使用 detours hook

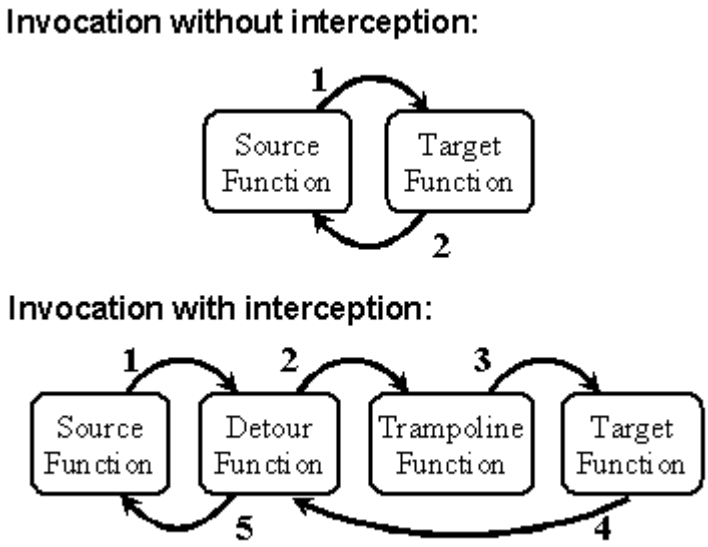
要实现hook目标函数(**Target Function**)，必须具备两个条件:一个是包含**Target Function**地址的目标指针，另一个是**Detours Function**。

为了正确拦截目标函数、**Detour Function**和目标指针必须具有完全相同的调用签名，包括参数数和调用约定。使用相同的调用约定可以确保适当地保留寄存器，并确保堆栈在detour函数和目标函数之间正确对齐。

用户代码必须包含`detours.h`头文件 并 链接`detours.lib`库。

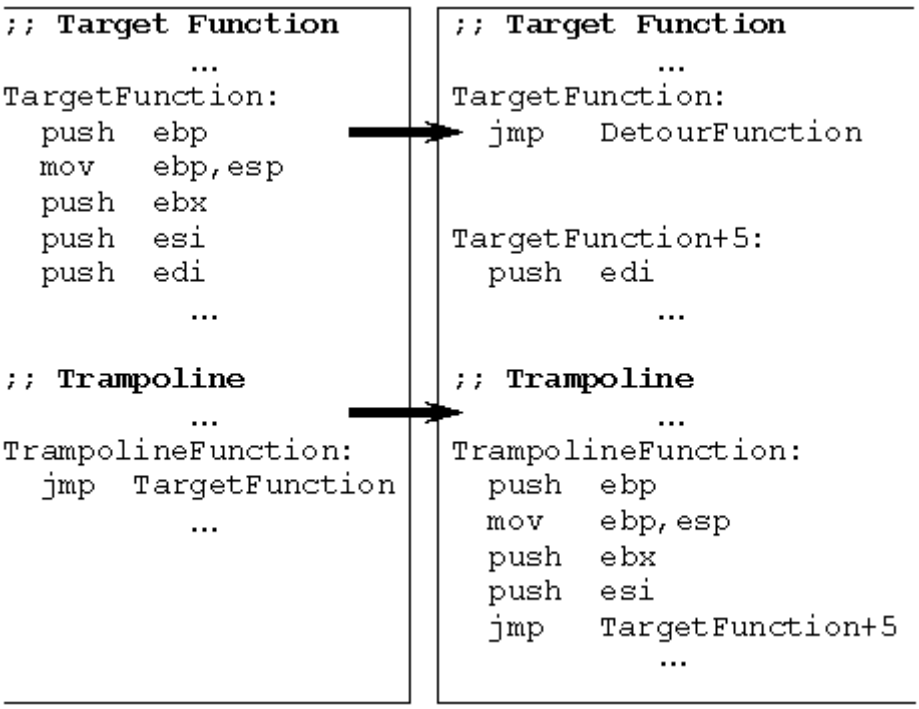
```
#include <cstdio>
#include "windows.h"
#include "include/detours.h"
#pragma comment(lib, "detours.lib")

int (WINAPI *pTrueMessageBox)(HWND hWnd, LPCWSTR lpText,LPCWSTR lpCaption,UINT
uType) = MessageBoxW;
//这是我们需要覆盖拦截的目标函数，即Target Function
int WINAPI MyMessageBox(HWND hWnd, LPCWSTR lpText,LPCWSTR lpCaption,UINT uType);
//这是我们的具体函数实现，即Detours Function
int WINAPI MyMessageBox(HWND hWnd, LPCWSTR lpText,LPCWSTR lpCaption,UINT uType)
{
    // pre
    return pTrueMessageBox(NULL, L"This message has been detoured",
lpCaption,MB_OK);
    // post
}

BOOL APIENTRY DllMain( HMODULE hModule, DWORD  ul_reason_for_call, LPVOID
lpReserved)
{
    switch (ul_reason_for_call)
    {
        case DLL_PROCESS_ATTACH:
            DetourRestoreAfterWith();
            DetourTransactionBegin();
            DetourUpdateThread(GetCurrentThread());
            DetourAttach(&(PVOID&)pTrueMessageBox, MyMessageBox);
            DetourTransactionCommit();
            break;
        case DLL_PROCESS_DETACH:
            DetourTransactionBegin();
            DetourUpdateThread(GetCurrentThread());
            DetourDetach(&(PVOID&)pTrueMessageBox, MyMessageBox);
            DetourTransactionCommit();
            break;

        case DLL_THREAD_ATTACH:
        case DLL_THREAD_DETACH:
            break;
    }
    return TRUE;
}
```

detours的基本概念



- **Target Function** 被hook的方法
- **Source Function** 调用了 **Target Function** 的方法
- **Detour Function** 由我们提供的hook方法
- **Trampoline Function** 用于保存原**Target Function**

detours原理



- detours会负责动态分配空间给**Trampoline Function**, 用于保存原**Target Function**
- 在**Target Function**的开头， 添加无条件跳转指令，跳转到**Detour Function**。 jmp指令需要占用一些空间，如果被Hook的Target Function小于5bytes， 则detours只能返回失败，（比如ntdll!DbgBreakPoint函数）
- 通过对 `DetourTransactionBegin` 和 `DetourTransactionCommit` 的调用来标记Hook事务。

- 通过`DetourAttach`在Hook事务中调用可以拦截目标函数。该`DetourAttach`带有两个参数：**Target Function**指针的地址和指向**Detour Function**函数的指针。

  `DetourAttach`分配和准备跳转调用目标函数。Hook事务提交时，将重写**Target Function**和跳转函数**Trampoline Function**，并更新目标指针以指向跳转函数

- 该`DetourUpdateThread`函数指定一个线程，并在事务提交后确保线程的指令被更新。

  `DetourUpdateThread` 这个函数实际上会挂起指定的线程(如果传入的线程就是自身线程， 它会直接忽略)。 应该用这个函数来确保进程中**所有可能用到被hook API的线程** 被挂起。 最直接的办法就是挂起所有线程。 远程线程需要考虑吗

  - 不考虑远程线程的情况， 代码实现进程内所有线程被挂起

```
#include <cstdio>
#include "windows.h"
#include "DllHook.h"
#include "TlHelp32.h"
#include "set"
#include "include/detours.h"
#pragma comment(lib, "detours.lib")
__attribute__((unused)) BOOL suspendAllThread() {

    bool needRepeat = FALSE;
    DWORD processID = GetCurrentProcessId();
    HANDLE hThreadSnap = CreateToolhelp32Snapshot(TH32CS_SNAPTHREAD,
processID);
    if (hThreadSnap == NULL) {
        return false;
    }

    THREADENTRY32 thread32;
    thread32.dwSize = sizeof(THREADENTRY32);
    if (!Thread32First(hThreadSnap, &thread32)) {
        return false;
    }

    do {
        if (thread32.th32OwnerProcessID == processID) {
            if (threadIdSet.find(thread32.th32ThreadID) ==
threadIdSet.end()) {
                continue;
            }

            needRepeat = TRUE;
            // 此处打开的所有线程要求在detours事务提交后再close
            HANDLE threadHandle = OpenThread(THREAD_SET_CONTEXT |
THREAD_SUSPEND_RESUME, false, thread32.th32ThreadID);
            if (threadHandle == NULL) {
                continue;
            }

            DetourUpdateThread(threadHandle);
            threadIdSet.insert(thread32.th32ThreadID);
        }
    }while(Thread32Next(hThreadSnap, &thread32));
```

```
        if (needRepeat) {
            return suspendAllThread();
        } else {
            return true;
        }
    }
```

- 远程线程的情况
  -

注入方式

远程线程注入

使用 `createRemoteThread()` 在需要注入的进程中创建一个线程用于加载我们的 注入Dll

```
#include <stdio.h>
#include <tchar.h>
#include <windows.h>

bool InjectProcess(DWORD pid, char* dllPath, int len) {
    HANDLE ProcessHandle = OpenProcess(PROCESS_QUERY_INFORMATION |
PROCESS_CREATE_THREAD |
                                       PROCESS_VM_OPERATION | PROCESS_VM_WRITE,
false, pid);
    if (ProcessHandle == NULL) {
        return false;
    }

    LPVOID memoryPointer = VirtualAllocEx(ProcessHandle, NULL, len + 1,
MEM_COMMIT, PAGE_READWRITE);
    if (memoryPointer == NULL) {
        return false;
    }

    if(!WriteProcessMemory(ProcessHandle, memoryPointer, (LPVOID)dllPath, len +
1, NULL)){
        return false;
    }

    FARPROC LoadLibraryFunctionPointer = GetProcAddress(
            GetModuleHandle(TEXT("Kernel32")),
            "LoadLibraryA"
    );
    if (LoadLibraryFunctionPointer == NULL) {
        printf("not find loadLib");
        return false;
    }
    HANDLE ThreadHandle = CreateRemoteThread(ProcessHandle, NULL, 0,

(PTHREAD_START_ROUTINE)LoadLibraryFunctionPointer,
                                             memoryPointer, 0, NULL);
```

```
    if (ThreadHandle == NULL) {
        printf("create remote thread failed");
        return false;
    }

    WaitForSingleObject(ThreadHandle, INFINITE);

    CloseHandle(ProcessHandle);
    CloseHandle(ThreadHandle);

    return true;
}
```

使用detours提供API 创建进程

`DetourCreateProcessWithDllEx` 可以创建一个新进程，并将指定的dll载入其中。大致实现原理： 使用 `CREATE_SUSPENDED` 属性 创建进程， detours会修改内存中映像文件， 将指定的dll插入到导入表的第一个位置。 之后触发resume， 进程会首先加载我们注入的dll和其他dll， 然后才执行程序。 几个注意点：

- 第六个参数 dwCreationFlags， 要加上 `CREATE_SUSPENDED`
- 指定的导入表必须导出 `ordinal #1`, 否则该函数会失败。 (可能是至少有一个导出函数的意思)
- 目前已经允许父进程是64位 子进程是32位 或 父进程是32位 子为64位。

```
#include <stdio.h>
#include <windows.h>
#include "include/detours.h"
#pragma comment(lib, "lib/detours.lib")
BOOL createProcessWithDll(char* exePath, char* dllPath)
{
    /* 参数定义 */
    _In_opt_ LPCSTR lpApplicationName = NULL;                   //传给 API
createProcess
    _Inout_opt_ LPSTR lpCommandLine = exePath;                  //传给 API
createProcess
    _In_opt_ LPSECURITY_ATTRIBUTES lpProcessAttributes = NULL;  //传给 API
createProcess
    _In_opt_ LPSECURITY_ATTRIBUTES lpThreadAttributes = NULL;   //传给 API
createProcess
    _In_ BOOL bInheritHandles = false;                          //传给 API
createProcess

    //传给 API createProcess, 不带CREATE_SUSPENDED好像可以
    _In_ DWORD dwCreationFlags = CREATE_DEFAULT_ERROR_MODE | CREATE_SUSPENDED;

    _In_opt_ LPVOID lpEnvironment = NULL;                       //传给 API
createProcess
    _In_opt_ LPCSTR lpCurrentDirectory = NULL;                  //传给 API
createProcess
```

```
    _In_ STARTUPINFOA lpStartupInfo = { 0 };                    //传给 API
createProcess
    _Out_ PROCESS_INFORMATION lpProcessInformation = { 0 };    //传给 API
createProcess 返回的进程信息
    _In_ LPCSTR lpDllName = dllPath;

    // 指向CreateProcess API 的指针， 允许为null
    _In_opt_ PDETOUR_CREATE_PROCESS_ROUTINEA pfCreateProcessA =
            reinterpret_cast<PDETOUR_CREATE_PROCESS_ROUTINEA>
((PDETOUR_CREATE_PROCESS_ROUTINEW) CreateProcessA);



    // 函数调用
    if (!DetourCreateProcessWithDllEx(lpApplicationName, lpCommandLine,
                                       lpProcessAttributes,lpThreadAttributes,
                                       bInheritHandles, dwCreationFlags,
                                       lpEnvironment, lpCurrentDirectory,
                                       &lpStartupInfo, &lpProcessInformation,
                                       lpDllName, pfCreateProcessA)){

        printf("create process with dll failed, error:%d \n", GetLastError());
        return FALSE;
    }

    ResumeThread(lpProcessInformation.hThread);
    WaitForSingleObject(lpProcessInformation.hProcess, INFINITE);

    return true;
}
```

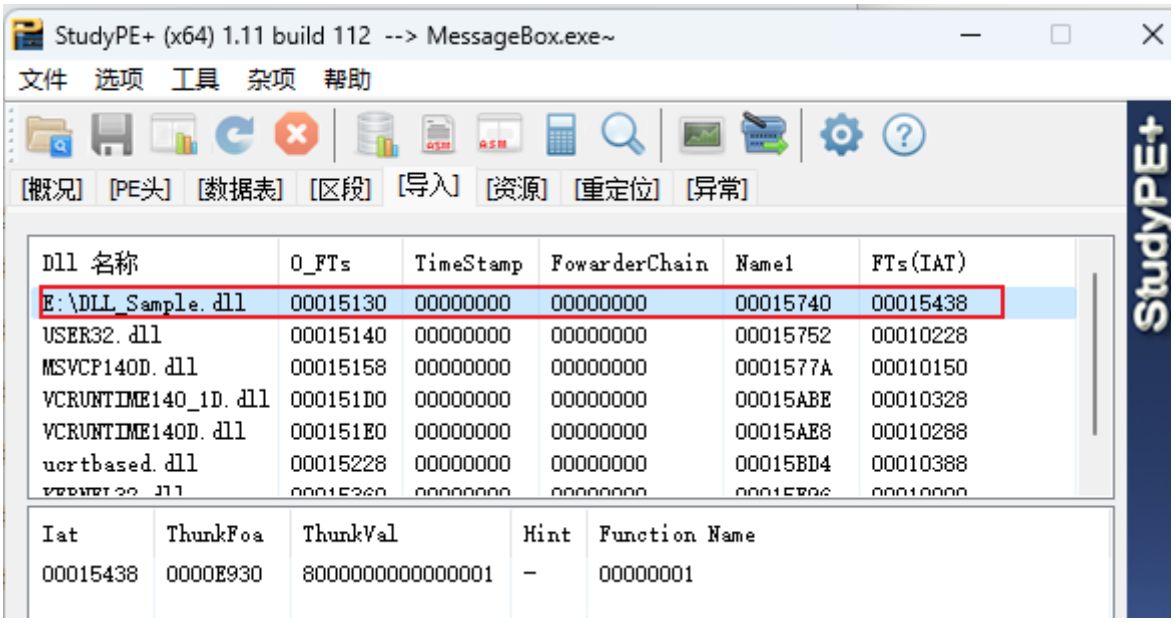
使用detours编辑二进制文件

detours提供了一些函数允许修改windows的二进制文件，通过修改PE文件的导入表信息， 可以实现dll注入。

```
#include <cstdio>
#include "windows.h"
#include "include/detours.h"
#pragma comment(lib, "detours.lib")

BOOL ALREADY_INSERT = FALSE;

static BOOL CALLBACK byway_callback(
        _In_opt_ PVOID pContext,
        _In_opt_ LPCSTR pszFile,
        _Outptr_result_maybenull_ LPCSTR *ppszOutFile) {

    if (!ALREADY_INSERT) {
        *ppszOutFile = (char*)pContext;
        ALREADY_INSERT = TRUE;
```

```
        return TRUE;
    }
    return TRUE;
}

BOOL editPE(char* exePath, char* newExePath, char* dllPath) {

    BOOL bRet = FALSE;
    HANDLE hFile = NULL;
    HANDLE newHFile = NULL;
    do {
         hFile = CreateFileA(exePath, GENERIC_READ,
                                 FILE_SHARE_READ, NULL, OPEN_EXISTING,
                                 FILE_ATTRIBUTE_NORMAL, NULL);
         newHFile = CreateFileA(newExePath, GENERIC_WRITE | GENERIC_READ,
                                    FILE_SHARE_READ, NULL, CREATE_ALWAYS,
                                    FILE_ATTRIBUTE_NORMAL |
FILE_FLAG_SEQUENTIAL_SCAN, NULL);

        if (hFile == NULL || newHFile == NULL) {
            printf("create file failed, %p %p", hFile, newHFile);
            break;
        }

        // 通过文件句柄得到文件二进制数据
        PDETOUR_BINARY fileBinary = DetourBinaryOpen(hFile);
        if (fileBinary == NULL) {
            printf("bin open failed, %ld", GetLastError());
            break;
        }
        DetourBinaryResetImports(fileBinary);

        /*设置回调函数, DetourBinaryEditImports通过回调函数来修改导入表的,  回调的触发机制
如下:
         * ** 假设这个pe文件导入了两个动态库  kernel32.dll 中的 CloseHandle() 和
CreateFileA()
         *                                msvcrt.dll   中的 _strdup
         *
         * ** 回调函数的触发顺序为       bywayCallback(..., NULL, ...)
         *                              FileCallback(..., kernel32.dll, ...)
         *                              SymbolCallback(..., CloseHandle, ...)
         *                              SymbolCallback(..., CreateFileA, ...)
         *                              bywayCallback(..., NULL, ...)
         *                              FileCallback(..., msvcrt.dll, ...)
         *                              SymbolCallback(..., _strdup, ...)
         *                              bywayCallback(..., NULL, ...)
         *                              CommitCallback(...)
         **/
        _In_opt_ PF_DETOUR_BINARY_BYWAY_CALLBACK BywayCallback = byway_callback;
        _In_opt_ PF_DETOUR_BINARY_FILE_CALLBACK FileCallback = NULL;
        _In_opt_ PF_DETOUR_BINARY_SYMBOL_CALLBACK SymbolCallback = NULL;
        _In_opt_ PF_DETOUR_BINARY_COMMIT_CALLBACK CommitCallback = NULL;
        if (!DetourBinaryEditImports(fileBinary, dllPath,
                                     BywayCallback, FileCallback, SymbolCallback,
CommitCallback)) {
            printf("edit pe failed, %ld", GetLastError());
```

```
            break;
        }

        // 保存
        if (!DetourBinaryWrite(fileBinary, newHFile)) {
            printf("DetourBinaryWrite failed: %ld\n", GetLastError());
            break;
        }
        bRet = TRUE;
    }while(0);

    if (hFile != NULL) {
        CloseHandle(hFile);
    }
    if (newHFile != NULL) {
        CloseHandle(hFile);
    }
    return bRet;
}
```



但是文件图标丢失了， 暂时不知道什么原因