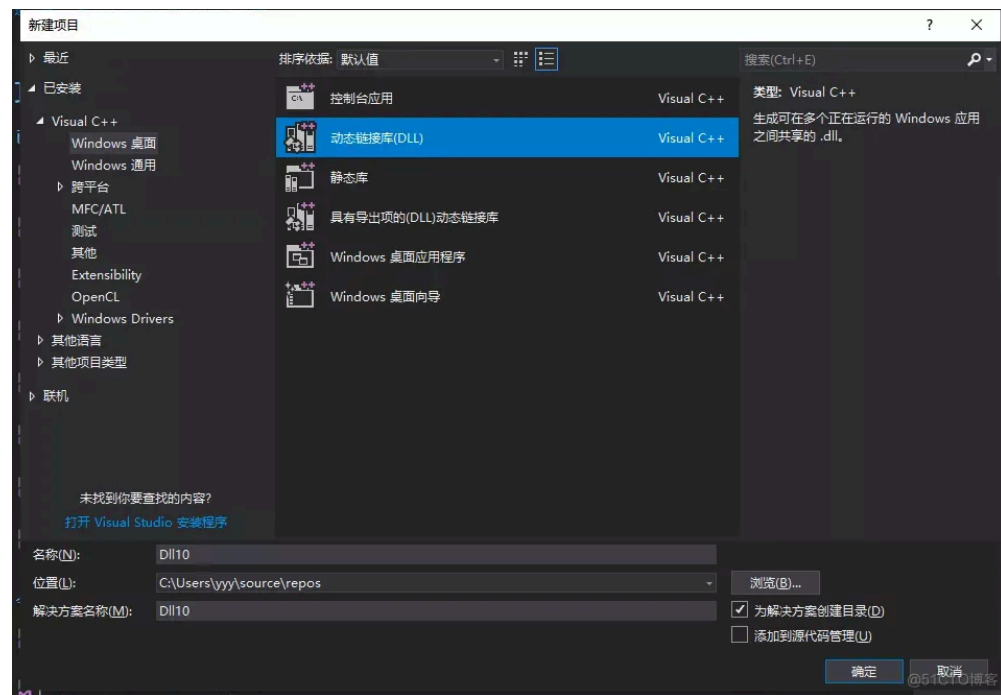
那么以MessageBoxA为例，我们实际上来体验一下IATHOOK的实现方式。

我这里采用的操作系统是Windows 10 20H2（19042.1288），集成开发环境采用的是Visual Studio 2017。那么我们先来创建一个DLL项目。步骤如下：

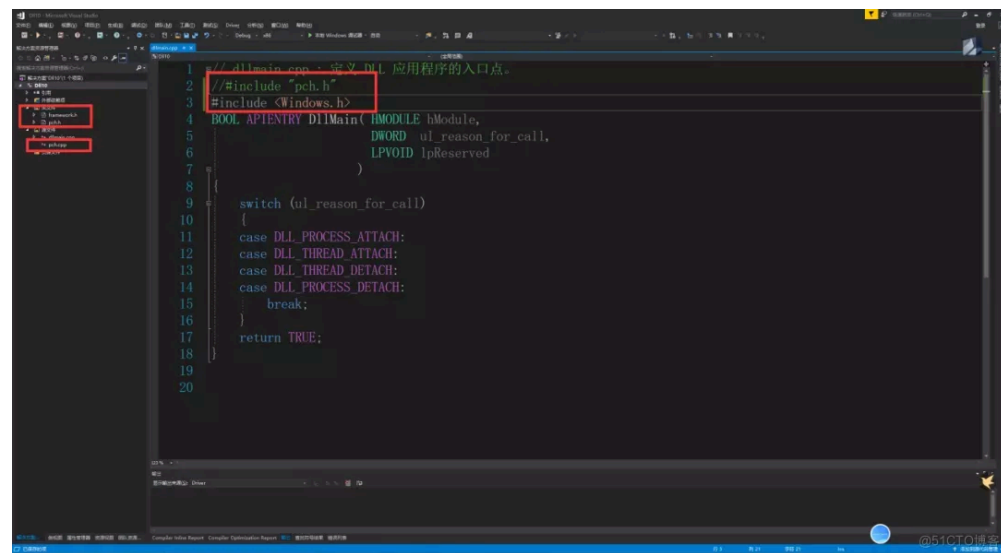
1.选择新建项目



2: 选择Windows桌面->动态链接库（DLL）， 点击确定



3: 注释#include "pch.h",添加#include <Windows.h>。删除framework.h、 pch.h以及pch.cpp文件。

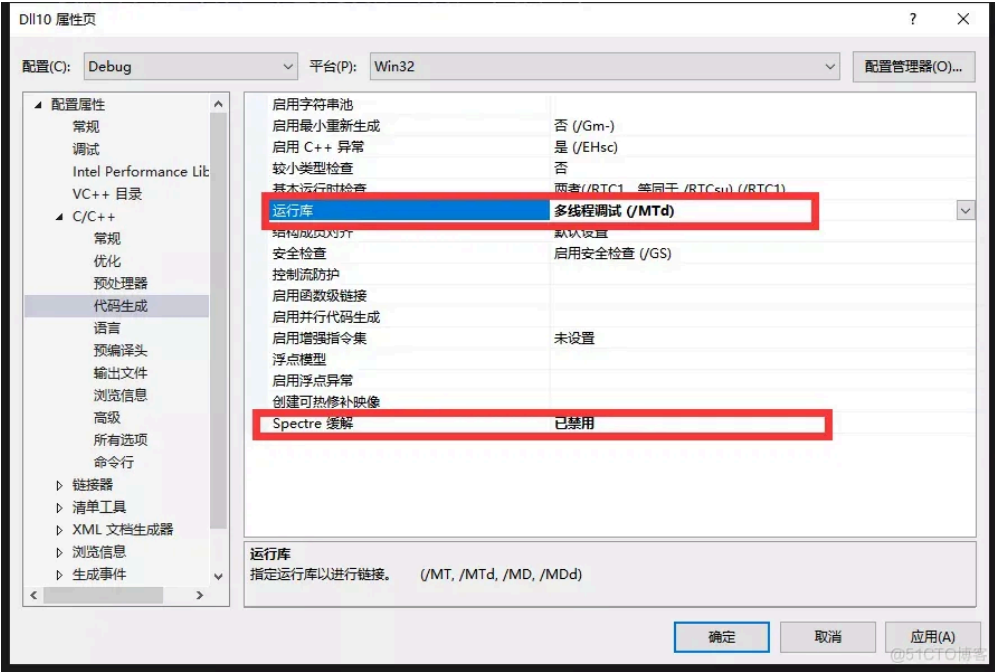


4: 配置

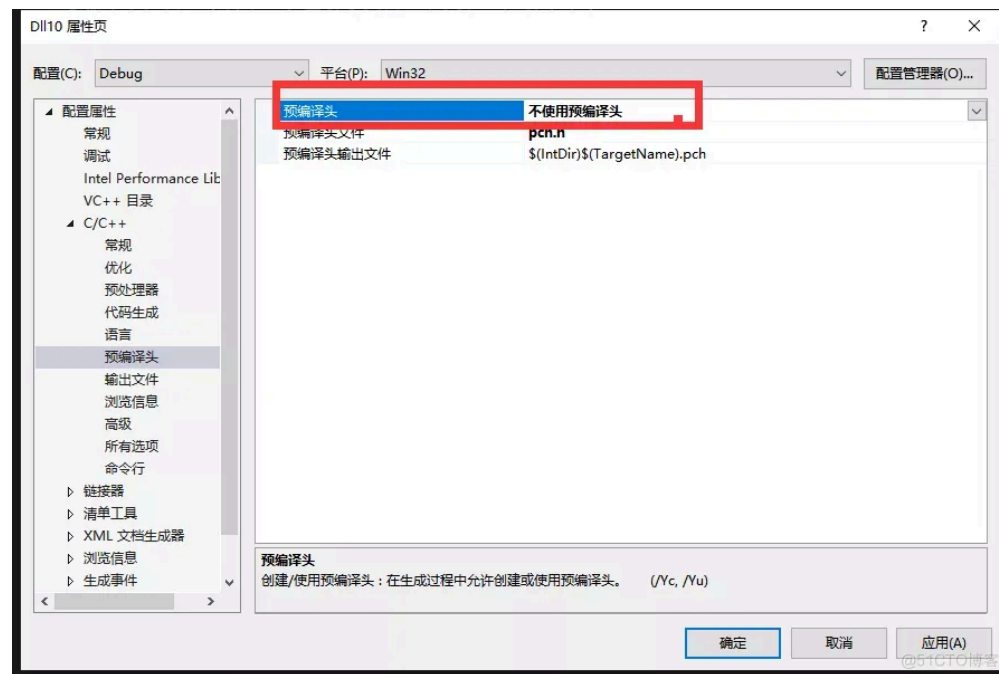
4.1 选择属性



4.2 修改运行库以及Spectre缓解，选择应用



4.3 修改预编译头，选择应用



5. 在每一个分支中，添加break，防止DLL注入失败。

```
BOOL APIENTRY DllMain( HMODULE hModule,
                      DWORD  ul_reason_for_call,
                      LPVOID lpReserved
                      )
{
    switch (ul_reason_for_call)
    {
        case DLL_PROCESS_ATTACH:
            break;
        case DLL_THREAD_ATTACH:
            break;
        case DLL_THREAD_DETACH:
            break;
        case DLL_PROCESS_DETACH:
            break;
    }
    return TRUE;
}
```

@51CTO博客

6. 首先我们需要先声明一个MessageBoxA的函数指针，通过在MessageBoxA上F12即可获得函数原型，然后修改成如下代码，并且声明一个对象，用来保存MessageBoxA原始函数地址：


```
typedef int
(WINAPI *
    FncMessageBoxA) (
    _In_opt_ HWND hWnd,
    _In_opt_ LPCSTR lpText,
    _In_opt_ LPCSTR lpCaption,
    _In_ UINT uType);
FncMessageBoxA OldMesageBoxA = NULL;
```

@51CTO博客

7. 然后我们还需要一个用来替换原始函数的HOOK函数，这里的内容可以任意处理，我这里只是修改参数后调用原函数，修改函数后，无论原来的内容是什么，都会替换成rkvir。

8. 接下来，我们就需要编写HOOK函数了，首先获取函数原始地址，并且解析PE格式，获取导入表。这里的映像地址通过GetModuleHandle(NULL)来进行获取。

```
VOID IATHook ()
{
    //获取需要HOOK的函数的原始地址
    OldMesageBoxA = (FncMessageBoxA)GetProcAddress(GetModuleHandleA("user32.dll"), "MessageBoxA");
    //分析PE文件，获取导入表中的函数地址表，并且替换
    //拿到了DOS头
    PIMAGE_DOS_HEADER pDosHeader = (PIMAGE_DOS_HEADER)GetModuleHandle(NULL);
    //获取NT头
    PIMAGE_NT_HEADERS pNtHeader = (PIMAGE_NT_HEADERS)((DWORD)pDosHeader + (DWORD)pDosHeader->e_lfanew);
    //扩展头
    PIMAGE_OPTIONAL_HEADER pOptionalHeader = (PIMAGE_OPTIONAL_HEADER)&pNtHeader->OptionalHeader;
    //获取数据目录表中的导入表
    DWORD dwImprotTableOffset = pOptionalHeader->DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].VirtualAddress;
    PIMAGE_IMPORT_DESCRIPTOR pImprotTable = (PIMAGE_IMPORT_DESCRIPTOR)((DWORD)pDosHeader + dwImprotTableOffset);
    @51CTO博客
}
```

9. 然后获取并且遍历导入地址表，找到我们的目标函数地址进行替换

```

DWORD * pFirstThunk;
//遍历导入表结构
//判断是否有效
while (pImportTable->Characteristics && pImportTable->FirstThunk != NULL)
{
    //获取数据
    pFirstThunk = (DWORD*)(pImportTable->FirstThunk + (DWORD)pDosHeader);
    //判断数组当前项是否有效
    while (*(DWORD*)pFirstThunk != NULL)
    {
        //如果相当了,就说明当前的数组元素就是我们要找的函数地址表中的函数地址
        if (*(DWORD*)pFirstThunk == (DWORD)OldMessageBoxA)
        {
            DWORD oldProtected;
            //修改内存属性为可读可写可执行,并且保存原有的属性
            VirtualProtect(pFirstThunk, 0x1000, PAGE_EXECUTE_READWRITE, &oldProtected);
            //替换原有的地址成我们自己的HOOK函数
            DWORD dwFuncAddr = (DWORD)MyMessageBoxA;
            memcpy(pFirstThunk, (DWORD *)&dwFuncAddr, 4);
            //还原内存属性
            VirtualProtect(pFirstThunk, 0x1000, oldProtected, &oldProtected);
        }
        pFirstThunk++;
    }
    pImportTable++;
}

```




@51CTO博客

10.接着在DLL_PROCESS_ATTACH里添加IATHOOK的调用。

11. 生成文件

12.取出文件到桌面或其他位置

yyy > source > repos > Dll9 > Debug

名称	修改日期	类型	大小
 Dll9.dll	2021/12/9 11:24	应用程序扩展	826 KB
 Dll9.ilink	2021/12/9 11:24	Incremental Link...	2,126 KB
 Dll9.pdb	2021/12/9 11:24	Program Debug...	6,276 KB

@51CTO博客

13.测试HOOK效果

13.1 首先写一个目标程序，代码如下

```
1 #include <Windows.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     system("pause");
7     MessageBoxA(NULL, "Test", "rkx", MB_OK);
8     system("pause");
9     MessageBoxA(NULL, "Test", "rkx", MB_OK);
10    system("pause");
11    return 0;
12 }
```

13.2使用注入器（自行编写或网上下载，这里我用的是自己写的）将我们生成的模块注入到目标进程中。

正常情况下：



HOOK后:



到了这里，我们就完成了整个IATHook的代码编写。

关于作者

作者: rkvir (榴莲老师)

简介: 曾任某安全企业技术总监; 看雪讲师; 曾任职国内多家大型安全公司; 参与*2国家级安全项目

擅长: C/C++/Python/x86/x64汇编/系统原理&

研究方向: 二进制LD/FUZZ/Windows内核安全/内网GF