

DLL Injection은 어떻게 이루어지는가?

By **bl4ck3y3**

(<http://hisjournal.net/blog>)

Abstract

루트킷을 비롯하여 바이러스, 악성코드 등 여러 분야에 두루 쓰이는 기법이 DLL Injection입니다. Windows에 한정되어 적용되는 것이지만, Windows 자체의 점유율이 높은 이유로 아주 효과적으로 공격자가 원하는 작업을 수행할 수 있는 방법이죠. 최근 루트킷에 대해 공부하면서 이 DLL Injection이 어떻게 이루어지는 알게 된 것을 정리해봅니다.

Cert-is

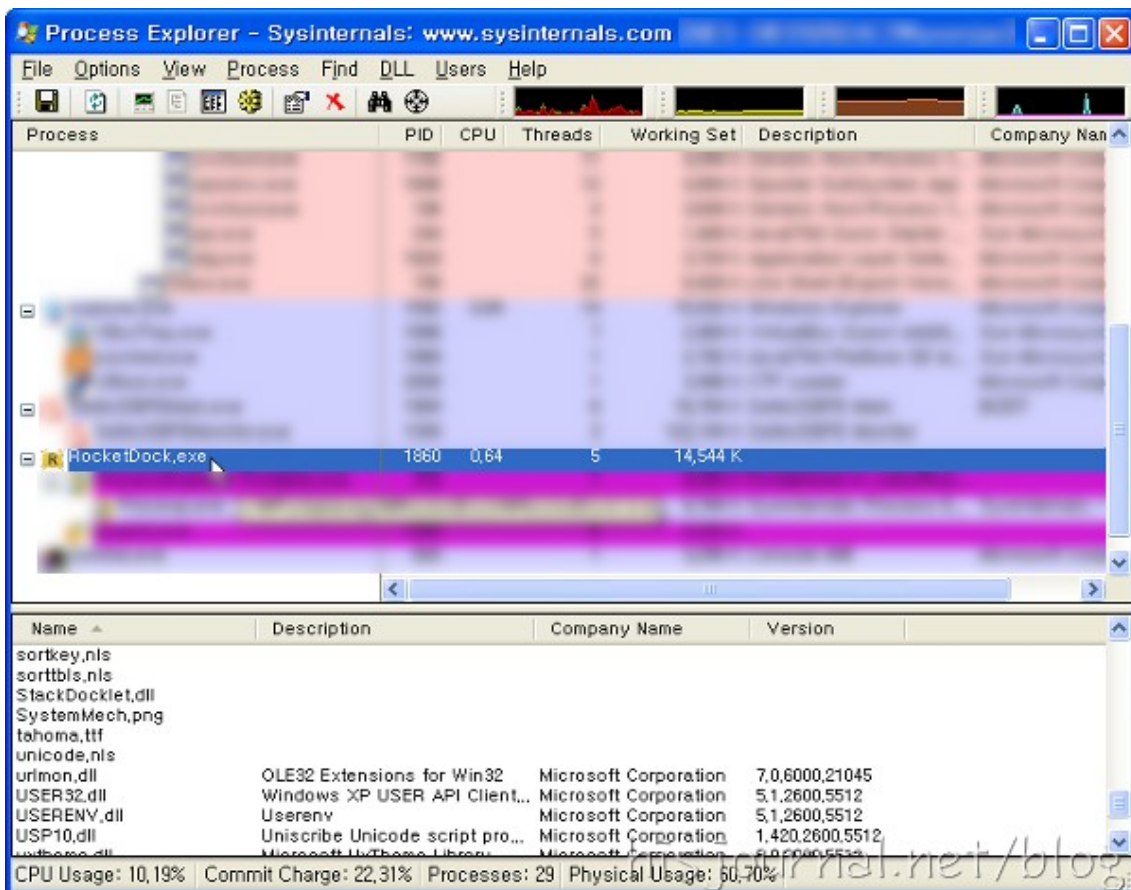
Content

1. DLL? 그게 뭐야?	3
2. 레지스트리 값을 조작하여 Injection	5
2.1. Applnit_DLLs	5
2.2. Email-Worm.Win32.Warezov.nf	6
3. 윈도우 후킹함수를 이용하여 Injection	7
3.1. SetWindowsHookEx ()	7
3.2. HookNotePadDll	8
4. 쓰레드를 생성하여 Injection	10
4.1. CreatRomoteThread ()	10
5. Reference	13

1. DLL? 그게 뭐야?

DLL은 Windows에서 사용되는 동적 연결 라이브러리 실행 파일입니다. 말이 어렵죠? 간단하게 말해서, 실행할 수 있는 아주 작은 프로그램의 단위라고 이해하시면 되겠습니다.

라이브러리라는 말은 아시나요? 영어로 Library, 도서관이라는 뜻이죠? 프로그램이 작업을 수행할 때 이 라이브러리를 참조하게 됩니다. 라이브러리에 포함된 함수, 구조체, (추천하는 방법은 아니지만) 변수 등을 이용하는 것이죠. 우리는 프로그래밍을 할 때 (C 언어 기준으로) 헤더 파일을 집어넣습니다. `stdio.h`, `string.h` 같은 표준 헤더부터, `cimg.h`, `curl.h` 같은 사용자 정의 헤더까지 다양하죠. 이런 것을 정적 라이브러리라고 합니다. 프로그램 자체에 포함되는 라이브러리죠.



[그림 1] RocketDock이 이용하는 많은 DLL 파일들

이에 반해 동적 연결 라이브러리는 [그림 1]에서처럼 프로그램과는 별도로 존재합니다. 가장 많이 볼 수 있는 `kernel32.dll`, `user32.dll`, `gdi32.dll` 등이 동적 연결 라이브러리 파일입니다. 프로그램은 필요할 때 이 DLL을 호출해서 그 안에 있는 함수를 이용하지요.

왜 이렇게 따로 하나구요? 이렇게 함으로써 두 가지 이익을 얻을 수 있기 때문입니다.

첫 째는 프로그램이 메모리에 적재되는 용량을 줄일 수 있다는 점입니다. 프로그램 자체에 처음부터 들어 있는 게 아니라, 필요할 때 호출하는 방식이기 때문에 메모리에 항상 적재되어 있을 필요가 없지요.

두 번째는 DLL의 코드가 변경되어도 원래의 함수 이름이나 인자가 그대로라면, 프로그램을 다시 컴파일 하거나 링킹을 할 필요가 없다는 점입니다. 정적 라이브러리는 변경될 때마다 매번 컴파일하고 링킹을 새로 해야하니 DLL은 정말 편리한 녀석이죠. [그림 1]에서 StackDocklet.dll을 이용하는 것을 볼 수 있죠? 플러그인을 이렇게 DLL 파일로 지원할 수도 있습니다. StackDocklet.dll의 내부 코드가 변경되어도 함수의 형태만 유지된다면 RocketDock 자체를 새로 컴파일할 필요는 없겠죠?

2. 레지스트리 값을 조작하여 Injection

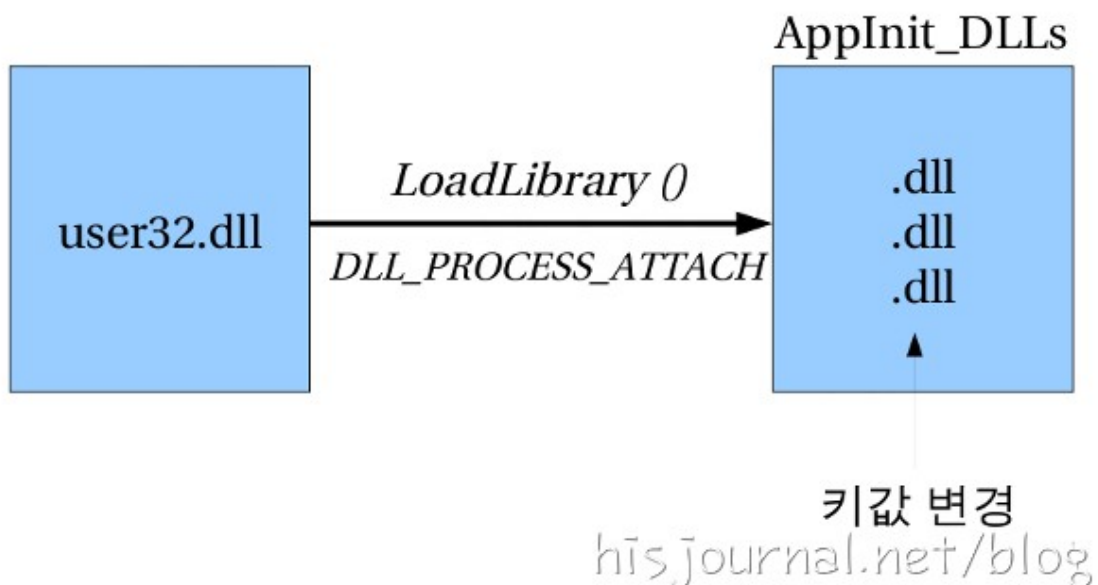
2.1. Applnit_DLLs

DLL Injection에는 여러 방법이 있습니다. 그중 루트킷에 이용되는 세 가지 방법을 알아보겠습니다. 첫 번째로 레지스트리를 조작하는 방법입니다.

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Windows\Applnit_DLLs

regedit에서 위 주소의 레지스트리 값을 확인해보면, 보통은 값이 비어있습니다. 그런데 어떠한 값이 들어있다면, 그리고 그것이 자신이 모르는 어떤 파일이라면, 그것은 악성 코드일 가능성이 있습니다. 이 레지스트리 값이 무엇이나구요? 바로 이 값이 DLL Injection에 참조되는 레지스트리 값입니다.

Windows의 많은 어플리케이션들이 user32.dll을 호출합니다. 이 DLL에는 창관리자를 호출하는 API 함수들이 있습니다. Windows가 기본적으로 GUI 환경이고 그 때문에 어플리케이션들도 user32.dll을 호출하기 마련이죠.



[그림 1] user32.dll이 AppInit_DLLs를 참조하는 개요

그런데 이 user32.dll은 특별한 기능을 가지고 있습니다. [그림 1]에서처럼 user32.dll은 LoadLibrary() 함수를 이용하여 Applnit_DLLs 안에 지정된 DLL들을 호출합니다. 이 때 LoadLibrary()의 인자값으로 DLL_PROCESS_ATTACH를 지정해주기 때문에 프로세스 중간에 DLL이 들어갈 수 있는 것입니다.

공격자가 이 레지스트리 값을 악의적으로 변경한다면 악성코드를 프로세스에 심을 수 있죠. 대신 이것이 적용될려면 재부팅이 필요합니다. 그러나 레지스트리 값이 변경되기 이전에 메모리에 적재된 프로세스일 경우에 재부팅이 필요한 것이지, 변경 된 이후에 메모리에 적재될 프로세스라면 굳이 재부팅이 안 되더라도 DLL이 삽입됩니다.

하지만 이 방법은 해당 레지스트리 값만 조사하면 금방 탐지될 수 있는 단점이 있습니다. 보통 정상적인 경우라면 위에서도 언급했듯이 값이 비어있기 때문이죠. 그렇더라도 레지스트리가 무엇인지 모르는 윈도우즈 사용자가 대부분이기 때문에 이것만으로도 꽤 치명적이라 할 수 있습니다.

2.2. Email-Worm.Win32.Warezov.nf

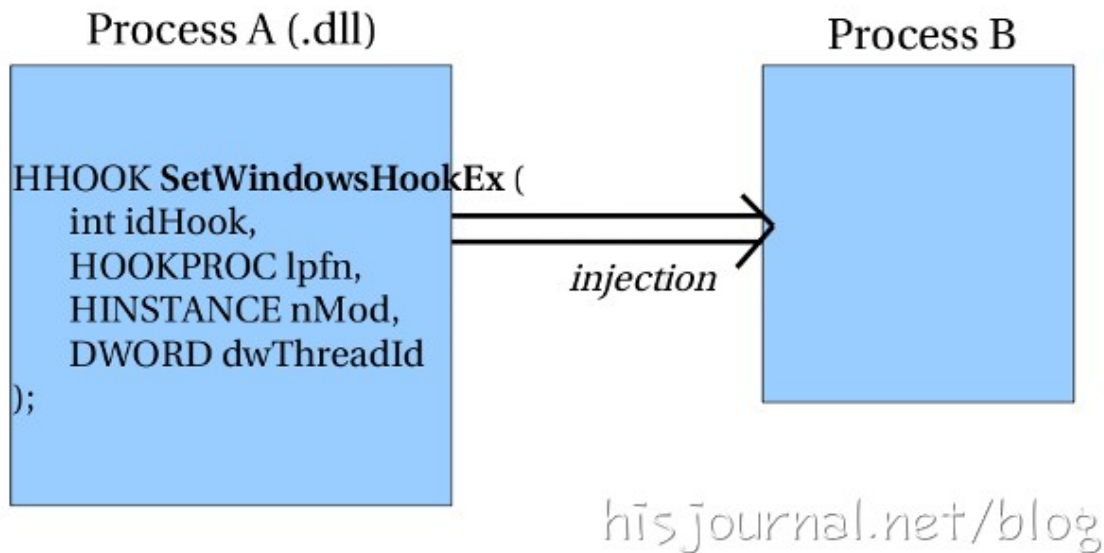
Email-Worm.Win32.Warezov.nf 이라는 웜이 Applnit_DLLs 값을 변경하여 DLL Injection을 시도하는 한 예입니다. 이것은 이메일의 첨부파일이 실행되면 악성 DLL을 C:\WINDOWS\system32에 생성합니다. 그리고 Applnit_DLLs 값을 변경하여 이후에 user32.dll을 호출하는 프로세스에 악성 DLL을 삽입하죠. 이 웜에 대한 자세한 내용은 아래 참고 문헌을 읽어주세요.

“Email-Worm.Win32.Warezov.nf” information
File size : 135579 bytes
MD5 : 88c8a817ae4645755fd1bb6704fabecb
SHA1 : 2018a1b49b99f1c87863b4f7d1d116a181007482
SHA256 : f9bc52f93108f61c26fde81d1a9767ea4d6535c6ee4dc2d2ddf91de168ba300a
TrID : File type identification 100.0% (.EXE) DOS Executable Generic (2000/1)
ssdeep : 3072:a68TxpnlbL+SzOIZeniqbhrPKdoH7rvUFsUloA0:a68TxIB0meicbTUFsuoA
PEiD :
packers (Kaspersky) : PE_Patch, UPack
RDS : NSRL Reference Data Set

3. 윈도우 후킹함수를 이용하여 Injection

3.1. SetWindowsHookEx ()

이번에는 Windows에서 정상적으로 지원하는 후킹 함수를 이용하는 방법을 알아보겠습니다. Windows는 OS와 프로세스 간에 메시지를 주고받는데요. 키보드를 누른다던지, 마우스를 클릭한다던지, 디버깅에 관한 메시지라던지 그런 것들일요. 그리고 그렇게 주고받는 메시지를 후킹할 수 있는 함수도 제공하고 있습니다.



[그림 1] SetWindowsHookEx() 함수의 개요

SetWindowsHookEx() 함수가 그것이지요. [그림 1]처럼 이 함수에 삽입할 DLL의 Handler, 즉 주소와 함수의 주소를 인자로 넣음으로써 메모리에 올라간 프로세스에 삽입하게 됩니다.

```
SetWindowsHookEx (WH_KEYBOARD, KeyHookProc, hModule, NULL);
```

위 예처럼 함수를 작성합니다. hModule은 삽입할 DLL의 주소입니다. 이것은 LoadLibrary() 함수로 구할 수 있죠.

그리고 KeyHookProc은 DLL의 함수인데, GetProcAddress() 함수로 구할 수 있고, CallBack 함수로 등록할 수도 있습니다. 주의할 것은 KeyHookProc는 CallNextHookEx() 함수를 반환해야 한다는 것입니다. 후킹이 끝나고 나서 원래의 작업을 하기 위해서요. 만약, 원래의 작업을 하지 않는다면, 사용자가 무엇인가가 잘

못 되었다는 것을 알 수 있고 자칫하면 블루스크린을 띄울 수도 있거든요.

[그림 1]에서 보면 네 번째 인자로 dwThreadId가 지정되어 있는데, 이것은 DLL을 삽입하려는 스레드의 ID입니다. GetCurrentThreadId() 함수로 구할 수 있습니다. 이 값이 0이라면, 현재 Windows 데스크탑의 모든 스레드가 대상이 되죠.

WH_CALLWNDPROC	WH_KEYBOARD
WH_CALLWNDPROCRET	WH_KEYBOARD_LL
WH_CBT	WH_MOUSE
WH_DEBUG	WH_MOUSE_LL
WH_FOREGROUNDIDLE	WH_MSGFILTER
WH_GETMESSAGE	WH_SHELL
WH_JOURNALPLAYBACK	WH_SYSMSGFILTER
WH_JOURNALRECORD	hisjournal.net/blog

[그림 2] idHook 테이블

첫 번째 인자가 남았는데 이것이 후킹할 메시지입니다. OS와 여기에 지정된 메시지를 주고받는 스레드를 후킹하지요. 이 인자에 들어갈 수 있는 값은 [그림 2]에 정리되어 있습니다. 키보드, 마우스, 디버깅, 셸과 관련된 15개의 메시지들이 있습니다. 이 메시지들에 대해 더 자세한 내용은 아래 참고 문헌에 나와 있으니 여기서는 생략하겠습니다.

마지막으로 작업이 모두 끝나면 메모리 반환을 위해서 UnhookWindowsHookEx() 함수로 후킹을 풀어줘야 합니다.

3.2. HookNotePadDll

아래는 SetWindowsHookEx() 함수를 이용하는 HookNotePadDll의 소스입니다.

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

```
#pragma data_seg(".npdata")
    HINSTANCE hModule=NULL;
    HHOOK hKeyHook=NULL;
    HWND hWndBeeper=NULL;
#pragma data_seg()

#pragma comment (linker, "/SECTION:.npdata,RWS")

LRESULT CALLBACK KeyHookProc(int nCode, WPARAM wParam, LPARAM lParam)
{
    if (nCode>=0)
    {
        SendMessage(hWndBeeper,WM_USER+1,wParam,lParam);
    }
    return CallNextHookEx(hKeyHook,nCode,wParam,lParam);
}

extern "C" __declspec(dllexport) void InstallHook(HWND hWnd)
{
    hWndBeeper=hWnd;
    hKeyHook=SetWindowsHookEx(WH_KEYBOARD,KeyHookProc,hModule,NULL);
}

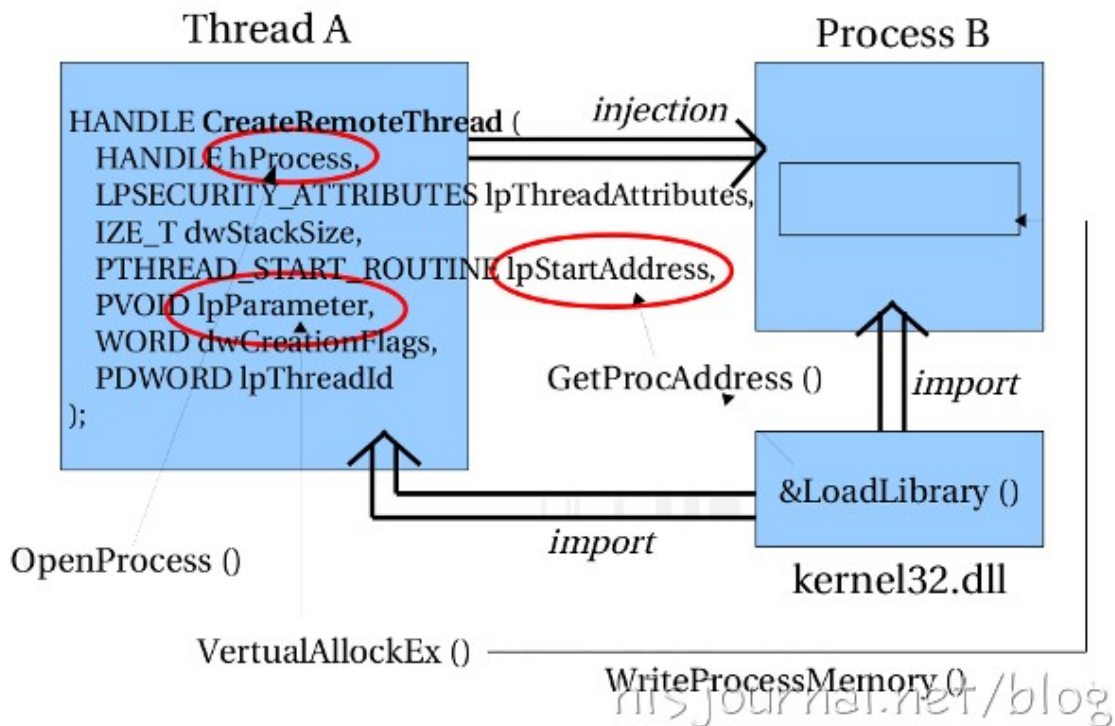
extern "C" __declspec(dllexport) void UninstallHook()
{
    UnhookWindowsHookEx(hKeyHook);
}

BOOL WINAPI DllMain(HINSTANCE hInst, DWORD fdwReason, LPVOID lpRes)
{
    switch (fdwReason)
    {
        case DLL_PROCESS_ATTACH:
            hModule=hInst;
            break;
        case DLL_PROCESS_DETACH:
            break;
    }
    return TRUE;
}
```

4. 쓰레드를 생성하여 Injection

4.1. CreatRemoteThread ()

DLL Injection의 마지막 세 번째 방법은 CreatRemoteThread() 함수를 이용하는 것입니다.



[그림 1] CreatRemoteThread() 의 개요

CreatRemoteThread() 함수는 이름 그대로 쓰레드를 만드는 API 함수입니다. 원격은 다른 프로세스에서 쓰레드를 생성하기 때문에 붙은 것이죠. [그림 1]과 같은 구조로 구성되어 있고, Windows NT 계열에서 지원합니다.

첫 번째 인자인 hProcess는 쓰레드를 생성할 프로세스를 가리킵니다. Windows에서 제공하는 OpenProcess() API 함수를 이용해서 구할 수 있으며, PID(Process ID) 값을 이용합니다. OpenProcess()의 원형은 아래와 같습니다.

```
HANDLE WINAPI OpenProcess(
    __in DWORD dwDesiredAccess,
    __in BOOL bInheritHandle,
```

```
__in DWORD dwProcessId  
);
```

네 번째 인자, lpStartAddress는 LoadLibrary() 함수를 가리킵니다. LoadLibrary()를 이용하여 DLL을 삽입하기 때문이죠. 원래는 쓰레드를 생성할 프로세스에서의 LoadLibrary()를 가리켜야 하지만, Windows는 각 프로세스의 kernel32.dll의 베이스주소를 일정하게 관리하기 때문에 DLL Loader에서 사용하는 kernel32.dll의 LoadLibrary()를 가리키더라도 문제 없다고 합니다. LoadLibrary()의 주소는 GetProcAddress()를 이용합니다. 원형은 아래와 같습니다.

```
FARPROC WINAPI GetProcAddress(
```

```
__in HMODULE hModule,  
__in LPCSTR lpProcName  
);
```

다섯 번째 인자인 lpParameter는 프로세스 안에서 쓰레드가 위치할 메모리 영역을 가리킵니다. CreateRemoteThread()는 프로세스 안에 쓰레드를 생성하는 것이므로 쓰레드를 위한 메모리를 별도로 생성해주어야 하지요. 이 때 사용하는 API 함수가 VirtualAllocEx()와 WriteProcessMemory()입니다. VirtualAllocEx()가 메모리를 할당하면, WriteProcessMemory()로 쓰는 것이죠. 각각의 원형은 아래와 같습니다.

```
LPVOID WINAPI VirtualAllocEx(
```

```
__in HANDLE hProcess,  
__in_opt LPVOID lpAddress,  
__in SIZE_T dwSize,  
__in DWORD flAllocationType,  
__in DWORD flProtect  
);
```

```
BOOL WINAPI WriteProcessMemory(
```

```
__in HANDLE hProcess,  
__in LPVOID lpBaseAddress,  
__in LPCVOID lpBuffer,  
__in SIZE_T nSize,  
__out SIZE_T *lpNumberOfBytesWritten
```

```
);
```

나머지 인자들은 lp*는 NULL로, dw*는 0으로 넣습니다. 이 중에서 dwCreationFlags에 0을 넣는 것은 쓰레드가 생성되는 즉시 실행하라는 뜻입니다.

전체적으로 정리해보죠. OpenProcess()로 DLL을 집어넣을 프로세스를 구합니다. 그리고 GetProcAddress()로 DLL을 로드하는 LoadLibrary()의 주소를 구합니다. 마지막으로 VirtualAllocEx()로 DLL이 위치할 메모리를 할당하고 WriteProcessMemory()로 쓰게 됩니다. 이 모든 것들을 CreateRemoteThread()의 인자값으로 넣어서 DLL Loader에서 실행합니다. 이러면 DLL Injection이 된 것입니다. 간단하죠?

하지만 이 방법은 간단한 대신 쉽게 탐지되기도 합니다. CreateRemoteThread()를 호출하면 CreatThreat()가 호출되고, 그 안에서 다시 BaseThreadStartThunk()가 호출된다고 합니다. 이 BaseThreadStartThunk()를 후킹해서 lpStartAddress가 위에서 설명한 대로 LoadLibray()나 GetProcAddress()를 가리킨다면 DLL Injection이라고 간주하는 것이죠.

Reference

루트킷 : 윈도우 커널 조작의 미학, Greg Hoglund, James Butler, 2007, 에이콘

Dynamic-link library :: http://en.wikipedia.org/wiki/Dynamic-link_library

Visual C++ DLL :: [http://msdn.microsoft.com/ko-kr/library/1ez7dh12\(VS.80\).aspx](http://msdn.microsoft.com/ko-kr/library/1ez7dh12(VS.80).aspx)

부팅할때 읽는 주요 파일 :: http://fd.igrus.kr/?mid=Study&document_srl=1013

Email-Worm.Win32.Warezov.nf 정보 :: <http://blog.daum.net/virusmyths/6041710>

SetWindowsHookEx() 함수를 이용한 방법 :: <http://blog.naver.com/amanahnr/90032952064>

Shield from DLL-Injection :: <http://nerd.egloos.com/2940078>

SetWindowsHookEx Function :: <http://msdn.microsoft.com/library/ms644990.aspx>

CreateRemoteThread Function :: [http://msdn.microsoft.com/en-us/library/ms682437\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms682437(VS.85).aspx)

OpenProcess Function :: [http://msdn.microsoft.com/en-us/library/ms684320\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms684320(VS.85).aspx)

GetProcAddress Function :: [http://msdn.microsoft.com/en-us/library/ms683212\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms683212(VS.85).aspx)

VirtualAllocEx Function :: [http://msdn.microsoft.com/en-us/library/aa366890\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa366890(VS.85).aspx)

WriteProcessMemory Function :: [http://msdn.microsoft.com/en-us/library/ms681674\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms681674(VS.85).aspx)