

Forum:[Hack Forums](#)

Thread: [x86 and x64 API hook \(C++\)](#)

AmericanExpress - June 20, 2017, 6:36 pm

Hello!

I have to post in this section due to my prestige.. :(

I found some old source code from when I was getting into API hooking from a few years ago and since I have no uses for it I decided I'd share it here for free to help anyone interested in it!

I actually sold the code to someone for them to put into a project of their own, an e-book or whatever they were doing, however we both have full rights to do the code and they don't mind if I share it either! So it is all good! ?

The code below will place a hook on NtOpenProcess once the DLL is injected into the process... It will use IsWow64Process to check if the process it is hooking is for x86 or x64 to decide on the hook function to use and what trampoline to use.

stdafx.h:

```
Code // stdafx.h : include file for standard system include files,  
// or project specific include files that are used frequently, but  
// are changed infrequently  
//
```

```
#pragma once
```

```
#include "targetver.h"
```

```
#define WIN32_LEAN_AND_MEAN           // Exclude rarely-used stuff from Windows headers  
// Windows Header Files:  
#include <windows.h>  
#include <winternl.h>
```

```
// TODO: reference additional headers your program requires here
```

```
targetver.h:
```

```
Code #pragma once
```

```
// Including SDKDDKVer.h defines the highest available Windows platform.
```

```
// If you wish to build your application for a previous Windows platform, include WinSDKVer.h and  
// set the _WIN32_WINNT macro to the platform you wish to support before including  
SDKDDKVer.h.
```

```
#include <SDKDDKVer.h>
```

```
dllmain.cpp:
```

```
Code #include "stdafx.h"
```

```
#pragma pack (1)
```

```

struct addrinf {
    unsigned char bytecode;
    ULONG uAddr;
};

typedef struct _CLIENT_ID {
    PVOID UniqueProcess;
    PVOID UniqueThread;
} CLIENT_ID, *PCLIENT_ID;

NTSTATUS NTAPI NtOpenProcessHook(PHANDLE ProcessHandle, ACCESS_MASK
DesiredAccess, POBJECT_ATTRIBUTES ObjectAttributes, PCLIENT_ID ClientId);

typedef NTSTATUS(NTAPI *func_NtOpenProcess)(PHANDLE ProcessHandle, ACCESS_MASK
DesiredAccess, POBJECT_ATTRIBUTES ObjectAttributes, PCLIENT_ID ClientId);

struct hfunstarget64 { UINT_PTR callbackAddress; int intPCount; LPVOID lpTrAddr; LPVOID
lpTAddr; };
struct hfunstarget32 { ULONG callbackAddress; int intPCount; LPVOID lpTrAddr; };

unsigned char byteArray64[] = { 0x48,0xB8,0x90,0x90,0x90,0x90,0x90,0x90,0x90,0x90,0x90,0xFF,0xE0
};
unsigned char byteArray642[] = {
0x48,0xBB,0x90,0x90,0x90,0x90,0x90,0x90,0x90,0x90,0xFF,0xE3 };

hfunstarget64 hfunctarget64[1] = {
    { (UINT_PTR)NtOpenProcessHook, 21 }
};

hfunstarget32 hfunctarget32[1] = {
    { (ULONG)NtOpenProcessHook, 5 }
};

bool is64 = false;

NTSTATUS NTAPI NtOpenProcessHook(PHANDLE ProcessHandle, ACCESS_MASK
DesiredAccess, POBJECT_ATTRIBUTES ObjectAttributes, PCLIENT_ID ClientId)
{
    HANDLE targetPid = ClientId->UniqueProcess;
    if (targetPid != 0)
    {
        if (targetPid == (HANDLE)5000)
        {
            return 0xC000000B;
        }
    }
    if (is64)
    {
        return ((func_NtOpenProcess)hfunctarget64[0].lpTrAddr)(ProcessHandle, DesiredAccess,
ObjectAttributes, ClientId);
    }
    else {
        return ((func_NtOpenProcess)hfunctarget32[0].lpTrAddr)(ProcessHandle, DesiredAccess,

```

```

ObjectAttributes, ClientId);
    }
}

bool dAPI_64(BYTE *lpTargetAddress, int hookId) {
    hfunstarget64 structInfo = hfunctarget64[hookId];
    DWORD dwPreserve = (structInfo.intPCount + 12;
    DWORD dwMemProtection = 0;
    if (!VirtualProtect(lpTargetAddress, dwPreserve, PAGE_EXECUTE_READWRITE,
&dwMemProtection))
    {
        return false;
    }
    structInfo.lpTrAddr = VirtualAlloc(NULL, dwPreserve, MEM_RESERVE | MEM_COMMIT,
PAGE_EXECUTE_READWRITE);
    memcpy(structInfo.lpTrAddr, lpTargetAddress, structInfo.intPCount);
    *(UINT_PTR*)(byteArray642 + 2) = (UINT_PTR)(lpTargetAddress + structInfo.intPCount);
    memcpy((PBYTE)structInfo.lpTrAddr + structInfo.intPCount, (PBYTE)&byteArray642, 12);
    *(UINT_PTR*)(byteArray64 + 2) = (UINT_PTR)(structInfo.callbackAddress);
    memcpy(lpTargetAddress, (PBYTE)&byteArray64, 12);
    hfunctarget64[hookId] = structInfo;
    VirtualProtect(lpTargetAddress, dwPreserve, dwMemProtection, &dwMemProtection);
    return true;
}

bool dAPI_32(ULONG lpTargetAddress, int hookId)
{
    addrinf *pAddrInf;
    DWORD dwOldRights = PAGE_EXECUTE_READWRITE;
    hfunstarget32 *structInfo = &hfunctarget32[hookId];
    DWORD nPreseve = structInfo->intPCount + 0x00000005;
    if (VirtualProtect((PVOID)lpTargetAddress, nPreseve, dwOldRights, &dwOldRights)) {
        structInfo->lpTrAddr = VirtualAlloc(NULL, nPreseve, MEM_RESERVE | MEM_COMMIT,
PAGE_EXECUTE_READWRITE);
        memcpy(structInfo->lpTrAddr, (PVOID)lpTargetAddress, structInfo->intPCount);
        pAddrInf = (addrinf *)lpTargetAddress;
        pAddrInf->bytecode = 0xE9;
        pAddrInf->uAddr = (structInfo->callbackAddress - (lpTargetAddress + 0x00000005));
        pAddrInf = (addrinf *)((ULONG)structInfo->lpTrAddr + (ULONG)structInfo->intPCount);
        pAddrInf->bytecode = 0xE9;
        pAddrInf->uAddr = ((ULONG)lpTargetAddress) - ((ULONG)structInfo->lpTrAddr +
0x00000005);
        return true;
    }
    return false;
}

bool checkprocess_64(HANDLE ProcessHandle)
{
    typedef BOOL(APIENTRY *pdef_IsWow64Process)(HANDLE, PBOOL);
    pdef_IsWow64Process IsWow64Proc;
    BOOL wow64check = false;
    IsWow64Proc = (pdef_IsWow64Process)GetProcAddress(LoadLibraryA("kernel32.dll"),

```

```

"IsWow64Process");
    if (IsWow64Proc != NULL)
    {
        if (IsWow64Proc(ProcessHandle, &wow64check))
        {
            return wow64check;
        }
    }
    return wow64check;
}

void main_setup()
{
    if (checkprocess_64((HANDLE)-1))
    {
        is64 = false;
        dAPI_32((ULONG)GetProcAddress(LoadLibraryA("ntdll.dll"), "NtOpenProcess"), 0);
    }
    else
    {
        is64 = true;
        dAPI_64((PBYTE)GetProcAddress(LoadLibraryA("ntdll.dll"), "NtOpenProcess"), 0);
    }
}

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

```

Iterator - June 23, 2017, 4:46 am

If IsWow64Process returns false that could mean that the process is 32bit running on 32bit OS or 64bit running on 64bit OS, so your code wouldn't work on 32bit systems.

AmericanExpress - June 23, 2017, 5:19 am

(Jun 23, 2017 04:46 AM)Iterator Wrote: [\[CITE\]](#) If IsWow64Process returns false that could mean that the process is 32bit running on 32bit OS or 64bit running on 64bit OS, so your code wouldn't work on 32bit systems. Yeah, I was developing on a 64-bit system. I apologize for that. Although the fix would be to check the OS is 32-bit or 64-bit and if it is 32-bit then just use the x86 24/7 and else use the IsWow64Process method. ?

Thanks for pointing that out!!