2007년 2학기 해킹 및 크래킹

분석자료 #1

MyCrackIt.exe 분석자료

분석 대상 프로그램 다운로드 : http://crackmes.de

분석 도구 : OllyDebugger

질문 연락처 : kyegeun.kim@initech.com

<문서 열람 후 질문이 있을 시에는 상기 메일주소로 연락할 것>

1. 개요

MyCrackIt.exe 프로그램은 기본적으로 대화상자 기반의 윈도우용 프로그램이다. 분석을 위해서 해당 프로그램에서 사용되는 API에 대한 내용을 사전에 조사하게 될 경우 빠른 분석이 가능하다. 따라서, 이번 MyCrackIt.exe 에서 사용하는 API 함수에 대해서 사전에 조사하는 것이 필요하며 이에 대한 설명은 프로그램을 분석하는 과정에서 설명하도록 한다.
사용되는 API 함수는 다음과 같다.

① DialogBoxParam
② SendMessage
③ GetDlgItem
④ StrToInt
⑤ GetModuleHandle
⑥ GetProcAddress
⑦ lstrcat
⑧ MessageBox
⑨ EndDialog

또한, 다음의 사항을 기본적으로 알고 있는 것이 분석을 위해서 도움이 된다.

1) **함수의 전처리 과정 (Prolog)** – 상응하는 후처리 과정이 존재하는 것은 숙지하도록 한다.
2) **스택의 위치와 계산법** – EBP, ESP 레지스터를 통한 스택의 계산
3) **자동변수, static 변수, 전역변수 등의 저장위치** – stack인지? heap인지? 어떻게 저장되는지? 저장시에 어떤 크기(bytes)단위로 할당이 되는지?
4) **약간의 어셈블리 언어 해석 능력** – 어셈블리 명령은 reference를 보면서 이해하면 된다
5) **윈도우 프로그래밍 개념** – 기본적인 구조와 API함수에 대한 지식

2. 분석 과정

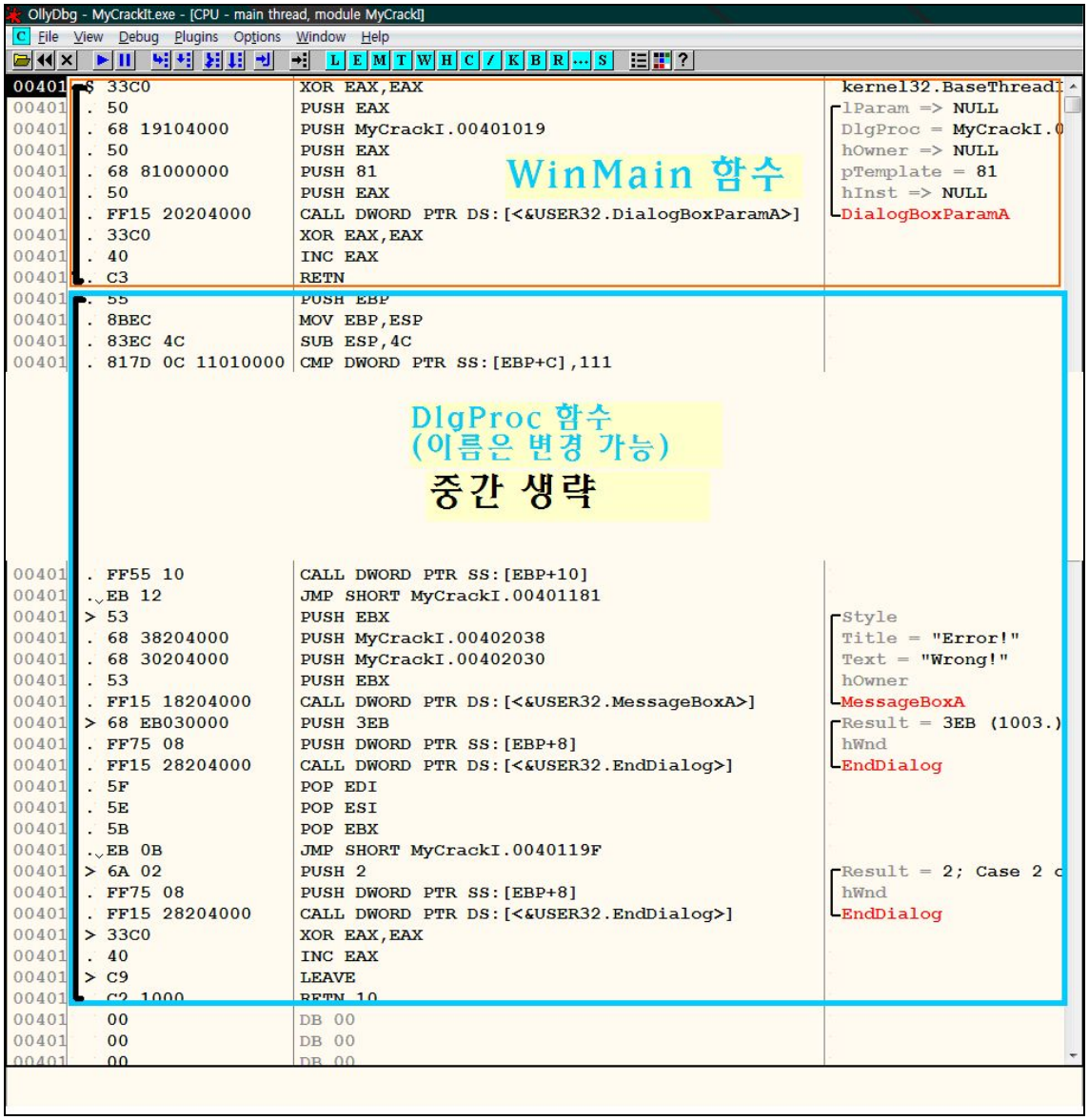
분석을 위해서 올리디버거를 통하여 역어셈블된 코드를 보면서 설명을 한다.

프로그램을 실행시켜 보면 알겠지만 해당 프로그램은 대화상자를 기반으로 동작하는 프로그램이다. 윈도우 프로그램을 분류하자면, SDI, MDI, Dialog Based 등으로 나뉠 수 있는데 MycrackIt.exe는 대화상자기반(Dialog Based)의 프로그램이다. 따라서, 대화상자 기반의 프로그램을 작성하기 위한 기본 골격을 알면 된다.
일반적으로 MFC가 아닌 SDK로 작성된 대화상자 기반의 프로그램은 WinMain과 rm 안에 위치한 DialogBox 혹은 DialogBoxParam과 메시지 처리함수(일반적으로 DlgProc를 많이 사용)로 이루어져있다.

즉, 다음과 같은 코드 형태를 지닌다.

```
int APIENTRY WinMain( 변수들 )
{
    …
    DialogBoxParam( 변수들 );
    …
}

BOOL CALLBACK DlgProc(변수들)
{
    switch(메시지) {
            case WM_XXXXX :
                                    break;
            case WM_XXXXX :
                                    break;
            case WM_XXXXX :
                                    break;
    }
}
```

올리디버거를 통해서 해당 프로그램을 역어셈블하여 살펴보게 되면 두 개의 함수로 이루어진 것을 쉽게 알 수 있다. 올리디버거에서 함수의 시작과 끝을 크게 연결시켜 보여주기 때문이다. 다음 그림을 보면 알 수 있다. 두 번째 칸의 왼쪽에 굵은 검정색으로 위 아래를 연결하는 굵은 검정색 선이 보일 것이다. 이게 함수의 시작과 끝을 알려주는 것으로 해석하면 된다.(화면 표시를 위하여 조금 강조해서 그렸음)



전체적인 구조를 확인하였으면 함수를 하나씩 분석해 보기로 한다.

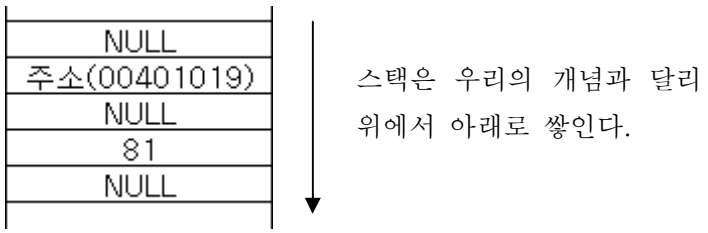
2.1. WinMain 함수 분석

분석과정에서 일반적인 윈도우 프로그램과 약간의 상이한 점이 있더라도 이 프로그램은 초보자를 위한 프로그램이므로 불필요한 부분은 삭제된 상태로 만들어졌다고 보면 된다.
이제 코드 분석을 위해서 올리디버거에서 역어셈블한 WinMain 함수 부분을 살펴보도록 한다.

```
00401000 XOR EAX,EAX              ;      kernel32.BaseThreadInitThunk
00401002 PUSH EAX                 ;      /lParam => NULL
00401003 PUSH MyCrackI.00401019   ;      |DlgProc = MyCrackI.00401019
00401008 PUSH EAX                 ;      |hOwner => NULL
00401009 PUSH 81                  ;      |pTemplate = 81
0040100E PUSH EAX                 ;      |hInst => NULL
0040100F CALL DWORD PTR DS:[<&USER32.DialogBoxParamA>]
                                  ;      \DialogBoxParamA
00401015 XOR EAX,EAX
00401017 INC EAX
00401018 RETN
```

위의 코드는 올리디버거에 나온 것을 복사해 놓은 것이다. PE(Portable Executable)형식의 윈도우 실행 프로그램은 적재시에 기본 시작주소번지인 00401000에서부터 시작되며 이 부분이 프로그램의 진입점(EntryPoint)가 된다. 또한, 주소번지 00401002에서부터 0040100F 까지를 보면 PUSH 명령과 CALL 명령의 조합으로 이루어져 있다. 이는 함수를 호출하기전에 함수로 전달되는 인자 값을 스택에 저장하고 함수를 호출하는 전형적인 형태로 함수를 호출하는 것을 알 수 있다. 호출하는 함수는 DialogBoxParam함수이다.

00401002번에서 0040100E에 이르기까지 코드가 차례대로 수행되면 스택에는 다음과 같은 데이터가 들어가 있게된다.



[그림 2-1] 스택에 넣은 인자 값

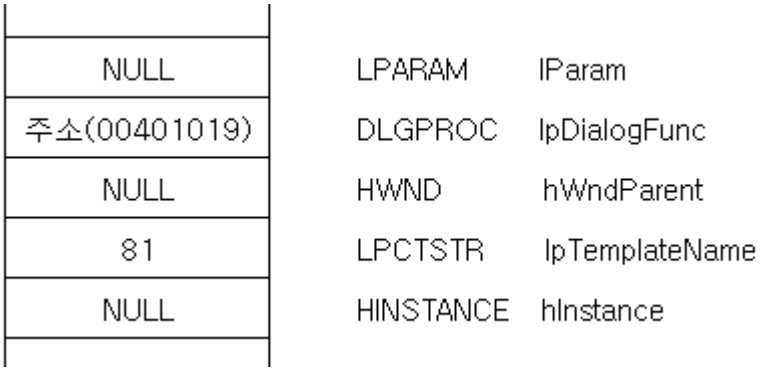
여기서 각각의 값에 해당하는 것은 API 함수와 비교해 보면 쉽게 알 수 있다. 주소번지 0040100F 번지의 내용은

```
0040100F CALL DWORD PTR DS:[<&USER32.DialogBoxParamA>]
```

으로 USER32.DLL 함수에 속한 DialogBoxParam 함수를 호출하라는 코드이다. 함수의 호출은 역어셈블을 하게되면 PUSH 명령과 CALL 명령의 조합으로 나타나게 되는데 함수의 인자로 전달되는 값을 스택에 넣고 CALL 하게되면 해당 인자를 호출된 함수에서 스택에서 꺼내어 사용하도록 되어있다.
DialogBoxParam 함수의 원형을 보게되면

```
INT_PTR DialogBoxParam(
        HINSTANCE hInstance,     // 대화상자템플릿을 정의한 실행프로그램의 핸들
        LPCTSTR lpTemplateName,  // 대화상자 템플릿(리소스편집기로 편집)
        HWND hWndParent,         // 대화상자를 소유한 부모 윈도우의 핸들
        DLGPROC lpDialogFunc,    // 대화상자로 들어오는 메시지를 처리할 함수
        LPARAM dwInitParam        // 대화상자 초기화 시에 사용할 LPARAM
                                  // WM_INITDIALOG 전송 시에 같이 보내어질
                                  // LPARAM 값을 의미
);
```

와 같이 나타난다. 여기에 그림1의 스택에 저장된 값을 대입해 보면



로 표시할 수 있다. 좀 더 정리를 해보면

가) hInstance 는 화면에 표시되는 대화상자의 resource를 포함한 실행파일에 대한 핸들이다.여기에서는 NULL을 넣었는데 이는 자기 자신을 의미한다고 볼 수 있다.

나) lpTemplateName은 대화상자의 템플릿을 가리키는데 Visual C++의 resource 편집기를 통해서 화면 설계를 할 때 각각은 고유 Control ID라는 정수 값을 가지며, 이 정수 값이 문자로 표시되는 이름과 연결된다. 여기서 lpTemplateName이라고 하는 것은 바로 고유 Control ID와 연결된 이름을 의미한다. 단, 이름은 문자열이므로 이에 연결되는 숫자(즉, Control ID)가 81임을 나타내는 것이다.

다) hWndParent는 대화상자를 표시한 응용프로그램에 대한 핸들을 가리키는데 이 프로그램은 대화상자가 바로 메인윈도우이므로 NULL을 넣게된다. 즉, 부모윈도우가 없다는 의미이다.

라) lpDialogFunc는 메시지를 처리하는 함수의 주소번지를 가리킨다. 여기서는 00401019번이 이

메시지 처리함수의 시작주소임을 나타내고 있다.

마) lParam은 대화상자가 처음 생성되는 초기화 과정(WM_INITDIALOG 이벤트 발생)에서 같이 보내어질 LPARAM 값을 설정해 주는 것이다. 여기서는 아무 설정을 사용하지 않아 NULL을 넣어주었다.

위의 내용을 비추어 볼 때 C언어로 작성된 코드는 아래와 같이 표시된다.

```
DialogBoxParam(NULL, MAKEINTRESOURCE(IDD_DIALOG_1), NULL, DialogProc, NULL);
```

이제 역어셈블된 코드의 남은 부분을 해석하기 위해보면 아래와 같은 코드를 볼 수 있다.

```
00401015 XOR EAX,EAX
00401017 INC EAX
00401018 RETN
```

해당 코드를 한 줄씩 분석해 보면

① EAX를 초기화, EAX = 0
② EAX의 값을 1 증가, EAX = 1
③ 함수를 종료하고 반환

의 의미가 있다. 그런데 함수가 종료하면 그 반환값은 EAX 레지스터를 통해서 반환되게 되며, 위의 코드를 보면 반환 값이 1임을 알 수 있다. WinMain 함수이므로 WinMain 함수의 return type이 int 이므로 정수 1을 반환할 것으로 추측된다. 즉, 완성된 WinMain 함수는 다음과 같을 것으로 추측 할 수 있다.

```
int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                     LPSTR lpszCmdParam, int nCmdShow)
{
      DialogBoxParam(NULL, MAKEINTRESOURCE(IDD_DIALOG1), NULL,
                     DialogProc, NULL);

      return 1;
}
```

위의 코드에서 DialogBoxParam함수의 두 번째 인자는 대화상자의 Control ID를 이야기하고 있으며, 이는 VC++을 이용해서 프로그램을 하는 과정에서 아래의 그림처럼 확인 할 수 있다. 다시 말해서 Control ID는 프로그램에 하드 코딩되어 있으며 이는 resource.h 에서 확인이 가능하다.

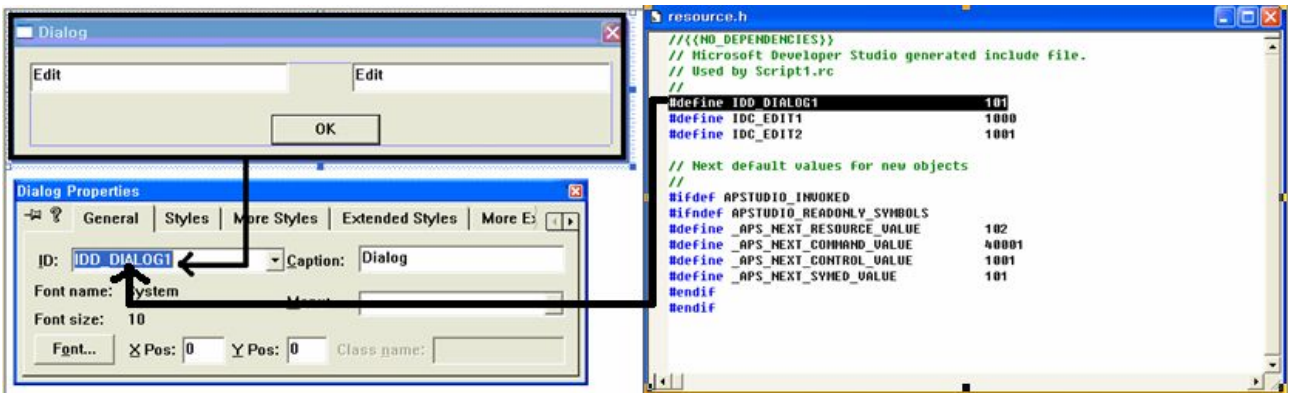
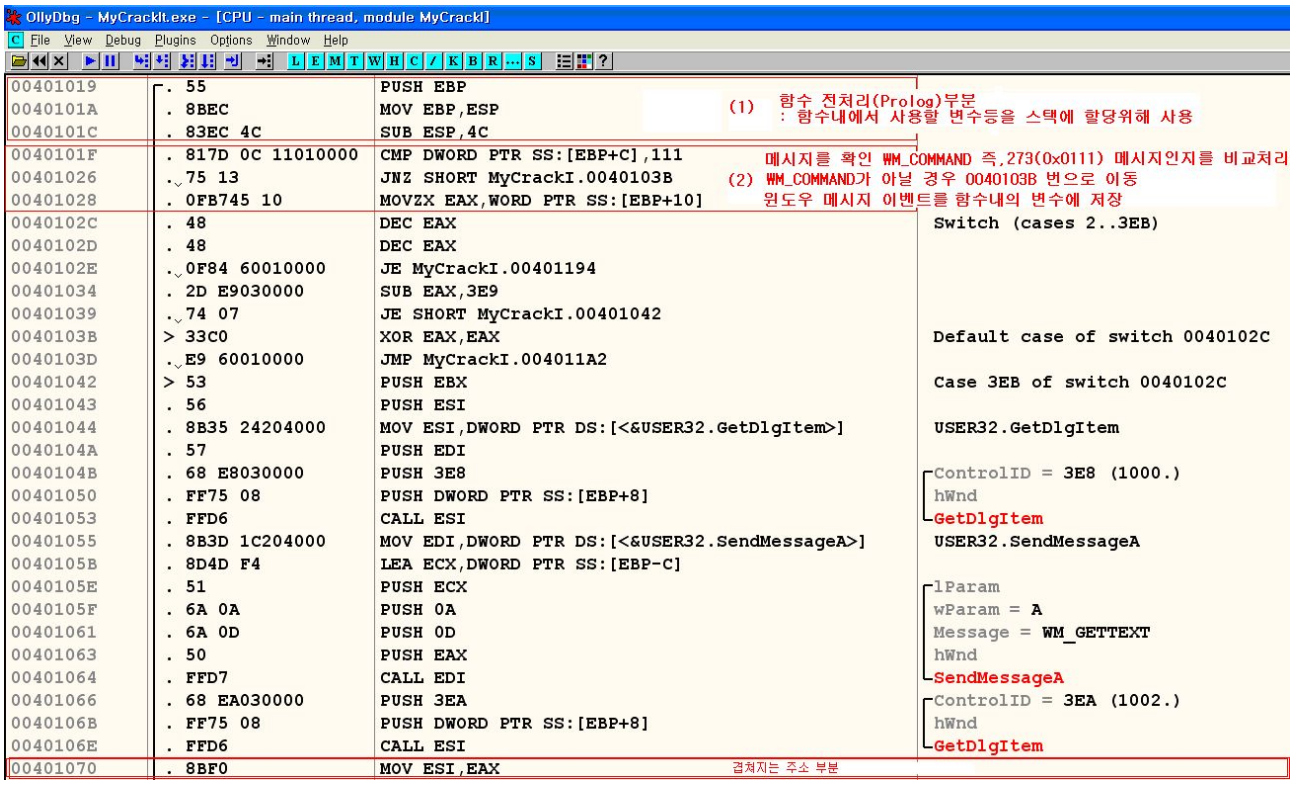


그림 2-2 대화상자의 Control ID 확인(윈도우 프로그램 작성 중에 확인한 화면)

2.2. 메시지 처리 함수 분석

WinMain 함수에서 호출한 DialogBoxParam 함수의 네 번째 인자는 해당 대화상자를 들어오는 윈도우 메시지를 처리하기 위한 함수를 지정하는 것이다. 따라서, 프로그래머는 별도의 함수를 지정하여 여기서 윈도우에서 기본으로 제공하는 메시지 처리 루틴이 동작하기 이전에 원하는 동작을 할 수 있도록 처리할 수 있다. 여기서 메시지 처리함수의 이름이 DialogProc라고 정하였으므로 메시지처리함수는 DialogProc로 선언하면 된다. 또한, 윈도우에서 메시지 처리함수는 BOOL CALLBACK 으로 선언하므로 해당 형식에 따라서 전달되는 인자를 설정해 주면 된다. 즉, 다음과 같이 선언될 것이다.

```
BOOL CALLBACK DialogProc(HWND hDlg, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
        … 생략 …
}
```



위의 그림에서 (1)번 사각형은 함수의 전처리 부분으로 함수에서 사용되는 자동변수들을 스택에 할당하기 위해서 사용된다. 여기서는 DialogProc 함수에서 사용하는 변수 등(변수 외에 별도의 내용을 저장하기 위한 공간도 할당하기 때문에 변수 등이라고 하였음)을 위한 스택 공간할당을 의미한다. 정확한 스택의 구조는 역어셈블 과정을 통해서 이 후에 설명하도록 한다. (2)번 부분은 메시지 처리함수인 DialogProc함수로 들어오는 메시지가 WM_COMMAND인지를 판단하는 루틴이라고 보면 된다.

Winuser.h 헤더파일에 보면 0x111 의 숫자를 갖는 메시지는 WM_COMMAND이다.
해당 코드와 연관된 부분을 정리해 보면 아래와 같이 나누어 볼 수 있다.

```
0040101F       CMP DWORD PTR SS:[EBP+C],111  ; 윈도우 메시지 확인
00401026       JNZ SHORT MyCrackI.0040103B   ; WM_COMMAND가 아니면 0040103B번지로 이동
… 생략 …
0040103B       XOR EAX,EAX                   ; 반환값을 만들기 위해 EAX 초기화
0040103D       JMP MyCrackI.004011A2         ; 004011A2로 이동
… 생략 …
004011A2       LEAVE                         ; 함수 후처리 과정을 지원
004011A3       RETN 10                       ; 함수 종료
```

이는 메시지 처리함수의 특정 인자를 통해 들어오는 값을 비교하여 어떤 처리과정을 거치도록 하고 있다. 우선 메시지 처리함수의 프로토 타입은 아래와 같다.

```
BOOL CALLBACK DlgProc(HWND hDlg,UINT iMessage,WPARAM wParam,LPARAM lParam)
```

위의 메시지 처리함수에서 EBP는 스택 프레임 포인터로 DlgProc함수에서 사용하는 스택의 경계를 가리킨다. 이 지점부터 낮은 주소는 함수 내부에서 사용하는 변수들이 저장되는 공간이고 높은 주소는 함수호출과 관련하여 저장된 값들이다. 함수가 호출되게 되면 스택에는 다음과 같은 기본 사항들이 저장되게 된다.

[그림 2-3] 메시지 처리함수의 스택 구조 (함수 전처리 과정 직후)

따라서 위의 역어셈블 코드 중 SS:[EBP+C] 번지에 위치한 값은 iMessage(윈도우 메시지코드)이며,

수신된 윈도우 메시지코드가 WM_COMMAND인지를 비교하는 부분이 첫 줄이다. 여기서 수신한 윈도우 메시지 코드가 0x111(WM_COMMAND)일 때와 아닐 때의 처리 방식이 나뉘게 된다. 우선 WM_COMMAND가 아닐 경우는 다음의 처리 과정을 갖는다.

주소 번지 (offset)	설명
0040101F	MyCrackIt의 0040103B 번지로 이동하라. 즉, WM_COMMAND가 아닐 경우 의 처리부분으로 이동하라는 명령
0040103B	EAX를 초기화하고 004011A2 번지로 이동하라. 즉, 함수의 반환 값을 저장하는 EAX를 0으로 만들어 반환 값을 0으로[1] 설정하라. 여기서 DlgProc함수는 BOOL 타입이므로 0 값은 FALSE를 의미
004011A2	LEAVE 명령을 수행(함수의 후처리 과정을 지원)하고 호출한 함수로 돌아가라

두 번째로 입력된 윈도우 메시지가 WM_COMMAND일 경우는 다음과 같이 처리한다.
SS:[EBP+10] 번의 위치에 있는 값을 EAX 레지스터로 이동하는데 SS:[EBP+10] 번지의 내용은 함수로 전달되는 3번째 인자인 wParam을 의미한다.[2]

이를 전체적인 구조의 C++ 형태로 보면 다음과 같다.

```
BOOL CALLBACK DlgProc(HWND hDlg,UINT iMessage,WPARAM wParam,LPARAM lParam)
{
        switch(iMessage)
        {
        case WM_COMMAND:
                                … 생략 …
        }
        return FALSE;
}
```

주소번지 00401028 부터는 윈도우 메시지를 확인한 결과 값이 WM_COMMAND일 경우를 처리하는 과정이다. WM_COMMAND 윈도우 메시지[3]는 일반적으로 wParam과 lParam 변수를 같이 쓰게되는데 주로 wParam은 어떤 컨트롤이 메시지를 보냈는지를 확인하는 것이다. 다시 말해서 윈도우화면에서 "확인" 버튼을 눌렀는지, "취소" 버튼을 눌렀는지를 구분할 수 있다. lParam은 메시지를 보낸 컨트롤의 윈도우 핸들이 저장된다. lParam의 경우 MyCrackIt 프로그램에서는 사용하지 않으므로 설명은 생략한다. 상세한 설명은 아래와 같다.

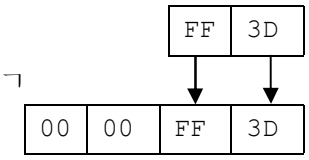
---

[1] DlgProc함수의 타입이 BOOL 이므로 값 0은 FALSE를 의미한다.
2 10쪽의 [그림 2-3] 메시지 처리함수의 스택 구조 (함수 전처리 과정 직후) 를 참고하라.
3 메뉴, 액셀러레이터를 선택했을 때 이 메시지가 전달되며 차일드 컨트롤이 부모 윈도우로 통지 메시지를 전달할 때도 이 메시지 형태로 전달된다. 각종 컨트롤로부터 값이 전달되며 또한 각 컨트롤은 다양한 통지 메시지를 보내므로 이 메시지는 일반적으로 이중 switch문으로 작성된다. 즉, 윈도우 화면에서 사용자가 버튼을 누르는 등의 특정행위를 했을 때 이를 처리하기 위한 메시지로 보면 된다.

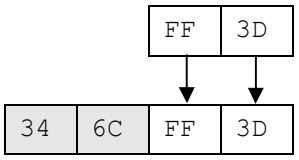
```
00401028        MOVZX EAX,WORD PTR SS:[EBP+10]          ..... (1)

0040102C        DEC EAX
0040102D        DEC EAX                                 ..... (2)
0040102E        JE MyCrackI.00401194

00401034        SUB EAX,3E9                             ..... (3)
00401039        JE SHORT MyCrackI.00401042
```

MyCrackIt 의 00401028 번지부터 00401039 번지까지는 위와 같이 3 부분으로 나뉘어 질 수 있다. 각 부분에 대한 설명은 다음과 같다.

A. (1) 번 구간은 SS:[EBP+10]번 메모리 주소에 있는 내용을 EAX레지스터로 옮기라는 명령이다. 단, MOV 명령과 다른 점은 소스에서 목적지로 복사할 때 크기가 다르면 남는 부분은 0으로 채워 넣으라는 것이 다르다.[4] 명령에서 복사할 내용은 WORD 크기로 2바이트임을 나타내고 복사할 곳의 번지는 SS:[EBP+10]으로 10쪽의 [그림 2-3]을 참조하면 wParam 을 의미한다. 따라서, 이는 SS:[EBP+10]번지에서 2바이트를 EAX로 4바이트로 확장하여 복사하면서 EAX의 나머지 상위 2바이트는 0으로 초기화 한다는 것을 의미한다.



< MOVZX 를 통한 복사 >

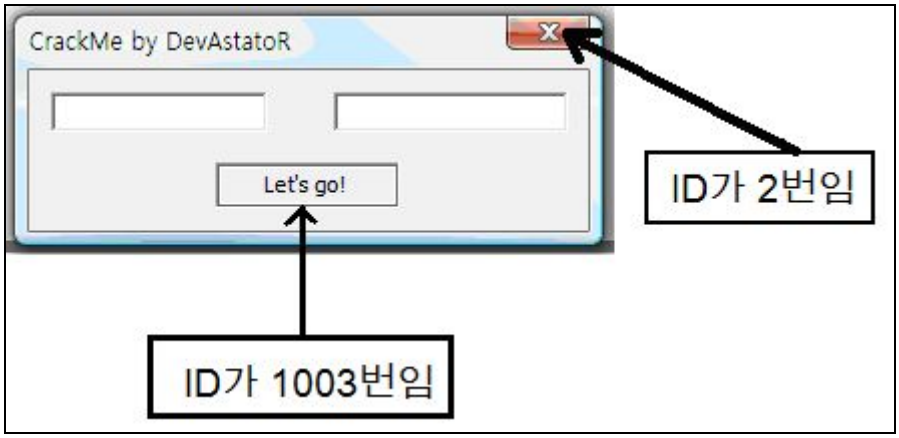


< MOV 를 통한 복사 >

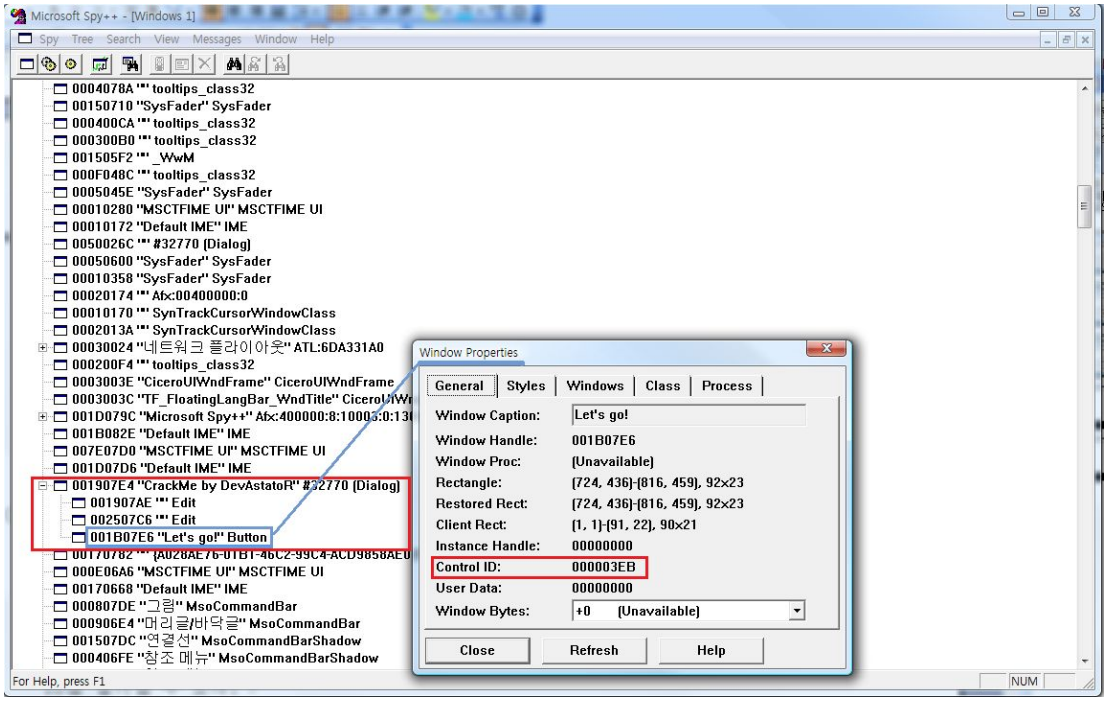
위의 그림에서 보듯이 MOVZX를 사용하면 상위 2바이트에 0 값을 채워놓아 확장이 되더라도 원하는 값이 그대로 저장되지만, MOV를 사용할 경우 하위 2바이트는 원하는 값이 복사되지만 상위 2바이트에는 어떤 값이 들어갈 지 모른다. 다시 말해서 이전에 있던 값의 상위 2 바이트 값이 남아 있어 전체적으로 4바이트 값을 보면 원하는 값이 복사된 것이 아님을 알 수 있다. 따라서, MOVZX는 하위 2바이트를 4바이트로 확장하여 옮기는 것을 의미한다.

B. (2) 번 구간은 위에서 설명한 바와 같이 이동한 wParam의 값에서 2를 뺀 뒤 이 값이 0과 같으면(wParam 값이 2이면) 00401194 번지로 이동하라는 명령이다. 주목할 것은 여기서 wParam의 값이 2 라는 것이 어떤 의미인가 하는 것이다. 일반적으로 WM_COMMAND와 함께 들어오는 wParam의 값은 통지 메시지를 보낸 항목의 ID 값을 나타낸다. 즉, ID 가 2번이라는 것인데 여기서는 좌측 상단의 종료메뉴를 의미한다.

[4] MOVZX 명령에 대한 자세한 내용은 어셈블리 명령을 참조하라.



C. (3) 번 구간은 EAX가 0x3E9(10진수 1001) 인지를 비교하는데 이는 원칙적으로 EAX에 원래 할당된 값, 다시 말해 SS:[EBP+10]번지에 저장되었던 값인 wParam이 0x3EB(10진수 1003)인지를 확인하는 것이다.(이미 앞에서 DEC EAX를 두 번하였기 때문에 2값을 더해야 한다.) 여기서 1003은 마찬가지로 ID 번호 1003을 의미하는데 이는 Microsoft Visual Studio에서 제공하는 유틸리티인 Spy++ 를 사용하여 해당 프로그램의 Control ID를 알아낼 수 있다.



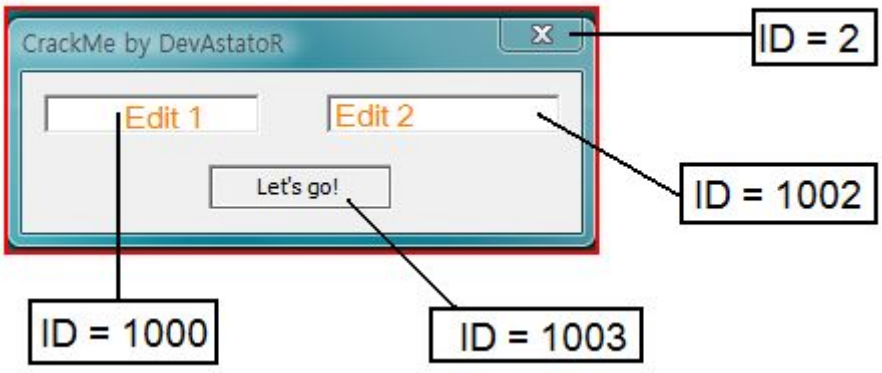
따라서, 위의 3 부분을 각각 분석한 결과를 합쳐서 설명하면 wParam 값이 2인 경우(종료버튼이 보낸 메시지인 경우)에는 00401194 번지로 이동하여 명령을 수행하고, 1003(0x3EB)인 경우(Let's go라고 쓰인 버튼인 경우)에는 00401042번지로 이동하여 명령을 수행하라는 의미이다. 이를 좀 더 쉽게 표현하자면 종료 버튼이 눌렸을 경우와 "Let's go"라는 버튼이 눌렸을 때의 동작을 의미한다.

다음 역어셈블 코드 부분은 아래와 같이 두 줄을 해석하면 된다.

```
0040101F      CMP DWORD PTR SS:[EBP+C],111
00401026      JNZ SHORT MyCrackI.0040103B
… 생략 …
0040103B      XOR EAX,EAX                    ;  Default case of switch 0040102C
0040103D      JMP MyCrackI.004011A2
… 생략 …
004011A2      LEAVE
004011A3      RETN 10
```

위의 내용을 보면 EAX를 초기화 하여 0으로 만들고 004011A2 번지로 이동하고 해당 번지를 보면 함수를 종료하는 것임을 알 수 있다. (004011A2번지 이하의 내용은 앞서 설명하였다.) 여기서, 0040103B번지로 어디서 왔는지를 위에서 참고하여 보면 00401026번지와 연관 지을 수 있다. 즉, 위의 코드는 수신한 윈도우 메시지가 WM_COMMAND가 아닐 경우 반환값을 0으로 초기화(FALSE)로 만든 후 함수 후처리 과정을 거쳐서 종료하는 것을 의미한다.

여기서 정리를 한 번 해보면 00401042번지부터는 WM_COMMAND메시지를 수신하고 Control ID가 1003일 경우(즉, Let's go 버튼을 누른 경우)에 수행되는 코드를 나타내고 있다. 여기서 MyCrackIt 프로그램의 각 아이템의 Control ID를 Spy++를 통해 확인해 보면 아래 그림과 같다.



[그림 2-4] 아이템의 Control ID 확인 (ID = 2는 Control ID 아님)

위 [그림 2-4] 에서 설명의 편의를 위하여 사용자 입력을 받아들이는 두 개의 편집상자에 Edit1 과 Edit2 라는 이름을 주었다. 우선 각 아이템의 Control ID는 그림에서 보는 바와 같으며, 시스템 메뉴의 종료상자(ID = 2)는 Spy++를 통해서 확인되지 않는다. 옵셋번지 00401042번부터 0040116D번까지가 두 개의 편집 상자에 모두 맞는 답을 넣었을 때 처리되는 과정이고 0040116F에서 00401192번까지는 답이 틀렸을 때 메시지박스를 나타내는 코드이다. 우선 이 부분의 설명을 위하여 역어셈블 코드에서 사용하는 내부변수에 대한 정리를 해 둘 필요가 있다. 메시지처리함수의 함수 전처리 과정에서 SUB ESP, 4C 라고 하여 76바이트를 내부 지역변수를 위하여 할당한 것은 확인 할 수 있으므로 역어셈블 코드에서 SS:[EBP-XX]라고 되어 있는 것들을 정리하여 분석과정에 사용하는 것이 편리하다.
이를 정리해보면 다음과 같다.

코드에서 사용	변수명	변수 타입	비고
SS:[EBP+14]	hDlg	HWND	메시지 처리함수로 전달되는 인자
SS:[EBP+10]	iMsg	UINT	
SS:[EBP+C]	wParam	WPARAM	
SS:[EBP+8]	lParam	LPARAM	
코드에서 사용	**임시로 사용할 변수명**	**예상되는 변수 타입(크기)**	**비고**
SS:[EBP-C]	Var_1	문자열 (12 Bytes)	메시지 처리함수에서 사용하는 지역변수
SS:[EBP-18]	Var_2	문자열 (12 Bytes)	
SS:[EBP-28]	Var_3	문자열 (16 Bytes)	
SS:[EBP-3A]	Var_4	문자열 (40 Bytes)	
SS:[EBP-4C]	Var_5	문자열 (58 Bytes)	

[표 2-1] 메시지 처리함수에서 사용하는 스택과 변수의 연관

```
00401042  PUSH EBX                          ; EBX 레지스터를 사용하기 위해 이전 값을 스택에 저장
00401043  PUSH ESI                          ; ESI 레지스터를 사용하기 위해 이전 값을 스택에 저장
00401044  MOV ESI,DWORD PTR DS:[<&USER32.GetDlgItem>]
0040104A  PUSH EDI
0040104B  PUSH 3E8
00401050  PUSH DWORD PTR SS:[EBP+8]
00401053  CALL ESI                          ..... (1)
```

User32.dll의 GetDlgItem 함수 호출임을 추정할 수 있는 부분

```
00401055  MOV EDI,DWORD PTR DS:[<&USER32.SendMessageA>]
0040105B  LEA ECX,DWORD PTR SS:[EBP-C]
0040105E  PUSH ECX
0040105F  PUSH 0A                           ..... (2)
00401061  PUSH 0D
00401063  PUSH EAX
00401064  CALL EDI
```

```
00401066  PUSH 3EA
0040106B  PUSH DWORD PTR SS:[EBP+8]         ..... (3)
0040106E  CALL ESI
```

```
00401070  MOV ESI,EAX
00401072  LEA EAX,DWORD PTR SS:[EBP-C]
00401075  PUSH EAX                          ..... (4)
00401076  CALL DWORD PTR DS:[<&SHLWAPI.StrToIntA>]
```

```
0040107C  LEA ECX,DWORD PTR SS:[EBP-18]
0040107F  PUSH ECX
00401080  PUSH 0C
00401082  SUB EAX,200E4                                    ..... (5)
00401087  PUSH EAX
00401088  PUSH ESI
00401089  CALL EDI
```

위 (1)번 구간은 GetDlgItem 함수를 실행시키는 부분이다. 우선 PUSH와 CALL의 조합은 함수호출이라는 것을 유념하고 분석을 한다. 코드를 보면 USER32.dll의 GetDlgItem함수를 호출하는 것을 알 수 있는데 이는 00401044번 에서 User32.dll의 GetDlgItem함수에 대한 포인터를 ESI레지스터에 넣고 00401053번에서 ESI 레지스터에 대해 CALL 명령을 수행한 것으로 알 수 있다. 분석을 쉽게 하기위해 GetDlgItem API를 참조해보면 다음과 같은 원형을 확인할 수 있다.

```
HWND GetDlgItem(HWND hDlg, int nIDDlgItem);
```

즉, 스택에 저장될 내용은 우선 대화상자의 핸들인 hDlg, 대화상자에 속한 아이템의 control ID인 nIDDlgItem 이다. 실제 API함수 호출을 위해 인자를 넣은 것은 CALL 바로 앞서 이루어지므로 맨 아래줄의 CALL ESI 가 User32.dll의 GetDlgItem함수를 호출한 것이므로(3번째 줄에 ESI에 해당 API의 주소번지를 저장하였으므로 CALL ESI는 함수 호출 동작임) 그 바로 위의 두줄에 걸친 PUSH가 인자를 넣는 부분이 된다. 따라서, 첫번째 인자인 hDlg는 SS:[EBP+8]이므로 메시지 처리함수로 들어오는 첫번째 인자를 가리키고, nIDDlgItem은 0x03E8 인 십진수 1000번을 의미한다. [그림 2-4]에서 1000번은 Edit1 이므로 아래의 3줄은 GetDlgItem함수를 통해서 대화상자의 Edit1 아이템에 대한 핸들을 얻기 위한 것으로 보면 된다. 또한, CALL ESI를 하게되면 함수가 호출되게 되고 함수가 호출한 이후의 반환 값은 EAX레지스터에 저장된다. 따라서, CALL ESI 를 수행하고 나면 (GetDlgItem 수행)이후에는 EAX에 해당 아이템의 핸들이 저장된다.

이를 C++ 코드로 작성하면 다음과 같다.

```
GetDlgItem(hDlg, IDC_EDIT_1);
```

이제 **ESI 레지스터에 GetDlgItem 함수에 대한 포인터**가 저장되어 있고, **EAX에는 Edit1 편집상자에 대한 핸들**이 저장되어있음을 숙지하고 살펴보기로 한다.

(2)번 구간은 결론적으로 볼 때 SendMessageA 함수를 호출하기 위한 과정이다. 따라서, SendMessage 함수에 대한 원형을 보면 다음과 같다.

```
LRESULT SendMessage(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam);
```

마찬가지로 CALL EDI라고 하는 마지막줄이 SendMessage를 호출하는 것이므로 함수에서 사용하는 인자 4개는 CALL명령 이전에 존재하게된다.

- EDI 레지스터에 user32.dll의 SendMessageA API함수의 주소를 저장
- SS:[EBP-C] 즉 지역변수인 Var_1 의 주소를 ECX레지스터가 가리키도록 함
- ECX 레지스터 값을 스택에 저장(Var_1변수가 할당 받은 메모리번지 저장)(**lParam인자**)
- 0x0A(십진수 10)를 스택에 저장함.(**wParam 인자**)
- 0x0D(십진수 13)를 스택에 저장함. 여기서 13은 윈도우 메시지 코드의 값으로 winuser.h를 참조하면 WM_GETTEXT를 말한다. WM_GETTEXT는 대상 아이템에서 문자열을 읽어오라는 명령으로 wParam과 연동되는데 wParam은 읽어들일 문자열의 최대길이를 말한다. 여기서는 위에서 0x0A를 저장했으므로 최대 10개의 문자를 읽어들인다. (**Msg인자**)
- EAX레지스터 값을 스택에 저장(EAX는 GetDlgItem 함수의 반환 값이 저장)(**hWnd인자**)

결국 (1)번, (2)번 구간을 묶어서 살펴보면 Edit1 편집상자에 대한 핸들을 얻고, 그 핸들을 이용하여 Edit1편집상자에 WM_GETTEXT 메시지를 전송하여 최대 10개의 문자를 읽어오도록 하는 것이다. (2)번 구간의 마지막 줄인 CALL EDI 를 수행하면 이제 EAX에는 SendMessage 함수를 실행하고 반환받은 결과값이 저장되었으므로 더 이상 Edit1 편집상자의 핸들을 갖지 않는다. 그러나, **ESI는 여전히 GetDlgItem 함수에 대한 포인터**를 가지고 있으며, (2)번 구간 첫 줄의 명령에 의해 **EDI는 SendMessage 함수에 대한 포인터**를 가지고 있음을 확인할 수 있다.

위에서 (3)번, (5)번 구간은 각각 (1)번, (2)번 구간과 유사한 코드를 보여주고 있다. 여기서 (3)번 구간은 대상 아이템의 번호만 0x03E8에서 0x03EA로 바뀌었을 뿐이다. 이는 [그림 2-4]를 보면 Edit2 편집상자의 Control ID가 할당되어 GetDlgItem을 수행하게 되므로 결국 (3)번 구간의 끝에서 EAX레지스터에는 Edit2 편집상자의 핸들이 저장되게 된다. (4)번 구간을 분석해 보면 EAX의 값을 우선 ESI레지스터에 넣는데 이는 반화값이 넘어가는 것을 의미하며 여기서는 Edit1 편집상자에서 읽어들인 문자열의 길이가 ESI레지스터로 저장됨을 의미한다. 이 후 3줄은 Var_1 (SS:[EBP-C])을 EAX레지스터에 할당하여 스택에 넣고 StrToInt함수를 호출한다. StrToInt 함수는 null 로 끝나는 문자열을 인자로 받아서 이를 정수로 변환하여 반환하는 함수로 원형은 다음과 같다.

```
int StrToInt(LPCTSTR lpSrc);
```

따라서, (4)번 구간의 의미는 (3)번 구간을 통해서 Edit1 편집상자에서 읽어들인 값을 정수로 변환하는 것을 의미하며 여기서 우리는 Edit1 편집상자는 최대 10자리까지의 숫자를 입력해야 한다는 것을 알 수 있다.

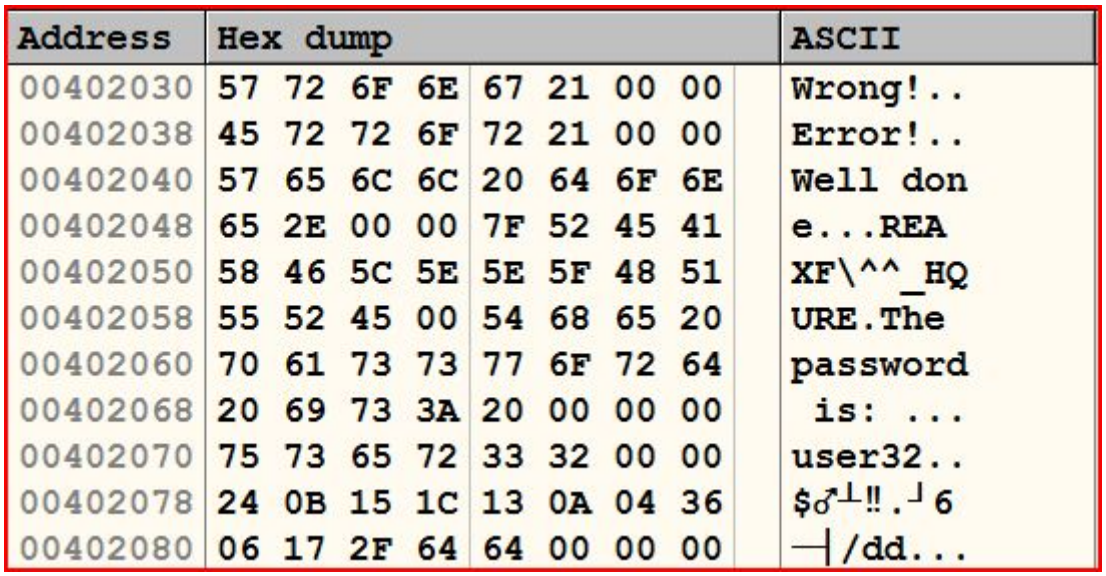
이 후 (5)번 구간은 (2)번 구간과 유사하나 좀더 자세히 보면 중간에 SendMessage함수에서 Msg에 대한 부분을 결정하는 부분이 무언가를 계산하도록 되어 있는 것을 알 수 있다.

```
00401082  SUB EAX,200E4
00401087  PUSH EAX
```

해당 부분을 보면 위의 코드와 같이 EAX에서 0x0200E4 를 빼고 그 값을 저장하도록 되어있다. 여기서 EAX는 StrToInt 함수의 반환 값으로 이는 Edit1 편집상자에 입력된 문자열을 정수로 형변환 한 값에서 0x0200E4(십진수 131300)을 뺀 나머지 값이 스택에 저장되는데 이 값이 윈도우 메시지코드로 사용되는 것을 확인할 수 있다. 유형이 (2)번 구간과 동일하므로 여기서는 결과 값이 0x0D (십진수 13)가 나와야 Edit2 편집상자에서 문자열을 읽어올 수 있도록 하는 것임을 유추할 수 있다. 따라서, Edit1 편집 상자에 들어가야 할 값은 131313 임을 알아낼 수 있다.

Edit1 편집상자에 대한 입력 값 : 131313

이제 0040108B 부터의 역어셈블코드에 대한 분석을 하여야 하는데 그 전에 미리 다음의 사항을 알아두고 넘어가도록 한다. 해당 부분에서부터는 프로그램의 BSS영역에 사전에 정의된 문자열 혹은 데이터 값을 넣어두고 있으며, 올리디버거를 통하여 해당 정보를 파악해 보면 아래와 같다.



Address	Hex dump	ASCII
00402030	57 72 6F 6E 67 21 00 00	Wrong!..
00402038	45 72 72 6F 72 21 00 00	Error!..
00402040	57 65 6C 6C 20 64 6F 6E	Well don
00402048	65 2E 00 00 7F 52 45 41	e...REA
00402050	58 46 5C 5E 5E 5F 48 51	XF\^^_HQ
00402058	55 52 45 00 54 68 65 20	URE.The
00402060	70 61 73 73 77 6F 72 64	password
00402068	20 69 73 3A 20 00 00 00	is: ...
00402070	75 73 65 72 33 32 00 00	user32..
00402078	24 0B 15 1C 13 0A 04 36	$♂┴‼.┘6
00402080	06 17 2F 64 64 00 00 00	─┤/dd...

[그림 2-5] 사전에 지정된 변수의 값 및 상수 값

각 메모리주소에는 프로그래머가 코딩시에 선언한 변수 등에 대한 고정 값이 있을 경우 이를 PE 파일 내에 저장하게 되며, 메모리에 적재되었을 경우에는 스택이 아닌 힙 혹은 bss 영역에 저장된다. 위의 화면 덤프는 해당 부분을 덤프한 것으로 프로그램 분석과정에서 쓰일 각 주소번지에 저장되어 있는 값들을 저장하면 아래의 표와 같다.

메모리 주소	16 진수 값	아스키 코드
00402030	57 72 6F 6E 67 21 00	W r o n g !
00402038	45 72 72 6F 72 21 00	E r r o r !
00402040	57 65 6C 6C 20 64 6F 6E 65 2E 00	W e l l d o n e .
0040204C	7F 52 45 41 58 46 5C 5E 5E 5F 48 51 55 52 45 00	R E A X F \ ^ ^ _ H Q U R E
0040205C	54 68 65 20 70 61 73 73 77 6F 72 64 20 69 73 3A 20 00	T h e p a s s w o r d i s :
00402070	75 73 65 72 33 32 00	u s e r 3 2
00402078	24 0B 15 1C 13 0A 04 36 06 17 2F 64 64	원래 16진 코드로 할당되었을 것으로 분석되어 아스키 코드로 변환 하지 않음

[표 2-2] 고정 변수 값 정리표

```
0040108B      MOV ESI,MyCrackI.00402078
00401090      LEA EDI,DWORD PTR SS:[EBP-28]
00401093      MOVSD          ; ESI가 가리키는 주소에서 EDI가 가리키는 주소로 4바이트 복사
00401094      MOVSD          ; 복사 후 ESI, EDI는 각 4바이트 뒤를 가리킴
00401095      MOVSD
00401096      MOVSW          ; ESI가 가리키는 주소에서 EDI가 가리키는 주소로 2바이트 복사
                             ; 복사 후 ESI, EDI는 각 2바이트 뒤를 가리킴
00401098      XOR EBX,EBX
0040109A      XOR ESI,ESI
0040109C      CMP BYTE PTR SS:[EBP-28],BL
0040109F      JE SHORT MyCrackI.004010A8
004010A1      INC ESI                                        ..... (1)
004010A2      CMP BYTE PTR SS:[EBP+ESI-28],BL
004010A6      JNZ SHORT MyCrackI.004010A1
004010A8      XOR ECX,ECX
004010AA      MOV EAX,ECX
004010AC      CDQ            ; EAX의 값을 EDX-EAX 조합으로 확장(32->64)
004010AD      IDIV ESI       ; EDX-EAX를 ESI로 나누어 몫을 EAX에 나머지를 EDX에 저장
004010AF      MOV AL,BYTE PTR SS:[EBP+EDX-28]
004010B3      XOR BYTE PTR SS:[EBP+ECX-18],AL
004010B7      INC ECX
004010B8      CMP ECX,0B
004010BB      JL SHORT MyCrackI.004010AA
```

```
004010BD      PUSH MyCrackI.00402070
004010C2      CALL DWORD PTR DS:[<&KERNEL32.GetModuleHandleA>]
004010C8      CMP EAX,EBX
004010CA      JE MyCrackI.0040116F                           ..... (2)
004010D0      LEA ECX,DWORD PTR SS:[EBP-18]
004010D3      PUSH ECX
004010D4      PUSH EAX
004010D5      CALL DWORD PTR DS:[<&KERNEL32.GetProcAddress>]
```

```
004010DB      CMP EAX,EBX                          ; 주소번지가 NULL인지 비교 (오류검사)
004010DD      MOV DWORD PTR SS:[EBP+10],EAX        ; 함수의 주소번지를 wParam에 저장
004010E0      JE MyCrackI.0040116F                 ; 오류 시 오류처리부분인 0040116F로 이동
```

위의 (1)번 구간의 코드를 분석하면 Var_3에 메모리 주소 00402078번에 저장된 값을 할당하는 작업을 한 뒤 이 값과 변수 Var_2에 Edit2 편집상자에서 얻은 값을 XOR 연산을 하게된다. 이 때 총 11번

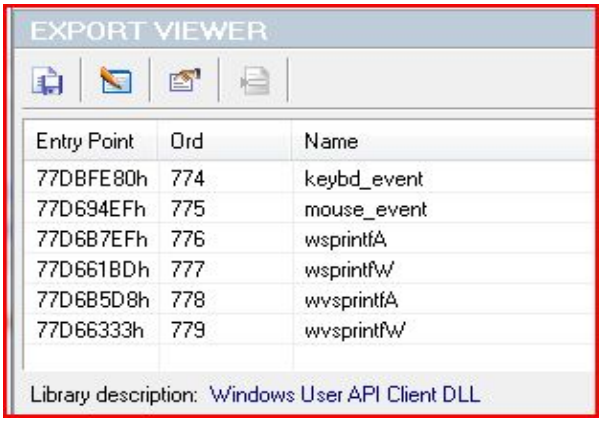
의 작업을 하게 되는데 결과 값은 다시 Var_2에 저장이 된다. 이 과정이 0040108B ~ 004010B8 번까지이다. 004010BB번에서 Var_2에 저장된 값을 살펴보면 Edit2 편집상자에 넣은 문자열과 Var_3에 들어있는 문자열을 XOR 시켜 생성된 문자열이 들어가 있게 된다. Edit2편집상자에 information을 넣었을 경우 MessageBoxA라는 문자열이 저장된다.

이 후 (2)번 구간은 00402070번 메모리번지에 저장된 "user32" 문자열을 스택에 넣고 GetModuleHandleA 함수를 호출한다. 이는 user32.dll 의 핸들을 얻기 위한 것으로 핸들은 EAX 레지스터에 저장된다. 세 번째, 네 번째 줄은 핸들을 제대로 얻지 못하였을 때의 오류 처리 루틴이다. 다섯 번째 줄에서 마지막 줄 까지는 GetProcAddress 함수를 호출 하는 것으로 함수의 인자로 필요한 모듈의 핸들과 함수의 이름을 넣는데 모듈의 핸들은 GetModuleHandleA 함수를 실행하고 난 뒤에 EAX에 저장되어 있고, 함수명은 Var_2 (SS:[EBP-18])에 "MessageBoxA"로 저장되어 있으므로 이를 스택에 넣고 나서 CALL 명령을 통해 GetProcAddress 함수를 호출한다. GetModuleHandle과 GetProcAddress 의 함수원형은 다음과 간다.

```
HMODULE WINAPI GetModuleHandle( __in_opt  LPCTSTR lpModuleName);

FARPROC WINAPI GetProcAddress( __in  HMODULE hModule,  __in  LPCSTR lpProcName);
```

즉, 위의 과정을 모두 거치고 난 뒤 EAX 레지스터에는 GetProcAddress 함수의 반환 값인 MessageBoxA 의 주소번지를 저장하고 있다.

여기서 우리는 두 번째 편집상자인 Edit2에 들어갈 문자열을 추측하는 과정이 있는데 이를 어떻게 추론해 냈는지 확인하지 않고 넘어왔다. 문자열 information이 들어가야 한다는 것은 위의 코드 분석 과정에서 Edit2 편집상자의 문자열과 조합하여 나온 문자열(Var_2 에 저장된 문자열)이 user32.dll의 export table에 저장되어 있어야 함을 추론할 수 있다. 또한, 변환과정에서 0x0B번 반복하여 문자열의 크기가 11글자임을 유추할 수 있다. 그리고, 이 중에서 함수를 호출할 때 사용되는 인자의 개수가 분석한 자료와 일치하는 API를 찾으면 된다. PE Explorer를 통해서 살펴본 user32.dll 의 export table에 존재하는 함수는 총 779개 임을 확인 할 수 있다.



[그림 2-6] user32.dll 의 Export 함수 목록

```
004010E6      MOV ESI,MyCrackI.0040205C
004010EB      LEA EDI,DWORD PTR SS:[EBP-4C]
004010EE      MOVSD
004010EF      MOVSD
004010F0      MOVSD
004010F1      MOVSD
004010F2      MOVSW
004010F4      XOR EAX,EAX
004010F6      CMP BYTE PTR SS:[EBP-C],BL
004010F9      LEA EDI,DWORD PTR SS:[EBP-3A]
004010FC      STOSD
004010FD      STOSD
004010FE      STOSD
004010FF      STOSW
00401101      STOSB
00401102      MOV ESI,MyCrackI.0040204C
00401107      LEA EDI,DWORD PTR SS:[EBP-28]
0040110A      MOVSD
0040110B      MOVSD
0040110C      MOVSD
0040110D      MOVSD
0040110E      MOV DWORD PTR SS:[EBP+C],EBX
00401111      JE SHORT MyCrackI.0040111F
00401113      INC DWORD PTR SS:[EBP+C]
00401116      MOV EAX,DWORD PTR SS:[EBP+C]
00401119      CMP BYTE PTR SS:[EBP+EAX-C],BL
0040111D      JNZ SHORT MyCrackI.00401113
0040111F      XOR ESI,ESI
00401121      CMP BYTE PTR SS:[EBP-18],BL
00401124      JE SHORT MyCrackI.0040112D
00401126      INC ESI
00401127      CMP BYTE PTR SS:[EBP+ESI-18],BL
0040112B      JNZ SHORT MyCrackI.00401126
0040112D      XOR ECX,ECX
0040112F      MOV EAX,ECX
00401131      CDQ
00401132      IDIV ESI
00401134      MOV EAX,ECX
```

```
00401136       MOVSX EDI,BYTE PTR SS:[EBP+EDX-18]
0040113B       CDQ
0040113C       IDIV DWORD PTR SS:[EBP+C]
0040113F       MOVSX EAX,BYTE PTR SS:[EBP+EDX-C]
00401144       CDQ
00401145       IDIV EDI
00401147       XOR BYTE PTR SS:[EBP+ECX-28],DL
0040114B       INC ECX
0040114C       CMP ECX,0F
0040114F       JL SHORT MyCrackI.0040112F
00401151       LEA EAX,DWORD PTR SS:[EBP-28]
00401154       PUSH EAX
00401155       LEA EAX,DWORD PTR SS:[EBP-4C]
00401158       PUSH EAX
00401159       CALL DWORD PTR DS:[<&KERNEL32.lstrcatA>]
0040115F       PUSH EBX
00401160       PUSH MyCrackI.00402040
00401165       LEA EAX,DWORD PTR SS:[EBP-4C]
00401168       PUSH EAX
00401169       PUSH EBX
0040116A       CALL DWORD PTR SS:[EBP+10]
0040116D       JMP SHORT MyCrackI.00401181
```

위의 역어셈블 코드는 Edit1, Edit2 편집상자에 입력된 값이 맞을 경우 메시지박스에 출력할 패스워드를 만들어서 메시지박스로 넣는 과정을 표현한 것이다.

분석자료는 추가 작성하여 올릴 예정이며 시험범위는 현재 분석된 자료까지로 제한할 것이니 참고바람.

[별첨 1] 윈도우 메시지

```
/*
 * Window Messages
 */

#define WM_NULL                         0x0000
#define WM_CREATE                       0x0001
#define WM_DESTROY                      0x0002
#define WM_MOVE                         0x0003
#define WM_SIZE                         0x0005

#define WM_ACTIVATE                     0x0006
/*
 * WM_ACTIVATE state values
 */
#define     WA_INACTIVE     0
#define     WA_ACTIVE       1
#define     WA_CLICKACTIVE  2

#define WM_SETFOCUS                     0x0007
#define WM_KILLFOCUS                    0x0008
#define WM_ENABLE                       0x000A
#define WM_SETREDRAW                    0x000B
#define WM_SETTEXT                      0x000C
#define WM_GETTEXT                      0x000D
#define WM_GETTEXTLENGTH                0x000E
#define WM_PAINT                        0x000F
#define WM_CLOSE                        0x0010
#define WM_QUERYENDSESSION              0x0011
#define WM_QUIT                         0x0012
#define WM_QUERYOPEN                    0x0013
#define WM_ERASEBKGND                   0x0014
#define WM_SYSCOLORCHANGE               0x0015
#define WM_ENDSESSION                   0x0016
#define WM_SHOWWINDOW                   0x0018
#define WM_WININICHANGE                 0x001A
```

```
#if(WINVER >= 0x0400)
#define WM_SETTINGCHANGE            WM_WININICHANGE
#endif /* WINVER >= 0x0400 */


#define WM_DEVMODECHANGE             0x001B
#define WM_ACTIVATEAPP               0x001C
#define WM_FONTCHANGE                0x001D
#define WM_TIMECHANGE                0x001E
#define WM_CANCELMODE                0x001F
#define WM_SETCURSOR                 0x0020
#define WM_MOUSEACTIVATE             0x0021
#define WM_CHILDACTIVATE             0x0022
#define WM_QUEUESYNC                 0x0023

#define WM_GETMINMAXINFO             0x0024
// end_r_winuser
/*
 * Struct pointed to by WM_GETMINMAXINFO lParam
 */
typedef struct tagMINMAXINFO {
    POINT ptReserved;
    POINT ptMaxSize;
    POINT ptMaxPosition;
    POINT ptMinTrackSize;
    POINT ptMaxTrackSize;
} MINMAXINFO, *PMINMAXINFO, *LPMINMAXINFO;

// begin_r_winuser
#define WM_PAINTICON                 0x0026
#define WM_ICONERASEBKGND            0x0027
#define WM_NEXTDLGCTL                0x0028
#define WM_SPOOLERSTATUS             0x002A
#define WM_DRAWITEM                  0x002B
#define WM_MEASUREITEM               0x002C
#define WM_DELETEITEM                0x002D
#define WM_VKEYTOITEM                0x002E
#define WM_CHARTOITEM                0x002F
```

```
#define WM_SETFONT                  0x0030
#define WM_GETFONT                  0x0031
#define WM_SETHOTKEY                0x0032
#define WM_GETHOTKEY                0x0033
#define WM_QUERYDRAGICON            0x0037
#define WM_COMPAREITEM              0x0039
#if(WINVER >= 0x0500)
#define WM_GETOBJECT                0x003D
#endif /* WINVER >= 0x0500 */
#define WM_COMPACTING               0x0041
#define WM_COMMNOTIFY               0x0044  /* no longer suported */
#define WM_WINDOWPOSCHANGING        0x0046
#define WM_WINDOWPOSCHANGED         0x0047

#define WM_POWER                    0x0048
/*
 * wParam for WM_POWER window message and DRV_POWER driver notification
 */
#define PWR_OK            1
#define PWR_FAIL          (-1)
#define PWR_SUSPENDREQUEST  1
#define PWR_SUSPENDRESUME   2
#define PWR_CRITICALRESUME  3

#define WM_COPYDATA                 0x004A
#define WM_CANCELJOURNAL            0x004B

// end_r_winuser

/*
 * lParam of WM_COPYDATA message points to...
 */
typedef struct tagCOPYDATASTRUCT {
   DWORD dwData;
   DWORD cbData;
   PVOID lpData;
} COPYDATASTRUCT, *PCOPYDATASTRUCT;
```

```
// begin_r_winuser

#if(WINVER >= 0x0400)
#define WM_NOTIFY                   0x004E
#define WM_INPUTLANGCHANGEREQUEST      0x0050
#define WM_INPUTLANGCHANGE           0x0051
#define WM_TCARD                    0x0052
#define WM_HELP                     0x0053
#define WM_USERCHANGED               0x0054
#define WM_NOTIFYFORMAT              0x0055

#define NFR_ANSI                        1
#define NFR_UNICODE                      2
#define NF_QUERY                        3
#define NF_REQUERY                       4

#define WM_CONTEXTMENU               0x007B
#define WM_STYLECHANGING             0x007C
#define WM_STYLECHANGED              0x007D
#define WM_DISPLAYCHANGE             0x007E
#define WM_GETICON                  0x007F
#define WM_SETICON                  0x0080
#endif /* WINVER >= 0x0400 */

#define WM_NCCREATE                 0x0081
#define WM_NCDESTROY                0x0082
#define WM_NCCALCSIZE               0x0083
#define WM_NCHITTEST                0x0084
#define WM_NCPAINT                  0x0085
#define WM_NCACTIVATE               0x0086
#define WM_GETDLGCODE               0x0087
#define WM_SYNCPAINT                0x0088
#define WM_NCMOUSEMOVE              0x00A0
#define WM_NCLBUTTONDOWN             0x00A1
#define WM_NCLBUTTONUP              0x00A2
#define WM_NCLBUTTONDBLCLK           0x00A3
#define WM_NCRBUTTONDOWN             0x00A4
#define WM_NCRBUTTONUP              0x00A5
```

```
#define WM_NCRBUTTONDBLCLK            0x00A6
#define WM_NCMBUTTONDOWN              0x00A7
#define WM_NCMBUTTONUP                0x00A8
#define WM_NCMBUTTONDBLCLK            0x00A9

#define WM_KEYFIRST                   0x0100
#define WM_KEYDOWN                    0x0100
#define WM_KEYUP                      0x0101
#define WM_CHAR                       0x0102
#define WM_DEADCHAR                   0x0103
#define WM_SYSKEYDOWN                 0x0104
#define WM_SYSKEYUP                   0x0105
#define WM_SYSCHAR                    0x0106
#define WM_SYSDEADCHAR                0x0107
#define WM_KEYLAST                    0x0108

#if(WINVER >= 0x0400)
#define WM_IME_STARTCOMPOSITION       0x010D
#define WM_IME_ENDCOMPOSITION         0x010E
#define WM_IME_COMPOSITION            0x010F
#define WM_IME_KEYLAST                0x010F
#endif /* WINVER >= 0x0400 */

#define WM_INITDIALOG                 0x0110
#define WM_COMMAND                    0x0111
#define WM_SYSCOMMAND                 0x0112
#define WM_TIMER                      0x0113
#define WM_HSCROLL                    0x0114
#define WM_VSCROLL                    0x0115
#define WM_INITMENU                   0x0116
#define WM_INITMENUPOPUP              0x0117
#define WM_MENUSELECT                 0x011F
#define WM_MENUCHAR                   0x0120
#define WM_ENTERIDLE                  0x0121
#if(WINVER >= 0x0500)
#define WM_MENURBUTTONUP              0x0122
#define WM_MENUDRAG                   0x0123
#define WM_MENUGETOBJECT              0x0124
```

```
#define WM_UNINITMENUPOPUP            0x0125
#define WM_MENUCOMMAND               0x0126
#endif /* WINVER >= 0x0500 */


#define WM_CTLCOLORMSGBOX             0x0132
#define WM_CTLCOLOREDIT              0x0133
#define WM_CTLCOLORLISTBOX            0x0134
#define WM_CTLCOLORBTN               0x0135
#define WM_CTLCOLORDLG               0x0136
#define WM_CTLCOLORSCROLLBAR          0x0137
#define WM_CTLCOLORSTATIC             0x0138


#define WM_MOUSEFIRST                0x0200
#define WM_MOUSEMOVE                 0x0200
#define WM_LBUTTONDOWN               0x0201
#define WM_LBUTTONUP                 0x0202
#define WM_LBUTTONDBLCLK              0x0203
#define WM_RBUTTONDOWN               0x0204
#define WM_RBUTTONUP                 0x0205
#define WM_RBUTTONDBLCLK              0x0206
#define WM_MBUTTONDOWN               0x0207
#define WM_MBUTTONUP                 0x0208
#define WM_MBUTTONDBLCLK              0x0209

#if (_WIN32_WINNT >= 0x0400) || (_WIN32_WINDOWS > 0x0400)
#define WM_MOUSEWHEEL                0x020A
#define WM_MOUSELAST                 0x020A
#else
#define WM_MOUSELAST                 0x0209
#endif /* if (_WIN32_WINNT < 0x0400) */

#if(_WIN32_WINNT >= 0x0400)
#define WHEEL_DELTA                  120    /* Value for rolling one detent */
#endif /* _WIN32_WINNT >= 0x0400 */
#if(_WIN32_WINNT >= 0x0400)
#define WHEEL_PAGESCROLL              (UINT_MAX) /* Scroll one page */
```

```
#endif /* _WIN32_WINNT >= 0x0400 */

#define WM_PARENTNOTIFY              0x0210
#define WM_ENTERMENULOOP             0x0211
#define WM_EXITMENULOOP              0x0212

#if(WINVER >= 0x0400)
#define WM_NEXTMENU                  0x0213
// end_r_winuser

typedef struct tagMDINEXTMENU
{
    HMENU   hmenuIn;
    HMENU   hmenuNext;
    HWND    hwndNext;
} MDINEXTMENU, * PMDINEXTMENU, FAR * LPMDINEXTMENU;

// begin_r_winuser
#define WM_SIZING                    0x0214
#define WM_CAPTURECHANGED             0x0215
#define WM_MOVING                    0x0216
// end_r_winuser
#define WM_POWERBROADCAST             0x0218     // r_winuser pbt
// begin_pbt

#define PBT_APMQUERYSUSPEND           0x0000
#define PBT_APMQUERYSTANDBY           0x0001

#define PBT_APMQUERYSUSPENDFAILED      0x0002
#define PBT_APMQUERYSTANDBYFAILED      0x0003

#define PBT_APMSUSPEND               0x0004
#define PBT_APMSTANDBY               0x0005

#define PBT_APMRESUMECRITICAL         0x0006
#define PBT_APMRESUMESUSPEND          0x0007
#define PBT_APMRESUMESTANDBY          0x0008
```

```
#define PBTF_APMRESUMEFROMFAILURE      0x00000001

#define PBT_APMBATTERYLOW             0x0009
#define PBT_APMPOWERSTATUSCHANGE       0x000A

#define PBT_APMOEMEVENT               0x000B
#define PBT_APMRESUMEAUTOMATIC         0x0012
// end_pbt

// begin_r_winuser
#define WM_DEVICECHANGE               0x0219

#endif /* WINVER >= 0x0400 */

#define WM_MDICREATE                  0x0220
#define WM_MDIDESTROY                 0x0221
#define WM_MDIACTIVATE                0x0222
#define WM_MDIRESTORE                 0x0223
#define WM_MDINEXT                    0x0224
#define WM_MDIMAXIMIZE                0x0225
#define WM_MDITILE                    0x0226
#define WM_MDICASCADE                 0x0227
#define WM_MDIICONARRANGE              0x0228
#define WM_MDIGETACTIVE               0x0229


#define WM_MDISETMENU                 0x0230
#define WM_ENTERSIZEMOVE              0x0231
#define WM_EXITSIZEMOVE               0x0232
#define WM_DROPFILES                  0x0233
#define WM_MDIREFRESHMENU             0x0234


#if(WINVER >= 0x0400)
#define WM_IME_SETCONTEXT             0x0281
#define WM_IME_NOTIFY                 0x0282
#define WM_IME_CONTROL                0x0283
#define WM_IME_COMPOSITIONFULL         0x0284
```

```
#define WM_IME_SELECT                   0x0285
#define WM_IME_CHAR                     0x0286
#endif /* WINVER >= 0x0400 */
#if(WINVER >= 0x0500)
#define WM_IME_REQUEST                  0x0288
#endif /* WINVER >= 0x0500 */
#if(WINVER >= 0x0400)
#define WM_IME_KEYDOWN                  0x0290
#define WM_IME_KEYUP                    0x0291
#endif /* WINVER >= 0x0400 */

#if(_WIN32_WINNT >= 0x0400)
#define WM_MOUSEHOVER                   0x02A1
#define WM_MOUSELEAVE                   0x02A3
#endif /* _WIN32_WINNT >= 0x0400 */

#define WM_CUT                          0x0300
#define WM_COPY                         0x0301
#define WM_PASTE                        0x0302
#define WM_CLEAR                        0x0303
#define WM_UNDO                         0x0304
#define WM_RENDERFORMAT                 0x0305
#define WM_RENDERALLFORMATS             0x0306
#define WM_DESTROYCLIPBOARD             0x0307
#define WM_DRAWCLIPBOARD                0x0308
#define WM_PAINTCLIPBOARD               0x0309
#define WM_VSCROLLCLIPBOARD             0x030A
#define WM_SIZECLIPBOARD                0x030B
#define WM_ASKCBFORMATNAME              0x030C
#define WM_CHANGECBCHAIN                0x030D
#define WM_HSCROLLCLIPBOARD             0x030E
#define WM_QUERYNEWPALETTE              0x030F
#define WM_PALETTEISCHANGING            0x0310
#define WM_PALETTECHANGED               0x0311
#define WM_HOTKEY                       0x0312

#if(WINVER >= 0x0400)
#define WM_PRINT                        0x0317
```

```
#define WM_PRINTCLIENT                0x0318

#define WM_HANDHELDFIRST              0x0358
#define WM_HANDHELDLAST               0x035F

#define WM_AFXFIRST                   0x0360
#define WM_AFXLAST                    0x037F
#endif /* WINVER >= 0x0400 */

#define WM_PENWINFIRST                0x0380
#define WM_PENWINLAST                 0x038F


#if(WINVER >= 0x0400)
#define WM_APP                        0x8000
#endif /* WINVER >= 0x0400 */

/*
 * NOTE: All Message Numbers below 0x0400 are RESERVED.
 *
 * Private Window Messages Start Here:
 */
#define WM_USER                       0x0400

#if(WINVER >= 0x0400)

/*  wParam for WM_SIZING message  */
#define WMSZ_LEFT         1
#define WMSZ_RIGHT        2
#define WMSZ_TOP          3
#define WMSZ_TOPLEFT      4
#define WMSZ_TOPRIGHT     5
#define WMSZ_BOTTOM       6
#define WMSZ_BOTTOMLEFT   7
#define WMSZ_BOTTOMRIGHT  8
#endif /* WINVER >= 0x0400 */

#ifndef NONCMESSAGES
```

[별첨 2]