Beist
Security Study Group

초보자를 위한 예제와 함께
배워보는 OllyDbg사용법
-1부-

By Beist Security Study Group
(http://beist.org)

요약: 이 문서는 Ollydbg 프로그램을 이용하여 Reverse Engineering을 하는 방법에 대해서 다룬다. 초보자를 위하여 작성된 문서이며 예제와 함께 Ollydbg의 각 기능에 대해서 알아본다. 주로 기초적인 내용을 다루고 있다.

0. Prolog

이 강좌는 Debugging과 Reverse Engineering을 하기 위한 강력하고 효과적인 프로그램인 OllyDbg의 사용법을 제공하기 위해 만들어졌습니다. OllyDbg를 사용하여 임시 예제의 소스와 프로그램을 분석하면서 OllyDbg의 사용방법을 하나씩 알아보는 방법으로 설명하겠습니다.

1. Debug & Debugging

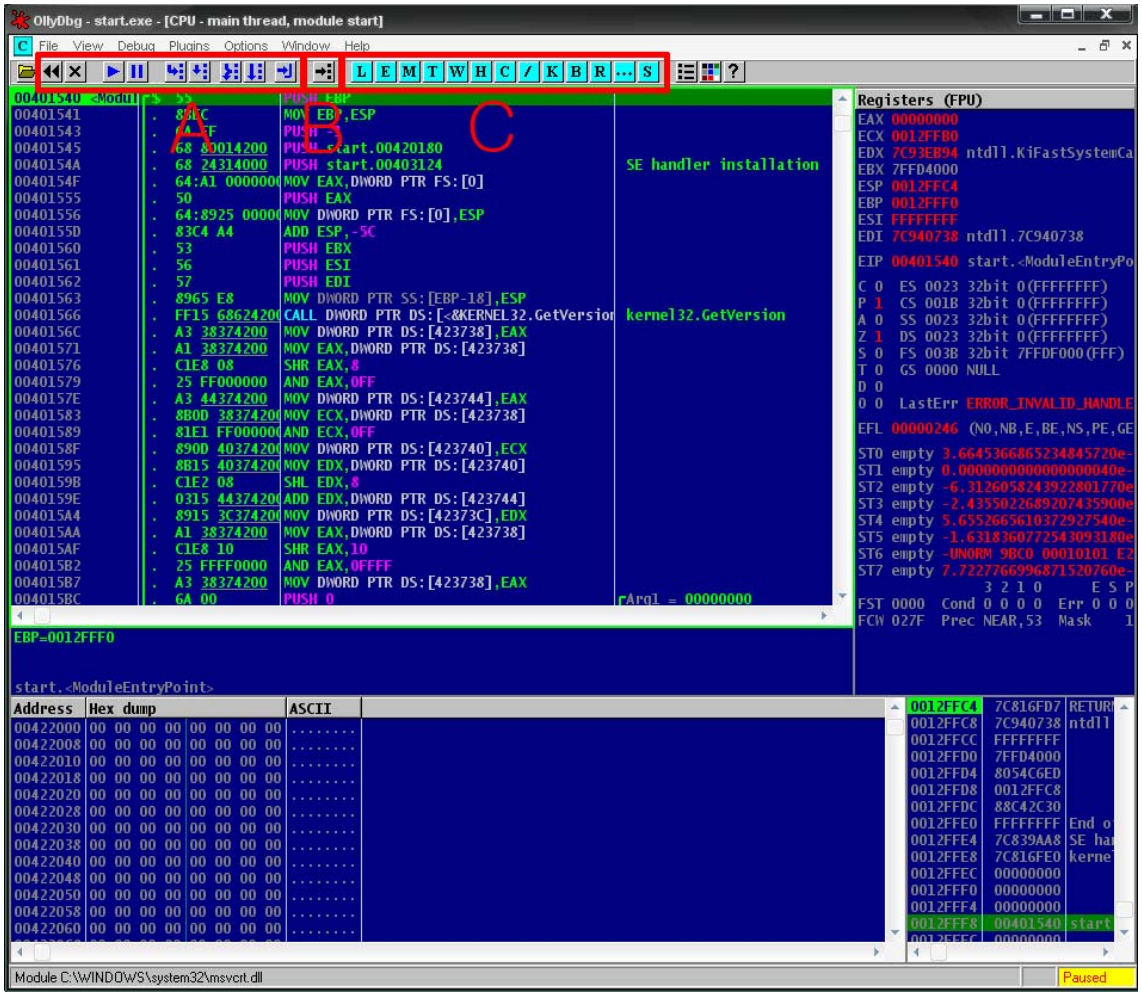
Debug란 프로그래머가 자신이 만든 프로그램의 오류를 찾고 그것을 해결하는 과정을 말합니다. Debug를 하기 위해서 3가지 방법이 있습니다.

1. 프로그래머의 눈으로 소스 코드의 내용을 일일이 따라가면서 오류를 찾아내는 방법이 있습니다. 하지만 이 방법은 시간과 체력을 많이 소진하고 프로그램이 커질수록 힘들어지는 경우가 많습니다.
2. 오류로 예상되는 부분의 앞뒤에 printf와 같은 함수를 사용하면서 프로그램의 실행 구조를 파악하는 방법입니다. 사람이 계산하는 것보다 쉽게 할 수 있습니다.
3. Debugging을 도와주는 프로그램을 이용하는 방법입니다. 디버깅 툴로는(gdb, softice, ollydbg 등이) 있습니다.

위 3번째에서 언급한 Debugging 툴을 이용하여 작업하는 것이 효율적입니다.

2. OllyDbg 메뉴

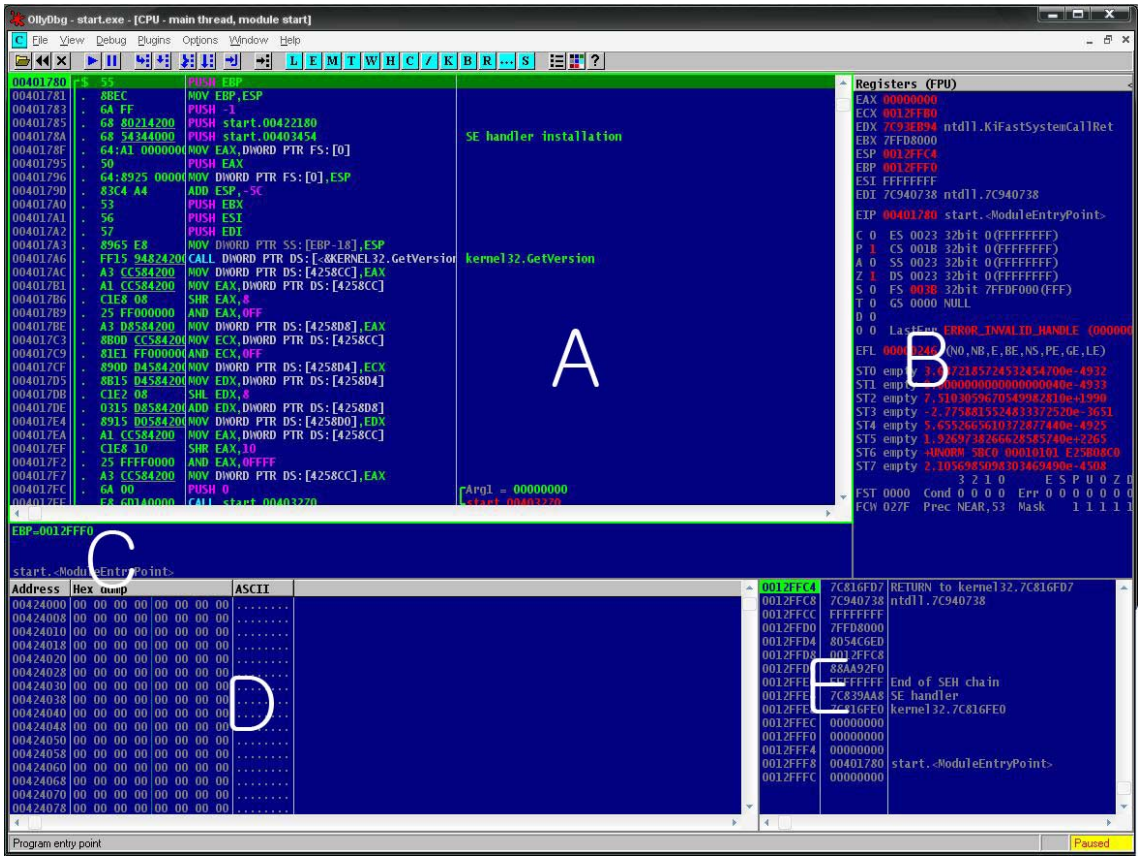
-2.1 Tool Bar



[그림 1]

[A]부분은 Debug툴 바 입니다. 프로그램을 restart, close, run 시키거나, 프로세스의 실행 단계에 대해서 조작할 수 있는 기능들을 제공합니다. [B]부분은 입력된 주소 값으로 이동해서 disassemble코드를 보여주는 역할을 합니다. [C]부분은 디버그하고 있는 프로그램에 대한 여러 정보를 윈도우 형식으로 표시해줍니다. thread, window, call stack, break point등의 정보를 볼 수 있습니다.

-2.2 작업 영역 Interface



[그림 2]

각각 5개의 부분으로 나눠져 있는 것을 볼 수 있습니다. 먼저 [A]부분은 disassembling된 코드와 OP코드를 나타내고 있는 CODE부분입니다. 여기서 프로그램의 흐름을 체크하고 분석합니다.

[B]부분은 CPU의 레지스터 상태를 나타냅니다. 이 부분을 이용해 각종 레지스터의 값이나 Flag를 직접 조작할 수도 있습니다. 예를 들어 Zero flag의 값을 변경해 다른 분기로도 점프 해보면서 프로그램을 분석해 나갈 수도 있습니다.

[C]부분은 상태 바 입니다. [A] 부분에서 실행되고 있는 각 해당 위치의 offset값과 변경된 메모리 주소, 레지스터의 내용 등을 나타내줍니다. [D]부분은 메모리의 각 값들을 HEX 코드와 ASCII 코드로 보여주는 부분으로, 메뉴 설정을 통해 다양한 데이터 형태로 값들을 확인할 수 있습니다.

[E]부분은 Stack의 내용을 보여주고 있습니다. 이 부분을 통해 원하는 주소의 Stack 영역의 값들을 확인 할 수 있습니다. [A], [D], [E] 각 창들은 인터페이스가 조금씩 차이가 있지만 기본적으로 프로세스의 모든 메모리 영역에 접근할 수 있다는 공통점이 있습니다.

3. 리버싱을 통해 알아보는 search & break point

-3.1 Test Code

아래 소스 코드는 사용자로부터 패스워드를 체크하는 간단한 프로그램입니다. 만약 패스워드를 맞추면 BINGO라는 메시지를 볼 수 있습니다.

```
#include <windows.h>
#include <stdlib.h>
#include <time.h>

#define ID_BUTTON1 104
#define ID_EDIT1 105
#define MAX_STRING 256

LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);

HINSTANCE g_hInst;
HWND hWndMain;
LPCTSTR lpszClass=TEXT("OllyTest");
HWND hwndEdit;
HWND hwndButton1;
char tmp_buf[MAX_STRING];
char str_guess[10]= "1ab23";



int APIENTRY WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance
                     ,LPSTR lpszCmdParam,int nCmdShow){
        HWND hWnd;
        MSG Message;
        WNDCLASS WndClass;
        g_hInst=hInstance;

        WndClass.cbClsExtra=0;
        WndClass.cbWndExtra=0;
```

```
	WndClass.hbrBackground=(HBRUSH)GetStockObject(WHITE_BRUSH);
	WndClass.hCursor=LoadCursor(NULL,IDC_ARROW);
	WndClass.hIcon=LoadIcon(NULL,IDI_APPLICATION);
	WndClass.hInstance=hInstance;
	WndClass.lpfnWndProc=(WNDPROC)WndProc;
	WndClass.lpszClassName=lpszClass;
	WndClass.lpszMenuName=NULL;
	WndClass.style=CS_HREDRAW | CS_VREDRAW;
	RegisterClass(&WndClass);

	hWnd=CreateWindow(lpszClass,lpszClass,WS_OVERLAPPEDWINDOW,
		0,0,180,150,
		NULL,NULL,hInstance,NULL);
	ShowWindow(hWnd,nCmdShow);


	while(GetMessage(&Message,0,0,0)) {
		TranslateMessage(&Message);
		DispatchMessage(&Message);
	}
	return Message.wParam;
}

LRESULT CALLBACK WndProc(HWND hWnd,UINT iMessage,WPARAM wParam,LPARAM
lParam){
	switch(iMessage){
	case WM_CREATE:
		CreateWindow(TEXT("static"),"Type password",WS_CHILD | WS_VISIBLE,
			10,10,150,30,hWnd,(HMENU)0, g_hInst, NULL);
		hwndEdit=CreateWindow(TEXT("edit"),NULL,WS_CHILD | WS_VISIBLE |
WS_BORDER | ES_LEFT | ES_MULTILINE |ES_AUTOVSCROLL,

	10,40,150,30,hWnd,(HMENU)ID_EDIT1,g_hInst,NULL);

	hwndButton1=CreateWindow(TEXT("button"),"Try",WS_CHILD|WS_VISIBLE|BS_PU
SHBUTTON,
```

```
        10,70,150,30,hWnd,(HMENU)ID_BUTTON1,g_hInst,NULL);

        return 0;


        case WM_COMMAND:
                switch(LOWORD(wParam)){
                case ID_BUTTON1:
                        GetDlgItemText(hWnd,ID_EDIT1,tmp_buf,MAX_STRING);
                        if(!strcmp(str_guess,tmp_buf)){
                                MessageBox(NULL,"BINGO","successful try",MB_OK);
                        }else{
                                MessageBox(NULL,"Try again","Mission failed",MB_OK);
                        }

                        break;

                }
                return 0;

        case WM_DESTROY:
                PostQuitMessage(0);
                return 0;
        }
        return(DefWindowProc(hWnd,iMessage,wParam,lParam));
}
```

-3.2 리버싱 시나리오

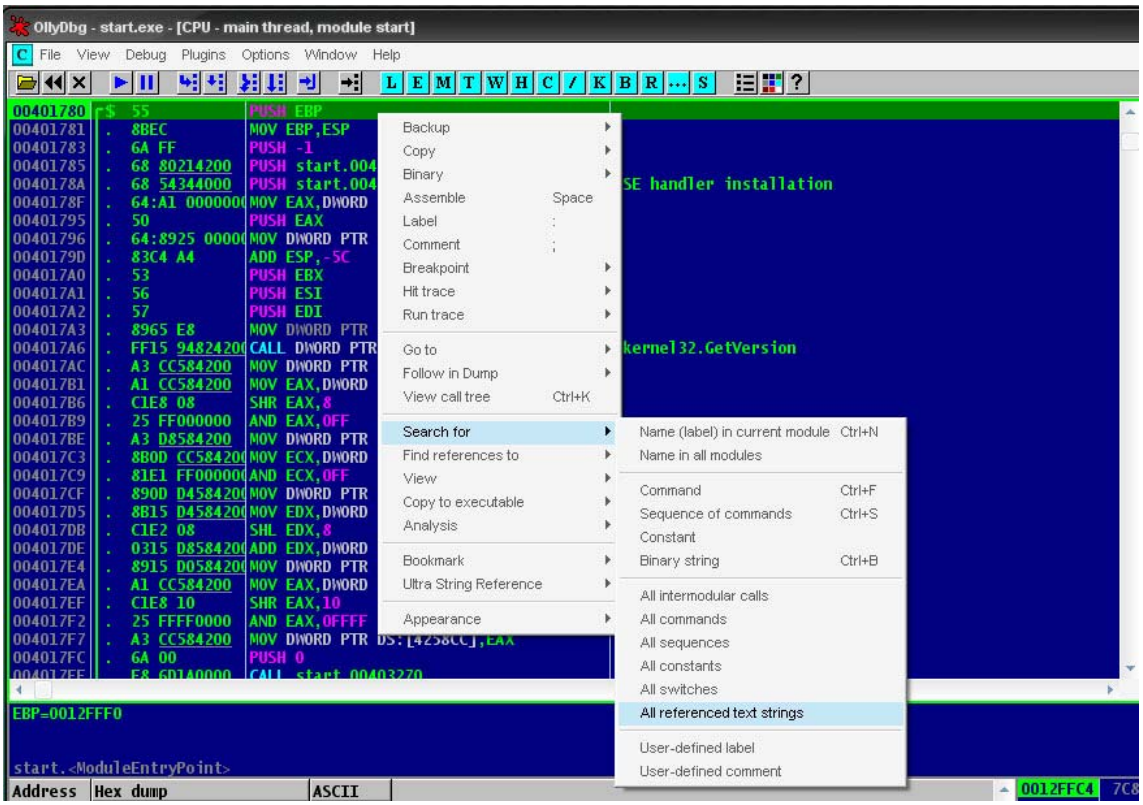
① 위 프로그램을 실행해 잘못된 패스워드를 입력하면 “Try again” 메시지를, 올바른 패스워드를 입력하면 “BINGO” 메시지를 볼 수 있습니다.

② 우리는 프로그램을 직접 실행해봄으로써 문자열을 띄우는 함수가 MessageBox 라는 함수인 것을 추측할 수 있습니다. 또한 “Try again”이라는 문자열은 MessageBox 함수의 인자로 들어가리라는 것도 추측할 수 있습니다.

③ OllyDbg를 실행시키고 위 프로그램을 로딩한 후 “Try again”이라는 문자열을 사용하는 코드가 있는지 찾아봅시다.
④ “Try again” 문자열을 사용하는 MessageBox 함수를 호출하는 코드를 찾았다면, 호출하는 코드가 불리기 전에 사용자가 입력한 문자열과 암호를 비교하는 코드를 찾아야 합니다.
⑤ 두 문자열을 비교하는 곳에서 암호 문자열과 암호 문자열의 위치를 알 수 있을 것입니다.

-3.2 Search & break point

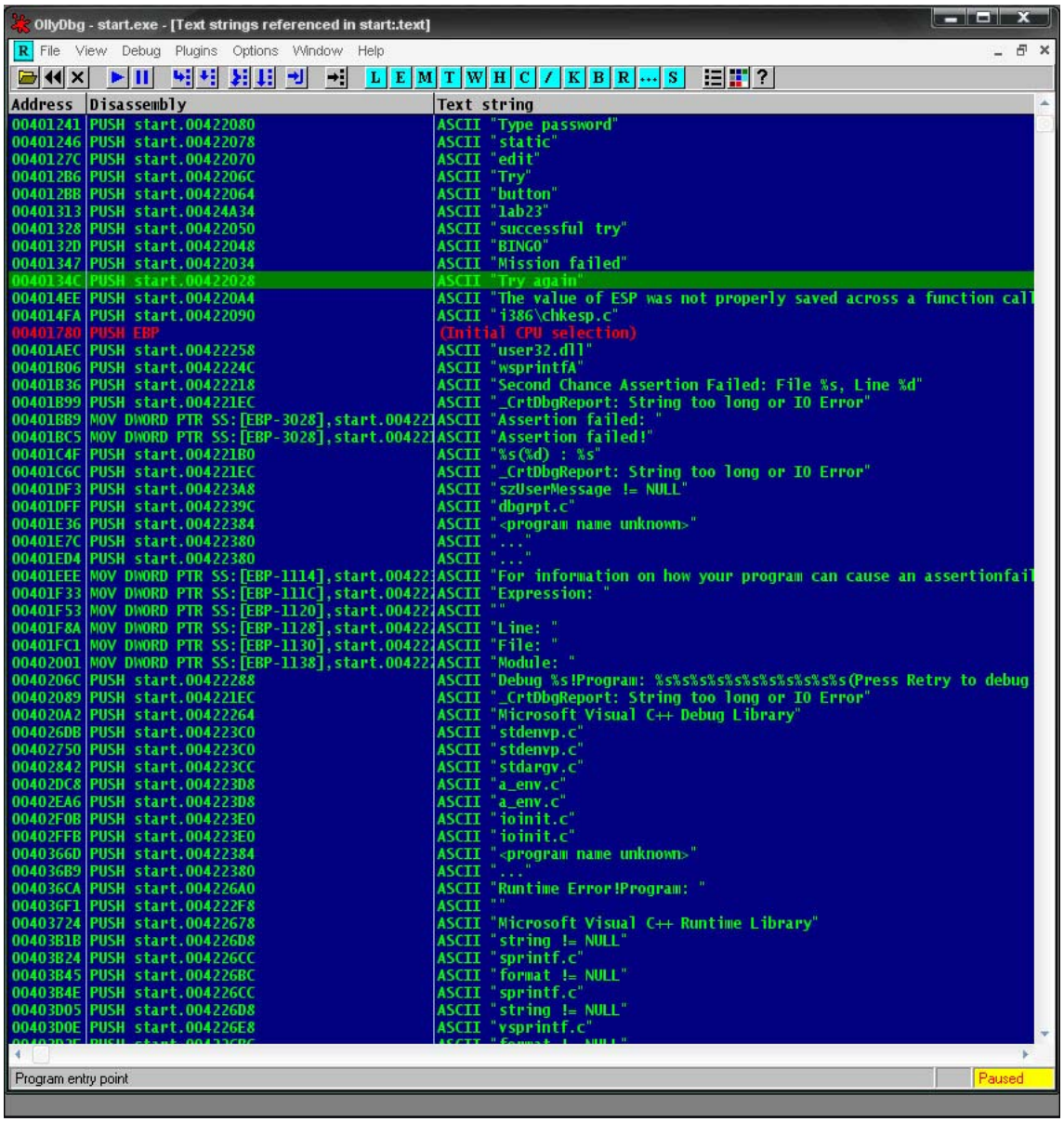
이제 위 시나리오에 따라 하나씩 진행해 나가겠습니다. OllyDbg를 실행시키고, 우리가 위 소스 코드를 컴파일 한 파일을 선택해서 열어줍니다.



[그림 3]

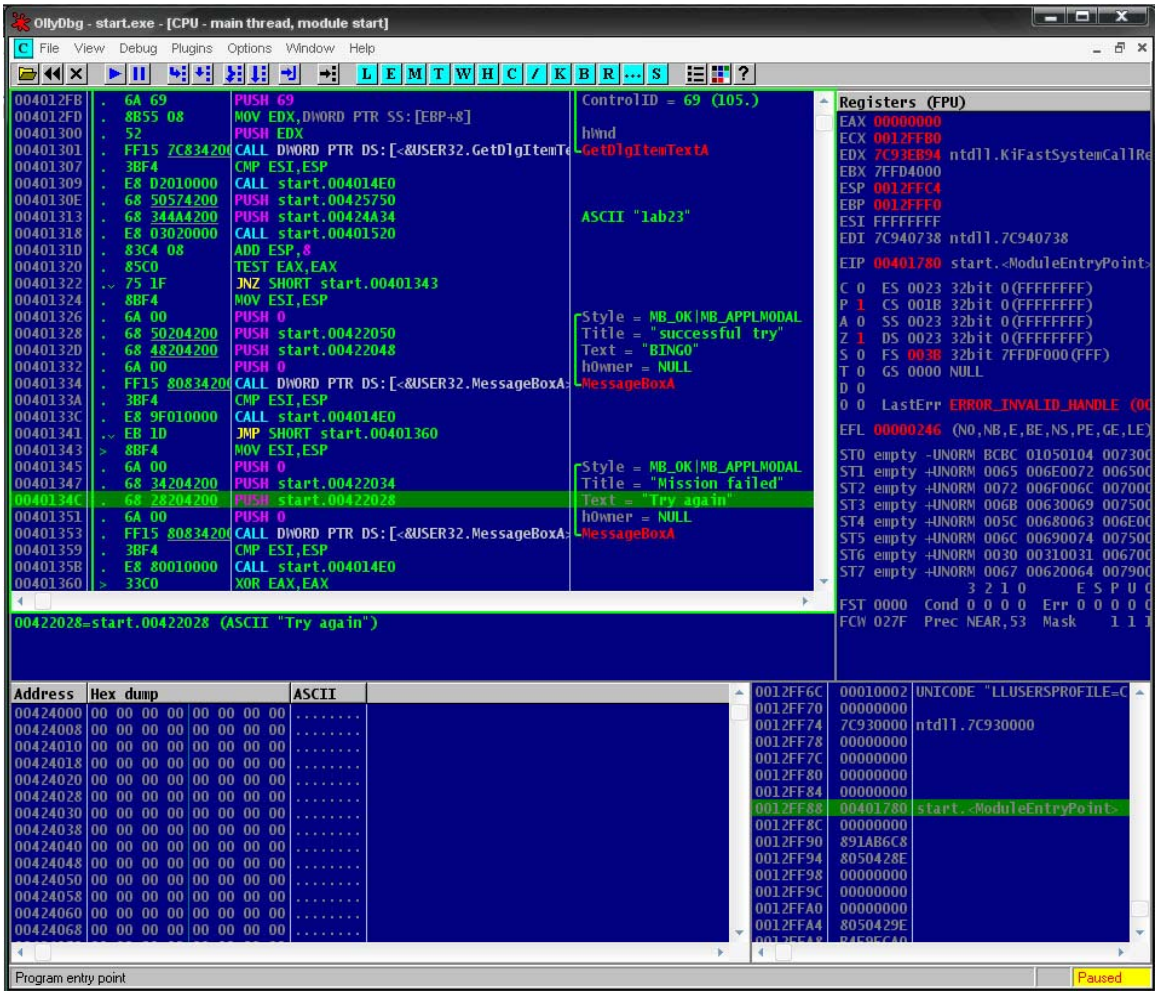
먼저 Disassembling한 코드에서 마우스 오른쪽 버튼을 클릭하여 POP-UP메뉴를 보면 여러 가지 기능이 존재합니다. 그 중에 Search for 메뉴를 이용하여 여러 가지 데이터 형태로 우리가 원하는 정보를 찾을 수 있습니다. 명령어, 바이너리 문자열, 모듈에서의 함수 이름 등으로 찾을 수 있는 기능이 제공됩니다. 우리가 입력한 암호가 다르면 발생하는 문자열인 “Try again”이라는 문자열을 참조한 위치를 알기 위해서 All referenced text string라는 기능

을 사용하겠습니다.



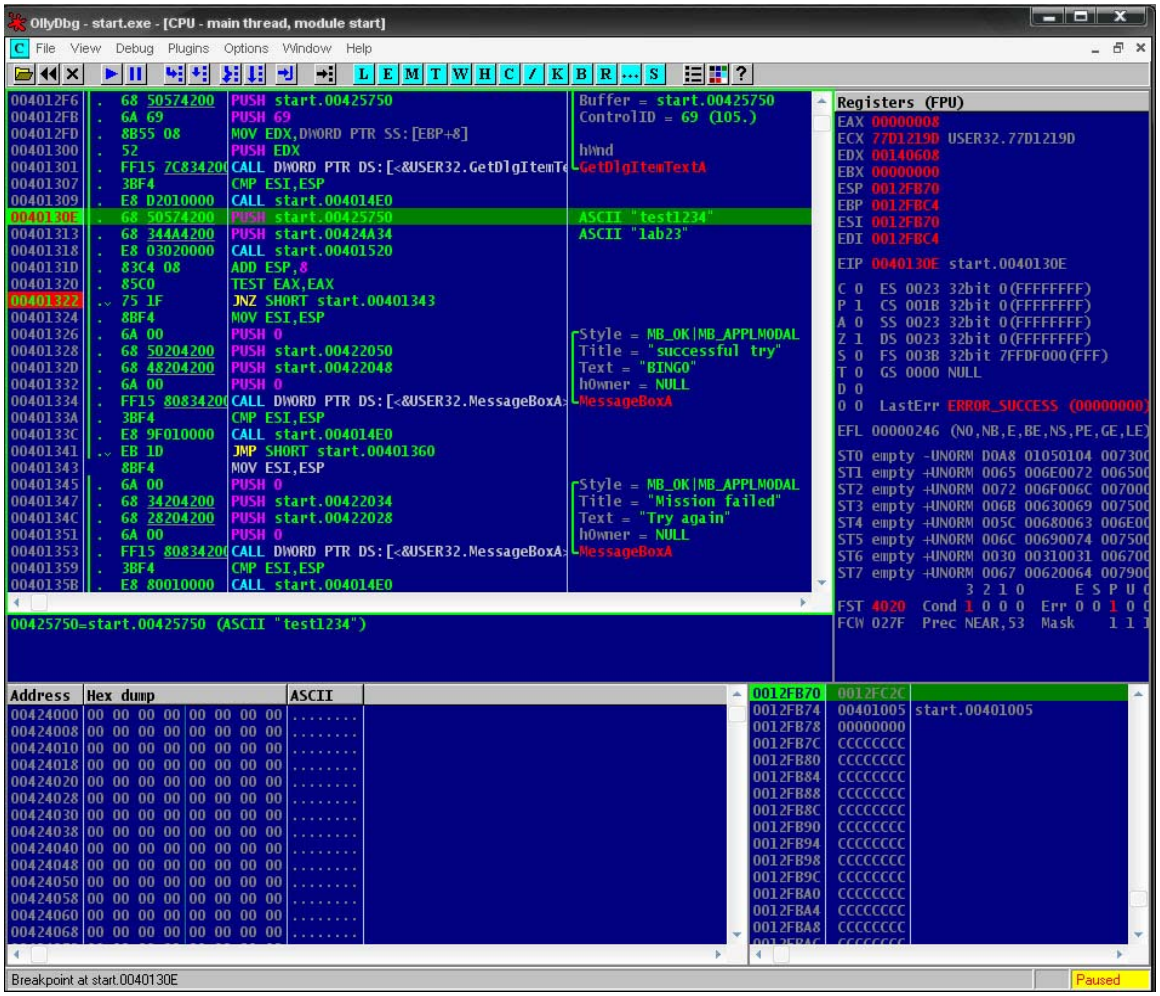
[그림 4]

[그림 4]에서 보듯이 해당 프로그램에서 참조된 문자열들이 출력되어있습니다. 우리는 여기서 “Try again” 문자열이 사용된 부분을 발견할 수 있습니다. 원하는 문자열을 선택하면 해당 코드 부분으로 이동할 수 있습니다.



[그림 5]

[그림 5]와 같이 “Try again” 문자열이 사용된 코드 부분으로 이동된 것을 볼 수 잇습니다. 위를 보면 “BINGO”라는 메시지를 사용하는 루틴도 볼 수 있습니다. 더 위를 보면 JNZ SHORT start.00401343를 볼 수 있습니다. 이것은 조건부 분기 코드인데, 어떠한 경우라면 00401343으로 분기합니다. 그 위를 보면 어떤 데이터 두 개를 push하고 특정 함수를 call 을 하고 있습니다. 이 상황은, 어떤 데이터 두 개중 하나는 사용자가 입력한 값이고 나머지 하나는 패스워드임을 추측할 수 있습니다. 그래서 만약 사용자가 입력한 값이 패스워드와 다르다면 “Try again”을 출력하는 코드로 분기하고, 같다면 “BINGO” 메시지를 출력하는 코드를 실행할 것입니다.



[그림 6]

지금까지의 추측이 맞는지 실제로 확인하기 위해서 Push start.00425750과, JNZ SHORT start.00401343 이 두 부분에 break point(F2)를 걸었습니다. (break point는 프로그램이 실행될 때 우리가 원하는 위치, 즉 원하는 상태에서의 메모리나 레지스터에 어떤 값이 들어가는지 알기 위해서 프로그램을 break 해주는 기능입니다.)

테스트를 위해 사용자는 test1234라는 패스워드를 입력했다고 가정하겠습니다. 첫 번째 break point에서는 입력한 값(test1234)이 push되는지 확인하였고 두 번째 break point에서는 입력한 값과 패스워드를 비교한 후 어디로 점프하는지 확인하였습니다.

[그림 6]에서와 같이 사용자가 입력한 값(test1234)가 push되고 있고, 입력 받은 부분과 패스워드를 비교하는 루틴이라는 추측이 맞았습니다. 그리고 두 번째 break point(JNZ SHORT start.00401343) 부분은, 이미 start.00401343으로 점프할 것이라 명시하고 있지만 어디로 점프하는지 직접 가보기 위해서 따라가 보면(Enter key) Wrong message를 출력하는 루틴으

로 이동하는 것을 확인할 수 있습니다. 결국 이 루틴은 사용자가 입력한 값과 패스워드를 비교하는 것이었습니다.

4. Plugin 소개

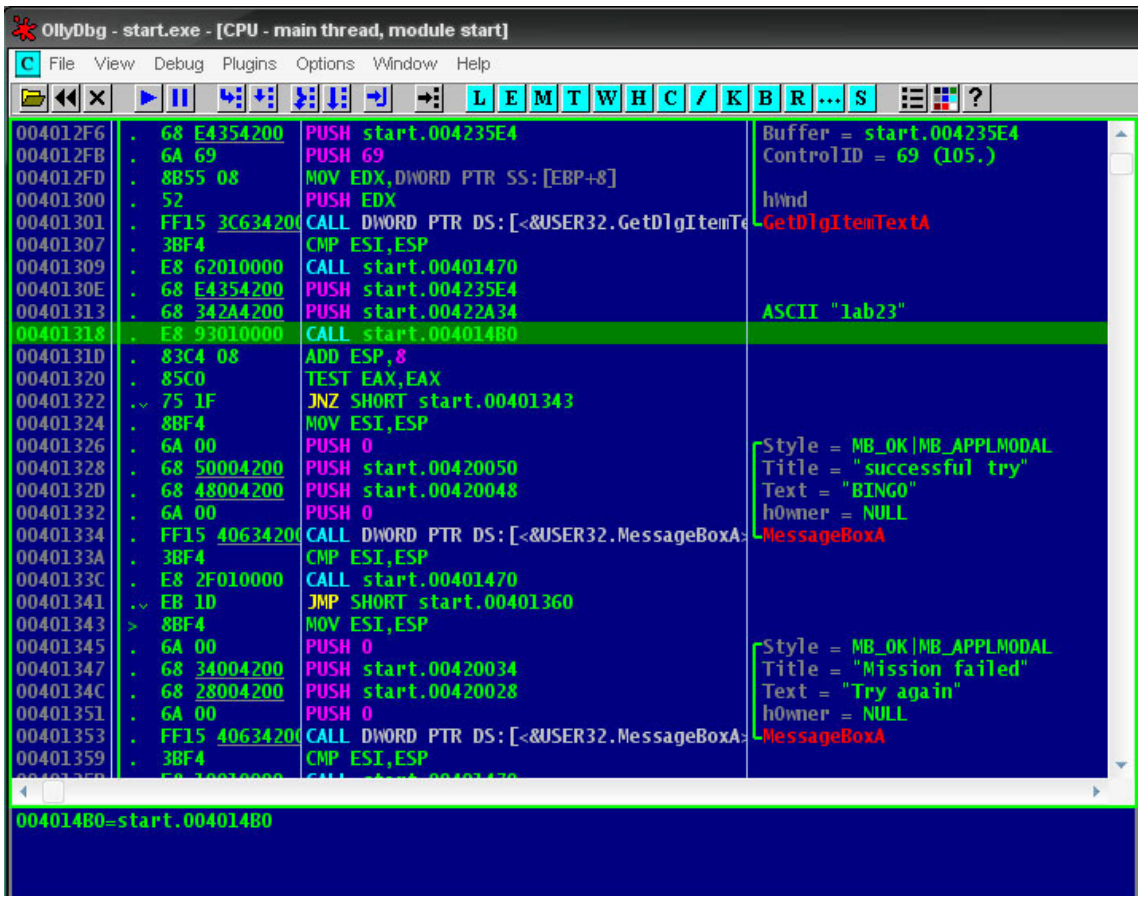
OllyDbg의 장점이자 특징 중 하나인 plugin에 대해서 설명하겠습니다. plugin은 사용자가 만들어서 배포 할 수도 있으며, plugin을 구할 수 있는 대표적인 사이트는 다음과 같습니다.

http://www.openrce.org/downloads/browse/OllyDbg_Plugins
http://www.pediy.com/tools/Debuggers/ollydbg/plugin.htm

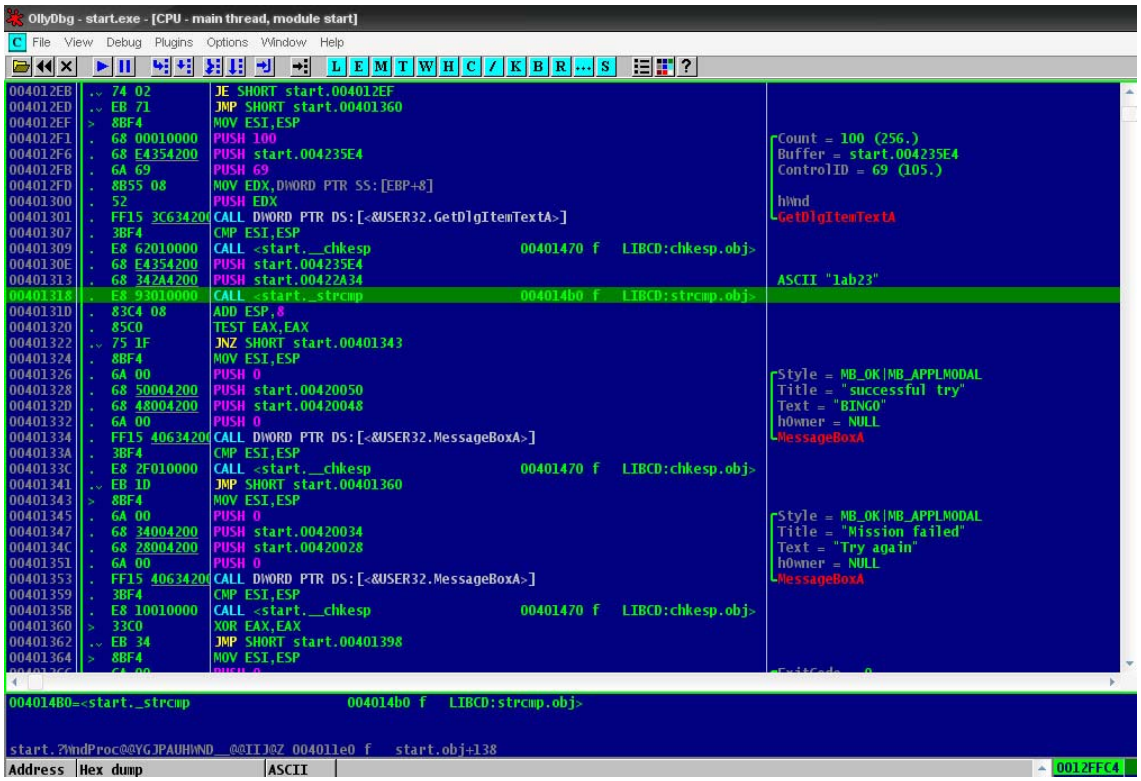
대부분 plugin은 dll 파일이나 소스 코드 형식으로 제공됩니다. dll 파일을 Ollydbg 폴더에 위치 시키고 실행시키면 plugin 메뉴에 해당 plugin이 추가됩니다. Plugin은 사용자가 직접 제작할 수도 있는데 이에 대해서는 ollydbg 홈페이지(http://www.ollydbg.de)를 참고하시기 바랍니다.

-4.1 MapConv

컴파일을 할 때 여러 가지 옵션을 사용할 수 있습니다. 그 중에서도 /MAP(vc++의 기능입니다.)이란 Linker 옵션은 컴파일을 하면서 동시에 프로그램의 map 파일을 생성해줍니다. Map 파일은 모듈 이름, 심벌의 이름 그리고 정의된 .obj 파일 등이 표시 됩니다. 이런 정보들을 이용하여 우리는 OllyDbg에 적용할 수 있고, 더 강력한 디버깅 정보를 제공 받게 됩니다.
MapConv라는 Ollydbg plugin을 이용해 map 파일을 로드 할 수 있습니다. Plugins -> MapConv -> Replace label을 선택한 뒤에 해당 *.map 파일을 선택합니다. Win32 API로 작성된 부분은 OllyDbg에서 label을 출력해 줍니다. 하지만 runtime libaray인 경우는 제대로 label을 달아주지 못합니다. 아래 그림에 위치한 부분이 CALL start.004014B0에서 CALL start.__strcmp로 바뀐 이유가 runtime library라서 label이 제대로 출력이 되지 않았기 때문입니다.
실제 Reverse Engineering 환경에서는 map 파일을 얻을 수 없는 경우가 많기 때문에 실용적이지는 않지만 참고할만한 plugin입니다.


OllyDbg - start.exe - [CPU - main thread, module start]
File View Debug Plugins Options Window Help
004012F6 . 68 E4354200 PUSH start.004235E4 Buffer = start.004235E4
004012FB . 6A 69 PUSH 69 ControlID = 69 (105.)
004012FD . 8B55 08 MOV EDX,DWORD PTR SS:[EBP+8]
00401300 . 52 PUSH EDX hWnd
00401301 . FF15 3C634200 CALL DWORD PTR DS:[<&USER32.GetDlgItemT GetDlgItemTextA
00401307 . 3BF4 CMP ESI,ESP
00401309 . E8 62010000 CALL start.00401470
0040130E . 68 E4354200 PUSH start.004235E4
00401313 . 68 342A4200 PUSH start.00422A34 ASCII "lab23"
00401318 . E8 93010000 CALL start.004014B0
0040131D . 83C4 08 ADD ESP,8
00401320 . 85C0 TEST EAX,EAX
00401322 . 75 1F JNZ SHORT start.00401343
00401324 . 8BF4 MOV ESI,ESP
00401326 . 6A 00 PUSH 0 Style = MB_OK|MB_APPLMODAL
00401328 . 68 50004200 PUSH start.00420050 Title = "successful try"
0040132D . 68 48004200 PUSH start.00420048 Text = "BINGO"
00401332 . 6A 00 PUSH 0 hOwner = NULL
00401334 . FF15 40634200 CALL DWORD PTR DS:[<&USER32.MessageBoxA MessageBoxA
0040133A . 3BF4 CMP ESI,ESP
0040133C . E8 2F010000 CALL start.00401470
00401341 . EB 1D JMP SHORT start.00401360
00401343 > 8BF4 MOV ESI,ESP
00401345 . 6A 00 PUSH 0 Style = MB_OK|MB_APPLMODAL
00401347 . 68 34004200 PUSH start.00420034 Title = "Mission failed"
0040134C . 68 28004200 PUSH start.00420028 Text = "Try again"
00401351 . 6A 00 PUSH 0 hOwner = NULL
00401353 . FF15 40634200 CALL DWORD PTR DS:[<&USER32.MessageBoxA MessageBoxA
00401359 . 3BF4 CMP ESI,ESP
004014B0=start.004014B0

[그림 7]



[그림 8]

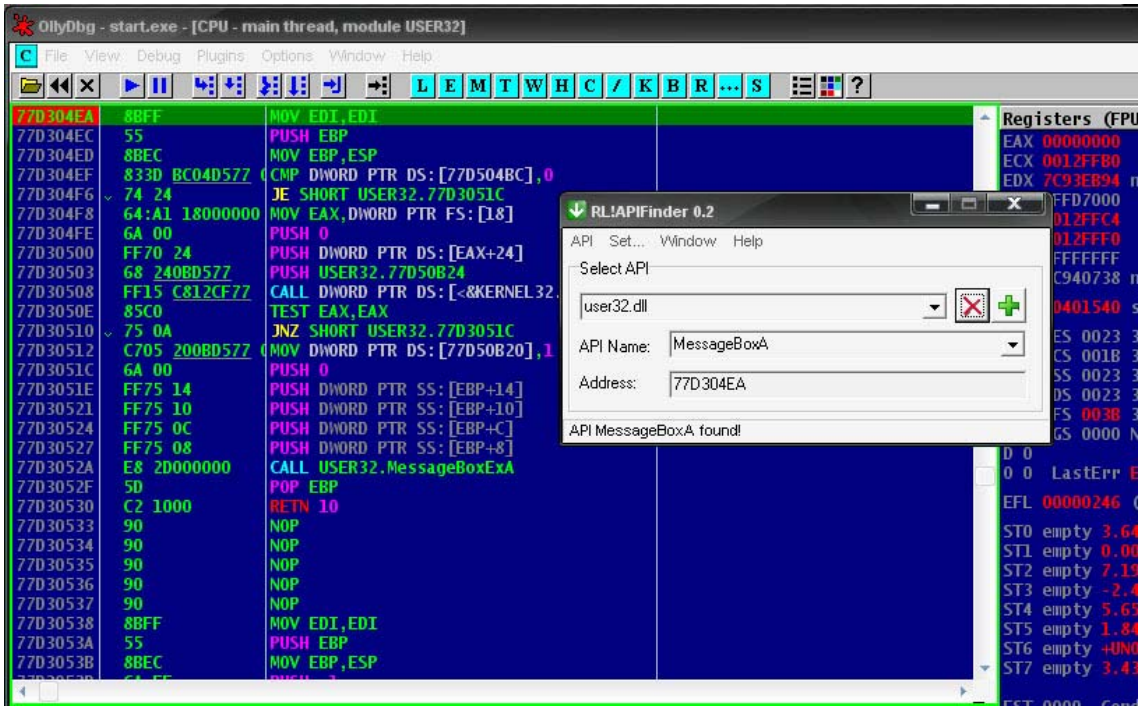
-4.2 OllyScript

OllyDbg에서 특정 스크립트를 실행시킬 수 있는 plugin입니다. 스크립트에 쓰이는 언어는 어셈블리어와 비슷한 형식을 가집니다. 주로 패킹된 프로그램의 OEP를 찾거나, packer로 인한 code obfuscation(코드난독화, junk code, scrambled code라고도 불립니다.)을 제거하는 script가 있습니다. 작업을 자동화할 수 있도록 도움을 주는 plugin이라 보시면 됩니다. 아래의 사이트에서 보다 많은 정보를 얻을 수 있습니다.

http://www.openrce.org/downloads/browse/OllyDbg_OllyScripts
http://www.pediy.com/tools/debuggers/ollydbg/script.htm

-4.3 ApiFinder

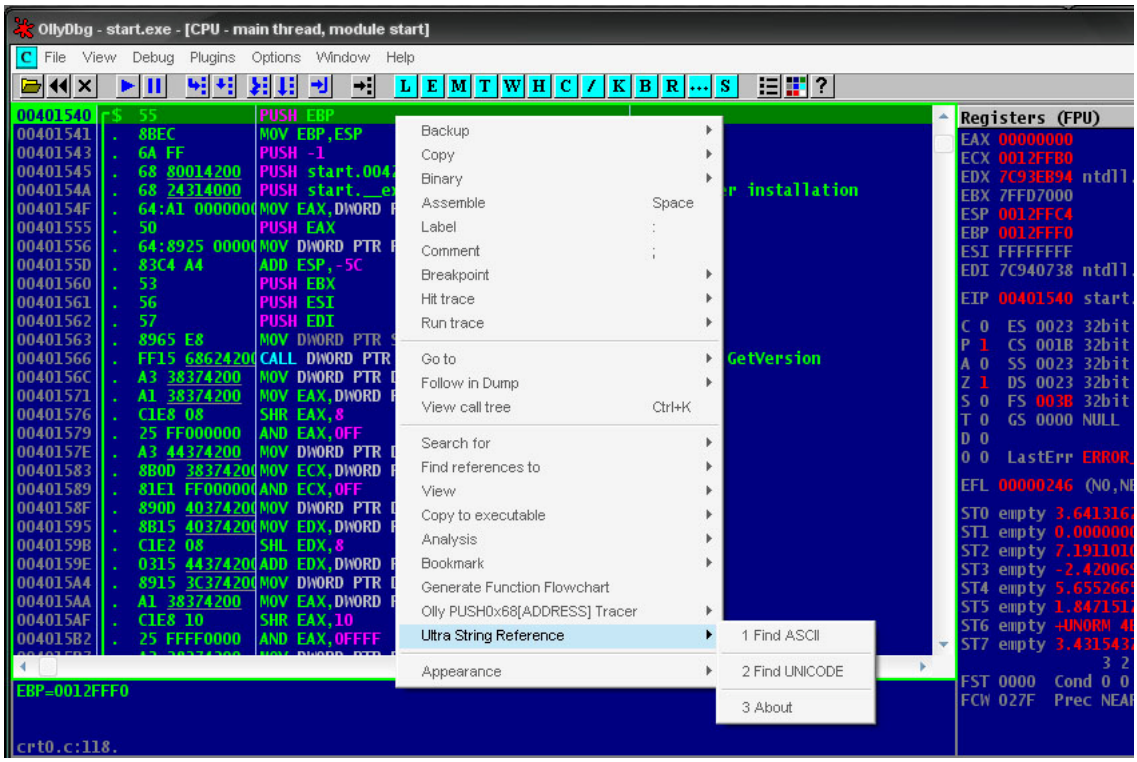
ApiFinder는 해당 프로그램에서 사용하는 dll들의 API들을 API 이름으로 검색한 뒤, 해당 API 루틴이 위치한 address로 이동해 break point를 쉽게 설치할 수 있는 기능을 제공합니다.



[그림 9]

-4.4 Ultra String Reference

ASCII와 UNICODE 검색을 지원합니다.



[그림 10]

5. 마치는 말

Ollydbg는 비교적 간단한 interface임에도 강력한 기능을 제공합니다. 이 강좌는 초보자를 위해 연재 형식으로 진행될 것이며 향후 연재에서는 plugin을 직접 제작하는 방법 등 보다 다양한 ollydbg 팁들에 대해 다룰 것입니다. 프로그램의 단축키나 보다 자세한 정보는 ollydbg 홈페이지에서 확인하시기 바랍니다.