개발자를 위한 윈도우 후킹 테크닉

WH_SHELL 훅을 사용해 다른 프로세스 윈도우 서브클래싱 하기

Win32 플랫폼에서는 컨텍스트 문제로 다른 프로세스에 존재하는 윈도우를 서브클래싱 하기가 쉽지 않다. WH_CBT 훅과 WH_SHELL 훅을 통해서 이러한 작업을 하는 방법을 살펴본다. 또한 브라우저 URL 하이재커와 팝업 제거 프로그램을 만드는 방법을 살펴본다.

목차

필자 소개

신영진 pop@jiniya.net, http://www.jiniya.net

"너는 네 세상 어디에 있느냐? 너에게 주어진 몇몇 해가 지나고 몇몇 날이 지났는데, 그래 너는 네 세상 어디쯤에 와 있느냐?" 2006 년도 이제 몇 달 남지 않았다. 후회가 되지 않는 한 해를 보내기란 버그 없는 프로그램을 만드는 것만큼 힘든 일인 것 같다.

연재 가이드

운영체제: 윈도우 2000/XP
개발도구: 마이크로소프트 비주얼 스튜디오 2003
기초지식: C/C++, Win32 프로그래밍
응용분야: 팝업 제거 프로그램, 브라우저 URL 하이재커

연재 순서

필자 메모

이번 달 샘플로 소개된 URL 하이재커는 처음에는 WH_CBT 훅으로 작성했다가 WH_SHELL 훅으로 변경했다. 훅 DLL 만 교체하면 되는 간단한 작업이었다. 그런데 신기한 것이 WH_SHELL 훅으로 변경하고 나서 HSHELL_WINDOWCREATED 가 전혀 호출되지 않는 것이었다. 필자는 몇 시간 동안 모든 코드를 찍어보면서 WH_SHELL 훅을 디버깅했다. 훅 DLL 디버깅과 구글 검색을 반복하기를 수 시간 문제가 훅 DLL 에 있지 않음을 알게 되었다. 문제의 원인은 훅 프로시저를 설치하던 메인 프로그램의 코드에서 WH_CBT 를 WH_SHELL 로 고치지 않았던 것이었다.

필자가 최고의 디버깅 덕목으로 여기는 것에는 두 가지가 있다. 냉정함과 상상력이 그것이다. 냉정하게 어디서 문제가 발생했는지 상상한다면 대부분의 문제는 쉽게 해결된다. 하지만 이번 사례에서 필자는 냉정함을 잃었고 지엽적인 문제에 집착함으로써 결과적으로 상당한 시간을 허비했다. 따라서 디버깅을 할 때에는 늘 냉정한 마음으로 문제의 원인을 분석해야 한다. 만약 냉정해 지기가 힘든 상황이라면 해당 문제를 잠시 잊고 휴식을 취하는 것이 좋다. 냉정함을 잃은 상태에서 문제를 해결하기는 굉장히 어렵기 때문이다.

Intro

윈도우 개발자가 특정 컨트롤의 기능을 확장 시키기 위해서 가장 자주 사용하는 기술은 서브클래싱이다. 주로 고객의 UI 욕구를 충족시키기 위해서 이러한 기법이 많이 사용된다. 이미지로 된 버튼, 이미지가 삽입된 리스트 컨트롤등이 대표적인 예라 할 수 있겠다. 하지만 이러한 것들은 늘 자신의 프로그램 내에서만 동작하도록 국한된다. SetWindowLong 을 통해서 해당 윈도우의 메시지 프로시저를 교체하더라도 해당 프로시저의 주소는 자신의 기준이지 다른 프로세스의 기준이 아니기 때문이다. 그렇다면 다른 프로그램의 윈도우를 서브클래싱 할 수는 없을까? 물론 할 수 있다. 메신저 플러스팩 등이 대표적이라고 할 수 있다.

다른 프로세스 윈도우 서브클래싱

다른 프로세스의 윈도우를 서브클래싱하지 못하는 가장 큰 이유는 컨텍스트가 다르기 때문이다. 이 문제를 해결하기 위해서 사용되는 방법은 여러가지가 있다. 직접적으로 해결하는 방법은 Dll injection 을 사용하는 것이다. 하지만 이 방법의 경우 복잡하고 생각해야 할 것이 많다. 간단하게는 우리가 이제껏 사용해왔던 훅을 이용하면 된다. 훅을 다른 스레드에 걸 경우 해당 스레드 컨텍스트에서 훅 프로시저가 수행되기 때문이다.

훅을 사용하는 방법은 두 가지 단점을 가지고 있다. 첫째는 훅 DLL 이 다른 컨텍스트에서 로드되고 해제되는 시점이 애매모호하다는 점이다. 이 과정은 전적으로 시스템에서 제어하기 때문이다. 둘째는 USER32.DLL 을 사용하지 않는 프로그램들(대표적으로 콘솔 프로그램)에는 사용할 수 없다는 점이다.

우리는 서브클래싱을 할 것이기 때문에 두 번째 단점은 문제가 되지 않는다. 또한 첫 번째 단점의 경우는 WH_CBT 나 WH_SHELL 훅을 사용함으로써 어느 정도 커버가 된다. 훅 DLL 은 해당 훅 프로시저가 처음으로 호출될 때 로드 된다. 키보드 훅은 해당 스레드에서 처음으로 키보드 메시지가 발생하는 시점에, 마우스 훅은 해당 스레드에서 처음으로 마우스 메시지가 발생하는 시점에 훅 DLL 이 해당 컨텍스트에서 로드 된다. WH_CBT 나 WH_SHELL 훅은 윈도우의 생성/소멸/활성화 등을 감지하는 역할을 하기 때문에 훅 프로시저 중에는 가장 먼저 다른 프로세스에서 로드 된다는 점이 특징이다. 훅 DLL 의 로드 지연이 가장 적고 윈도우의 활동을 감시하기 때문에 서브클래싱에 사용하기에 가장 적합한 훅이다.

박스 1 DLL Injection

DLL injection 이란 DLL 을 다른 프로세스의 주소 공간으로 침투시키는 방법을 말한다. 일반적으로 프로그램에서 필요한 DLL 을 LoadLibrary 함수를 사용해서 로드한다. DLL injection 의 핵심은 다른 프로세스에서 자신의 DLL 을 강제로 LoadLibrary 하도록 만드는 것이다. 이렇게 함으로써 해당 DLL 의 코드가 침투한 프로세스와 동일한 주소 공간에서 수행되도록 할 수 있다. 실제 구현 방법이 궁금한 독자 분들은 "Programming Applications for Microsoft Windows"를 참고하도록 하자.

WH_CBT 훅

WH_CBT 훅은 컴퓨터 베이스드 트레이닝에 사용되기 위해서 고안된 훅이다. 이 훅의 주된 역할은 윈도우의 생성/파괴/이동/활성화 등을 검사하는 일이다. 훅 함수의 원형은 아래와 같다.

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

code – [입력] 훅 프로시저가 호출된 이유를 알리는 코드 값이다(<표 1>을 참고). 0 보다 작은 경우에는 훅 프로시저를 수행하지 않고 CallNextHookEx 를 호출한 후 바로 리턴 해야 한다.

표 1 code 값의 의미

코드 값	의미
HCBT_ACTIVATE	윈도우가 활성화 된 경우다.
HCBT_CLICKSKIPPED	마우스 메시지가 시스템 메시지에서 제거된 경우다.
HCBT_CREATEWND	윈도우가 생성된 경우다. 이 훅 프로시저는 WM_CREATE, WM_NCCREATE 메시지가 윈도우에 전달되기 이전에 호출된다. 훅 프로시저에서 0 이 아닌 값을 리턴할 경우 해당 윈도우는 생성되지 않는다. CreateWindow 함수는 NULL 을 리턴 한다. 0 을 리턴 할 경우에는 정상적으로 윈도우가 생성된다. 이 훅 프로시저가 수행되는 시점에 윈도우는 생성되었으나, 최종 크기, 위치, 부모 윈도우등은 결정되지 않은 상태다. 이 상태에서 훅 프로시저는 해당 윈도우로 메시지를 전송할 수 있으나, 해당 윈도우의 WM_CREATE, WM_NCCREATE 프로시저가 아직 수행되지 않은 단계라는 것을 알아야 한다. CBT_CREATEWND 구조체의 hwndInsertAfter 값을

	변경함으로써 생성될 윈도우의 z-order 를 변경할 수 있다.
HCBT_DESTROYWND	윈도우가 파괴된 경우다.
HCBT_KEYSKIPPED	키보드 메시지가 시스템 메시지 큐에서 제거된 경우다.
HCBT_MINMAX	윈도우가 최소화/최대화가 된 경우다.
HCBT_MOVESIZE	윈도우가 옮겨지거나 크기가 변경된 경우다.
HCBT_QS	시스템 메시지 큐로부터 WM_QUEUESYNC 메시지를 전달받은 경우다.
HCBT_SETFOCUS	윈도우가 키보드 포커스를 가지게 된 경우다.
HCBT_SYSCOMMAND	시스템 명령(WM_SYSCOMMAND)이 수행되는 경우다.

wParam, lParam – code 값에 따라서 다른 값이 넘어온다(<표 2> 참고)

표 2 code 값에 따른 wParam 및 lParam 정보

code	wParam	lParam
HCBT_ACTIVATE	활성화된 윈도우 핸들	CBTACTIVATESTRUCT 구조체 포인터
HCBT_CLICKSKIPPED	마우스 메시지가 제거되었는지를 나타내는 플래그	MOUSEHOOKSTRUCT 구조체 포인터
HCBT_CREATEWND	생성된 윈도우 핸들	CBT_CREATEWND 구조체 포인터
HCBT_DESTROYWND	파괴될 윈도우 핸들	사용안함
HCBT_KEYSKIPPED	가상 키 코드	WM_KEYDOWN, WM_KEYUP 메시지의 lParam 값(반복 횟수, 스캔 코드 등을 포함)
HCBT_MINMAX	최대/최소화된 윈도우 핸들	하위 워드는 ShowWindow 플래그가 저장되어 있다(SW_SHOW, SW_HIDE 등). 상위 워드는 사용되지 않는다.
HCBT_MOVESIZE	이동/크기 변경된 윈도우 핸들	변경된 영역을 담고 있는 RECT 구조체 포인터
HCBT_QS	사용안함	사용안함
HCBT_SETFOCUS	포커스를 얻은 윈도우 핸들	포커스를 잃은 윈도우 핸들
HCBT_SYSCOMMAND	시스템 커맨드(SC_CLOSE, SC_HOTKEY, ...)	해당 시스템 커맨드에 대한 lParam 정보

wParam 과 lParam 에 사용되는 대부분의 구조체와 정보가 이제껏 소개한 것이기 때문에 새로 나오는 구조체에 대한 정보만 살펴보도록 하자. 아래는 HCBT_ACTIVATE 인 경우에 LPARAM 으로 전달되는 CBTACTIVATESTRUCT 구조체의 원형과 필드 별 설명이다.

```
typedef struct {
   BOOL fMouse; // 마우스에 의해서 활성화 되었는지를 나타내는 플래그
   HWND hWndActive; // 활성화된 윈도우 핸들
} CBTACTIVATESTRUCT, *LPCBTACTIVATESTRUCT;
```

아래는 HCBT_CREATEWND 인 경우에 LPARAM 으로 전달되는 CBT_CREATEWND 구조체의 원형과 필드 별 설명이다. HCBT_CREATEWND 인 경우에 윈도우는 생성되었으나 각종 정보가 아직 설정되지 않은 단계다. 이 단계에서는 GetWindowText 등으로 윈도우의 텍스트를 구할 수 없다. 윈도우 명을 구하기 위해서는 lpcs 필드 내에 있는 lpszName 을 참조해서 구해야 한다.

```
typedef struct {
   LPCREATESTRUCT lpcs; // WM_CREATE 메시지의 LPARAM 으로 전달되는 윈도우 생성 정보
   HWND hwndInsertAfter; // z-order 상에 생성될 윈도우보다 먼저 위치될 윈도우 핸들
} CBT_CREATEWND, *LPCBT_CREATEWND;
```

리턴 값: code 값이 HCBT_CLICKSKIPPED, HCBT_KEYSKIPPED, HCBT_QS 인 경우는 리턴 값이 무시된다. 그 외의 경우에는 0 을 리턴 할 경우 해당 작업이 정상적으로 처리되고, 1 을 리턴 할 경우 해당 작업이 취소된다.

WH_SHELL 훅

WH_SHELL 훅은 쉘 이벤트가 발생할 때 호출되는 훅이다. WH_CBT 훅과 비슷하게 윈도우의 생성/소멸/활성화를 감지하는 기능이 있다. 하지만 WH_CBT 와는 달리 모두 탑 레벨 윈도우의 변화만 감지하기 때문에 호출 빈도가 낮다. 따라서 좀 더 효율적이라 할 수 있다.

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

code – [입력] 훅 프로시저가 호출된 이유를 알려주는 코드 값이다(<표 3>을 참고). 0 보다 작은 경우에는 훅 프로시저를 수행하지 않고 CallNextHookEx 를 호출한 후 바로 리턴 해야 한다.

표 3 code 값의 의미

코드 값	의미

HSHELL_ACCESSIBILITYSTATE	Windows 2000/XP: accessibility 상태가 변경되었음.
HSHELL_ACTIVATESHELLWINDOW	쉘 메인 윈도우가 활성화 되어야 하는 경우다.
HSHELL_APPCOMMAND	Windows 2000/XP: 사용자의 입력 이벤트에 의해서 WM_APPCOMMAND 가 발생한 경우다. WM_APPCOMMAND 는 볼륨 조절등의 추가적인 키 입력을 처리하는 메시지다.
HSHELL_GETMINRECT	윈도우가 최소화/최대화된 경우다.
HSHELL_LANGUAGE	키보드 언어가 변경되거나 새로운 키보드 레이아웃이 로드된 경우다.
HSHELL_REDRAW	태스크바에 있는 윈도우 타이틀이 새로 그려지는 경우다.
HSHELL_TASKMAN	사용자가 태스크 리스트를 선택한 경우다. 테스트해 본 결과 Ctrl + Esc 키를 누른 경우 훅 프로시저가 호출되었다. 시작 버튼을 눌렀을 때엔 호출되지 않는다.
HSHELL_WINDOWACTIVATED	탑 레벨 윈도우의 활성화 상태가 변경된 경우다.
HSHELL_WINDOWCREATED	탑 레벨 윈도우가 생성된 경우다. 이 훅이 호출되는 시점에 윈도우는 생성된 상태다.
HSHELL_WINDOWDESTROYED	탑 레벨 윈도우가 파괴된 경우다. 이 훅이 호출되는 시점에 윈도우는 존재한다.
HSHELL_WINDOWREPLACED	Windows XP: 탑 레벨 윈도우가 대체(replaced)된 경우다.

실제로 훅 프로시저가 호출되는 시점에 관한 문서 정보고 애매한 점도 있고, 일부는 문서의 내용과 호출되는 시점이 달랐다. XP 컴퓨터에서 테스트 해 본 결과 code 값이 HSHELL_ACTIVATESHELLWINDOW, HSHELL_WINDOWREPLACED 로 호출되는 경우는 없었다. HSHELL_GETMINRECT 는 최소화될 때는 호출되었으나, 최대화될 경우에는 호출되지 않았다. HSHELL_TASKMAN 은 Ctrl + Esc 를 누른 경우 호출되었다. 반면 마우스로 시작 버튼을 클릭한 경우에는 호출되지 않았다. HSHELL_ACCESSIBILITYSTATE 는 필터키, 고정키 등의 설정이 변경될 때 호출되었고, HSHELL_APPCOMMAND 는 볼륨 조절 등의 키보드를 누른 경우에 호출되었다.

wParam, lParam – <표 4> 참고

표 4 code 값에 따른 wParam 및 lParam 정보

code	wParam	lParam

HSHELL_ACCESSIBILITYSTATE	상태가 변경된 accessibility 기능을 나타낸다. ACCESS_FILTERKEYS, ACCESS_MOUSEKEYS, ACCESS_STICKYKEYS 중 하나의 값을 가진다.	사용 안함.
HSHELL_ACTIVATESHELLWINDOW	사용 안함.	사용 안함.
HSHELL_APPCOMMAND	WM_APPCOMMAND 가 전달된 원래 윈도우 핸들.	사용 안함.
HSHELL_GETMINRECT	최대화/최소화된 윈도우 핸들.	RECT 구조체 포인터.
HSHELL_LANGUAGE	윈도우 핸들.	키보드 레이아웃 핸들.
HSHELL_REDRAW	새로 그려질 윈도우 핸들.	플래시되는 경우에는 TRUE, 그렇지 않은 경우에는 FALSE.
HSHELL_TASKMAN	사용 안함.	사용 안함.
HSHELL_WINDOWACTIVATED	활성화된 윈도우 핸들.	풀스크린 모드인 경우 TRUE, 아닌 경우 FALSE.
HSHELL_WINDOWCREATED	생성된 윈도우 핸들.	사용 안함.
HSHELL_WINDOWDESTROYED	파괴된 윈도우 핸들.	사용 안함.
HSHELL_WINDOWREPLACED	대체될 윈도우 핸들.	새로운 윈도우 핸들.

리턴 값: code 값이 HSHELL_APPCOMMAND 인 경우엔 1 을, 아닌 경우엔 0 을 리턴 해야 한다.

브라우저 URL 하이재커

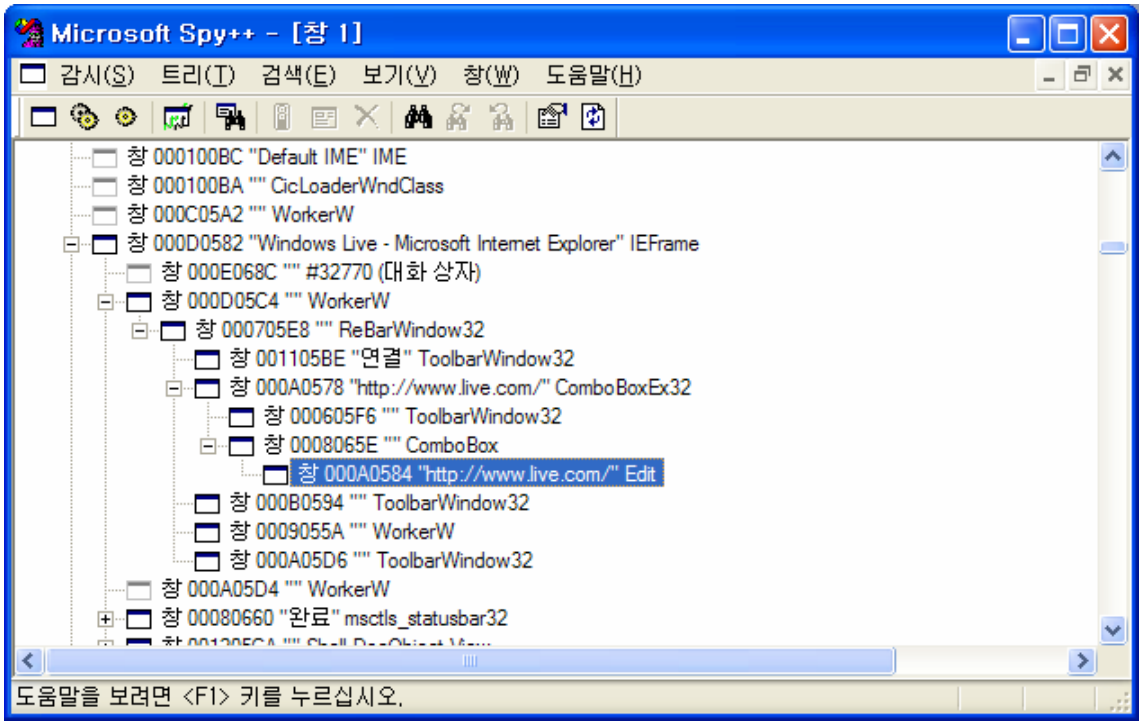
이번 시간에 우리가 WH_SHELL 훅을 사용해 만들어 볼 프로그램은 브라우저 URL 하이재커다. 작업을 간단하게 하기 위해서 "마이크로소프트"란 하나의 키워드만 검사하도록 한다. 주소 창에 "마이크로소프트"가 입력될 경우 http://www.imaso.co.kr 로 이동 시키면 된다. 우리는 이 작업을 WH_SHELL 훅을 통해서 브라우저의 URL 입력 상자를 서브클래싱 해서 구현할 것이다.

박스 2 하이재킹

하이재킹(hijacking)또한 후킹의 일종이라고 할 수 있다. 하이재킹은 주로 실행 경로를

중간에 가로채서 다른 경로로 변경 시키는 것을 의미한다. 브라우저의 URL 하이재커가 가장 많이 알려져 있다. URL 하이재커는 특정 키워드가 URL 주소에 입력될 경우 다른 주소로 이동 시키거나 자사의 검색엔진 결과를 출력하도록 만든다. 물론 이러한 URL 하이재커 외에도 하이재킹 대상에 따라서 다양한 종류의 하이재커가 있다.

URL 하이재커를 만들기 위해서 우리가 가장 먼저 해야 할 일은 브라우저의 주소 창을 찾는 일이다. 주소 창을 찾는 가장 손쉬운 방법은 윈도우의 계층 구조를 조사하는 것이다. <화면 1>에는 Internet Explorer 의 주소 창에 대한 윈도우 계층 구조가 나타나 있다. 선택된 부분이 실제 우리가 서브클래싱할 에디터 상자다.



화면 1 브라우저 주소창의 윈도우 계층 구조

<리스트 1>에는 위 정보를 사용해서 주어진 탑 레벨 윈도우로부터 주소 창을 찾아서 리턴하는 함수가 나와있다. 주소 창이 없는 경우엔 NULL 이 리턴 된다. Spy++ 제작할 때 사용했던 GetWindow 함수를 사용해서 구현되어 있다. 좀 더 엄밀히 구현한다면 GetWindow 의 리턴 값이 NULL 인 경우 바로 NULL 을 리턴 하도록 처리해야 한다. 여기서는 구현의 핵심만 보이기 위해서 그러한 에러 처리 부분을 생략했다.

리스트 1 윈도우 주소창을 찾는 함수

```
HWND GetAddressBarWnd(HWND hwnd)
{
```

```
    TCHAR buf[MAX_PATH];

    GetClassName(hwnd, buf, sizeof(buf));
    if(_tcsicmp(buf, "IEFrame") != 0)
        return NULL;

    hwnd = GetWindow(hwnd, GW_CHILD);       // #32770 대화상자
    hwnd = GetWindow(hwnd, GW_HWNDNEXT);    // WorkerW
    hwnd = GetWindow(hwnd, GW_CHILD);       // ReBarWindow32
    hwnd = GetWindow(hwnd, GW_CHILD);       // ToolbarWindow32
    hwnd = GetWindow(hwnd, GW_HWNDNEXT);    // ComboBoxEx32
    hwnd = GetWindow(hwnd, GW_CHILD);       // ToolbarWindow32
    hwnd = GetWindow(hwnd, GW_HWNDNEXT);    // ComboBox
    hwnd = GetWindow(hwnd, GW_CHILD);       // Edit

    GetClassName(hwnd, buf, sizeof(buf));
    if(_tcsicmp(buf, "Edit") != 0)
        return NULL;

    return hwnd;
}
```

주소 창을 찾았으면 다음으로 할 일은 서브클래싱을 해야 한다. 서브클래싱 작업의 경우 여러 곳에서 반복적으로 사용되기 때문에 함수 셋을 정의해두면 나중에도 다시 사용할 수 있다. <리스트 2>에는 이러한 서브클래싱 작업을 도와줄 함수 셋이 나와 있다.
서브클래싱을 할 경우 이전 주소를 저장해야 한다. 주소를 전역 맵으로 저장하는 대신에 간단하게 윈도우 프로퍼티를 사용해서 구현했다. 서브클래싱을 할 때에 프로퍼티를 설정하고 해제할 때에 프로퍼티를 삭제하는 방식이다. 그리고 서브클래싱을 할 때에 주의해야 할 점은 유니코드 윈도우 인지를 판단해서 서브클래싱을 해야 한다는 점이다. IsWindowUnicode 함수를 사용해서 유니코드 윈도우 인지를 판별해서 매칭되는 정확한 함수를 사용해서 서브클래싱을 해야 한다. 그렇지 않을 경우 문자열 메시지에서 오류가 발생할 수 있다.

리스트 2 서브클래싱 관련 함수들

```
#define SUBCLASS_PROP TEXT("SubClassData")

typedef struct _SUBCLASSDATA
{
    HWND       hwnd;
    WNDPROC     oldProc;
} SUBCLASSDATA, *PSUBCLASSDATA;

// 윈도우의 서브클래스 구조체를 구해 온다.
PSUBCLASSDATA GetSubclassData(HWND hwnd)
{
    PSUBCLASSDATA data = (PSUBCLASSDATA) GetProp(hwnd, SUBCLASS_PROP);
```

```
    if(!data)
        return NULL;
    if(data->hwnd != hwnd)
        return NULL;

    return data;
}

// 윈도우를 서브클래싱 한다.
BOOL SubclassWindow(HWND hwnd, WNDPROC fn)
{
    PSUBCLASSDATA data = new SUBCLASSDATA;

    data->hwnd = hwnd;
    data->oldProc = (WNDPROC) GetWindowLongPtr(hwnd, GWL_WNDPROC);
    SetProp(hwnd, SUBCLASS_PROP, data);

    if(IsWindowUnicode(hwnd))
        SetWindowLongPtrW(hwnd, GWL_WNDPROC, (LONG_PTR) fn);
    else
        SetWindowLongPtrA(hwnd, GWL_WNDPROC, (LONG_PTR) fn);
    return TRUE;
}

// 서브클래싱을 해제한다.
BOOL UnSubclassWindow(HWND hwnd)
{
    PSUBCLASSDATA data = GetSubclassData(hwnd);
    if(!data)
        return FALSE;

    if(IsWindowUnicode(hwnd))
        SetWindowLongPtrW(hwnd, GWL_WNDPROC, (LONG_PTR) data->oldProc);
    else
        SetWindowLongPtrA(hwnd, GWL_WNDPROC, (LONG_PTR) data->oldProc);
    RemoveProp(hwnd, SUBCLASS_PROP);
    return TRUE;
}

// 서브클래싱된 윈도우인지 체크한다.
BOOL IsWindowSubclassed(HWND hwnd)
{
    PSUBCLASSDATA data = GetSubclassData(hwnd);
    if(!data)
        return FALSE;

    return TRUE;
}
```

이제 주소 창을 서브클래싱한 프로시저를 작성하면 된다. <리스트 3>에 그 코드가 나와있다. 특별한 내용은 없다. 엔터 키가 눌린 경우 내용을 검사해서 "마이크로소프트"라면

http://www.imaso.co.kr 로 대체하는 것과 WM_DESTROY 인 경우에는 서브클래싱을 복원시키는 것이 전부다.

리스트 3 새로운 주소창 프로시저

```
LRESULT CALLBACK UrlHijackProc(HWND hwnd, UINT msg, WPARAM w, LPARAM l)
{
    PSUBCLASSDATA data = (PSUBCLASSDATA) GetProp(hwnd, SUBCLASS_PROP);
    if(!data)
        return DefWindowProc(hwnd, msg, w, l);

    if(msg == WM_KEYDOWN && w == VK_RETURN)
    {
        TCHAR buf[MAX_PATH] = {0,};
        GetWindowText(hwnd, buf, sizeof(buf));
        if(_tcscmp(buf, TEXT("마이크로소프트")) == 0)
        {
            // 하이재킹
            SetWindowText(hwnd, "http://www.imaso.co.kr");
        }
    }
    else if(msg == WM_DESTROY)
    {
        UnSubclassWindow(hwnd);
    }

    return CallWindowProc(data->oldProc, hwnd, msg, w, l);
}
```

이제 WH_SHELL 훅 프로시저를 살펴보자. <리스트 4>에 훅 프로시저의 전문이 나와있다. 우리는 HSHELL_WINDOWACTIVATED 를 사용해서 서브클래싱을 하도록 했다. 이렇게 해야 후킹 되기 이전의 주소 창까지 서브클래싱을 할 수 있기 때문이다.
HSHELL_WINDOWCREATED 를 사용하면 이전에 생성된 윈도우는 서브클래싱 할 수 없다.

박스 3 IME 완료 상태

에디터 상자의 입력 문자열의 한글 조합 상태가 완료되지 않은 상태에서 에디터의 내용을 변경하는 경우 조합 중인 글자가 찌꺼기로 남는 경우가 있다. 이러한 경우에는 에디터의 내용을 변경하기 전에 IME 조합 상태를 취소하거나 완료시킬 필요가 있다. 이전 텍스트의 내용이 필요 없는 경우라면 아래와 같이 IME 조합 상태를 취소함으로써 그런 문제를 없앨 수 있다.

```
HIMC imc = ImmGetContext(hwnd);
ImmNotifyIME(imc, NI_COMPOSITIONSTR, CPS_CANCEL, 0);
ImmReleaseContext(hwnd, imc);
```

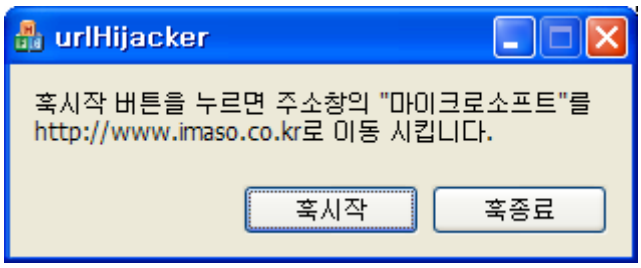
리스트 4 훅 프로시저

```
LRESULT CALLBACK SubclassProc(int code, WPARAM w, LPARAM l)
{
   if(code < 0)
      return CallNextHookEx(NULL, code, w, l);

   if(code == HSHELL_WINDOWACTIVATED)
   {
      HWND hwnd = GetAddressBarWnd((HWND) w);
      if(hwnd && !IsWindowSubclassed(hwnd))
      {
         // 하이재킹
         SubclassWindow(hwnd, UrlHijackProc);
      }
   }

   return 0;
}
```

이제 모든 작업이 끝났다. 위의 함수를 가진 DLL 을 만들고, SubclassProc 을 전역 WH_SHELL 훅으로 설치만 하면 된다.



화면 2 URL 하이재커 실행 화면

이번 달 샘플로 제작된 URL 하이재커를 실행하면 <화면 2>와 같은 윈도우가 나타난다. 훅시작 버튼을 누르면 주소창의 URL 을 하이재킹 한다. 훅종료를 누르면 해당 훅을 종료한다.

서브클래싱 스택

서브클래싱은 가장 원시적인 방법의 후킹이기 때문에 서브클래싱한 순서에 따라서 스택과 같은 구조가 생기게 된다. 이렇게 여러 단계의 서브클래싱이 걸린 경우는 제거하는데 생각을 많이 해야 한다. 그렇지 않으면 잘못된 연산 오류나 프로그램 오동작을 일으킬 수

있기 때문이다. <그림 1>에는 네 개의 프로시저가 서브클래싱을 한 경우의 구조를 보여주고 있다. 1 번이 오리지널 메시지 핸들러이고, 그것을 2,3,4 번이 순차적으로 서브클래싱 했다고 생각하면 된다.

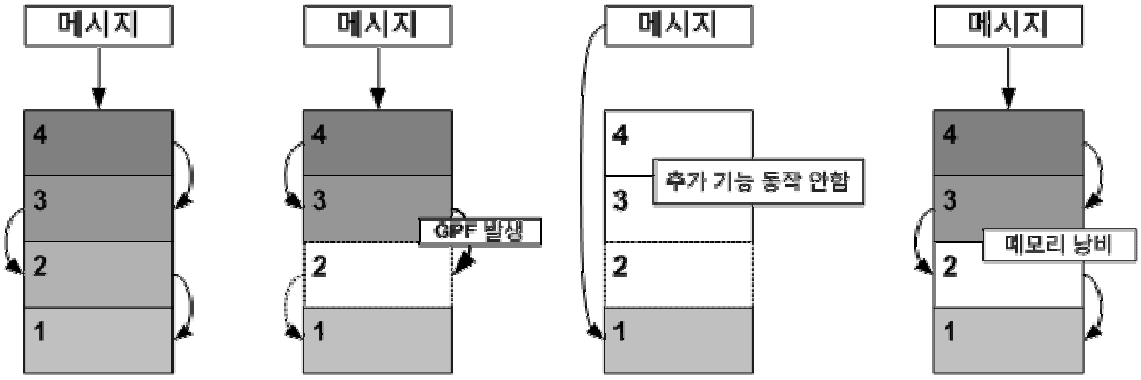


그림 1 서브클래싱 스택

위와 같은 경우에 2 번 프로시저가 서브클래싱을 끝내려고 한다고 생각을 해보자. 아무런 처리 없이 빠진 경우는 두 번째 그림과 같이 된다. 3 번이 2 번을 참조하곤 있지만 그곳은 더 이상 합법적인 공간이 아닌 것이다. 따라서 GPF 가 발생한다. 일반적인 서브클래싱 해제 루틴처럼 자신이 가지고 있는 주소로 복원 시킨 경우는 세 번째 그림처럼 된다. 이 경우는 자신보다 나중에 후킹한 3,4 번 프로시저의 추가 기능이 동작을 하지 않는다는 문제가 발생한다. 이러한 문제를 우아하게 해결한 방식이 최종 그림과 같은 형태다. 이 방식은 자신이 빠지지 않고 메모리에 자신의 메시지 핸들러를 남겨두는 것이다. 그리곤 자신은 단지 메시지를 포워딩 시키는 기능만 해준다. 이 경우에 단점으론 메모리 낭비를 들 수 있다. 불필요하게 포워딩을 위한 메모리를 차지하고 있는 셈이기 때문이다.

그렇다면 네 번째 그림과 같이 구현하기 위해서는 어떻게 해야 할까? 여러 가지 방법이 있지만 가장 손쉬운 방법은 서브클래싱 프로시저를 훅 프로시저가 있는 DLL 이 아닌 별도의 DLL 에 두는 방법이다. 이렇게 한 후에 훅 DLL 에서 서브클래싱을 할 때에 해당 서브클래싱 DLL 을 로드해서 해당 프로시저를 사용하면 된다. 프로그램이 종료되더라도 훅 DLL 만 시스템이 제거하기 때문에 서브클래싱 DLL 은 계속 메모리에 상주한 상태가 된다.

이러한 방법 외에도 프로그램 종료를 지연시키는 방법을 사용할 수도 있다. 사용자에게 현재 다른 프로그램이 사용하고 있기 때문에 종료할 수 없다는 메시지를 보내는 것이다. 이 경우는 서브클래싱 카운터 등을 공유 변수로 두고 추적해서 메시지를 출력하면 된다. 하지만 이 방법도 강제 종료 시에는 속수무책이다.

팝업 킬러

팝업은 한 때 브라우저의 문제로 여겨지던 시절이 있었다. 하지만 요즘 필자는 팝업이 더 이상 브라우저 속에서만 벌어지는 일이 아님을 뼈저리게 느끼고 있다. 필자의 삶에 가장 큰 충격을 준 사실은 넥슨의 카트라이더에 삽입된 팝업이다. 얼마 전부터 넥슨의 카트라이더 게임을 하고 나면 <화면 3>와 같은 광고 화면이 나타난다. 여기에 더 심각한 사실은 클로즈 버튼 또한 없다는 것이다. 광고가 끝나고 나면 클로즈 버튼이 생긴다. 실제로 alt+f4 등의 단축키를 모르는 사람들은 이러한 팝업에 속수 무책일 수 밖에 없다. 이것이 비단 넥슨만의 문제는 아니다. 랜섬웨어의 대부분이 악성코드도 아닌 것을 치료하라는 팝업을 계속 띄운다. 이러한 브라우저 밖의 팝업을 차단하는데 WH_CBT 훅은 강력한 역할을 한다.

화면 3 넥슨 광고 팝업

위의 넥슨 광고를 차단하는 훅 프로시저가 <리스트 5>에 나와있다. 이번에는 윈도우 생성을 차단해야 하기 때문에 HCBT_CREATEWND 를 검사했다. 이 경우에 팝업 생성을 취소하기 위해서는 간단하게 TRUE 를 리턴 해 주면 된다.

리스트 5 넥슨 광고 팝업을 제거하는 CBT 훅 프로시저

```
LRESULT CALLBACK PopupBlockerProc(int code, WPARAM w, LPARAM l)
{
    if(code < 0)
        return CallNextHookEx(NULL, code, w, l);

    if(code == HCBT_CREATEWND)
    {
        HWND hwnd = (HWND) w;
        TCHAR className[MAX_PATH] = {0,};

        // 생성될 윈도우의 클래스명을 구한다.
        GetClassName(hwnd, className, sizeof(className));
```

```
        // 넥슨 팝업이면 생성을 취소 시킨다.
        if(_tcsicmp(className, "NexonADBallon") == 0)
            return TRUE;
    }

    return 0;
}
```

도전 과제

이번 달은 여러분에게 풍성한 도전 과제를 내 줄 수 있을 것 같다. 더러는 유용한 것도 있기 때문에 한번씩 꼭 만들어 보도록 하자. 명심해야 할 사실은 눈으로 읽은 사실은 여러분의 지식이 아니라는 점이다. 필자가 몇 년간 코딩을 하면서 느낀 점은 실제로 자신이 입력하고 문제를 해결했을 때 진정으로 자신의 지식이 된다는 점이다.

1. 제공되는 샘플은 서브클래싱 종료 문제를 해결하지 않은 버전이다. 이것을 위에서 제시된 우아한 방법으로 종료되도록 수정해 본다. 훅 프로시저를 설치하고, 인터넷 익스플로러를 실행한 다음 훅 프로그램을 종료해서 문제가 없으면 된다.
2. 위의 문제를 종료를 지연시키도록 만들어본다. 사용자가 종료하려고 하면 다른 프로그램에서 사용하고 있다고 알려주는 것이다.
3. URL 하이재킹을 좀 더 쓸모 있게 만들어 본다. 여러 개의 URL 을 검사해서 사용자가 매칭시킨 곳으로 이동시키도록 만들어본다.
4. 팝업 킬러를 좀 더 유용하게 만들어 본다. 이전에 Spy++ 제작에서 배운 윈도우 탐색 방식을 사용해서 사용자가 팝업을 실시간으로 추가할 수 있고, 추가된 팝업을 모두 제거하는 형태로 프로그램을 제작해 본다.

참고자료

참고자료 1. Jeffrey Richter. <<Programming Applications for Microsoft Windows (4/E)>> Microsoft Press

참고자료 2. 김상형, <<Windows API 정복>> 가남사

참고자료 3. 김성우, <<해킹/파괴의 광학>> 와이미디어

참고자료 4. http://www.codeguru.com/cpp/w-p/win32/messagebox/article.php/c4541/ WH_CBT 훅을 사용해서 메시지 박스 위치를 변경하는 방법