

개발자를 위한 윈도우 후킹 테크닉

마우스 훅을 통한 화면 캡처 프로그램 제작하기.

지난 강좌에서 우리는 기초적인 후킹 API의 사용법과 그것들을 사용해서 키보드 모니터링을 하는 방법을 배웠다. 이번 강좌에서는 마우스 후킹을 통해서 스크린 캡처를 수행하는 프로그램을 만들어 보도록 하자. 이 과정에서 후킹 함수가 호출되는 쓰레드 컨텍스트가 어떤 의미를 가지는지 살펴보도록 하자.

목차

목차.....	1
필자 소개	1
연재 가이드.....	1
연재 순서	2
필자 메모	2
Intro	3
연장통	3
마우스 메시지	4
WH_MOUSE	6
WH_MOUSE_LL.....	7
마우스 후킹을 해보자.....	9
캡처.....	13
도전 과제	18
참고 자료	19

필자 소개

신영진 pop@jiniya.net

부산대학교 정보,컴퓨터공학부 4학년에 재학 중이다. 모자란 학점을 다 채워서 졸업하는 것이 꿈이되 버린 소박한 괴짜 프로그래머. 병역특례 기간을 포함해서 최근까지 다수의 보안 프로그램 개발에 참여했으며, 최근에는 모짜르트에 심취해 있다.

연재 가이드

운영체제: 윈도우 2000/XP

개발도구: 마이크로소프트 비주얼 스튜디오 2003

기초지식: C/C++, Win32 프로그래밍

응용분야: 화면 캡처 프로그램

연재 순서

2006. 05 키보드 모니터링 프로그램 만들기

2006. 06 마우스 훅을 통한 화면 캡처 프로그램 제작

2006. 07 메시지훅 이용한 Spy++ 훅내내기

2006. 08 SendMessage 후킹하기

2006. 09 Spy++ 클론 imSpy 제작하기

2006. 10 저널 훅을 사용한 매크로 제작

2006. 11 WH_SHELL 훅을 사용해 다른 프로세스 윈도우 서브클래싱 하기

2006. 12 WH_DEBUG 훅을 이용한 훅 탐지 방법

2007. 01 OutputDebugString 의 동작 원리

필자 메모

처음 개발자의 세계에 입문하면 누구나 빠른 시간 안에 고수가 되기를 원한다. 이러한 초심자들에게 도움이 될만한 필자가 재미를 본 한 가지 비법이 있어서 소개를 해볼까 한다. 이름하여 "우려먹기식 프로그래밍"으로, 한 가지 기술을 사용해서 닥치는 대로 프로그램을 짜보는 것이다. 이 방식의 기원은 다음과 같다.

필자가 고등학교 시절엔 게임 프로그래밍에 많은 흥미를 가지고 있었다. 그 때 한창 VGA 의 13h 모드인 300x200 화면 그래픽에 심취해있었다. 메모리 한 바이트와 화면의 한 픽셀이 일대일로 맵핑되는 단순함에 매료되어 근 세달 동안 그 놀만을 잡고 이것저것 만들고 있었다. 그걸 본 필자의 친구가 그랬다. "그만 좀 우려먹어라."

이 방법은 여러 가지 장점을 가지고 있다. 자신이 배운 범위 내에서 이것 저것 해봄으로써 자신이 배운 기술에 대해서 깊이 있게 이해할 수 있게 해준다. 또한 이 과정에서 배운 것을 조금씩 변형해 가면서 새로운 것을 만들기 때문에 몰라서 물어보는데 많은 시간을 보내지 않아도 되고, 코딩 량이 부족한 초보 개발자들에게 많은 코딩 량을 선사한다는 장점이 있다. 프로그래밍 언어도 실제 언어와 다를 바가 없어서 많이 사용해 볼수록 표현력이 늘고, 새로운 문제를 표현하는 방법도 다양해진다. 이번 달 강좌를 보고 여러분이 만약 마우스 후킹을 사용하는 프로그램을 천 개 정도 작성해 본다면, 그 후에는 마우스 후킹에 있어서는 누구보다도 높은 수준의 이해력과 통찰력을 가지게 될 것이란 사실을 명심하자.

Intro

잇을 만 하면 한 통씩 필자의 메일 함으로 후킹 관련 질문들이 들어오곤 한다. 그 중 팔 할은 훅 프로시저의 실행 컨텍스트에 관한 문제들이다. 사실 컨텍스트란 개념이 어려운 것이 아니다. 누구나 들을 땐 쉽게 당연히 그렇지 하고 지나가는 것들이 실제로 적용할 때에는 문제가 되는 것이다. 우리는 이번 강좌에서 WH_MOUSE 훅과 WH_MOUSE_LL 훅이라는 서로 다른 컨텍스트에서 실행되는 두 가지 종류의 훅을 수행해 볼 것이다. 이 둘을 통해서 훅 프로시저의 호출 컨텍스트가 어떤 의미를 가지는 지 그리고 결과적으로 어떤 차이를 가지는 지를 이해하도록 하자.

연장통

분야별로 약방에 감초처럼 자주 사용되는 API 들이 있다. 후킹을 하면서 자주 사용하게 되는 API 를 몇 가지 살펴보도록 하자. API 가 단순히 사용법만 보고 기능만 안다고 자신의 것이 되진 않는다. 실제로 코드에서 사용해보고 리턴 값의 의미를 분석해 볼 때에 자신의 것으로 체득된다는 것을 명심하자. 모두 간단한 API 들이기 때문에 한번씩 사용해 보고 결과 값을 관찰해 보도록 하자.

```
HANDLE GetCurrentThread(void);  
DWORD GetCurrentThreadId(void);  
HANDLE GetCurrentProcess(void);  
DWORD GetCurrentProcessId(void);
```

위 네 가지 함수는 현재의 실행 컨텍스트를 구하는 역할을 한다. 위에서부터 현재의 실행 스레드 핸들, 현재 실행 스레드 아이디, 현재 실행 프로세스 핸들, 현재 실행 프로세스 아이디를 반환하는 역할을 한다. 이를 통해서 우리는 후킹 함수가 동작하는 시점이 어느 프로세스의 어느 스레드 컨텍스트 인지를 알 수 있다.

```
DWORD GetWindowThreadProcessId(HWND hWnd, LPDWORD lpdwProcessId);
```

위 함수는 특정 윈도우가 동작하는 스레드와 프로세스의 아이디를 구하는 함수다. hWnd 로 프로세스와 스레드 아이디를 구할 윈도우 핸들을 넣어주면 lpProcessId 로는 프로세스 아이디를, 리턴 값으로 스레드 아이디를 넘겨준다. lpProcessId 가 필요 없을 경우 NULL 로 지정하면 된다.

```
BOOL DuplicateHandle(  
    HANDLE hSourceProcessHandle,  
    HANDLE hSourceHandle,  
    HANDLE hTargetProcessHandle,  
    LPHANDLE lpTargetHandle,
```

```
DWORD dwDesiredAccess,  
BOOL bInheritHandle,  
DWORD dwOptions  
);
```

위 함수는 핸들을 복사 시키는 역할을 한다. Win32 에서는 기본적으로 핸들이 프로세스를 넘어서 공유되지 않는다. 핸들을 공유하기 위해서는 위의 함수를 사용해서 핸들을 복사 시켜야 한다. 첫 번째 인자로 원본 프로세스 핸들을, 두 번째 인자로 복사할 원본 핸들을, 세 번째 인자로 대상 프로세스 핸들을, 네 번째 인자로 복사된 핸들을 리턴받을 포인터를 넣어주면 된다. 일반적으로 다음 인자들로는 0, FALSE, DUPLICATE_SAME_ACCESS 를 전달해 주면 된다. 이렇게 할 경우 동일한 권한을 가지도록 핸들이 복사된다. 보다 자세한 옵션은 MSDN 을 참고하도록 하자.

마우스 메시지

마우스는 Windows 의 필수 입력 장치중의 하나이다. 그만큼 마우스와 관련된 메시지도 많다. 마우스 후킹에 앞서서 어떤 마우스 메시지가 있는지 한번 살펴보도록 하자. <표 1>에 Windows 에서 지원하는 모든 마우스 메시지 목록이 나와있다.

표 1 마우스 메시지

메시지	발생 상황
WM_LBUTTONDOWNBLCLK	왼쪽 버튼을 더블 클릭했을 때 발생함.
WM_LBUTTONDOWN	왼쪽 버튼을 눌렀을 때 발생함.
WM_LBUTTONUP	왼쪽 버튼을 뗐을 때 발생함.
WM_MBUTTONDOWNBLCLK	가운데 버튼을 더블 클릭했을 때 발생함.
WM_MBUTTONDOWN	가운데 버튼을 눌렀을 때 발생함.
WM_MBUTTONUP	가운데 버튼을 뗐을 때 발생함.
WM_RBUTTONDOWNBLCLK	오른쪽 버튼을 더블 클릭했을 때 발생함.
WM_RBUTTONDOWN	오른쪽 버튼을 눌렀을 때 발생함.
WM_RBUTTONUP	오른쪽 버튼을 뗐을 때.
WM_XBUTTONDOWNBLCLK	추가 버튼을 더블 클릭했을 때.
WM_XBUTTONDOWN	추가 버튼을 눌렀을 때.
WM_XBUTTONUP	추가 버튼을 뗐을 때.
WM_NCLBUTTONDOWNBLCLK	클라이언트 영역이 아닌 곳에서 왼쪽 버튼을 더블 클릭했을 때.
WM_NCLBUTTONDOWN	클라이언트 영역이 아닌 곳에서 왼쪽 버튼을 눌렀을 때.
WM_NCLBUTTONUP	클라이언트 영역이 아닌 곳에서 왼쪽 버튼을 뗐을 때.

WM_NCRBUTTONDBLCLK	클라이언트 영역이 아닌 곳에서 오른쪽 버튼을 더블 클릭했을 때.
WM_NCRBUTTONDOWN	클라이언트 영역이 아닌 곳에서 오른쪽 버튼을 눌렀을 때.
WM_NCRBUTTONUP	클라이언트 영역이 아닌 곳에서 오른쪽 버튼을 뗐을 때.
WM_NCMBUTTONDBLCLK	클라이언트 영역이 아닌 곳에서 가운데 버튼을 더블 클릭했을 때.
WM_NCMBUTTONDOWN	클라이언트 영역이 아닌 곳에서 가운데 버튼을 눌렀을 때.
WM_NCMBUTTONUP	클라이언트 영역이 아닌 곳에서 가운데 버튼을 뗐을 때.
WM_NCXBUTTONDBLCLK	클라이언트 영역이 아닌 곳에서 추가 버튼을 더블 클릭했을 때.
WM_NCXBUTTONDOWN	클라이언트 영역이 아닌 곳에서 추가 버튼을 눌렀을 때.
WM_NCXBUTTONUP	클라이언트 영역이 아닌 곳에서 추가 버튼을 뗐을 때.
WM_MOUSEHOVER	마우스가 특정 윈도우 영역 안으로 들어올 때.
WM_MOUSELEAVE	마우스가 특정 윈도우 영역 밖으로 빠져나갈 때.
WM_MOUSEMOVE	마우스가 움직일 때.
WM_MOUSEWHEEL	마우스의 휠이 돌아갈 때.
WM_MOUSEACTIVATE	마우스를 통해서 응용 프로그램을 활성화 시킬 때.



그림 1 마우스의 X 버튼

<표 1>을 보면 알겠지만 마우스 메시지 이름은 규칙적으로 지어져 있다. 마우스 버튼의 종류를 나타내는 데에는 L, R, M, X 기호가 사용된다. L은 왼쪽 버튼을, R은 오른쪽 버튼을, M은 가운데 버튼을, X는 마이크로소프트 마우스 등에 왼쪽에 달린 버튼으로 <그림 1>에 그 위치가 나와있다. 위에서부터 XBUTTON1, XBUTTON2가 된다.

NC는 클라이언트 영역이 아닌 곳에서 메시지가 발생 했음을 알려준다. 클라이언트 영역이 아닌 곳이란 윈도우의 캡션이나 사이즈 조절 부위 등을 의미한다. WM_MOUSEHOVER와 WM_MOUSELEAVE는 롤오버 버튼을 구현할 때에 많이 사용된다. 특정 윈도우 영역 안으로 마우스가 들어올 때 HOVER 메시지가 발생하며, 나갈 때 LEAVE 메시지가 발생한다. 이 메시지는 기본적으로는 통지 되지 않으며 TrackMouseEvent 함수를 통해서 통지를 요청한 윈도우에만 포스팅 된다.

WH_MOUSE

WH_MOUSE 혹은 실행 프로그램이 메시지 큐에서 마우스 관련 메시지를 읽어가는 경우에 발생한다. 따라서 이 후크 코드는 마우스 메시지가 발생한 윈도우의 쓰레드 컨텍스트에서 실행된다.

```
LRESULT CALLBACK MouseProc(int nCode, WPARAM wParam, LPARAM lParam);
```

nCode – [입력] code 값이 0 보다 작은 경우는 후크 프로시저를 수행하지 않고 바로 CallNextHookEx 를 호출한 다음 리턴해야 한다. 이 값의 의미는 <표 2>를 참고하자.

표 2 nCode 값의 의미

코드값	의미
HC_ACTION	wParam 과 lParam 이 마우스 정보를 담고 있다.
HC_NOREMOVE	wParam 과 lParam 이 마우스 정보를 담고 있다. 이 값은 메시지 큐에서 메시지가 제거되지 않을 때 설정된다(PeekMessage 의 PM_NOREMOVE 플래그가 설정된 경우다).

wParam – [입력] 발생한 마우스 메시지 ID(WM_LBUTTONDOWN, WM_LBUTTONUP, ...).

lParam – [입력] MOUSEHOOKSTRUCT 구조체 포인터. MOUSEHOOKSTRUCT 구조체는 아래와 같이 정의되어 있으며. 필드별 의미는 <표 3>을 참고하자.

```
typedef struct {  
    POINT pt;
```

개발자를 위한 윈도우 후킹 테크닉: 마우스 훅을 통한 화면 캡처 프로그램 제작하기.

```
HWND hwnd;  
UINT wHitTestCode;  
ULONG_PTR dwExtraInfo;  
} MOUSEHOOKSTRUCT, *PMOUSEHOOKSTRUCT;
```

표 3 MOUSEHOOKSTRUCT 구조체 필드별 의미

필드명	의미
pt	메시지가 발생한 마우스 좌표.
hwnd	메시지가 발생한 윈도우.
wHitTestCode	히트 테스트 코드.
dwExtraInfo	메시지와 관련된 추가적인 정보.

WH_MOUSE_LL

WH_MOUSE_LL 혹은 WH_MOUSE 와 동일하게 마우스 훅을 수행한다. 하지만 동작 방식에 있어서는 WH_MOUSE 훅과 몇 가지 다른 점이 존재한다. WH_MOUSE 와 WH_MOUSE_LL 의 차이점은 <표 4>를 참고하자 (WH_KEYBOARD 와 WH_KEYBOARD_LL 에도 동일하게 적용된다).

표 4 WH_MOUSE 훅과 WH_MOUSE_LL 훅의 차이점

항목	WH_MOUSE	WH_MOUSE_LL
호출 시점	마우스 메시지가 메시지 큐에서 제거되거나 참조될 때.	마우스 메시지가 메시지 큐에 포스팅 될 때.
호출 컨텍스트	마우스 메시지가 발생한 윈도우의 스레드 컨텍스트.	훅을 수행한 스레드의 컨텍스트.
발생 윈도우 구분	어느 윈도우에서 메시지가 발생했는지 구분 가능함.	어느 윈도우에서 메시지가 발생했는지 알 수 없음.
실제 이벤트와 가상 이벤트 구분	구분 불가.	구분 가능(드라이버에서 발생한 마우스 이벤트와 mouse_event 등의 함수를 통해서 발생한 이벤트가 구분 가능함)
후킹 방식	지역 또는 전역.	전역만 가능함.

<표 4>의 내용중에 가장 주의해서 보아야 할 점은 호출 컨텍스트 부분이다.

WH_MOUSE_LL 의 경우 훅을 수행한 스레드 컨텍스트에서 호출되기 때문에 훅 프로시저가

DLL 에 존재할 필요도, DLL 에 존재한다고 하더라도 쓰레드 컨텍스트 사이에서 데이터를 공유하기 위해서 별도의 방법을 사용할 필요가 없다.

위의 사실이 장점만 가지는 것은 아니다. 시스템에서 쓰레드 컨텍스트를 전환 시키기 위해서 메시지를 사용한다. 그래서 WH_MOUSE_LL 을 사용할 경우 훅을 수행한 쓰레드가 반드시 메시지 루프를 가지고 있어야 한다. 그렇지 않을 경우 엉뚱한 동작을 할 수 있다. 실제로 지난 시간에 만든 콘솔용 hkldr 을 사용해서 WH_MOUSE_LL 을 테스트 할 경우 마우스가 이상하게 움직이는 현상을 발견할 수 있다. 아래는 WH_MOUSE_LL 훅 프로시저 함수의 원형이다.

```
LRESULT CALLBACK LowLevelMouseProc(int nCode, WPARAM wParam, LPARAM lParam);
```

nCode - [입력] nCode 값이 0 보다 작은 경우는 훅 프로시저를 수행하지 않고 바로 CallNextHookEx 를 호출한 다음 리턴해야 한다. 그렇지 않을 경우 이 값은 HC_ACTION 을 가진다. 이 경우에 wParam 과 lParam 에는 마우스 메시지와 관련된 정보가 들어 있다.

wParam - [입력] 발생한 마우스 메시지 ID. WH_MOUSE_LL 훅의 경우 NC 계열 메시지와 DBLCLK 메시지는 받지 못한다. 히트테스트가 수행되기 전이므로 NC 계열 메시지의 경우 그냥 일반 메시지로 넘어온다.

lParam - [입력] MSLHOOKSTRUCT 구조체 포인터. MSLHOOKSTRUCT 구조체는 아래와 같이 정의되어 있다. 각 필드가 의미하는 바는 <표 5>를 참고하자.

```
typedef struct {
    POINT pt;
    DWORD mouseData;
    DWORD flags;
    DWORD time;
    ULONG_PTR dwExtraInfo;
} MSLHOOKSTRUCT, *PMSLHOOKSTRUCT;
```

표 5 MSLHOOKSTRUCT 필드별 의미

필드명	의미
pt	마우스 메시지가 발생한 좌표 (이 좌표는 스크린 기준이다).
mouseData	마우스 메시지가 WM_MOUSEHWHEEL 인 경우엔, 상위 워드는 휠의 이동 값이고 하위 워드는 사용되지 않는다. 메시지가 XBUTTONDOWN 과 관련된 경우엔, 상위 워드는 XBUTTONDOWN 상태(XBUTTONDOWN1, XBUTTONDOWN2 를 사용해서 어떤 종류의 XBUTTONDOWN 이 눌러 졌는지 체크할 수 있다)가 하위 워드는 사용되지 않는다. 그 외의 메시지의 경우엔 이 필드 값이 사용되지 않는다.

flags	이벤트가 mouse_event 등의 함수로 생성된 것인지 구분하는 역할을 한다. LLMHF_INJECTED 플래그가 설정될 경우 mouse_event 등으로 메시지가 생성되었음을 나타낸다.
time	마우스 메시지가 발생한 시점의 타임스탬프 값.
dwExtraInfo	마우스 메시지와 관련된 추가적인 정보.

마우스 후킹을 해보자.

기본적인 후킹 DLL의 골격은 지난 시간에 살펴본 키보드 후킹과 다른 점이 없기 때문에 중요한 부분만 살펴보도록 하자. <리스트 1>에는 마우스 메시지를 출력하기 위해서 정의한 구조체와 배열이 나타나 있다. WM_MOUSEMOVE의 경우 불필요하게 많이 발생하는 경향이 있기 때문에 출력에서 제외시켰다.

리스트 1 mousemsg.h

```
#ifndef MOUSEMSG_H
#define MOUSEMSG_H

typedef struct _MOUSEMSGDESC
{
    UINT    id;
    LPCTSTR name;
} MOUSEMSGDESC, *PMOUSEMSGDESC;

MOUSEMSGDESC g_msgDesc[] =
{
    {WM_LBUTTONDOWN, "WM_LBUTTONDOWN"},
    {WM_RBUTTONDOWN, "WM_RBUTTONDOWN"},
    {WM_XBUTTONDOWN, "WM_XBUTTONDOWN"},
    {WM_MBUTTONDOWN, "WM_MBUTTONDOWN"},

    // ... 중략 ...

    /*{WM_MOUSEMOVE, "WM_MOUSEMOVE"},*/
    {WM_MOUSEACTIVATE, "WM_MOUSEACTIVATE"},
    {WM_MOUSEWHEEL, "WM_MOUSEWHEEL"}
};

static const int
DESCSIZE = sizeof(g_msgDesc)/sizeof(g_msgDesc[0]);

#endif
```

WM_MOUSEMOVE 훅의 코드는 <리스트 2>에 나와있다. 발생한 마우스 메시지 종류와 컨텍스트 정보, 그리고 넘어온 구조체의 윈도우와 히트테스트 코드를 출력하도록 해 두었다. 실행을

개발자를 위한 윈도우 후킹 테크닉: 마우스 혹은 통한 화면 캡처 프로그램 제작하기.

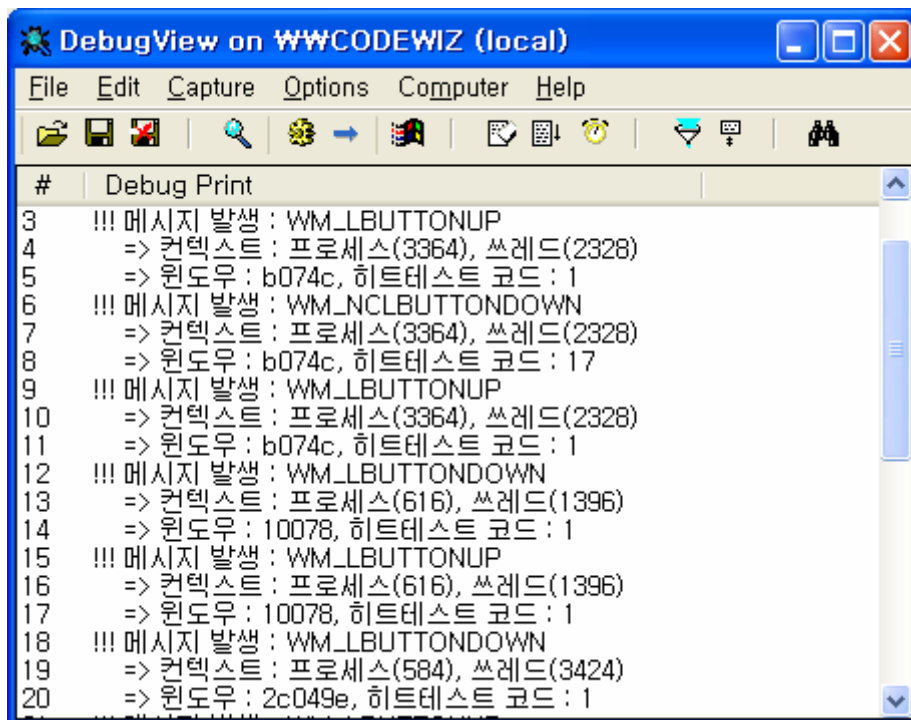
해서 디버그 뷰로 출력을 캡처해 보면 컨텍스트가 계속 변화되는 것을 볼 수 있다(<화면 1> 참고).

리스트 2 WH_MOUSE 후킹 프로시저

```
LRESULT CALLBACK
MouseProc(int code, WPARAM w, LPARAM l)
{
    if(code >= 0)
    {
        PMOUSEHOOKSTRUCT mhs = (PMOUSEHOOKSTRUCT) l;

        for(int i=0; i<DESCSIZE; ++i)
        {
            if(w == g_msgDesc[i].id)
            {
                XTRACE("!!! 메시지 발생 : %s", g_msgDesc[i].name);
                XTRACE("    => 컨텍스트 : 프로세스(%d), 쓰레드(%d)", GetCurrentProcessId(),
                GetCurrentThreadId());
                XTRACE("    => 윈도우 : %x, 히트테스트 코드 : %d\n", mhs->hwnd, mhs->wHitTestCode);
            }
        }

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



화면 1 WH_MOUSE 훅 실행 화면

WH_MOUSE_LL 훅의 코드는 <리스트 3>에 나타나있다. 기본 적으로 동일한 정보를 출력한다. 구조체에서는 인젝트된 메시지인지 실제 하드웨어에서 발생한 메시지인지를 구분해서 출력하도록 해 두었다. 실행해 보면 WH_MOUSE 메시지와 달리 서로 다른 프로세스에서 발생하는 메시지도 모두 동일한 스레드 컨텍스트에서 호출되는 것을 알 수 있다(<화면 2> 참고).

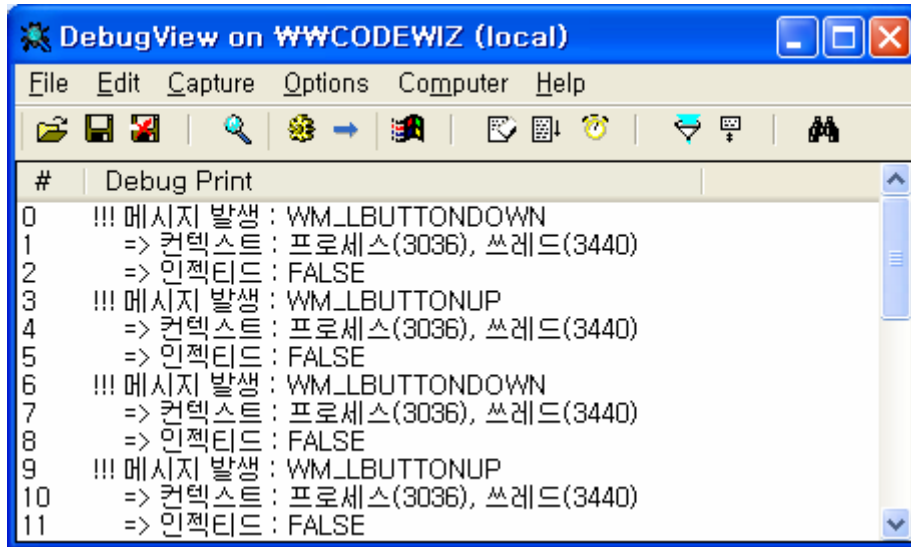
리스트 3 WH_MOUSE_LL 훅 프로시저

```
LRESULT CALLBACK
MouseLLProc(int code, WPARAM w, LPARAM l)
{
    if(code >= 0)
    {
        PMSLLHOOKSTRUCT hs = (PMSLLHOOKSTRUCT) l;

        for(int i=0; i<DESCSIZE; ++i)
        {
            if(w == g_msgDesc[i].id)
            {
                XTRACE("!!! 메시지 발생 : %s", g_msgDesc[i].name);
                XTRACE("    => 컨텍스트 : 프로세스(%d), 스레드(%d)", GetCurrentProcessId(),
                GetCurrentThreadId());
                XTRACE("    => 인젝티드 : %s\n", (hs->flags & LLMHF_INJECTED) ? "TRUE" :
                "FALSE");
            }
        }

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

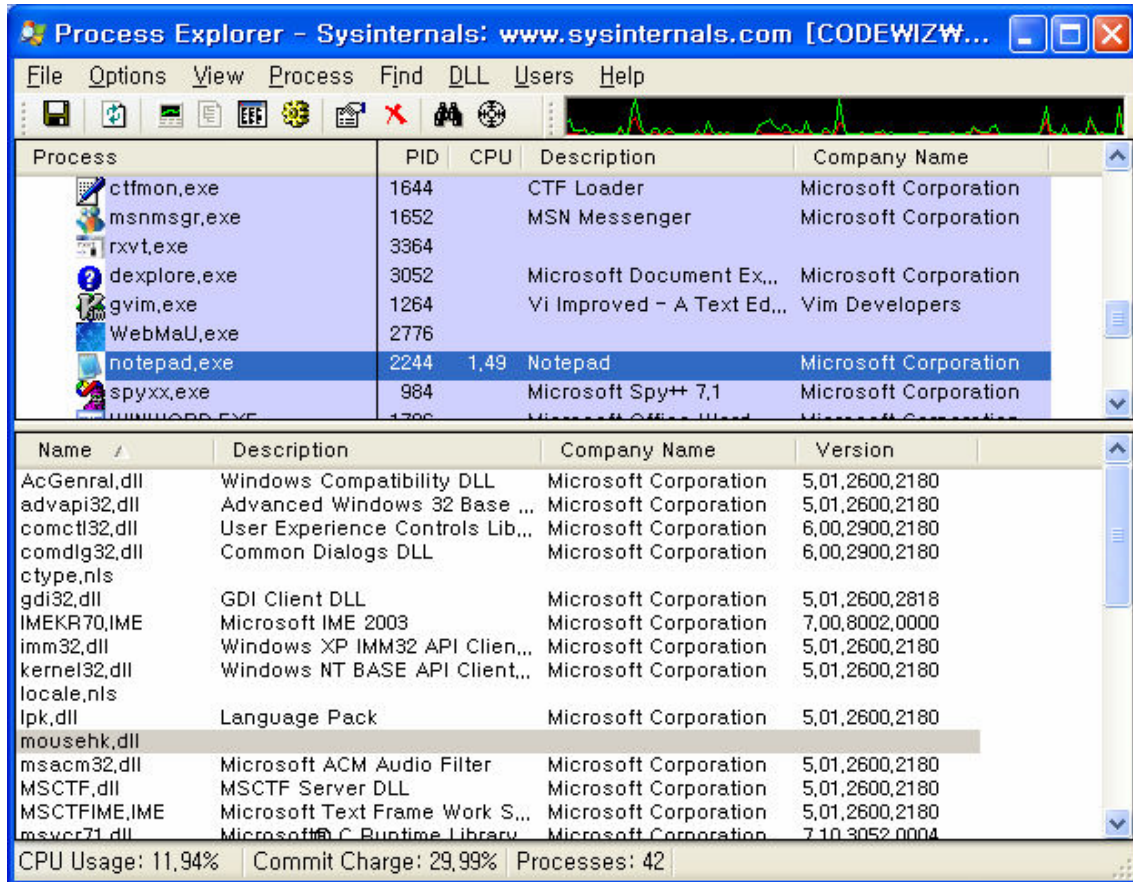
개발자를 위한 윈도우 후킹 테크닉: 마우스 훅을 통한 화면 캡처 프로그램 제작하기.



화면 2 WH_MOUSE_LL 훅 실행 화면

끝으로 WH_MOUSE_LL 훅의 실행 컨텍스트가 가지는 장점을 하나 더 살펴보도록 하자. 일반적으로 메시지 훅의 경우 다른 프로세스의 쓰레드를 후킹하는 경우 DLL 이 해당 프로세스로 인젝트된다. <화면 3>을 살펴보면 notepad.exe 의 실행 영역에 mousehk.dll 이 인젝트 된 것을 확인할 수 있다. 하지만 WH_MOUSE_LL 의 경우 자신의 컨텍스트에서 실행되기 때문에 별도로 DLL 이 다른 프로세스로 인젝트되지 않는다.

개발자를 위한 윈도우 후킹 테크닉: 마우스 클릭 통한 화면 캡처 프로그램 제작하기.



화면 3 프로세스 익스플로러를 통해서 인젝트된 DLL 을 확인하는 화면

캡처

윈도우를 캡처하는 작업은 간단하다. 캡처할 대상 윈도우의 DC 를 구한 후 해당 DC 의 영역을 생성된 비트맵으로 복사하면 모든 일이 완료된다. <리스트 4>에 특정 윈도우의 화면을 캡처하는 함수의 소스가 나와있다. GetBoundsRect 는 특정 DC 에서 화면에서 표시되는 영역을 구한다. 윈도우의 경우 스크린 밖으로 나가는 경우도 있기 때문에, 스크린에 표시되는 부분만 캡처하기 위해서 GetBoundsRect 를 사용했다. GetWindowRect 를 사용해서 전체 윈도우를 캡처하게 되면 화면 밖으로 나간 부분은 검은 색으로 캡처된다.

리스트 4 캡처 함수 소스

```
/*--
    특정 윈도우를 캡처한 비트맵을 얻어온다.

    파라미터
    - hwnd - [입력] 캡처할 윈도우 핸들.

    리턴
```

```
- 성공 시 비트맵 핸들, 실패 시 NULL.
- */
HBITMAP WINAPI
CaptureWindow(HWND hwnd)
{
    HBITMAP captureBmp;
    HBITMAP oldBmp;

    HDC hWindowDC;
    HDC hMemDC;

    RECT rc;
    SIZE sz;

    // 윈도우 DC 를 구한다.
    hWindowDC = GetWindowDC(NULL);

    // 윈도우 영역을 구한다.
    GetBoundsRect(hWindowDC, &rc, 0);
    sz.cx = rc.right - rc.left;
    sz.cy = rc.bottom - rc.top;

    if(sz.cx <= 0 && sz.cy <= 0)
        return NULL;

    // 윈도우 DC 와 호환되는 메모리 DC 및 비트맵을 생성한다.
    hMemDC = CreateCompatibleDC(hWindowDC);
    captureBmp = CreateCompatibleBitmap(hWindowDC, sz.cx, sz.cy);

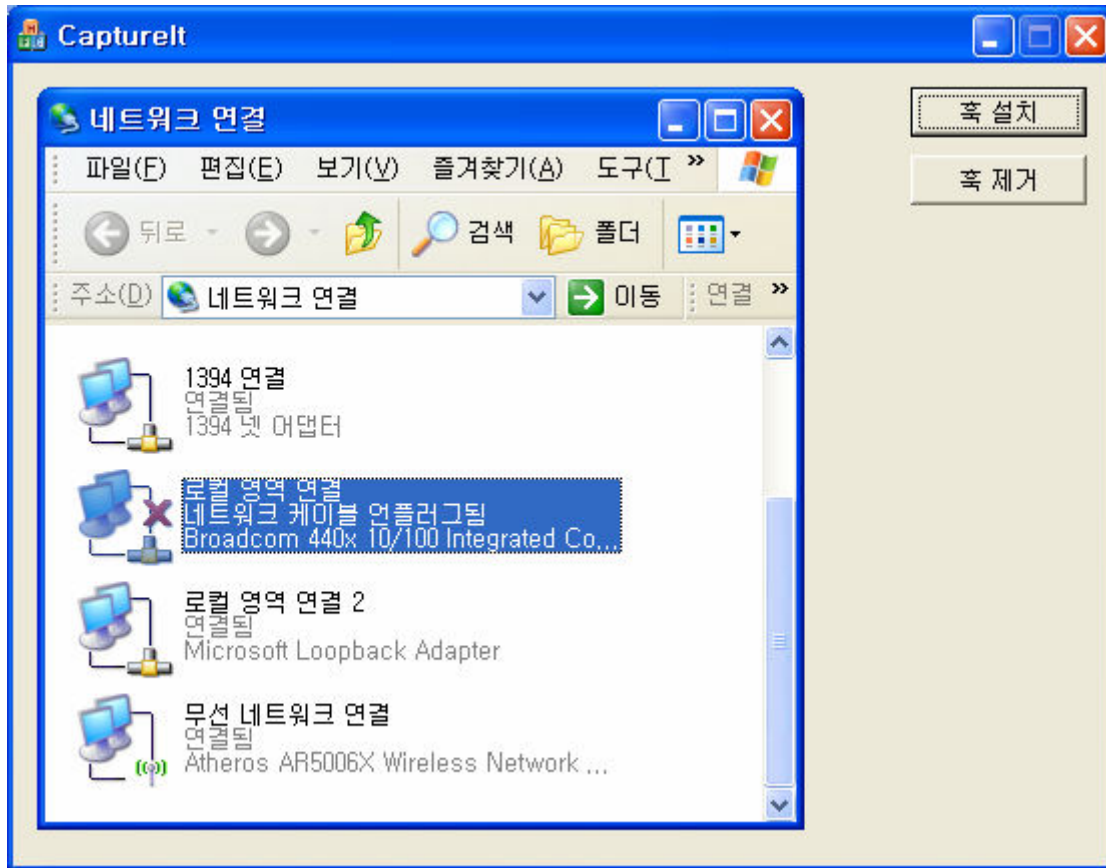
    // 비트맵을 복사한다.
    oldBmp = (HBITMAP) SelectObject(hMemDC, captureBmp);
    BitBlt(hMemDC, 0, 0, sz.cx, sz.cy, hWindowDC, rc.left, rc.top, SRCCOPY);
    SelectObject(hMemDC, oldBmp);

    // 할당된 자원을 반환한다.
    DeleteDC(hMemDC);
    ReleaseDC(hwnd, hWindowDC);

    return captureBmp;
}
```

이제 캡처 프로그램을 만들 모든 준비가 되었다. 우리가 만들 캡처 프로그램은 Ctrl 키를 누른 상태에서 마우스 왼쪽 버튼을 누르면 해당 윈도우를 캡처해서 보여주는 프로그램이다. <화면 4>에 프로그램의 실행 화면이 나타나 있다. 훅 설치 버튼을 누른 다음 캡처할 윈도우로 이동해서 Ctrl + 왼쪽 클릭을 하면 해당 윈도우가 캡처되어서 다이얼로그에 표시된다.

개발자를 위한 윈도우 후킹 테크닉: 마우스 혹은 통한 화면 캡처 프로그램 제작하기.



화면 4 CaptureIt 프로그램을 통해서 네트워크 연결 창을 캡처한 화면

<리스트 5>는 캡처 프로그램에 사용된 마우스 훅 프로시저다. WM_NCLBUTTONDOWN 인지 WM_LBUTTONDOWN 인지 체크한다. 그리고 Ctrl 키를 눌렀는지 체크하기 위해서 GetAsyncKeyState 함수를 사용하였다.

리스트 5 캡처 훅 프로시저

```
LRESULT CALLBACK
CaptureMouseProc(int code, WPARAM w, LPARAM l)
{
    if(code >= 0 && (w == WM_NCLBUTTONDOWN || w == WM_LBUTTONDOWN) &&
        (GetAsyncKeyState(VK_CONTROL) & 0x8000))
    {
        PMOUSEHOOKSTRUCT mhs = (PMOUSEHOOKSTRUCT) l;

        if(IsWindow(g_targetWnd))
        {
            SendMessageTimeout( g_targetWnd,
                                g_callbackMsg,
                                (WPARAM) mhs->hwnd,
                                0,
```

```
        SMTO_BLOCK|SMTO_ABORTIFHUNG,  
        50,  
        NULL    );  
    }  
}  
  
return CallNextHookEx(NULL, code, w, l);  
}
```

WH_MOUSE_LL의 경우는 그렇지 않지만, WH_MOUSE 훅을 사용할 경우 메시지가 윈도우의 계층 구조를 따라서 여러 번 발생하는 현상이 생긴다. 이 경우 동일한 윈도우를 캡처하는 코드가 빈번하게 수행되기 때문에 비효율적이다. 여러 번 수행되는 것을 줄이기 위해서 처리를 지연시키는 방법이 많이 사용된다. <리스트 6>과 <리스트 7>의 코드가 그러한 방법을 보여주고 있다. <리스트 6>은 캡처 메시지 핸들러이다. 반복 작업을 줄이기 위해서 메시지 핸들러에서 바로 캡처하지 않고 캡처할 윈도우 핸들을 리스트에 추가하는 일만 한다. 실제로 화면을 캡처하는 것은 <리스트 7>의 타이머 핸들러에서 한다. 캡처하기 전에 리스트의 unique 함수를 통해 동일한 핸들을 제거하기 때문에 효율적인 구조가 된다.

리스트 6 캡처 메시지 핸들러

```
LRESULT CCaptureItDlg::OnCapture(WPARAM w, LPARAM l)  
{  
    HWND dest = (HWND) w;  
    HWND root = ::GetAncestor(dest, GA_ROOT);  
  
    if(!m_mutex.Lock(5000))  
        return 0;  
  
    m_hwnds.push_back(root);  
    m_mutex.Unlock();  
    return 0;  
}
```

리스트 7 타이머 메시지 핸들러

```
void CCaptureItDlg::OnTimer(UINT nIDEvent)  
{  
    if(nIDEvent == 1)  
    {  
        KillTimer(1);  
  
        if(m_mutex.Lock(5000))  
        {  
            // 중복된 윈도우를 제거한다.  
            m_hwnds.unique();  
  
            if(!m_hwnds.empty())  
            {  

```



```
m_bmpCapture.DeleteObject();

HBITMAP bmp = CaptureWindow(m_hwnds.front());
if(bmp)
{
    m_bmpCapture.Attach(bmp);
    m_stcCapture.SetBitmap(m_bmpCapture);
}
m_hwnds.pop_front();
}

m_mutex.Unlock();
}

SetTimer(1, 1000, NULL);
}

CDialog::OnTimer(nIDEvent);
}
```

초기에는 좀 더 근사한 캡처 프로그램을 만들고 싶었지만 늘 그렇듯이 자원의 압박으로 아주 간단하게 만들 수 밖에 없었다. 그만큼 간단하다는 것은 개선의 여지를 많이 담고 있다는 것을 의미한다. 자신의 내공을 한 단계 업그레이드 하고 싶다면 다음과 같은 기능을 구현해 보도록 하자.

오너드로 리스트 박스를 통해서 후킹된 이미지를 목록으로 표시하는 기능
리스트 박스에서 특정 이미지를 선택할 경우 오른쪽 화면에 크게 표시하는 기능
캡처한 화면을 비트맵이나 JPG 파일로 저장 기능
특정 영역 캡처 기능
마우스 포인터 밑의 윈도우 캡처 기능

박스 1 캡처 FAQ

1. 미디어 플레이어 화면은 캡처되지 않나요?

미디어 플레이어 화면을 캡처해 보면 동영상 재생되는 부분은 검게 표시된다. 이는 미디어 플레이어의 동영상 재생 부분은 화면 출력을 빠르게 하기 위해서 GDI를 거치지 않고 직접 오버레이 화면에 표시하기 때문이다. 이를 캡처하기 위해서는 오버레이 화면을 캡처해서 GDI에서 캡처한 것과 합치거나 아니면 다른 공수를 쓰는 방법이 있다(<참고자료 2> 참고). 이것 외에도 비디오 어댑터의 하드웨어 가속 기능을 꺼서 캡처하는 방법이 있다. 하드웨어 가속 기능을 끄면 GDI를 통해서 화면이 표시된다.

2. 최소화된 창은 캡처할 수 없나요?

기본적으로 Windows 에서 캡처할 수 있는 화면은 스크린에 표시되는 것들이다. 왜냐하면 화면에 표시되지 않는 것까지 모두 그리는 것은 Windows 입장에서는 굉장히 비효율적인 일이기 때문에 화면에 표시되지 않는 윈도우의 DC 는 출력되지 않도록 되어 있다. 하지만 질문과 같은 의도를 가진 프로그램을 위해서 WM_PRINT 와 WM_PRINTCLIENT 메시지가 존재한다. 이 메시지를 윈도우로 전송할 경우 윈도우는 전송된 파라미터의 DC 에 자신을 그려야 한다. 그러나 많은 윈도우가 이 메시지를 지원하지 않기 때문에 사실 이 것은 거의 무용지물과 다를 없다. 필자가 테스트 해 본 바에 의하면 윈도우의 기본 컨트롤과 기본 프레임의 경우 원활하게 잘 그려지고 그 외의 것들은 제대로 그려지지 않았다.

3. 메뉴 영역만 캡처할 순 없나요?

Windows 는 매우 추상화가 잘 된 시스템이다. 추상화가 잘 되어 있다는 말은 대부분의 객체가 일관적인 방식으로 추상화 되어 있다는 의미다. 따라서 화면에 표시되는 메뉴도 특별하지 않다. 그것도 일반적인 것과 똑 같은 윈도우다. 단지 인식하지 못하는 이유는 메뉴의 경우 그 특성상 다른 부위를 클릭하면 바로 사라지기 때문이다. Windows 의 기본 메뉴의 클래스명은 "#32768" 이다. 따라서 메뉴를 캡처하고 싶다면 해당 윈도우를 찾아서 캡처하면 된다. 찾는 방법은 FindWindow 를 사용하면 된다.

FindWindow("#32768", NULL);로 하면 메뉴 윈도우가 찾아진다.

물론 모든 메뉴가 위의 방식대로 캡처되진 않는다. 일부 프로그램의 경우 메뉴 자체를 직접 구현한 경우가 더러 있기 때문이다. 대표적인 경우가 마이크로소프트 오피스의 제품군이다. 오피스 2003 의 경우 메뉴 클래스명이 "MsoCommandBarPopup" 이다. 그리고 탐색기의 즐겨찾기 메뉴의 경우 툴바로 구현된 것이다.

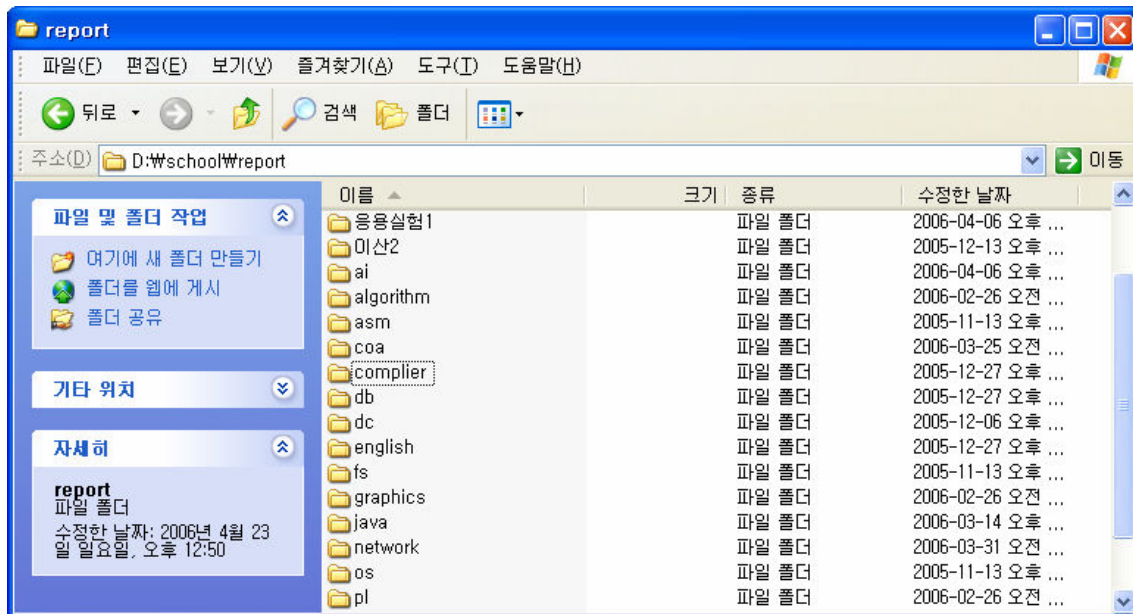
이러한 변종 메뉴를 포함해서 메뉴를 가장 확실하게 캡처할 수 있는 방법은 마우스 포인터를 사용하는 방법이다. 마우스 포인터 아래 있는 윈도우를 찾아서 캡처하는 방법이 가장 확실하다. 이 경우 GetCursorPos 를 통해서 마우스 포인터를 찾은 후에 WindowFromPoint 를 통해서 포인터 아래 있는 윈도우를 찾아서 캡처하면 된다.

도전 과제

아마도 대부분의 프로그래머들이 GUI 보다는 콘솔 창에서 작업하는 것을 편하게 느끼는 경우가 많다. 특히나 배치 작업을 해야 하는 경우는 더욱 더 그렇다. 필자의 경우 폴더에서 마우스 오른쪽 버튼을 누를 경우에 바로 해당 폴더에서 콘솔 창이 열리도록 레지스트리를 수정해서 사용하고 있다. 하지만 때때로 이 또한 귀찮을 때가 있다. 그래서 생각 해 본 기능

하나가 탐색기의 빈 영역을 어떻게 클릭했을 때 해당 폴더에서 콘솔 창을 열도록 하는 기능이다.

<화면 5>을 보면 d:\school\report 화면이 열려있다. 여기서 주소창이나 아니면 폴더 창의 여백 부분을 특수하게 클릭할 경우(Ctrl + 클릭등) 해당 폴더 창이 열리도록 만들어 보자.



화면 5 탐색기

참고 자료

- 참고자료 1. 다이렉트 X 스크린 캡처 방법 -
<http://www.mvps.org/directx/articles/screengrab.htm>
- 참고자료 2. 다양한 스크린 캡처 방법 -
<http://www.codeproject.com/dialog/screencap.asp>
- 참고자료 3. 오버레이 화면 캡처 방법 -
<http://www.codeproject.com/audio/capvidscrn.asp>
- 참고자료 4. 프로세스 익스플로러 -
<http://www.sysinternals.com/Utilities/ProcessExplorer.html>