Windows Hook

Jerald Lee
Contact Me : lucid7@paran.com

본 문서는 저자가 Windows Hook을 공부하면서 알게 된 것들을 정리할 목적으로 작성되었습니다. 본인이 Windows System에 대해 아는 것의 거의 없기 때문에 기존에 존재하는 문서들을 짜집기 한 형태의 문서로밖에 만들 수가 없었습니다. 문서를 만들면서 참고한 책, 관련 문서 등이 너무 많아 일일이 다 기술하지 못한 점에 대해 원문 저자들에게 매우 죄송스럽게 생각합니다.
본 문서의 대상은 운영체제와 Win32 API를 어느 정도 알고 있다는 가정하에 쓰여졌습니다. 본 문서에 기술된 일부 기법에 대한 자세한 설명은 책을 참조하시길 바랍니다.

제시된 코드들은 Windows XP Service Pack2, Visual Studio .net 2003에서 테스트 되었습니다. 문서의 내용 중 틀린 곳이나 수정해야 할 부분이 있으면 연락해 주시기 바랍니다.

목 차

1. 사용되는 기법들

필요한 기초 지식은 앞에서 다 다루었다. 깊이가 부족하긴 하지만 그럭저럭 이해하는 데는 별 어려움이 없으리라 본다.
Windows Hook 또는 API Hook을 위해 사용되는 기법은 매우 다양하며 그 논의는 매우 오래전부터 있어왔다. 본 문서에서는 현재까지 제시된 대부분의 기법을 다루어 보기로 한다.

1.1. Message Hook (Sub-Classing)

Windows는 메시지 기반 운영체제이다. 메시지는 큐에 저장된 형태로 차례차례로 어플리케이션에게 전달되는데 Hook 프로시저를 만들어 여기에서 메시지를 먼저 전달 받은 다음 어플리케이션으로 다시 메시지를 보내는 것이 핵심이다.
이를 서브 클래싱이라고 하는데 후킹으로 분류하기에는 좀 모호한 감이 없지 않아 있지만 누구나 다 만든다는 키보드 후킹 프로그램이 거의 이 방법으로 만들어지기 때문에 본 문서에서도 한번 제작을 해 보게 되었다.

Windows 메시지 후킹의 경우 API 레벨에서 지원하고 있으며 함수는 아래와 같다.

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

첫 번째 변수 idHook은 설치하고자 하는 훅의 타입을 지정하며 WH_로 시작되는 매크로 상수 중 하나를 써주면 된다.
lpfn은 가로 챈 메시지를 넘겨받을 프로시저의 주소를 지정하며 hMod는 lpfn이 들어있는 Dll의 주소를 나타낸다.
dwThreadId는 훅 프로시저가 감시할 스레드의 ID를 나타내며(만약 스레드 단위로 훅을 하고 싶다면) 이 값이 0일 경우 특정 스레드가 아닌 시스템의 모든 스레드에서 발생하는 메시지를 훅할 수 있다. 자신의 메시지를 넘겨주고 싶을 때는 GetCurrentThreadId 함수로 현재 스레드의 ID를 넘겨주면 된다. 시스템의 모든 메시지를 감시하고자 한다거나 다른 프로그램의 메시지를 감시하고자 할 경우 lpfn은 반드시 분리된 DLL에 있어야 하며 이때 hMod는 이 DLL의 핸들이어야 한다.

```
지역 훅 : SetWindowsHookEx(idHook, lpfn, NULL, GetCurrentThreadId());
```

```
전역 훅 : SetWindowsHookEx(idHook, lpfn, hDll, 0);
```

SetWindowsHookEx는 HHOOK 타입의 훅 핸들을 반환하며 에러 발생 시 NULL을 반환한다.

```
BOOL UnhookWindowsHookEx( HHOOK hhk );
```

해제하고자 하는 훅 핸들을 넘겨주기만 하면 된다. 훅을 설치한 프로그램은 종료되기 전에 반드시 훅 프로시저를 해제해 주어야 한다. 훅 프로시저가 설치되면 해당 타입의 메시지는 목표 윈도우로 보내지기 전에 훅 프로시저에게 먼저 전달되는데 훅 프로시저는 메시지를 살펴본 후 특별한 이유가 없으면 메시지를 훅 체인의 다음 훅 프로시저에게 전달해 주어야 한다. 이때는 다음 함수를 사용한다.

```
LRESULT CallNextHookEx(
    HHOOK hhk,
    int nCode,
    WPARAM wParam,
    LPARAM lParam
);
```

hhk는 현재 처리하고 있는 훅의 핸들인데 SetWindowsHookEx 함수가 리턴한 값이다. 나머지 세 인수는 운영체제가 훅 프로시저에게 전달해준 인수들이다. 훅 체인에 포함된 훅 프로시저의 목록은 운영체제가 직접 관리하기 때문에 훅을 설치한 응용 프로그램은 다음 훅 프로시저의 번지를 따로 저장할 필요없이 이 함수만 호출해 주면 훅 체인을 따라 모든 훅 프로시저가 순서대로 호출되며 최종적으로 목표 윈도우로 메시지가 전달될 것이다.

훅 프로시저는 전달 받은 메시지를 다음 훅 프로시저에게 꼭 전달해 주어야 할 의무는 없으며 메시지를 아예 없애버리려면 전달하지 않아도 상관없으며 원하는 대로 변경할 수도 있다. 물론 이 때는 메시지를 없애버리거나 변경한 후의 효과에 대해 확실히 책임질 수 있어야 한다. 이 함수는 훅 체인에서 다음 훅 프로시저를 호출하고 훅 프로시저가 리턴하는 값을 다시 리턴해 주는데 현재의 훅 프로시저는 이 리턴값을 또 그대로 리턴해 주어야 한다. 그래서 훅 프로시저의 끝은 보통 return CallNextHookEx(...) 호출문이 온다.

Windows 에서 제공하는 훅은 다음과 같다.

훅 타입	설명
WH_CALLWNDPROC	SendMessage 함수로 메시지를 보내기 전에
WH_CALLWNDPROCRET	WH_CALLWNDPROC 훅 프로시저가 호출되며 윈도우 프로시저가 메시지를 처리한 후에 WH_CALLWNDPROCRET 훅 프

	로시저가 호출된다. WH_CALLWNDPROCRET 훅은 훅 프로시저에게 CWPRETSTRUCT 구조체를 전달하는데 이 구조체에는 메시지와 메시시를 처리한 리턴값을 담고 있다
WH_CBT	윈도우를 생성, 파괴, 활성화, 최대, 최소, 이동, 크기변경하기 전에, 시스템 명령을 처리하기 전에, 마우스나 키보드 메시지를 메시지 큐에서 제거하기 전에 이 훅 프로시저가 호출된다. 이 훅은 컴퓨터를 이용한 훈련 프로그램(Computer Based Training)에서 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_JOURNPLAYBACK 훅에서 재생할 수 있다. 이 훅은 전역으로만 설치할 수 있으며 특정 스레드에만 설치할 수는 없다.
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. 이 훅은 전역으로만 설치할 수 있으며 특정 스레드에만 설치할 수는 없다.
WH_KEYBOARD	WM_KEYDOWN, WM_KEYUP 등의 키보드 메시지를 감시한다.
WH_MOUSE	마우스 메시지를 감시한다.
WH_MSGFILTER	메뉴, 스크롤 바, 메시지 박스, 대화상자 등에 의해 처리되는 메시지와 사용자의 Alt+Tab키, Alt+Esc키 입력에 의한 포커스 이동을 감시한다. 훅 프로시저를 설치한 프로그램에 대해서만 동작한다.
WH_SYSMSGFILTER	WH_MSGFILTER과 같으며 모든 프로그램에 대해 동작한다.
WH_SHELL	쉘 프로그램이 활성화되거나 새로운 최상위 윈도우가 만들어지거나 파괴될 때 이 훅 프로시저가 호출된다.
WH_FOREGROUNDIDLE	포그라운드 스레드가 한가해질 때 이 훅 프로시저가 호출된다. 아이들 시에 우선 순위가 낮은 작업을 하고 싶을 때 이 훅을 사용한다.
WH_KEYBOARD_LL	스레드의 입력 큐에 붙여지는 키보드 입력 메시지를 감시한다. WH_KEYBOARD보다 더 저수준의 메시지를 받을 수 있지만 NT 4.0 SP3 이후에만 사용할 수 있다.
WH_MOUSE_LL	스레드의 입력 큐에 붙여지는 마우스 입력 메시지를 감시한다.

훅을 설치하는 코드는 다음과 같이 작성할 수 있다.

```
HOOKPROC hHookProc;
```

```
static HINSTANCE hLoaDll;
static HHOOK hHook;

hLoadDll = LoadLibrary("Keyboard.dll");

hHookProc = (HOOKPROC) GetProcAddress(hLoadDll, "HookProc");
hHook = SetWindowsHookEx(WH_GETMESSAGE, hHookProc, hLoadDll, 0);
```

Keyboard.dll을 작성한 뒤 LoadLibrary 함수를 사용해 메모리로 불러온다. Keyboard.dll에 포함된 HookProc 함수의 엔트리를 찾은 후 SetWindowsHookEx 함수로 훅을 설치한다.

설치될 HookProc는 다음과 같이 구현될 수 있다.

```
extern "C" __declspec(dllexport) LRESULT CALLBACK HookProc (
  int nCode, WPARAM wParam, LPARAM lParam)
{
        if( nCode >= 0 )
        {
                ...키보드 메시지를 파일로 저장...
        }

        return CallNextHookEx( g_Hook , nCode , wParam , lParam );
}
```

nCode 변수에 넘어오는 값이 0보다 작으면 CallNextHookEx를 실행하고 HC_ACTION이면 전달된 메시지를 훅 프로시저(CallNextHookEx)에서 처리한다.
wParam 변수는 넘어온 메시지가 큐에서 제거되었는지를 알려주는 플래그이며 PM_NOREMOVE 또는 PM_REMOVE 가 전달된다.
lParam 변수는 후킹한 메시지의 구조체 정보가 담기게 되며 그 구조체는 아래와 같다.

```
typedef struct tagMSG {
    HWND   hwnd;        // 메시지를 가져온 윈도우의 핸들
    UINT   message;     // 메시지 번호
    WPARAM wParam;
    LPARAM lParam;      // 메시지에 따르는 파라메터들
    DWORD  time;        // 메시지가 posting된 시간
    POINT  pt;          // 메시지가 posting 되었을 때 커서 위치
```

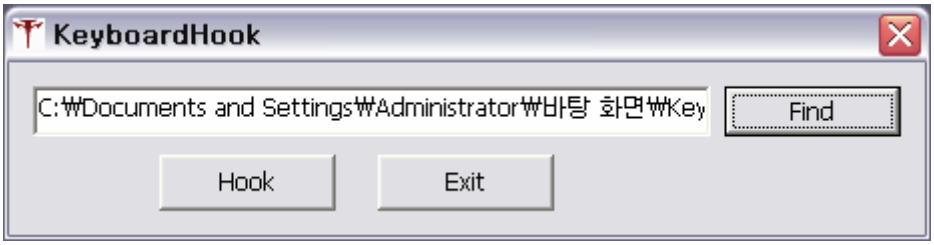
```
} MSG;
```

이제 키보드 후킹 프로그램을 제작하기 위한 것이 모두 준비되었다. 제작된 프로그램은 아래와 같으며 로드 할 dll 파일 이름은 다이얼로그 창에서 받아들이도록 제작하였다.
미리 말해두지만 해당 프로그램은 키보드 메시지 중 WM_CHAR일 경우에만(도스창에서 문자 입력할 때는 동작하지 않는다.) 동작하며 한번에 50글자 이상 입력하면 에러가 발생한다 -_-; 소스가 지저분해도 차분히 고쳐 쓰는 습관을 기르도록 하자.

이 프로그램은 이동우 님의 메시지 후킹에 공개된 소스를 바탕으로 조금 수정을 하였다.(원본 소스는 깔끔하다 ㅠ_ㅠ).
WH_GETMESSAGE로 전역 후킹을 한 후 WM_CHAR만 뽑아내어 파일에 저장하는 방식을 사용하였으며 위에도 나와 있듯이 WH_KEYBOARD 또는 WH_KEYBOARD_LL 을 이용한 방법도 있다. WH_KEYBOARD를 사용할 경우에는 전역 후킹으로 사용할 수 없으며 WH_KEYBOARD_LL을 사용한 후킹은 온라인 게임 클라이언트에서 특정 키보드를 막기 위해 많이 사용되는 방법이다.

소스 파일 : http://consult.skinfosec.co.kr/~lucid7/Document/KeyboardHook.zip
실행 파일 : http://consult.skinfosec.co.kr/~lucid7/Document/KeyboardHook-EXE.zip

소스 및 실행 파일은 위의 주소에서 받을 수 있으며 이 프로그램의 실행 모습은 아래와 같다.



참고1)
WH_KEYBOARD_LL을 사용한 키보드 후킹 예제도 아래에 올려놓았으니 분석해보면 좋을 듯하다. C#으로 작성되었으며 C++로 변환하는 데 전혀 어려움이 없다.

http://consult.skinfosec.co.kr/~lucid7/Document/Keyboardhook-C#.zip

참고2)
위의 방법을 사용해서 제작된 Keylogger는 레지스트리에 기록이 남게 되므로(정규 필터 드라이버의 사용 기록이 남는다고 한다.) Keylogger로서의 효용성이 없다. Phrack이나 Codeguru,

Codeproject 등에서 실행 파일 감추기 등등의 이름으로 여러 가지 방법이 나왔지만 결국은 레지스트리에 기록이 남아버리는 문제점이 있다. 상용 툴이나 중국애들이 맹그는 것은 이런 문제점을 피하기 위해서 다른 방법을 쓴다고 하는데 somma 님 블로그에 하나의 기법이 소개되어 있다.(소개만 되어있고 뭐 소스라든가 그런거는 없다. -__-)

somma 님 블로그 http://somma.egloos.com/1470909
(이 somma님이 예전의 그 소마 행님인지 잘 모르겠다. 소마 행님 외국에 있었는데...들어오셨나? =_=;)

1.2. 참고문서

WebSite

1. Three Ways to Inject Your Code into Another Process
   (http://www.codeproject.com/threads/winspy.asp)

2. API Hooking Revealed
   (http://www.codeproject.com/system/HookSys.asp)

3. 동우의 홈페이지
   http://dasomnetwork.com/~leedw/mywiki/moin.cgi/

Book

1. Windows 시스템 실행 파일의 구조와 원리, 이호동 저, 한빛미디어
2. API로 배우는 Windows 구조와 원리, 야스무로 히로카즈 저, 한빛미디어
3. Win32 System Programming,