

```
#####  
# #  
# Detours를 사용하기 & iAT Hooking 걸기 #  
# 2008-10-31 #  
# written by crattack #  
# crattack@naver.com #  
#####
```

0x0000. prologue

어느덧 Reverse를 시작한지 4년째가 되었습니다. 4년이라는 시간 동안 조금씩 이것 저것 맛만 보던 저에게 Hooking이라는 기술을 본격적으로 이용해야 하는 시기가 왔습니다. 회사에서 Project로 함수에서 사용하는 buff에 대한 로그를 확인해야 하는 과정이 필요했는데 Code Injection을 하기엔 너무 코드의 양이 많더군요. 그래서 Hooking을 선택했습니다. Hooking 하는데 있어 여러 가지 방법이 사용되고 있지만 저에겐 최대한 빨리 작업하여 Output을 내야 하기 때문에 Detours라는 Library을 이용하기로 하였습니다. 하지만 Detours의 성격상 제가 원하는 부분은 Hooking이 안 걸리더군요. 그래서 iAT Hooking을 통하여 문제를 해결 했습니다. Project하면서 문제가 발생했던 Hooking 과정을 조금 더 쉽게 많은 사람들이 사용하여 응용하였으면 좋겠다는 생각으로 이 문서를 만들었습니다.

보시다가 문제가 생길 수 있는 부분이나, 또는 잘못된 지점은 가차없이 메일을 주시면 수정하겠습니다. 저도 배우는 입장으로 잘못된 지식을 가지고 있으면 고쳐나가 차후에 똑 같은 실수로 문서를 작성하지 않을 수 있으니 이 글을 읽으시는 많은 분들의 가차 없는 질타를 기다리겠습니다. 또한 문서에서 존칭은 prologue와 epilogue에서만 사용하고 본문에서는 반어를 사용하겠습니다. 이점 양해해주시고 이제부터 시작하겠습니다. Here we go~~♪

0x0100. Detours ?

Detours는 Microsoft사에서 제공하는 Hooking Library이다. API를 Hooking 하는데 있어서 가장 뛰어나면서도 Simple한 최고의 Library 이다. Detours에 관한 자료는 google 형님한테 물어보면 많은 내용을 확인 할 수 있을 것이다. 또한 codeproject라든지, opensource 기반의 code를 참조하여 Detours 사용 Skill을 늘려나가면 될 것이다.

현재 detours는 MS에서 아래의 사이트에 가면 구할 수 있다. 현재 버전은 2.1 버전이므로 DetoursExpress2.1 버전을 다운 받으면 된다.

(Detours 배포 : <http://research.microsoft.com/sn/detours/>)

## 0x0200. Detours Example

### 0x0210) Setting ……

먼저 코드를 보도록 하자. Detours Express 2.1을 설치하면 다음과 같은 기본 경로에 Source가 설치되게 된다.

C:\Program Files\Microsoft Research\Detours Express 2.1\samples 이곳에서 쉽게 Code를 수정하여 사용할 수 있다. 자! Detours를 사용하기 위해선 먼저 실행 파일을 setdll.exe, withdll.exe 파일이 존재해야 한다. setdll.exe 파일은 exe 파일 내에서 Hooking 역할을 하는 DLL을 static injection 하여 사용하게 하는 반면에, withdll.exe는 exe 파일 안에 동적으로 Hooking 역할을 하는 DLL을 injection하여 사용하게 한다. 즉, exe 파일 안에 Hooking 역할을 하는 DLL이 고정으로 들어가느냐, 아니면 일회용으로 사용할 때만 들어가느냐의 차이이다.

Test할 때는 withdll을 이용하여 하면 보다 쉽게 test 할 수 있을 것이다. 그럼 시작해보도록 하자. setdll과 withdll 파일을 컴파일을 먼저 해보자.

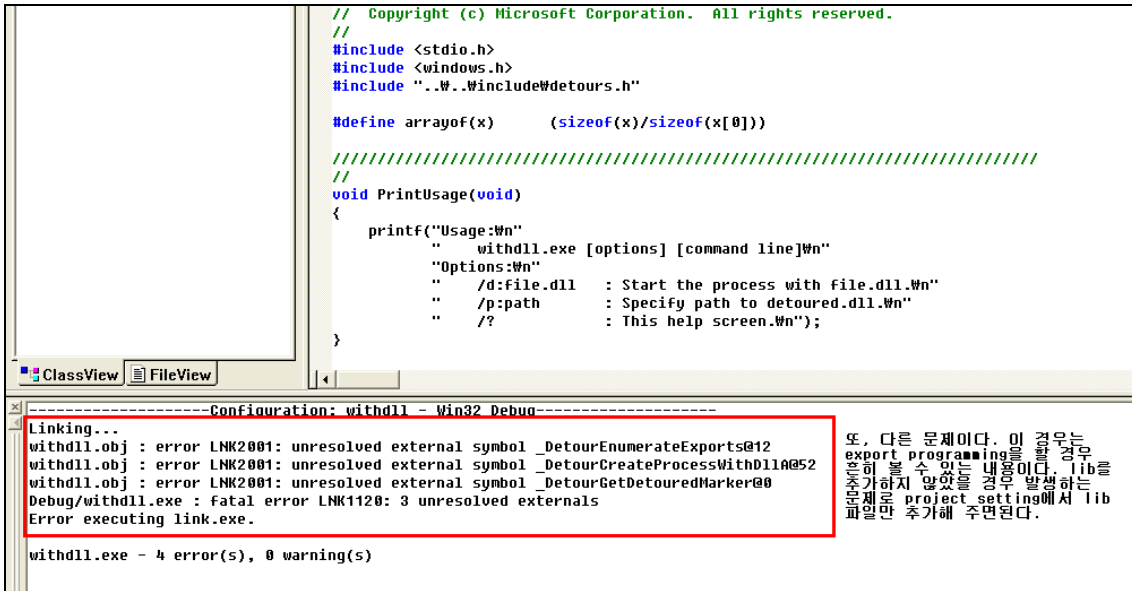
필자가 좋아하는 withdll을 먼저 컴파일 하도록 하겠다. 경로는 다음과 같다. “C:\Program Files\Microsoft Research\Detours Express 2.1\samples\withdll\withdll.cpp” 을 열어서 컴파일을 하면 제일 먼저 나오는 문제가 바로 include 경로 문제가 발생하게 된다.

```
////////////////////////////////////  
//  
// Test DetourCreateProcessWithDll function (withdll.cpp).  
//  
// Microsoft Research Detours Package, Version 2.1.  
//  
// Copyright (c) Microsoft Corporation. All rights reserved.  
//  
#include <stdio.h>  
#include <windows.h>  
#include <detours.h>  
  
#define arrayof(x) (sizeof(x)/sizeof(x[0]))  
  
////////////////////////////////////  
//  
void PrintUsage(void)  
{  
    printf("Usage:\n"  
        "  withdll.exe [options] [command line]\n"  
        "Options:\n"  
        "  /d:file.dll : Start the process with file.dll.\n"  
        "  /p:path    : Specify path to detoured.dll.\n"  
        "  /?       : This help screen.\n");  
}
```

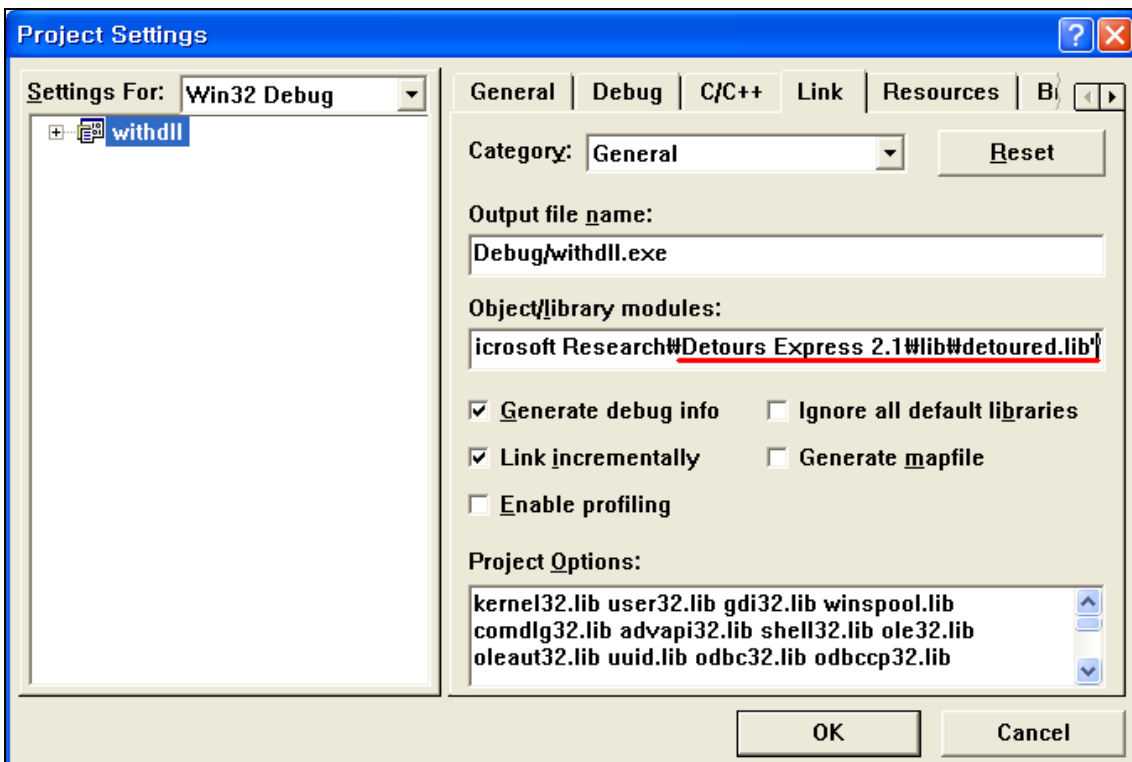
[그림 1] 첫 번째 문제. [Include] 경로 문제

두 번째는 Export된 함수에 대한 Library 파일 추가 문제이다. 임시 방편으로 추가해주는 방식을 선택하였지만 차후에도 detours를 자주 사용할 예

정이라면 Visual Studio 환경에 Library 파일을 추가 하여 사용하는 것이 좋은 것 같다. (※ WsrcW 디렉토리에 detours.h, detoured.h 파일이 존재한다. 밖으로 include 파일로 추출하여 사용하는 게 편할 것이다.)

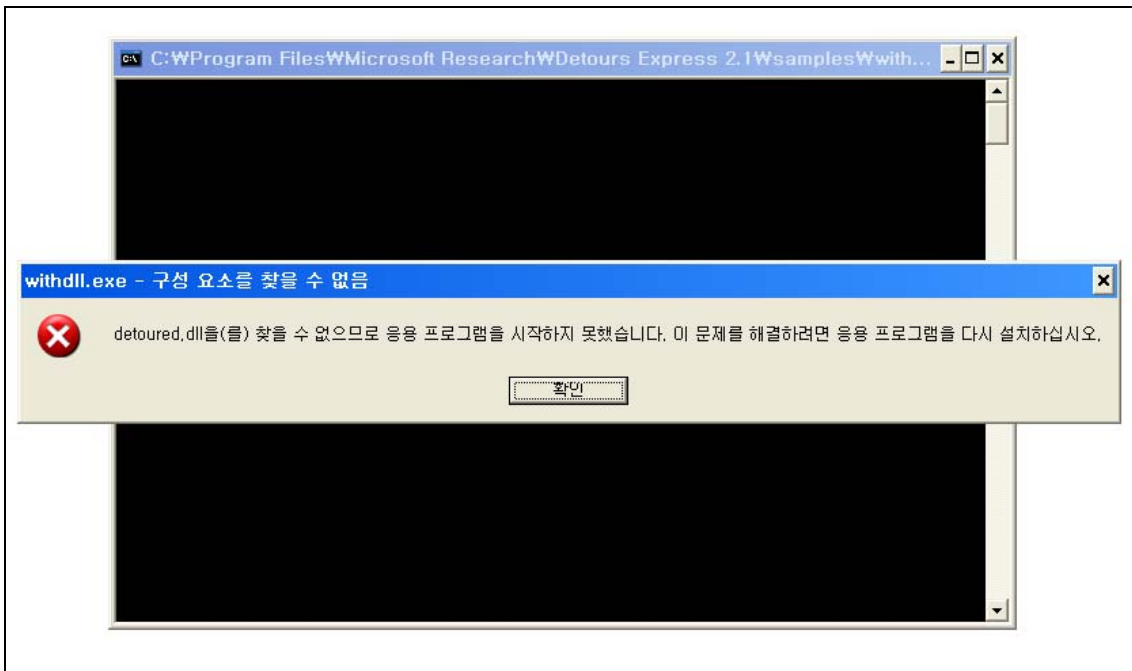


[그림 2] 두 번째 문제. [Export] 문제



[그림 3] 두 번째 문제 해결책. (detours.lib, detoured.lib 파일 추가)

자, 이젠 컴파일은 완료된다. 하지만 또 하나의 문제가 발생하였다. 바로 detoured.dll 파일이 필요하다. 2.1 버전 이하인 1.5 버전에서는 detoured.dll 파일이 필요 하지 않았다. 하지만 2.1로 업그레이드 되면서 detoured.dll 파일이 필수적으로 필요하게 되었다. 만약 detoured.dll 파일을 사용하지 않고 쓰고 싶다면 아래의 링크로 재 컴파일 하길 바란다.  
(<http://blog.naver.com/crattack/70036911168>)



[그림 4] Detours Express 2.1 버전 업 되면서 필수 조건이 된 detoured.dll 파일 요구

하지만 이렇게 개인별로 컴파일 하지 않아도 한 명령어로 모든 파일이 컴파일 될 수 있다. 바로 nmake all 이다. Detours Express 2.1에서 배포할 때 makefile을 제공하여 한번에 모든 samples 디렉토리에 있는 모든 내용과 src 디렉토리에 있는 모든 내용을 컴파일 해주도록 만들어 놓았다. 그럼 nmake를 이용하여 컴파일을 하자.

```
C:\WINDOWS\system32\cmd.exe
tracelnk.cpp
  Creating library ..\..\bin\tracelnk.lib and object ..\..\bin\tracelnk.exp
  cd "C:\Program Files\Microsoft Research\Detours Express 2.1\samples\imp
unge"
    if not exist ..\..\bin mkdir ..\..\bin
    cl /nologo /Zi /MD /Gm- /W4 /WX /O1 "/I..\..\include" "/I..\..\include" /Gs
/DDETOURS_X86=1 /D_X86_ /Fe..\..\bin\impunge.exe /Fd..\..\bin\impunge.pdb imp
unge.cpp /link /release /machine:x86 "..\lib\syelog.lib" "..\lib\detours.li
b" "..\lib\detoured.lib" kernel32.lib gdi32.lib user32.lib shell32.lib imag
hlp.lib /subsystem:console /incremental:no
impunge.cpp
  cd "C:\Program Files\Microsoft Research\Detours Express 2.1\samples"
  cd "C:\Program Files\Microsoft Research\Detours Express 2.1"

C:\Program Files\Microsoft Research\Detours Express 2.1>"C:\Program Files\Micros
oft Research\Detours Express 2.1\bin\withdll.exe"
Usage:
  withdll.exe [options] [command line]
Options:
  /d:file.dll      : Start the process with file.dll.
  /p:path          : Specify path to detoured.dll.
  /?              : This help screen.

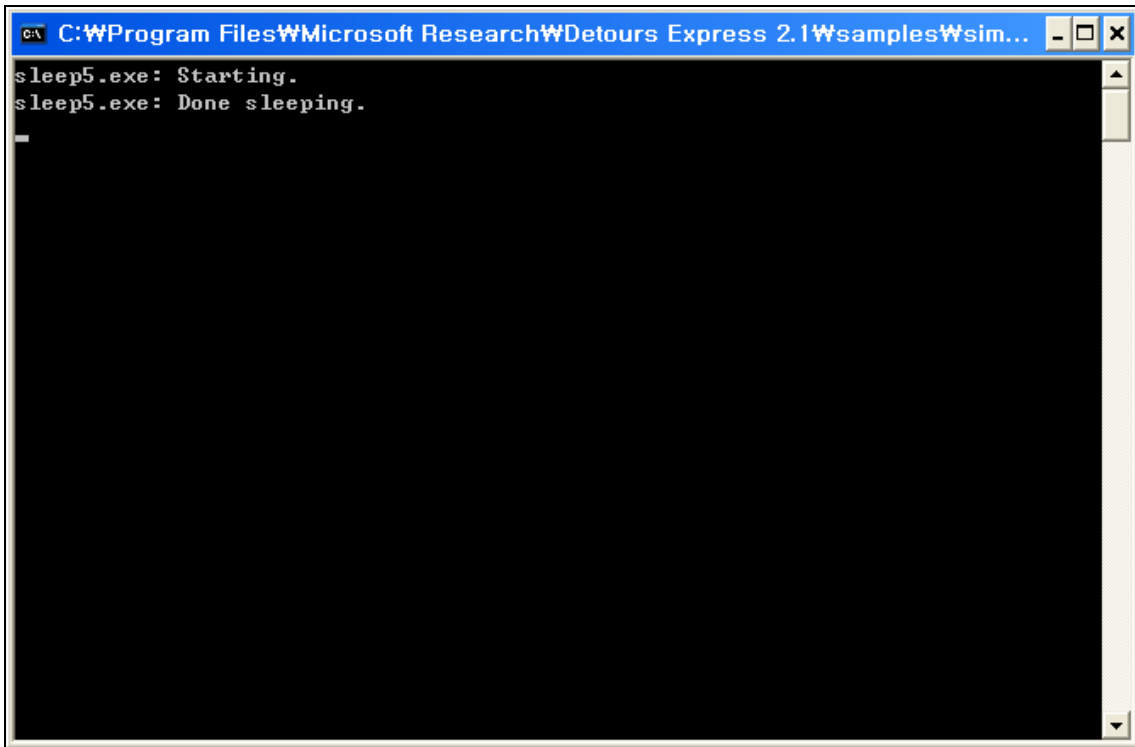
C:\Program Files\Microsoft Research\Detours Express 2.1>nmake all_
```

[그림 5] Detours Express 2.1 컴파일 하기

그럼 이제 준비가 다 되었다. 컴파일도 되었고 2.1에서 제일 필요한 withdll.exe와 detoured.dll 파일이 완성되었다. nmake all가 완성되면 .\bin\ 디렉토리에 잔뜩 결과물들이 생긴다. 그 곳에서 withdll.exe 파일과 detoured.dll 파일을 제외하고 다 지우도록 하자. 필요할 때마다 하나 하나 컴파일 하여 적용하여 사용하면 훨씬 많은 도움이 될 것이므로 다 지우고 이제부터 본격적으로 수정에 들어가도록 하자.

0x0220) Here we go .....

Target 프로그램은 sample 디렉토리에 있는 sleep5.exe 프로그램을 Target으로 잡겠다. 먼저 컴파일을 통하여 exe 파일을 만들어 보자.

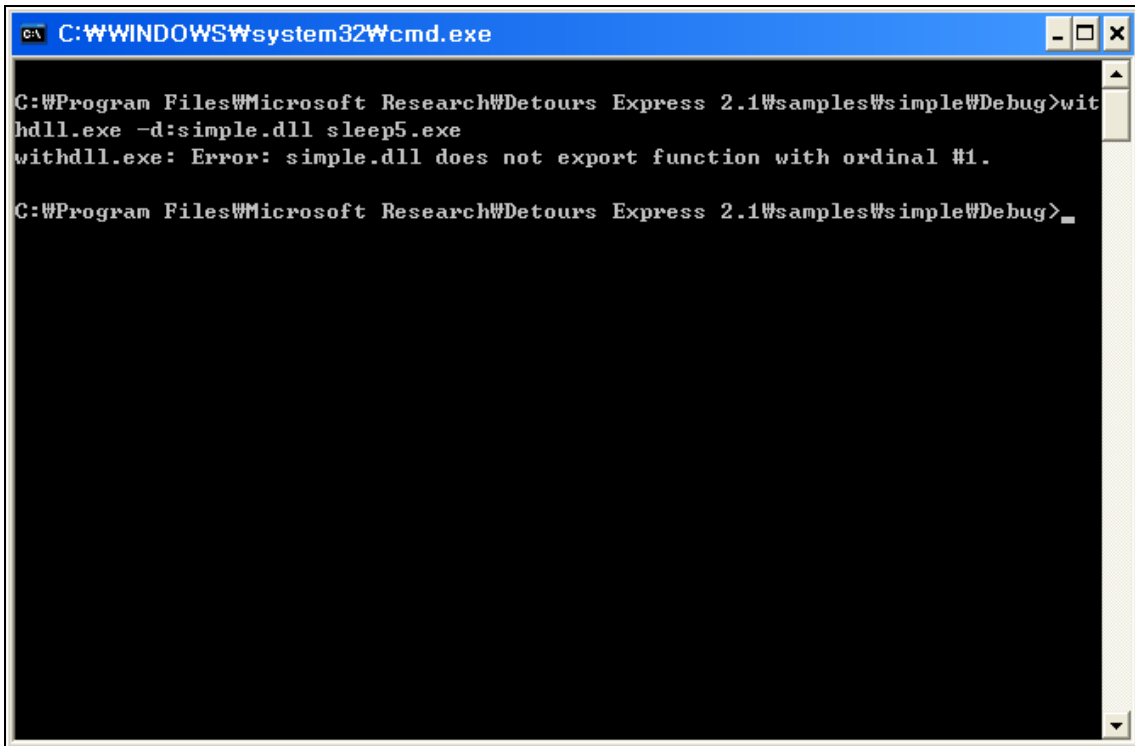


[그림 6] simple 파일 실행 화면

단순 sleep()함수를 통하여 delay를 걸어주는 프로그램이다. 이것 simple.dll 파일을 이용하여 API Hooking을 걸어보도록 하겠다. simple.cpp 파일을 Win32 Dynamic-Link Library의 project로 생성하여야 한다. dll 파일로 생성하여 withdll 을 통하여 sleep.exe 파일에 attach해야 하기 때문에 Visual Studio에서의 Project 중 Win32 Dynamic-Link Library 선택하여 Project를 생성하도록 한다.

simple.cpp 파일을 열면 역시나 include 경로에 대해서 문제가 발생한다. 역시나 [그림 1]과 같이 경로를 재정의 해서 사용하면 아무런 문제가 발생하지 않는다. include 경로 문제가 끝나면 [그림 2]와 같은 문제가 발생하므로 또한, [그림 3]과 같이 처리하여 컴파일을 완성해야 한다.

하지만, dll이 생성되었다고 한들 실행하면 다음과 같은 메시지가 나오게 된다.

A screenshot of a Windows command prompt window. The title bar reads "C:\WINDOWS\system32\cmd.exe". The command prompt shows the following text:

```
C:\Program Files\Microsoft Research\Detours Express 2.1\samples\simple\Debug>withdll.exe -d:simple.dll sleep5.exe  
withdll.exe: Error: simple.dll does not export function with ordinal #1.  
C:\Program Files\Microsoft Research\Detours Express 2.1\samples\simple\Debug>
```

[그림 7] 일반적인 컴파일로 생성된 DLL 파일을 컴파일 했을 때 발생하는 오류

export된 dll 파일이 ordinal을 찾을 수 없어서 발생하는 문제이므로 에러가 발생한다. 하지만 이것 간단한 문제이며, 쉽게 해결 할 수 있다. 바로 nmake를 이용하는 것이다. 먼저 makefile을 열어서 내용을 알아보도록 하자.

```
#####  
##  
## API Extention to Measure time slept.  
##  
## Microsoft Research Detours Package, Version 2.1.  
##  
## Copyright (c) Microsoft Corporation. All rights reserved.  
##  
CLIB=/MT  
!include ..\common.mak # Export를 위한 컴파일 옵션이 들어가 있는 파일  
#####  
all: dirs #  
    $(BIND)\simple.dll #  
    $(BIND)\sleep5.exe #  
    #  
!IF $(DETOURS_SOURCE_BROWSING)==1  
    $(BIND)\simple.bsc #  
    $(BIND)\sleep5.bsc #  
!ENDIF  
#####  
dirs:  
    if not exist $(BIND) mkdir $(BIND)  
$(BIND)\simple.dll $(BIND)\simple.lib: simple.cpp $(DEPS)  
    cl /LD $(CFLAGS) /Fe$@ /Fd$(BIND)\simple.pdb simple.cpp #  
        /link $(LINKFLAGS) /incremental:no /subsystem:console #  
        /entry:$(DLLENTY) #  
        /export:TimedSleep #  
        $(LIBS)  
$(BIND)\simple.bsc : simple.obj  
    bscmake /v /n /o $@ simple.sbr  
$(BIND)\sleep5.exe : sleep5.cpp $(DEPS)  
    cl $(CFLAGS) /Fe$@ /Fd$(BIND)\sleep5.pdb sleep5.cpp #  
        /link $(LINKFLAGS) $(LIBS) #  
        /subsystem:console /incremental:no  
$(BIND)\sleep5.bsc : sleep5.obj  
    bscmake /v /n /o $@ sleep5.sbr  
#####  
clean:  
    -del *~ *.obj *.sbr 2>nul  
    -del $(BIND)\simple.* 2>nul  
    -del $(BIND)\sleep5.* 2>nul  
#####  
test: $(BIND)\sleep5.exe $(BIND)\simple.dll  
    @echo ----- Reseting test binaries to initial state. -----
```

[그림 8] makefile 설정 환경

내용을 보면 컴파일시 추가해줘야 하는 컴파일 옵션들이 들어가 있다. 그럼 이제 nmake명령어를 통하여 컴파일을 해보도록 하자.



```

C:\WINDOWS\system32\cmd.exe
2008-11-05 오후 03:24          537 sleep5.dsw
2008-11-05 오후 03:24       33,792 sleep5.ncb
2008-11-05 오후 02:19      10,470 sleep5.obj
2008-11-05 오후 03:24      54,784 sleep5.opt
2008-11-05 오후 03:12        1,168 sleep5.plg
          15개 파일          225,536 바이트
          4개 디렉터리 19,944,628,224 바이트 남음

C:\Program Files\Microsoft Research\Detours Express 2.1\samples\simple>nmake al

Microsoft (R) Program Maintenance Utility   Version 6.00.9782.0
Copyright (C) Microsoft Corp 1988-1998. All rights reserved.

        if not exist ..\..\bin mkdir ..\..\bin
        cl /LD /nologo /Zi /MT /Gm- /W4 /WX /O1 "/I..\..\include" "/I..\..\include"
        /Gs /DDETOURS_X86=1 /D_X86_ /Fe..\..\bin\simple.dll /Fd..\..\bin\simple.pdb sim
        ple.cpp /link /release /machine:x86 /incremental:no /subsystem:console /entry:
        _DllMainCRTStartup@12 /export:TimedSleep "..\..\lib\syelog.lib" "..\..\lib\detour
        s.lib" "..\..\lib\detoured.lib" kernel32.lib gdi32.lib user32.lib shell32.lib
        simple.cpp
        Creating library ..\..\bin\simple.lib and object ..\..\bin\simple.exp

C:\Program Files\Microsoft Research\Detours Express 2.1\samples\simple>_

```

[그림 9] nmake를 이용한 simple.cpp 파일 컴파일

이제 됐으니 테스트 하도록 하자. 아까 만들어놓은 withdll.exe 파일을 가  
지고 sleep5.exe 파일에 Hooking을 걸겠다.

```

C:\WINDOWS\system32\cmd.exe
2008-11-05 오후 05:23      94,295 simple.dll
2008-11-05 오후 05:23         588 simple.exp
2008-11-05 오후 05:23         1,966 simple.lib
2008-11-05 오후 05:23     274,432 simple.pdb
2008-11-05 오후 02:19     73,815 sleep5.exe
2008-11-05 오후 02:19     28,760 withdll.exe
          8개 파일          4,448,606 바이트
          2개 디렉터리 19,944,173,568 바이트 남음

C:\Program Files\Microsoft Research\Detours Express 2.1\bin>withdll.exe -d:simpl
e.dll sleep5.exe
withdll.exe: Starting: 'sleep5.exe'
withdll.exe:  with 'C:\Program Files\Microsoft Research\Detours Express 2.1\bin
\simple.dll'

withdll.exe:  marked by 'C:\Program Files\Microsoft Research\Detours Express 2.
1\bin\detoured.dll'

simple.dll: Starting.
simple.dll: Detoured $sleep().
sleep5.exe: Starting.
sleep5.exe: Done sleeping.
simple.dll: Removed $sleep() (result=0), slept 5000 ticks.

C:\Program Files\Microsoft Research\Detours Express 2.1\bin>

```

[그림 10] sleep5.exe 파일을 Hooking한 결과

그럼 어떻게 Hooking이 걸리는지 체크 해보도록 하자.

```
////////////////////////////////////  
//  
// Detours Test Program (sleep5.cpp of sleep5.exe)  
//  
// Microsoft Research Detours Package, Version 2.1.  
//  
// Copyright (c) Microsoft Corporation. All rights reserved.  
//  
  
#include <windows.h>  
#include <stdio.h>  
  
int __cdecl main(int argc, char ** argv)  
{  
    if (argc == 2) {  
        Sleep(atoi(argv[1]) * 1000);  
    }  
    else {  
        printf("sleep5.exe: Starting.\n");  
        Sleep(5000);  
        printf("sleep5.exe: Done sleeping.\n");  
    }  
    return 0;  
}  
//  
//////////////////////////////////// End of File.
```

Hooking 할 함수

Sleep(5000);

[그림 11] sleep5 Program Source

[그림 11]과 같이 Sleep()를 사용하고 있다. Sleep()에 [그림 12]와 같이 Hooking을 걸어 의미의 Message를 출력하게 할 것이다.

```

#include <stdio.h>
#include <windows.h>
#include "..\..\WinInclude\detours.h"

static LONG dwSlept = 0;
static VOID (WINAPI * TrueSleep)(DWORD dwMilliseconds) = Sleep;

// Hooking한 Sleep()를 Customize 한 Function
VOID WINAPI TimedSleep(DWORD dwMilliseconds)
{
    DWORD dwBeg = GetTickCount();
    TrueSleep(dwMilliseconds);
    DWORD dwEnd = GetTickCount();

    InterlockedExchangeAdd(&dwSlept, dwEnd - dwBeg);
}

BOOL WINAPI DllMain(HINSTANCE hinst, DWORD dwReason, LPVOID reserved)
{
    LONG error;
    (void)hinst;
    (void)reserved;

    if (dwReason == DLL_PROCESS_ATTACH) {
        printf("simple.dll: Starting.\n");
        fflush(stdout);

        // DetourCreateProcessWithDll()을 통하여 process 생성한 후 import table의 정보를 memory에 저장.
        DetourRestoreAfterWith();

        // attaching 또는 detaching 하기 위한 준비 상태.
        DetourTransactionBegin();

        // 현재 Process의 Thread Handle를 얻어옴.
        DetourUpdateThread(GetCurrentThread());

        // 핵심 부분 : TimedSleep를 TrueSleep()로 대체 한다.
        // TrueSleep는 Slee()를 재정의 한것이다.
        DetourAttach(&(PVOID&)TrueSleep, TimedSleep);

        // Detours 적용.
        error = DetourTransactionCommit();

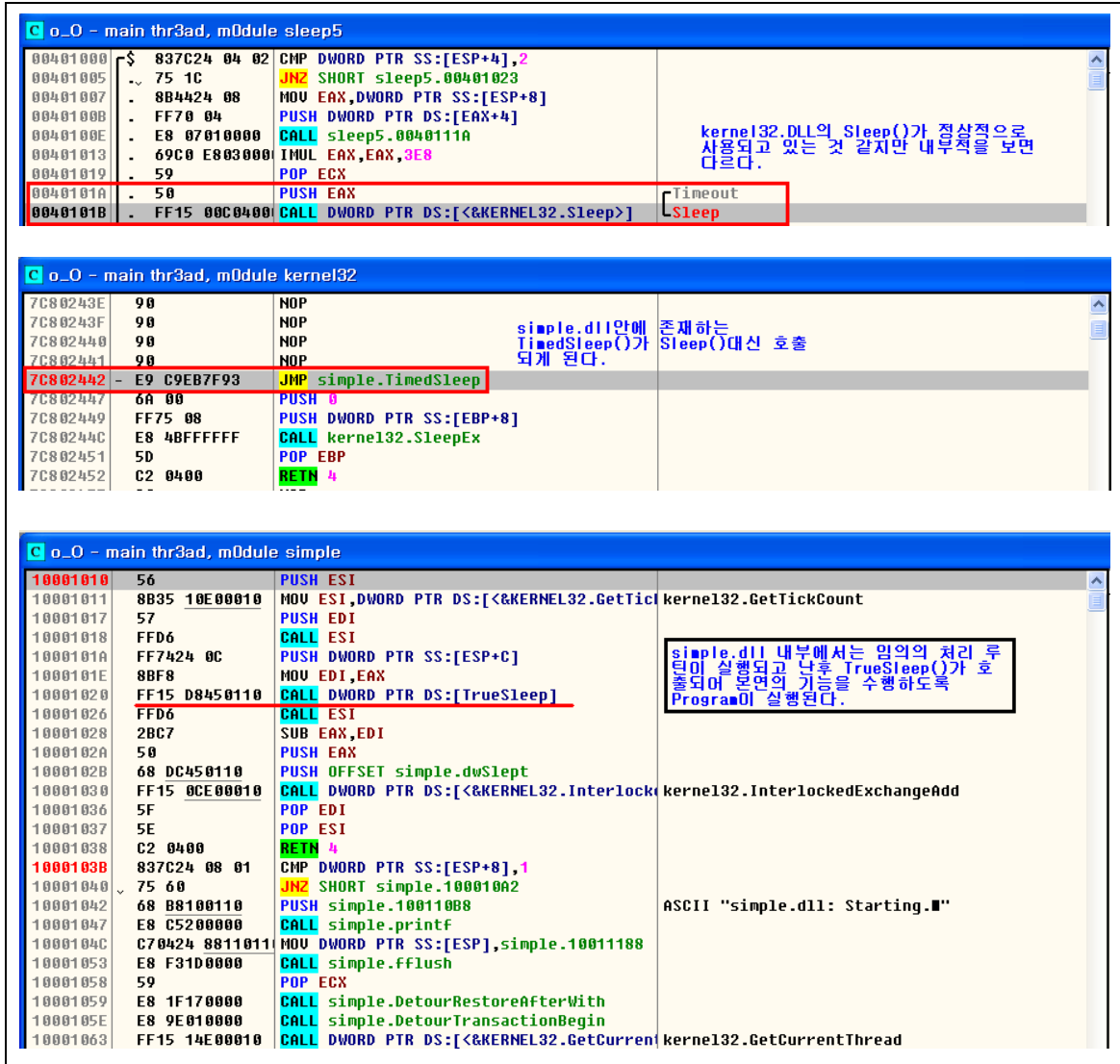
        if (error == NO_ERROR) {
            printf("simple.dll: Detoured Sleep().\n");
        }
        else {
            printf("simple.dll: Error detouring Sleep(): %d\n", error);
        }
    }
    else if (dwReason == DLL_PROCESS_DETACH) {
        // detaching하기 위한 준비.
        DetourTransactionBegin();
        // 현재 Pcess의 Thread Handle를 얻어옴.
        DetourUpdateThread(GetCurrentThread());
        // 원상 복구 - TimedSleep()를 TrueSleep()로 변환.
        DetourDetach(&(PVOID&)TrueSleep, TimedSleep);
        // Detours 적용.
        error = DetourTransactionCommit();

        printf("simple.dll: Removed Sleep() (result=%d), slept %d ticks.\n",
            error, dwSlept);
        fflush(stdout);
    }
    return TRUE;
}

```

[그림 12] simple Program Source

[그림 12]에서 알 수 있듯이 Sleep()의 함수를 Hooking하기 위해선 Sleep()의 원형을 제시해야 한다. 위와 같은 절차에 의해서 Detours Program이 운영 된다. 그럼 이제 Hooking된 상태를 Debugger로 확인 해보도록 하자.



[그림 13] Hooking 된 sleep5.exe 파일 내부 Sleep() 흐름도

또한, Memory Map을 통하여 Memory 상에 Load된 List를 확인할 경우 simple.dll과 detoured.dll이 Load된 것을 확인할 수 있다.

Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
00413000	00001000	sleep5	.detour	imports	Imag	R	RWE	
0F000000	00001000	detoured		PE header	Imag	R	RWE	
0F001000	00001000	detoured	.text	code	Imag	R	RWE	
0F002000	00001000	detoured	.rdata	imports,exp	Imag	R	RWE	
0F003000	00001000	detoured	.data	data	Imag	R	RWE	
0F004000	00001000	detoured	.rsrc	resources	Imag	R	RWE	
0F005000	00001000	detoured	.reloc	relocations	Imag	R	RWE	
10000000	00001000	simple		PE header	Imag	R	RWE	
10001000	0000D000	simple	.text	code	Imag	R	RWE	
1000E000	00003000	simple	.rdata	imports,exp	Imag	R	RWE	
10011000	00005000	simple	.data	data	Imag	R	RWE	
10016000	00002000	simple	.reloc	relocations	Imag	R	RWE	
6FF00000	00010000				Priv	R E	RWE	

[그림 14] detoured.dll 과 simple.dll 이 Memory에 Load된 상태

0x0230) 실무 적용

이제 실전에서 사용할 수 있는 간단한 예로 “MessageBox” Hooking을 걸어보도록 합시다. 먼저 Hooking할 대상 Source Code를 보도록 하겠습니다.

```

void CTestDlg::OnOK()
{
    // OK 버튼 클릭....
    char buff[18];

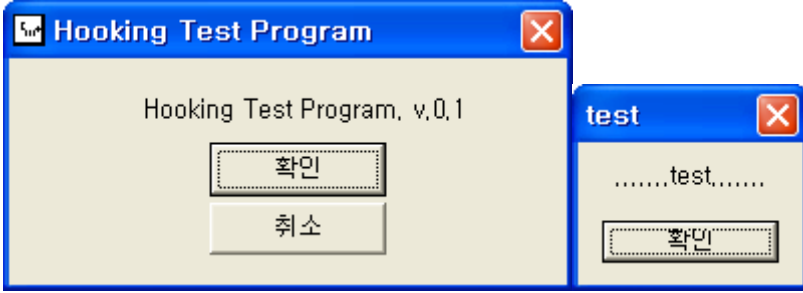
    memcpy(buff, ".....test.....", 18);
    buff[18] = '\0';

    MessageBox(buff, 0, MB_OK);

    OutputDebugString(buff);
    CDialog::OnOK();
}

```

O.K 버튼 Click시  
MessageBox 출력



The screenshot shows two windows. The main window, titled "Hooking Test Program", contains a text label "Hooking Test Program, v.0.1" and two buttons: "확인" (OK) and "취소" (Cancel). A smaller dialog box titled "test" is open, showing the text ".....test....." and an "확인" (OK) button.

[그림 15] Hooking Test Program Source Code

[그림 15]와 같이 단순 Click을 통해서 MessageBox를 출력해주고 있다.

MessageBox의 내용은 “.....test.....” 문자열이다. 그럼 “test” 문자열을 “change” 로 바꾸는 Hooking Program을 만들어 보자.

```
#include <stdio.h>
#include <windows.h>
#include "..\..\WinInclude\detours.h"

static LONG dwSlept = 0;

/*
int MessageBox(          HWND hWnd,
    LPCTSTR lpText,
    LPCTSTR lpCaption,
    UINT uType
);
*/
static int (WINAPI *TrueMeg)(HWND hWnd, LPCTSTR lpText, LPCTSTR lpCaption, UINT uType) = MessageBox;
// Hooking한 MessageBox 내용.
int WINAPI FakeMeg(HWND hWnd, LPCTSTR lpText, LPCTSTR lpCaption, UINT uType)
{
    int temp;
    temp = TrueMeg(hWnd, lpText, lpCaption, uType);
    lpText = ".....Change.....";
    return TrueMeg(hWnd, lpText, lpCaption, uType);
}

BOOL WINAPI DllMain(HINSTANCE hinst, DWORD dwReason, LPVOID reserved)
{
    LONG error;
    (void)hinst;
    (void)reserved;

    if (dwReason == DLL_PROCESS_ATTACH) {
        printf("simple.dll: Starting.\n");
        fflush(stdout);

        // DetourCreateProcessWithDll()을 통하여 process 생성한 후 import table의 정보를 memory에 저장.
        DetourRestoreAfterWith();

        // attaching 또는 detaching 하기 위한 준비 상태.
        DetourTransactionBegin();

        // 현재 Process의 Thread Handle를 얻어옴.
        DetourUpdateThread(GetCurrentThread());

        // 핵심 부분 : fakeMeg를 TrueMeg()로 대체 한다.
        // TrueMeg는 MessageBox()를 재정의 한것이다.
        DetourAttach(&(PVOID)&TrueMeg, FakeMeg);

        // Detours 적용.
        error = DetourTransactionCommit();
    }
}
```

```

// Detours 적용.
error = DetourTransactionCommit();

if (error == NO_ERROR) {
    printf("simple.dll: Detoured Sleep().\n");
}
else {
    printf("simple.dll: Error detouring Sleep(): %d\n", error);
}
}
else if (dwReason == DLL_PROCESS_DETACH) {
    // detaching하기 위한 준비.
    DetourTransactionBegin();
    // 현재 Process의 Thread Handle를 얻어옴.
    DetourUpdateThread(GetCurrentThread());

    // 원상 복구 - fakeMeg()를 TrueMeg()로 변환.
    DetourDetach(&(PVOID&)TrueMeg, fakeMeg);

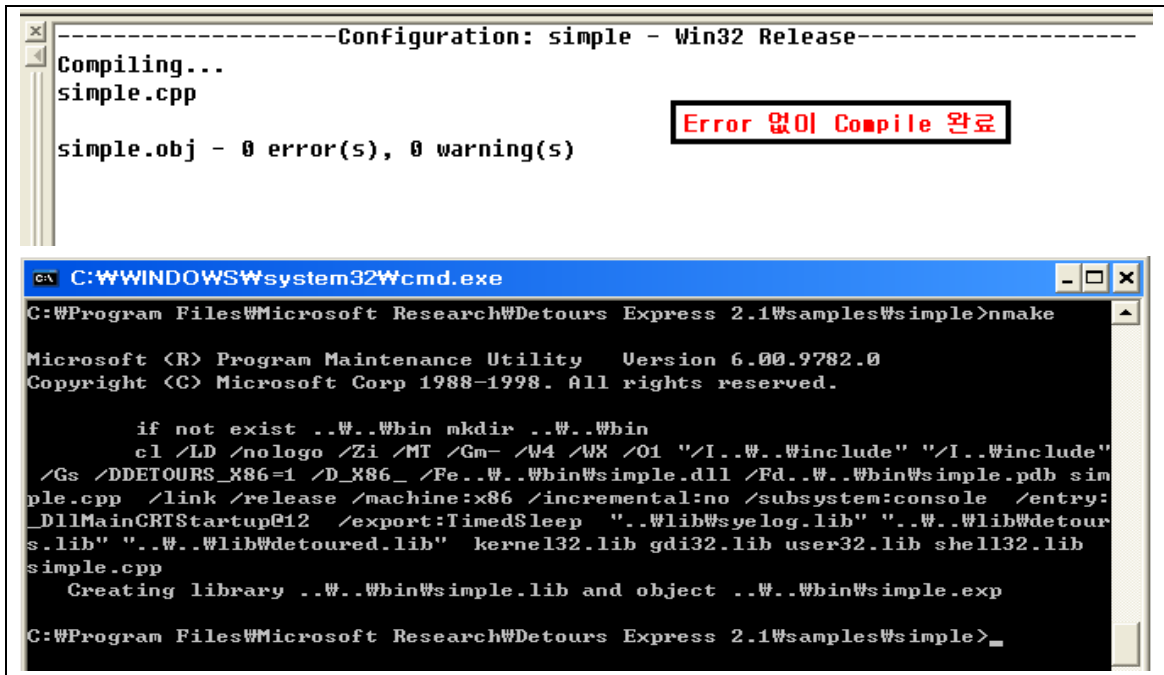
    // Detours 적용.
    error = DetourTransactionCommit();

    printf("simple.dll: Removed Sleep() (result=%d), slept %d ticks.\n",
        error, dwSlept);
    fflush(stdout);
}
return TRUE;
}
}

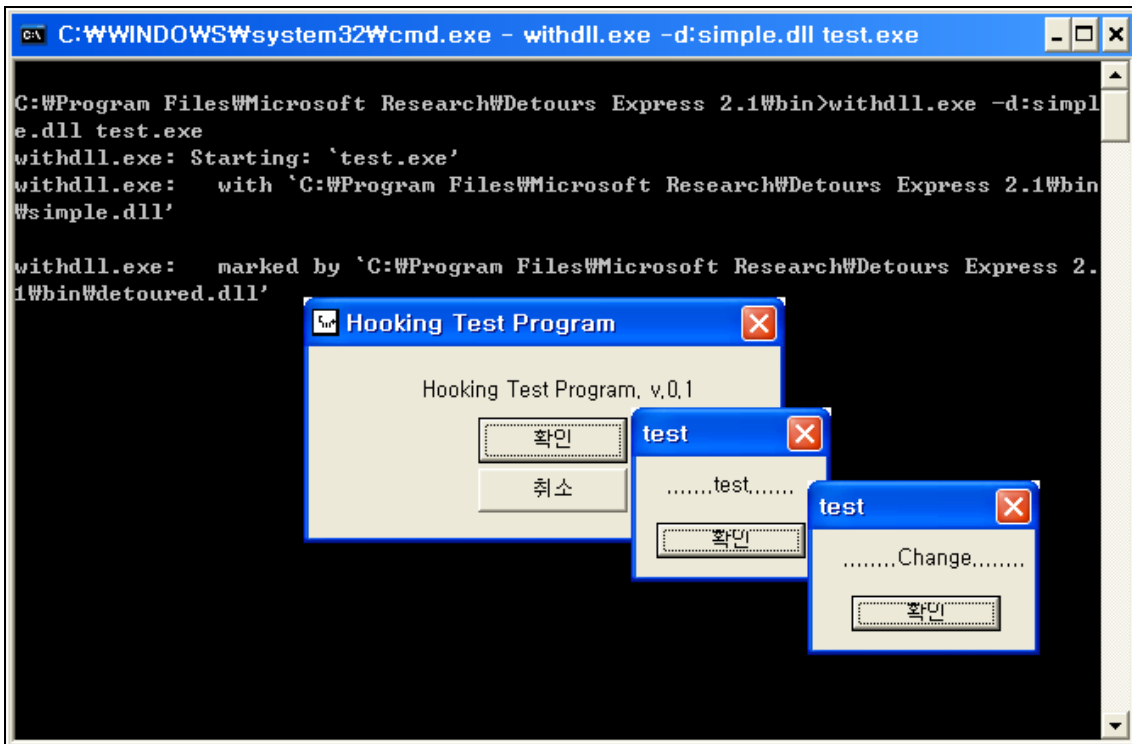
```

[그림 16] MessageBox() Hooking Source Code

MessageBox()의 원형을 기반으로 쌍둥이 함수를 만들어야 한다. 여기에서 MessageBox()의 원형을 기반으로 fakeMeg()를 만들었다. nmake를 하기전에 Compile을 통해서 Error 부분이 있는지를 체크 후 nmake을 통하여 DLL을 만든다.



[그림 17] simple.DLL 생성



[그림 18] Hooking 성공 화면

위의 절차를 따라서 함수를 Hooking을 하면 원하는 결과를 얻을 수 있을 것이다. 또한, detours Hooking 방식을 언제 사용해야 하는지에 대한 선택



방법론은 마지막에 언급하도록 하겠다. 이젠 iAT Hooking으로 넘어가도록 하자.

#### 0x0300. iAT Hooking ?

Detours의 API 함수를 이용한 Simple한 Hooking 방법이라면 iAT Hooking 방식은 Import Table Address를 수정하여 Hooking 작업을 하는 방법으로 Detours 보다 조금 복잡한 방법을 걸치게 된다. 하지만 기존에 사용했던 Tutorial이 많이 존재하므로 쉽게 접근할 수 있다.

iAT(Import Address Table)에 대한 내용은 이 문서에서는 언급하지 않겠다. 궁금한 내용이 있으면 Google 형님에게 물어보길 바란다. ^^

자 그럼 이제 실습을 하도록 하자.

#### 0x0310) iAT Hooking Example

##### 0x0311) DLL Injection

Target Program은 [그림 15]를 기반으로 하겠다. 단순 Message를 출력해 주는 프로그램이다. 내부에서 사용하는 memcpy 함수를 Hooking 하여 “.....test.....” 문자열을 “.....change.....” 로 바꾸는 Hooking Program을 만들어 보도록 하자.

먼저 해야 하는 순서는 실행중인 Process에 DLL을 Attach 시킬 수 있는 Injection Program을 만들어야 한다. Injection Program은 아래의 순서를 따르게 된다.

1. **Target Process** 선택
2. Attach 할 **DLL 파일 경로**

아주 간단하다. 자 그럼 위의 순서를 기반으로 Injection Program을 작성해 보자. 그럼 Target Process를 어떻게 선택할 것인가?, Target Process를 선택 하였다고 했을 때 Windows에서는 어떻게 처리하는가?를 먼저 생각해야 한다. 해결 포인트는 바로 **Handle** 이다. Handle을 이용해서 선택된 Program에 Memory에 DLL 을 Injection 시켜주게 된다.

그럼 Target Process의 handle을 얻는 부분을 보도록 하자.

(※ 코드는 그림으로만 표기하겠다. 그래야지 한번이라도 작성 할 수 있으므로 @^^@)

```

int main(int argc, char* argv[])
{
    char buff[1024] = "";
    // FindWindow()의 Handle.
    HWND hWnd;
    // FindWindow()의 ProcessId.
    DWORD pid;

    if (argc < 2) {
        printf ("wt#####\n");
        printf ("wtwt  DLL Injection Tool v.0.1\n");
        printf ("wtwtwt  program by crattack\n");
        printf ("wtwtwt  data. 2008. 11. 11\n");
        printf ("wt#####\n");
        printf ("Usage : %s [DLL Path] [Target Window Title]\n", argv[0]);
        printf ("ex) %s Hook_iAT.DLL w"Hooking Test Program"\n", argv[0]);
        return 0;
    } else
        hWnd = FindWindow(0, argv[2]);
        Hooking할 Target Window가 들어 있는
        argv[2]의 문자열로 FindWindow()를 이
        이용하여 Handle을 얻는다.

    if(hWnd == 0)
    {
        wsprintf(buff, "Injection Program ..... %s를 먼저 실행 시키세요.", argv[1]);
        OutputDebugString(buff);
        return 0;
    }

    GetWindowThreadProcessId(hWnd, &pid);
    FindWindow()에서 얻어진 Handle
    값을 이용하여 ProcessId를 얻어
    Hooking할 초석을 뒀는다.

    wsprintf(buff, "Injection Program 1. .... Process ID : %d", pid);
    OutputDebugString(buff);

    if (DllInject(pid, argv[1]) != 0)
        printf("DLL Injection Error\n");
    return 0;
}

```

[그림 19] Target Program Handle & ProcessId 얻기

그럼 이전 Injection할 ProcessID를 얻었다. 그럼 메모리를 확보하여 DLL을 Injection하는 부분을 보도록 하자. DLL Injection 하는 부분은 바로 **"DllInject()"** 가 그 역할을 한다. 내부를 보도록 하자.

```

    if (DllInject(pid, argu[1]) != 0)
        printf("DLL Injection Error\n");
    return 0;
}

// DLL Inject : 다른 프로세스의 주소 공간에 특정 DLL 삽입.
int DllInject(DWORD pid, char* path)
{
    char buff[1024] = "";
    // process 권한 관련 변수.
    HANDLE hProcess;
    // "Kernel32.DLL"의 모듈 handle
    HMODULE hKernel32;
    // "LoadLibraryA" 함수 Address
    PTHREAD_START_ROUTINE pLoadLibrary;
    // "VirtualAlloc" 함수의 Handle
    void* pVirtualAlloc;

    wsprintf(buff, "Injection Program 2. .... pid : 0x%08X", pid);
    OutputDebugString(buff);

    // Process 권한 모두 Open
    hProcess = OpenProcess(PROCESS_ALL_ACCESS, 0, pid);

    if (!hProcess) return 1;
    else {
        wsprintf(buff, "Injection Program 3. .... hProcess : 0x%08X", hProcess);
        OutputDebugString(buff);
    }

    // 내 프로세스에 있는 KERNEL32.DLL의 주소와 LoadLibrary의 주소를 구한다.
    hKernel32 = GetModuleHandle("kernel32.dll"); // kernel32.dll 모듈의 핸들값을 GetModuleHandle로 구한다.
    if (!hKernel32) return 1;

    wsprintf(buff, "Injection Program 4. .... hDll : 0x%08X", hKernel32);
    OutputDebugString(buff);

    // PTHREAD_START_ROUTINE은 미리 정의되어 있는 스레드 함수 모양의 함수 포인터 type
    pLoadLibrary = (PTHREAD_START_ROUTINE)GetProcAddress(hKernel32, "LoadLibraryA");
    if (!pLoadLibrary) return 1;

    wsprintf(buff, "Injection Program 5. .... pLoadLibrary : 0x%08X", pLoadLibrary);
    OutputDebugString(buff);

    // 주소공간에 메모리를 할당하고, DLL의 경로를 복사
    // VirtualAlloc은 가상 메모리를 할당하는 API이지만, Ex가 붙으면 다른 프로세스의 주소공간을 할당 가능.
    pVirtualAlloc = VirtualAllocEx(hProcess, 0, // 원하는 주소(0은 알아서 해달라)
        strlen(path)+1, // 크기
        MEM_RESERVE | MEM_COMMIT, // 예약과 동시 확정
        PAGE_READWRITE);
}

```

```

if (!pVirtualAlloc)
{
    wsprintf(buff, "Injection Program 6. .... VirtualAlloc Error");
    OutputDebugString(buff);
    return 0;
}

wsprintf(buff, "Injection Program 6. .... pVirtualAlloc : 0x%08X", pVirtualAlloc);
OutputDebugString(buff);

// 이제 DLL의 경로를 담은 문자열 복사
// WriteProcessMemory()는 다른 프로세스 주소 공간에 write 가능
DWORD len;

if (!WriteProcessMemory(hProcess, pVirtualAlloc, path, strlen(path)+1, &len))
    return 1;

// 새로운 스레드를 만든다.
// CreateRemoteThread는 다른 프로세스에 스레드를 생성시킨다.
HANDLE hThread = CreateRemoteThread(hProcess, // Target 핸들
    0, 0, pLoadLibrary, pVirtualAlloc, // 함수, 파라미터
    0, 0);

wsprintf(buff, "Injection Program 7. .... hThread : 0x%08X", hThread);
OutputDebugString(buff);

wsprintf(buff, "Injection Program 8. .... path : %s", path);
OutputDebugString(buff);

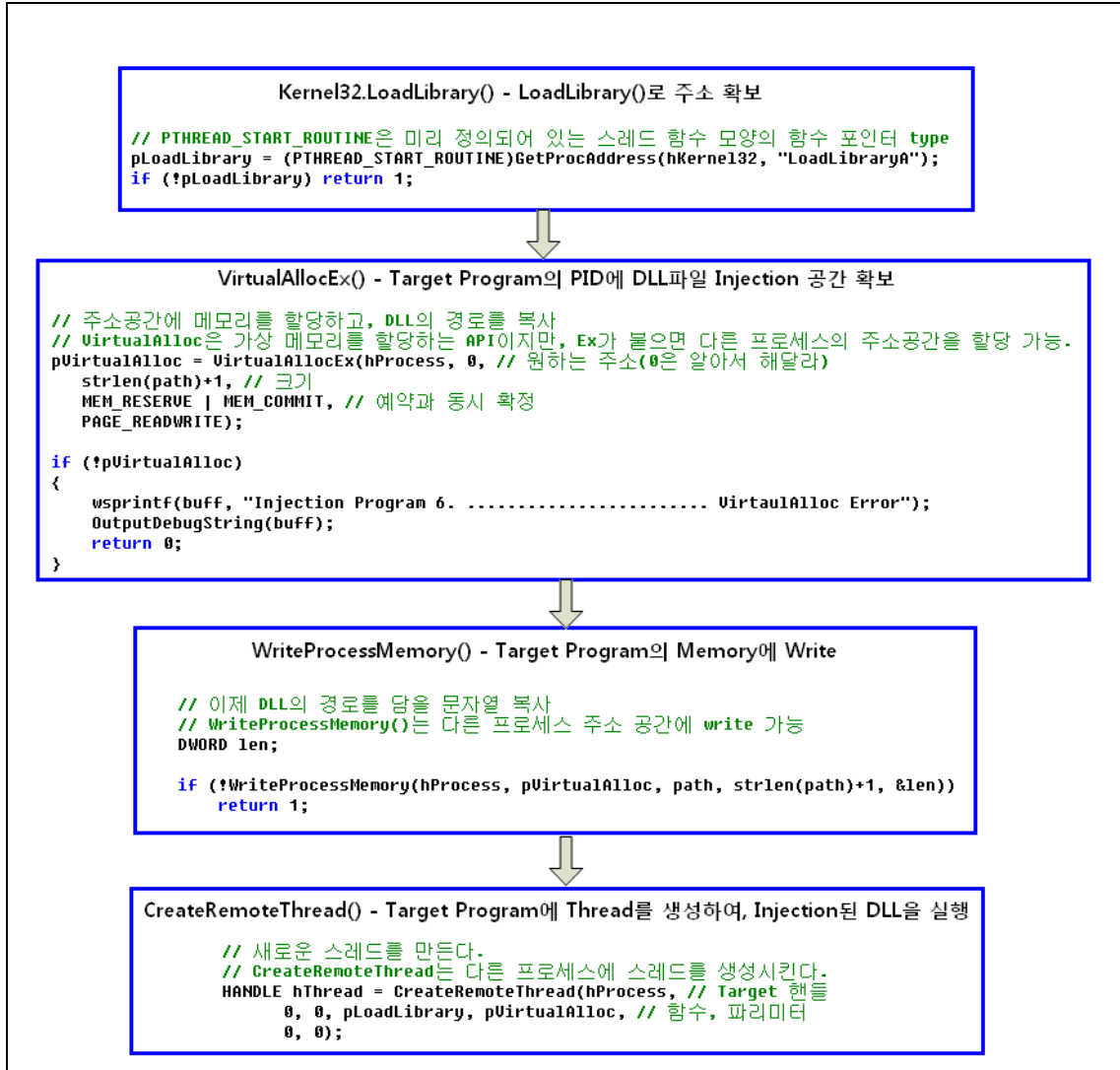
CloseHandle(hThread);
CloseHandle(hProcess);

return 0;
}

```

[그림 20] DllInject()의 Code

자, [그림 20]에서 보면 주석처리를 해놓았다. 하나하나 따져 보도록 하자. 절차는 이렇다.

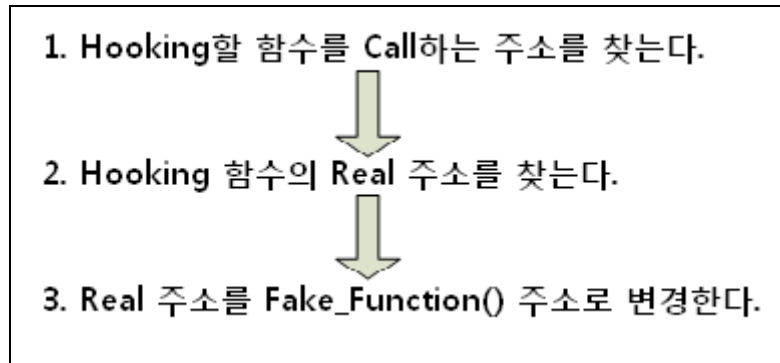


[그림 21] DLL이 Target Process에 Injection 되는 과정

[그림 21] 과정을 제대로 이해한다면 이젠 다른 Process에 청자가 원하는 프로그램을 쉽게 넣을 수 있을 것이다. 만약 이해가 안됐다면 각각의 함수에 대해서 공부를 더 해보길 바란다. **DLL Injection에 필요한 최소한의 함수만을 추출해서 정리한 것이므로 꼭! 알고 있어야 한다.**

## 0x0312) Hooking Code

Hooking Code는 함수를 호출하는 주소를 임의로 만든 Fake\_Function()으로 변경하는 과정을 보여줄 것이다. 그럼 들어가기 전에 Hooking Code가 처리되는 과정을 보도록 하자.



[그림 22] Hooking Code 처리 절차

Hooking Code의 예는 “memcpy()” 를 Hooking 할 것이다. [그림 22]의 순서와 같이 Code를 분석해보자.

먼저, 1번 순서를 따르겠다. “Hooking할 함수를 Call하는 주소를 찾는다.”

00401B48	CALL	DWORD PTR DS:[<&USER32.GetSystemMe	USER32.GetSystemMetrics
004020A0	CALL	<JMP.&MSUCRTD._initterm>	MSUCRTD._initterm
004020DB	CALL	<JMP.&MSUCRTD._initterm>	MSUCRTD._initterm
00401CA8	CALL	<JMP.&MSUCRTD.memcpy>	MSUCRTD.memcpy
<del>00401FA1</del>	<del>CALL</del>	<del>DWORD PTR DS:[&lt;&amp;MSUCRTD._onexit&gt;]</del>	<del>MSUCRTD._onexit</del>
00401CCA	CALL	DWORD PTR DS:[<&KERNEL32.OutputDeb	kernel32.OutputDebugStringA
0040205A	CALL	DWORD PTR DS:[<&MSUCRTD._p_commo	MSUCRTD._p_commode
0040204C	CALL	DWORD PTR DS:[<&MSUCRTD._p_fmode	MSUCRTD._p_fmode
0040230C	CALL	DWORD PTR DS:[<&MSUCRTD.setmbcp>]	MSUCRTD.setmbcp

00401C9D	. 6A 12	PUSH 12	n = 12 (18.) src = test.004153D4 dest memcpy
00401C9F	. 68 D4534100	PUSH test.004153D4	
00401CA4	. 8D45 E8	LEA EAX,DWORD PTR SS:[EBP-18]	
00401CA7	. 50	PUSH EAX	
00401CA8	. E8 43030000	CALL <JMP.&MSUCRTD.memcpy>	

00401FEF	CC	INT3	
00401FF0	FF25 A476410	JMP DWORD PTR DS:[&MSUCRTD.memcpy]	MSUCRTD.memcpy
00401FF6	CC	INT3	
00401FF7	CC	INT3	
00401FF8	CC	INT3	
00401FF9	CC	INT3	
00401FFA	CC	INT3	
00401FFB	CC	INT3	
00401FFC	CC	INT3	
00401FFD	CC	INT3	
00401FFE	CC	INT3	
00401FFF	CC	INT3	
00402000	55	PUSH EBP	
00402001	8BEC	MOV EBP,ESP	
00402003	6A FF	PUSH -1	
00402005	68 60544100	PUSH test.00415460	
0040200A	68 68224000	PUSH <JMP.&MSUCRTD._except_handler3>	SE handler installation
0040200F	64:A1 000000	MOV EAX,DWORD PTR FS:[0]	
00402015	50	PUSH EAX	
00402016	64:8925 0000	MOV DWORD PTR FS:[0],ESP	
0040201D	83C4 94	ADD ESP,-6C	

DS:[004176A4]=102199C0 (MSUCRTD.memcpy)  
Local call from 00401CA8

**DS:[004176A4]=102199C0 (MSUCRTD.memcpy)**  
**Local call from 00401CA8**

[그림 23] 1. Hooking할 함수를 Call하는 주소를 찾는다.

여기에서 필요한 부분은 [그림 23]의 마지막 Capture에서 나온 주소가 필요하다. DS:[004176A4]=102199C0 (MSUCRTD.memcpy) 좋다. 이젠 공격 포인트를 찾았으니 공격하기 위한 코드를 작성해보자. 실질적으로 Real Function은 필요가 없다. 단지, 어떤 순서로 Hooking이 이뤄지는지를 보여주기 위해서 찾은 것이니 너무 깊이 생각하지 않아도 될 것 같다.

단지, Fake\_Function의 주소와 Real의 주소를 정확하게 바꿔주면 iAT Hooking이 완료된다. 그럼 코드를 나눠서 설명하겠다.

```

pOrg_Addr = (int*)fake_memcpy;
sprintf(buff, "Hooking DLL 3. .... fake_memcpy address : 0x%08X", pOrg_Addr);
OutputDebugString(buff);

sprintf(buff, "Hooking DLL 4. .... IAT Patched.. [Org Function Addr=%08X] Wt [Now is=%08X] Wt [Target Point Addr=%08X]",
iGetProcAddr, pOrg_Addr, TargetAddr);
OutputDebugString(buff);

// iAT의 내용을 수정.
// DS:[004176A4]=102199C0 (MSUCRTD.memcpy) -> DS:[004176A4]=pOrg_Addr (Hook_IAT.fake_memcpy)
_asm {
    mov eax, TargetAddr
    mov ebx, pOrg_Addr
    mov DWORD PTR DS:[eax], ebx
}

sprintf(buff, "Hooking Complete. .... [원본 함수 주소 : %08X] -> [Hooking 함수 주소 : %08X] 로 변경 되었습니다.", iGetProcAddr, pOrg_Addr);
OutputDebugString(buff);
}
return TRUE;
}

```

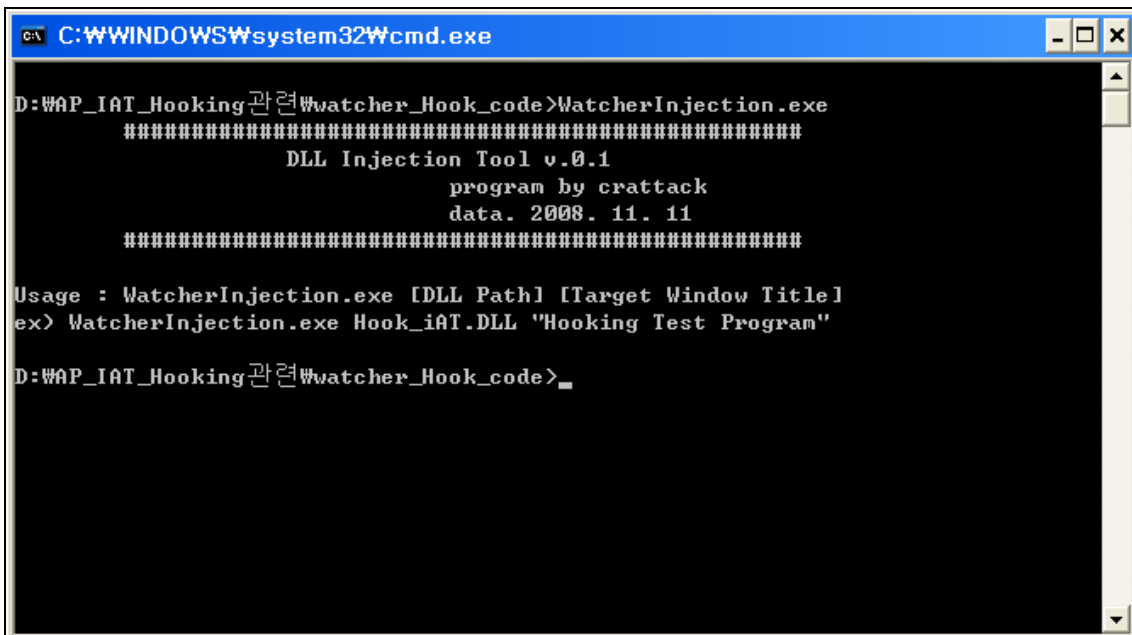
[그림 24] 3. Real 주소를 Fake\_Function() 주소로 변경한다.

Fake\_Function()인 fake\_memcpy()는 DebugView에 Log 찍는 역할만 하도록 변경해 놓았다.

```
// 함수 원형.  
//void * __cdecl memcpy(void *, const void *, size_t);  
  
void __cdecl fake_memcpy(void *arg1, const char *arg2, size_t count)  
{  
  
    char *buff = "";  
    memset(buff, NULL, sizeof(buff));  
  
    sprintf(buff, "Hooking DLL 6. .... Original Data : %s\n", arg2);  
    OutputDebugString(buff);  
    memcpy(arg1, ".....change.....", 18);  
    sprintf(buff, "Hooking DLL 7. .... Chagne data : %s\n", arg1);  
    OutputDebugString(buff);  
  
    arg1 = NULL;  
    arg2 = NULL;  
    count = 0;  
}
```

[그림 25] Fake\_Function()인 fake\_memcpy()의 Source Code

이제는 여태까지 작성해 놓은 프로그램을 기반으로 하나하나 실행 과정을 보여주도록 하겠다. 따라서 해보자. 먼저 “DLL Injection Tool”을 실행 시킨다.



[그림 26] DLL Injection Tool 실행



```

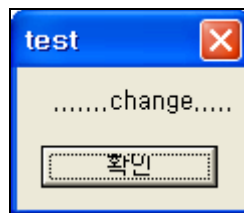
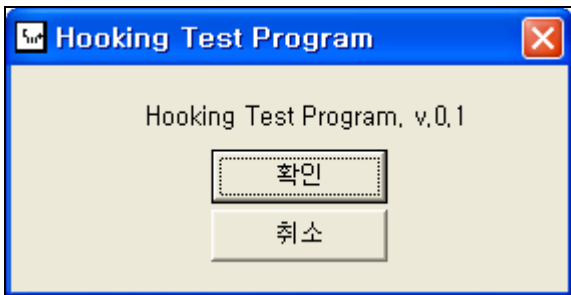
C:\WINDOWS\system32\cmd.exe
D:\WAP_IAT_Hooking\관련\watcher_hook_code>WatcherInjection.exe
#####
          DLL Injection Tool v.0.1
          program by crattack
          data. 2008. 11. 11
          #####

Usage : WatcherInjection.exe [DLL Path] [Target Window Title]
ex> WatcherInjection.exe Hook_IAT.DLL "Hooking Test Program"

D:\WAP_IAT_Hooking\관련\watcher_hook_code>WatcherInjection.exe "D:\WAP_IAT_Hooking\
관련\watcher_hook_code\Hook_IAT.dll" "Hooking Test Program"

D:\WAP_IAT_Hooking\관련\watcher_hook_code>

```



```

DebugView on WWCsLEE (local)
File Edit Capture Options Computer Help
#####
# Time Debug Print
0 0.00000000 [5272] Injection Program 1. .... Process ID : 2972
1 0.00002251 [5272] Injection Program 2. .... pid : 0x00000B9C
2 0.00005027 [5272] Injection Program 3. .... hProcess : 0x000007C0
3 0.00007612 [5272] Injection Program 4. .... hDll : 0x7C800000
4 0.00010114 [5272] Injection Program 5. .... pLoadLibrary : 0x7C801D77
5 0.00012915 [5272] Injection Program 6. .... pVirtualAlloc : 0x00B40000
6 0.00030443 [5272] Injection Program 7. .... hThread : 0x000007B8
7 0.00040185 [5272] Injection Program 8. .... path : D:\WAP_IAT_Hooking\관련\watcher_hook_code\Hook_IAT.dll
8 0.00106522 [2972] Hooking DLL 1. .... Preparing to patch IAT Current [GetModule Addr=10200000] [Hooking Function Address=102199C0]
9 0.00108704 [2972] Hooking DLL 2. .... Target Point Address : 0x004176A4
10 0.00115211 [2972] Hooking DLL 3. .... fake_memcpy address : 0x1000100A
11 0.00119539 [2972] Hooking DLL 4. .... IAT Patched. [Org Function Addr=102199C0] [Now is=1000100A] [Target Point Addr=004176A4]
12 0.00640466 [2972] Hooking Complete. .... [원본 함수 주소 : 102199C0] -> [Hooking 함수 주소 : 1000100A] 로 변경 되었습니다.

```

[그림 27] DLL Injection 성공

**R Found intermodular calls**

Address	Disassembly	Destination
00401AFB	CALL <JMP.&MFC42D.#3365>	MFC42D.#3365
00401B0F	CALL <JMP.&MFC42D.#413>	MFC42D.#413
00401B20	CALL <JMP.&MFC42D.#2993>	MFC42D.#2993
00401B28	CALL <JMP.&MFC42D.#4475>	MFC42D.#4475
00401B34	CALL DWORD PTR DS:[&USER32.GetSystemMe	USER32.GetSystemMetrics
00401B3C	CALL <JMP.&MSUCRTD._chkesp>	MSUCRTD._chkesp
00401B48	CALL DWORD PTR DS:[&USER32.GetSystemMe	USER32.GetSystemMetrics
00401B50	CALL <JMP.&MSUCRTD._chkesp>	MSUCRTD._chkesp
00401B5B	CALL <JMP.&MFC42D.#454>	MFC42D.#454
00401B67	CALL <JMP.&MFC42D.#2324>	MFC42D.#2324
00401B6F	CALL <JMP.&MFC42D.#5072>	MFC42D.#5072
00401B85	CALL <JMP.&MFC42D.#3201>	MFC42D.#3201
00401B88	CALL <JMP.&MFC42D.#1906>	MFC42D.#1906
00401BBF	CALL <JMP.&MFC42D.#646>	MFC42D.#646
00401BC9	CALL <JMP.&MFC42D.#3960>	MFC42D.#3960
00401DE0	CALL <JMP.&MSUCRTD._chkesp>	MSUCRTD._chkesp
00401CA8	CALL <JMP.&MSUCRTD.memcpy>	Hook IAT.10001020
00401C0F	CALL <JMP.&MFC42D.#9547>	MFC42D.#9547
00401CCA	CALL DWORD PTR DS:[&KERNEL32.OutputDeb	kernel32.OutputDebugStringA
00401CD2	CALL <JMP.&MSUCRTD._chkesp>	MSUCRTD._chkesp
00401CDA	CALL <JMP.&MFC42D.#3948>	MFC42D.#3948
00401CE7	CALL <JMP.&MSUCRTD._chkesp>	MSUCRTD._chkesp
00401FA1	CALL DWORD PTR DS:[&MSUCRTD._onexit]	MSUCRTD._onexit
00401FB0	CALL <JMP.&MSUCRTD._d1lonexit>	MSUCRTD._d1lonexit
0040202F	CALL DWORD PTR DS:[&MSUCRTD._set_app	MSUCRTD._set_app_type
0040204C	CALL DWORD PTR DS:[&MSUCRTD._p_fnode	MSUCRTD._p_fnode
0040205C	CALL DWORD PTR DS:[&MSUCRTD._p_fnode	MSUCRTD._p_fnode

**C o\_O - main thr3ad, m0dule test**

00401C86	. 53	PUSH EBX	
00401C87	. 56	PUSH ESI	
00401C88	. 57	PUSH EDI	
00401C89	. 51	PUSH ECX	
00401C8A	. 8D7D A8	LEA EDI,DWORD PTR SS:[EBP-58]	
00401C8D	. B9 16000000	MOV ECX,16	
00401C92	. B8 CCCCCCCC	MOV EAX,CCCCCCCC	
00401C97	. F3:AB	REP STOS DWORD PTR ES:[EDI]	
00401C99	. 59	POP ECX	
00401C9A	. 894D FC	MOV DWORD PTR SS:[EBP-4],ECX	
00401C9D	. 6A 12	PUSH 12	
00401C9F	. 68 D4534100	PUSH test.004153D4	
00401CA4	. 8D45 E8	LEA EAX,DWORD PTR SS:[EBP-18]	
00401CA7	. 50	PUSH EAX	
00401CA8	. E8 43030000	CALL <JMP.&MSUCRTD.memcpy>	dest memcpy
00401CAD	. 83C4 0C	ADD ESP,0C	
00401CB0	. C645 FA 00	MOV BYTE PTR SS:[EBP-6],0	

n = 12 (18.)  
src = test.004153D4  
  
dest  
memcpy

[그림 28] Hooking 된 Test Program을 Reverse 한 결과 (1)

o_o - main thr3ad, m0dule test			
00401FE5	. 48	DEC EAX	
00401FE6	. 5D	POP EBP	
00401FE7	. C3	RETN	
00401FE8	> FF25 AC76410	JMP DWORD PTR DS:[&MSUCRTD.__CxxFrameHa	MSUCRTD.__CxxFrameHandler
00401FEE	CC	INT3	
00401FEF	CC	INT3	
00401FF0	⚡- FF25 A476410	JMP DWORD PTR DS:[<&MSUCRTD.memcpy>]	Hook_IAT.1000100A
00401FF6	CC	INT3	
00401FF7	CC	INT3	
00401FF8	CC	INT3	
00401FF9	CC	INT3	
00401FFA	CC	INT3	
00401FFB	CC	INT3	
00401FFC	CC	INT3	
00401FFD	CC	INT3	
00401FFE	CC	INT3	
00401FFF	CC	INT3	

o_o - main thr3ad, m0dule Hook_IAT			
1000100A	E9 11000000	JMP Hook_IAT.10001020	
1000100F	CC	INT3	
10001010	CC	INT3	
10001011	CC	INT3	
10001012	CC	INT3	
10001013	CC	INT3	
10001014	CC	INT3	
10001015	CC	INT3	
10001016	CC	INT3	
10001017	CC	INT3	
10001018	CC	INT3	
10001019	CC	INT3	
1000101A	CC	INT3	
1000101B	CC	INT3	
1000101C	CC	INT3	
1000101D	CC	INT3	
1000101E	CC	INT3	
1000101F	CC	INT3	
10001020	8B4424 08	MOV EAX,DWORD PTR SS:[ESP+8]	
10001024	56	PUSH ESI	
10001025	50	PUSH EAX	
10001026	68 7C1A0110	PUSH Hook_IAT.10011A7C	ASCII "Hooking DLL 6. .... Original Data : %s"
1000102B	68 005D0110	PUSH Hook_IAT.10015D00	ASCII "Hooking Complete. .... [원본 함수 주소 : 102199C0] ->
10001030	C705 005D0110	MOV DWORD PTR DS:[10015D00],0	
1000103A	E8 03020000	CALL Hook_IAT.10001242	
1000103F	8B35 A4810110	MOV ESI,DWORD PTR DS:[<&KERNEL32.Output kernel32.OutputDebugStringA	
10001045	83C4 0C	ADD ESP,0C	
10001048	68 005D0110	PUSH Hook_IAT.10015D00	ASCII "Hooking Complete. .... [원본 함수 주소 : 102199C0] ->
1000104D	FFD6	CALL ESI	

[그림 29] Hooking 된 Test Program을 Reverse 한 결과 (2)

이상으로 iAT Hooking 방법에 대한 과정을 마치겠다. 몇 가지 주의 해야 하는 상황은 epilogue에서 설명하도록 하고 iAT Hooking Session을 마친다.

0x0400. epilogue

0x0401. Detours를 선택해야 하나? iAT Hooking을 선택해야 하나?

뻑뻑한 내용을 끝까지 읽어주셔서 감사합니다. (또는, 아직 안 읽고 epilogue만 먼저 읽고 계시는 분일지라도^^)

이런 생각이 들것입니다. 도대체 어떨 때 iAT Hooking을 써야 하고 어떨 때 detours를 써야 하는지 내용은 간단합니다. 바로, JMP로 Real 함수로 가느냐 아니면 Call 포인터로 가느냐의 차이입니다. 자, MessageBox()를 호출하는 2가지 방식을 보여드리도록 하겠습니다. 첫 번째 방식은 Static DLL Option으로 Compile한 것이고, 두 번째 방식은 Share DLL Option으로

Compile을 한 것이다.

00480F91	. 8B55 08	MOV EDX,DWORD PTR SS:[EBP+8]	
00480F94	. 52	PUSH EDX	
00480F95	. 8B4D F8	MOV ECX,DWORD PTR SS:[EBP-8]	
00480F98	. E8 DF6B0500	CALL test.004D7B7C	
00480F9D	. 50	PUSH EAX	
00480F9E	. FF15 1CFE5E00	CALL DWORD PTR DS:[<&USER32.MessageBoxA>]	hOwner MessageBoxA
00480FA4	. 8945 FC	MOV DWORD PTR SS:[EBP-4],EAX	
00480FA7	. 8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]	

[그림 30] Compile시 Static DLL Option 사용

00401CB4	. 6A 00	PUSH 0	
00401CB6	. 6A 00	PUSH 0	
00401CB8	. 8D4D E8	LEA ECX,DWORD PTR SS:[EBP-18]	
00401CBB	. 51	PUSH ECX	
00401CBC	. 8B4D FC	MOV ECX,DWORD PTR SS:[EBP-4]	
00401CBF	. E8 B8020000	CALL <JMP.&MFC42D.#3517>	

[그림 31] Compile시 Share DLL Option 사용

[그림 30]과 [그림 31]을 보시면 함수를 Call하는 방식이 틀리게 됩니다. **CALL DWORD PTR DS:[주소]** 방식과 **CALL JMP.주소** 방식입니다. 따라서, JMP로 호출되는 함수는 iAT Hooking을 이용하시면 되며, 포인터 주소로 넘겨지는 함수 호출 방식은 detours로 이용하면 됩니다. 이상으로 API Hooking과 iAT Hooking 방법에 대한 문서를 마칩니다. 10월 31일부터 쓴 문서가 11월 12일에 끝나네요. 업무와 겹쳐서 정리한다고 정리한게 많이 늦었습니다. 이제부터 그럼 Hooking의 세계로 여행을 떠나보시길 바랍니다..^^ here we go~~♪

#### 0x0402. nmake 사용하지 않기

nmake를 분석한 결과 nmake는 이란 함수를 export로 선언하여 컴파일 하여 일반 함수를 외부에서 호출할 수 있도록 해주는 간단한 script라는 것을 알았습니다. 그래서 DLL 파일을 Programming을 함에 있어서 export로 사용할 수 있도록 선언하고 컴파일 하면 굳이 nmake를 사용하지 않아도 됩니다. 선언 하는 방법은 아래와 같습니다.

```
__declspec(dllexport) LPSTR WINAPI fakeStrCat(LPSTR lpString1, LPCSTR lpString2)
{
    LPSTR buff;
    buff = "";
    sprintf(buff, ".....original String : %s", lpString2);
    OutputDebugString(buff);

    ●
    ●
    ●
}
```

[그림 32] export로 선언하여 nmake를 사용하지 않고 compile 하는 방법

그럼 즐거운 Detours와 iAT Hooking의 세계로 빠져 들어가길 바랍니다.