

## Forum:[Hack Forums](#)

Thread: [\[Solved\] Simple DLL injection w/ Function Hooking Not Working Correctly...](#)

**Goodies** - May 2, 2016, 11:34 am

Solution:

All I had to do was return TRUE from the DllMain function. Apparently, if you don't return true, it will unload the DLL immediately.

I wrote a couple quick classes. These are split into 3 monolithic applications.

HookDLL- This is a .dll library.

InjectDLL - This is an .exe used to inject DLLs.

TestMe - This is an .exe that just loops calling message box over and over to test when it has been hooked.

EasyHook.hpp

PHP Code

```
<?php
#ifndef EASYHOOK_HPP
#define EASYHOOK_HPP
#include <cstdio>
#include <iostream>

#define WIN32_LEAN_AND_MEAN
#include <Windows.h>

#ifdef __cplusplus
#define EC_BEG extern "C" {
#define EC_END }
#else
#define EC_BEG
#define EC_END
#endif

namespace EasyHook {
    enum opcode {
        LONGJUMP_SIZE = 0x05,
        RELJUMP = 0xe9,
        NOP = 0x90,
    };

    template<typename T, typename... Args>
    struct WrapperSTD {
        typedef T(WINAPI *Functor)(Args...);
        WrapperSTD() = default;
    };
}
```

```

template <typename T, typename... Args>
struct WrapperC {
    typedef T(*Functor)(Args...);
    WrapperC() = default;
};

template<typename T, typename... Args>
WrapperC<T, Args...> wrap(T(*func)(Args...)) {
    return WrapperC<T, Args...>{};
}

template<typename T, typename... Args>
WrapperSTD<T, Args...> wrap(T(WINAPI *func)(Args...)) {
    return WrapperSTD<T, Args...>{};
}

class Hook {
    ULONG oldProtection = NULL;
public:
    Hook() = default;
    LPVOID HookLong(LPVOID target, LPVOID Destination, ULONG jumpSize = 5) {
        LPVOID trampoline = NULL;
        ULONG OldProtection = 0;
        if (jumpSize < 5)
            return 0;
        if (!VirtualProtect((LPVOID)target, 1, PAGE_EXECUTE_READWRITE, &OldProtection))
            return 0;

        if (!(trampoline = VirtualAlloc(NULL, 1, MEM_COMMIT | MEM_RESERVE,
PAGE_EXECUTE_READWRITE))) {
            VirtualProtect((LPVOID)target, 1, OldProtection, &OldProtection);
            return 0;
        }

        memset(trampoline, opcode::NOP, jumpSize);

        *(BYTE*)trampoline = opcode::RELJUMP;
        *(ULONG*)((ULONG)trampoline + 1) = jumpSize;
        trampoline = (LPVOID)((ULONG)trampoline + opcode::LONGJUMP_SIZE);
        memcpy(trampoline, (LPVOID)target, jumpSize);
        *(BYTE*)((ULONG)trampoline + jumpSize) = opcode::RELJUMP;
        *(ULONG*)((ULONG)trampoline + jumpSize + 1) = (ULONG)target - (ULONG)trampoline -
opcode::LONGJUMP_SIZE;
        *(ULONG*)((ULONG)trampoline + jumpSize + 1 + sizeof(ULONG)) = (ULONG)target;
        memset((LPVOID)target, opcode::NOP, jumpSize);
        *(BYTE*)target = opcode::RELJUMP;
        *(ULONG*)((ULONG)target + 1) = (ULONG)Destination - (ULONG)target - opcode::
LONGJUMP_SIZE;
        VirtualProtect((LPVOID)target, 1, OldProtection, &OldProtection);
        return trampoline;
    }

    LPVOID UnhookLong(LPVOID trampoline) {

```

```

        LPVOID original = NULL;
        ULONG jumpSize = 0;
        trampoline = (LPVOID)((ULONG)trampoline - sizeof(ULONG));
        jumpSize = *(PULONG)trampoline;
        original = (LPVOID)((ULONG)trampoline + jumpSize + opcode::LONGJUMP_SIZE);
        return original;
    }
};
}
#endif

```

Source.cpp

PHP Code

```
<?php
```

```
#include "EasyHook.hpp"
```

```
#include <iostream>
```

```
#include <iomanip>
```

```
EC_BEG
```

```

using MessageBoxAF = decltype(EasyHook::wrap(MessageBoxA))::Functor;
static int WINAPI MessageBoxA_H(_In_opt_ HWND hWnd, _In_opt_ LPCSTR lpText,
    _In_opt_ LPCSTR lpCaption, _In_ UINT uType);
static MessageBoxAF trampoline;
EC_END

```

```

static int WINAPI MessageBoxA_H(_In_opt_ HWND hWnd, _In_opt_ LPCSTR lpText,
    _In_opt_ LPCSTR lpCaption, _In_ UINT uType) {
    std::cout << "Testing!\n";
    lpText = "Hook Successful";
    std::cout << (ULONG)trampoline << std::endl;
    if (trampoline)
        return trampoline(hWnd, lpText, lpCaption, uType);
    return 0;
}

```

```

BOOL WINAPI DllMain(_In_ HINSTANCE hinstDLL, _In_ DWORD fdwReason,
    _In_ LPVOID lpvReserved) {
    switch (fdwReason) {
        case DLL_PROCESS_ATTACH:
        case DLL_THREAD_ATTACH:
            std::cout << "Working!\n";
            std::cout << std::hex << MessageBoxA_H << std::endl;
            trampoline = (MessageBoxAF)EasyHook::Hook().HookLong(
                GetProcAddress(GetModuleHandleA("user32.dll"), "MessageBoxA"), MessageBoxA_H
            );
            std::cout << std::hex << trampoline << std::endl;
            break;
    };
}

```

The DLL injector EXE has 2 files:

InjectDLL.hpp

## PHP Code

```
<?php
#ifdef INJECTDLL_HPP
#define INJECTDLL_HPP

#define WIN32_LEAN_AND_MEAN
#include <Windows.h>
#include <tlhelp32.h>

#include <iostream>
#include <sstream>
#include <cstdio>
#include <vector>
#include <string>

namespace injection {
#   define ERROR(error) std::cerr << error
#   define SUCCESS(message) std::cout << message

    using std::vector;
    using std::string;
    using std::stringstream;

    vector<string> &split(const string &s, char delim, vector<string> &elems);
    vector<string> split(const string &s, char delim);
    string filename(string sFilePath);

    typedef HMODULE(WINAPI* LoadLibraryA_F)(LPCTSTR lpFileName);

    class dllinjector {
    public:
        dllinjector() = default;
        bool inject(LPCSTR szProcName, LPCSTR szDLLPath) const;
// provide a process name and DLL path
        bool inject(DWORD dwPID, LPCSTR szDLLPath) const; // provide a process ID and DLL path
        bool inject(HANDLE hProc, LPCSTR szDLLPath) const;
// provide a handle to a process and DLL path
    };
}
#endif//INJECTDLL_HPP
```

## InjectDLL.cpp

## PHP Code

```
<?php
#include "InjectDLL.hpp"

namespace injection {
    vector<string> &split(const std::string &s, char delim, std::vector<std::string> &elems) {
        stringstream ss(s);
        string item;
        while (getline(ss, item, delim))
```

```

        elems.push_back(item);
    return elems;
}

vector<string> split(const string &s, char delim) {
    vector<string> elems;
    split(s, delim, elems);
    return elems;
}

string filename(string sFullPath) {
    return split(sFullPath, '\\').back();
}

bool dllinjector::inject(HANDLE hProc, LPCSTR szDLLPath) const {
    bool success = FALSE;
    LPVOID lpBase = NULL;
    HANDLE hThread = NULL;
    FARPROC LoadLibraryAF = NULL;
    DWORD dwPathSize = strlen(szDLLPath) + 1;
    lpBase = VirtualAllocEx(hProc, NULL, dwPathSize, MEM_COMMIT, PAGE_READWRITE);
    if (lpBase) {
        if (WriteProcessMemory(hProc, lpBase, szDLLPath, dwPathSize, NULL)) {
            LoadLibraryAF = GetProcAddress(GetModuleHandleA("Kernel32"), "LoadLibraryA");
            hThread = CreateRemoteThread(hProc, NULL, 0, (LPTHREAD_START_ROUTINE)
LoadLibraryAF, lpBase, 0, NULL);
            if (hThread) {
                WaitForSingleObject(hThread, INFINITE);
                success = TRUE;
            } else return false;
        }
        VirtualFreeEx(hProc, lpBase, 0, MEM_RELEASE);
    } else return false;
    return success;
}

bool dllinjector::inject(DWORD dwPID, LPCSTR szDLLPath) const {
    HANDLE hProc = OpenProcess(PROCESS_ALL_ACCESS, FALSE, dwPID);
    if (!hProc)
        return false;
    return inject(hProc, szDLLPath);
}

bool dllinjector::inject(LPCSTR szProcName, LPCSTR szDLLPath) const {
    PROCESSENTRY32 entry;
    entry.dwSize = sizeof(entry);
    HANDLE snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, NULL);
    if (Process32First(snapshot, &entry) == TRUE) {
        while (Process32Next(snapshot, &entry) == TRUE) {
            if (!strcmp(entry.szExeFile, szProcName)) {
                HANDLE hProc = OpenProcess(
                    PROCESS_CREATE_THREAD | PROCESS_QUERY_INFORMATION |
PROCESS_VM_OPERATION | PROCESS_VM_WRITE |

```

```

        PROCESS_VM_READ | PROCESS_ALL_ACCESS, FALSE, entry.th32ProcessID
    );
    if (hProc)
        return inject(hProc, szDLLPath);
    }
}
}
CloseHandle(snapshot);
return false;
}
}
}

```

Source.cpp

PHP Code

```

<?php
#include "InjectDLL.hpp"
#include <string>
#include <fstream>

#define DLLPATH "c:/users/austin archer/documents/visual studio 2015/Projects/HookDLL/Release/
HookDLL.dll"
#define PID 11852 //PID of target application

using std::string;
using std::ifstream;

bool exists(string name) {
    ifstream f(name.c_str());
    bool ret = f.good();
    return f.close(), ret;
}

int main(int argc, char **argv) {
    if (!exists(DLLPATH))
        return 0;
    injection::dllinjector injector;
    injector.inject(PID, DLLPATH);
    //std::cin.ignore();
    return 0;
}

```

The PID I change to the PID of the target application. This is the target:

PHP Code

```

<?php
#include <iostream>
#include <sstream>
#include <string>

#define WIN32_LEAN_AND_MEAN
#include <Windows.h>

using std::string;
using std::stringstream;

```

```

int main() {
    unsigned long i = 0;
    stringstream ss;
    stringstream message;
    message << "Testing my message. PID = " << GetCurrentProcessId() << "\0";
    bool cont = true;
    while (cont) {
        ss.swap(stringstream());
        ss << "Test #" << i++ << "\0";
        cont = !(MessageBoxA(NULL, message.str().c_str(), ss.str().c_str(), MB_OKCANCEL) ==
IDCANCEL);
    }
}

```

The trouble is that this works fine in my local application. If I were to only run the EasyHook.hpp function and hook the MessageBoxA function, it works just fine. However, the trouble is that it is not working when I inject it into the target application. I am sure it has something to do with the way I define the functions or something, but I just don't know. This is my first experience with windows API, so any help is appreciated.

Video of what happens when I inject: [\(Click to View\)](#)  
<https://i.imgur.com/vONZN5S.gifv>

Video of it working correctly in a local application without injection: [\(Click to View\)](#)  
<https://i.imgur.com/vsAeZDY.gif>

**Benoit balls** - May 2, 2016, 12:26 pm

Try running the code inside the DllMain in a new thread.

**excr4sh** - May 2, 2016, 12:39 pm

You are using std::cout and other functions in the DLL so you're assuming that the target process is always a console application, that could cause some problems. You can use OutputDebugString function in Windows to help in debugging.

I highly recommend not using standard libraries in a DLL, unless you are initialize them properly, specially if you are using global variables.

**Goodies** - May 2, 2016, 12:50 pm

(May 2, 2016 12:26 PM)?Leonardo Da Vinci Wrote: [\[CITE\]](#) Try running the code inside the DllMain in a new thread.

Okay. How can I fix that?

(May 2, 2016 12:39 PM)excr4sh Wrote: [\[CITE\]](#) \* You are using std::cout and other functions in the DLL so you're assuming that the target process is always a console application, that could cause some problems. You can use OutputDebugString function in Windows to help in debugging.

\* I highly recommend not using standard libraries in a DLL, unless you are initialize them properly, specially if you are using global variables.

In this case, it is a console application. That's not meant to be in the final 'product' by any means... it's strictly for testing.

excr4sh - May 2, 2016, 5:16 pm

(May 2, 2016 12:50 PM)Goodies Wrote: [CITE](#) In this case, it is a console application. That's not meant to be in the final 'product' by any means... it's strictly for testing. Okay, yet, If you want to follow the flow of execution I recommend using OutputDebugString function, It's fast and easy to use.

I also recommend using a disassembler, eg HDE. It's not the best disassembler but it will work in most cases, try this dll:

```
Code #include <Windows.h>
#include <hde32.h>

#pragma comment(linker, "/ENTRY:DllMain")
#pragma comment(linker, "/SUBSYSTEM:WINDOWS")

#define HOOK_SIZE (5)
#define REL_DWORD_JMP(dest, src) ((DWORD)((LPBYTE)((LPBYTE)(dest) - (LPBYTE)(src)) - HOOK_SIZE))

typedef int (WINAPI* TdMessageBoxW)(_In_opt_ HWND hWnd, _In_opt_ LPCWSTR lpText,
_In_opt_ LPCWSTR lpCaption, _In_ UINT uType);

BOOL bHooked = FALSE;
LPVOID StubCode;

int WINAPI MessageBoxWHook(
    _In_opt_ HWND hWnd,
    _In_opt_ LPCWSTR lpText,
    _In_opt_ LPCWSTR lpCaption,
    _In_ UINT uType
)
{
    OutputDebugStringW(lpText);
    return ((TdMessageBoxW)StubCode)(hWnd, lpText, lpCaption, uType);
}

BOOL HookFunction(_In_ LPVOID TargetFunction, _In_ LPVOID HookFunction)
{
    BOOL bResult = FALSE;
    SIZE_T IDisam = 0;
    hde32s Disam;
    BYTE HookCode[HOOK_SIZE];
    DWORD OldProtect;

    while (IDisam < HOOK_SIZE && hde32_disasm((LPVOID)((LPBYTE)TargetFunction + IDisam),
&Disam))
    {
        IDisam += Disam.len;
    }
}
```

```

StubCode = VirtualAlloc(NULL, IDisam + HOOK_SIZE, MEM_RESERVE | MEM_COMMIT,
PAGE_EXECUTE_READWRITE);
if (StubCode)
{
    HookCode[0] = 0xE9;
    *((LPDWORD)((LPBYTE)HookCode + 1)) = REL_DWORD_JMP(HookFunction,
TargetFunction);

    RtlCopyMemory(StubCode, TargetFunction, IDisam);
    *((LPBYTE)StubCode + IDisam) = 0xE9;
    *((LPDWORD)((LPBYTE)StubCode + IDisam + 1)) =
REL_DWORD_JMP((LPBYTE)TargetFunction + IDisam, (LPBYTE)StubCode + IDisam);

    if (VirtualProtect(
        TargetFunction, HOOK_SIZE, PAGE_EXECUTE_READWRITE, &OldProtect))
    {
        RtlCopyMemory(TargetFunction, HookCode, HOOK_SIZE);
        VirtualProtect(TargetFunction, HOOK_SIZE, OldProtect, &OldProtect);
        bResult = TRUE;
    }
}

return bResult;
}

```

```

BOOLEAN APIENTRY DllMain(
    _In_ HMODULE hModule,
    _In_ DWORD Reason,
    _In_ LPVOID lpReserved
)
{
    switch (Reason)
    {
        case DLL_PROCESS_ATTACH:
            if (!bHooked)
            {
                bHooked = HookFunction(MessageBoxW, MessageBoxWHook);
            }
            break;
    }
    return TRUE;
}

```

I use HDE, you can download it from here:

<https://github.com/kimperator/MologieDetours/tree/master/hde32>

**Benoit balls** - May 3, 2016, 12:57 am

(May 2, 2016 12:50 PM)Goodies Wrote: [\[CITE\]](#) Okay. How can I fix that?

In this case, it is a console application. That's not meant to be in the final 'product' by any means... it's strictly for testing.

Try this :

<http://pastebin.com/eaLPS7tR>

**Goodies** - May 3, 2016, 10:54 am

(May 3, 2016 12:57 AM)?Leonardo Da Vinci Wrote: [\[CITE\]](#) Try this :

<http://pastebin.com/eaLPS7tR>

That did not work. Arguable, it made it worse. I cannot even call a simple function from a new thread. Even without running the hook at all, it still gives an error when creating the thread.

**excr4sh** - May 3, 2016, 5:55 pm

(May 3, 2016 10:54 AM)Goodies Wrote: [\[CITE\]](#) That did not work. Arguable, it made it worse. I cannot even call a simple function from a new thread. Even without running the hook at all, it still gives an error when creating the thread.because the runtime library is not initialized properly

**Goodies** - May 4, 2016, 2:26 pm

(May 3, 2016 05:55 PM)excr4sh Wrote: [\[CITE\]](#) because the runtime library is not initialized properly That's not it at all. The solution is really very easy... I can't believe no one spotted it because apparently it's a very common issue.

The DllMain function needs to return TRUE, otherwise it just gets unloaded immediately. All I had to do was return true and it worked...