introduction to standard dll injection (dll injection part 1)

By Zer0Mem0ry

**DLL injection** is a technique used for running code within the address space of another process by forcing it to load a dynamically linked library. DLL Injection techniques are operating system specific, because they rely on the APIs & System Calls exposed by the operating system in question.

The most common approach in windows systems are based on the API functions CreateRemoteThread(), WriteProcessMemory() and LoadLibrary(). This method is called the CreateRemoteThread & LoadLibrary technique. It works simply by:

1. Allocating the amount of memory needed to copy the path of the dll (for example C:\\TargetDll.dll) to the target process using VirtualAllocEx()
2. Copying the path of the dll (C:\users\public\desktop\test.dll) to the memory allocated on step 1. Using WriteProcessMemory()
3. Calling CreateRemoteThread() with the argument of the address of LoadLibraryA in the remote process, and the path to the dll as an argument.

A simple example of this technique may look something like this:

LPCSTR Dll = "C:\\TargetDll.dll";

// Step 1

// Open Handle to the target process.

HANDLE hProcess = OpenProcess(PROCESS\_ALL\_ACCESS, FALSE, 24476);

// Step 2

// Allocate memory for the dll path in the target process (length of the dll path string).

LPVOID DllPath = VirtualAllocEx(hProcess, NULL, strlen(Dll) + 1, MEM\_COMMIT,

PAGE\_READWRITE);

// Step 3

// Write the path "C:\\TargetDll.dll" to the address of the allocated memory

// in the target process.

WriteProcessMemory(hProcess, DllPath, (void\*)Dll, strlen(Dll) + 1, NULL);

// Step 4

// Create A Remote Thread in the Target process, which calls LoadLibraryA

// With the "C:\\TargetDll.dll" as an argument.

HANDLE hThread = CreateRemoteThread(hProcess, NULL, 0,

(LPTHREAD\_START\_ROUTINE)GetProcAddress(GetModuleHandleA("KERNEL32.DLL"),

"LoadLibraryA"), DllPath, 0, NULL);

// Step 5

// Wait for execution of the thread to finish.

WaitForSingleObject(hThread, INFINITE);

// Step 6

// Free the memory we allocated for "C:\\TargetDll.dll" in the

// target process in step 2.

VirtualFreeEx(hProcess, DllPath, strlen(Dll) + 1, MEM\_RELEASE);

// Step 7

// Close all handles.

CloseHandle(hThread); CloseHandle(hProcess);

* VirtualAllocEx().

“Reserves, commits, or changes the state of a region of memory within the virtual address space of a specified process. The function initializes the memory it allocates to zero.” <https://msdn.microsoft.com/en-us/library/windows/desktop/aa366890(v=vs.85).aspx>

In another words; VirtualAllocEx() is an API function used to allocate memory for another process instead of the current process.

It is needed in this technique because the path of the dll needs to be copied to the target process, and for the path to be copied in the target process, memory must be reserved for it.

* WriteProcessMemory()

“Writes data to an area of memory in a specified process. The entire area to be written to must be accessible or the operation fails.” <https://msdn.microsoft.com/en-us/library/windows/desktop/ms681674(v=vs.85).aspx>

In another words; WriteProcessMemory() is used to write data to another process’ memory, to a specific address.

It is needed in this technique because the characters of the path need to be copied into the address we used VirtualAllocEx() on.

* CreateRemoteThread()

“Creates a thread that runs in the virtual address space of another process.” <https://msdn.microsoft.com/en-us/library/windows/desktop/dd405484(v=vs.85).aspx>

In another words; CreateRemoteThread() is used to create a thread in another process’ address space, this can be used for e.g running a function in another process, like LoadLibraryA().

* GetProcAddress()

**“**Retrieves the address of an exported function or variable from the specified dynamic-link library (DLL).” <https://msdn.microsoft.com/fi-fi/library/windows/desktop/ms683212(v=vs.85).aspx>

In another words; GetProcAddress() gets the address of a function exported by a dll loaded by the local process.

Because windows API Libraries and their exported functions like kernel32.dll’s LoadLibraryA() are always mapped to the same address on each Win32 Process, we can simply just use this function on our own process to get the address of a wanted API function, like in this case the address of LoadLibraryA().

**More on standard Dll Injection**

<http://reverseengineering.stackexchange.com/questions/2252/what-is-dll-injection-and-how-is-it-used-for-reversing>

<https://en.wikipedia.org/wiki/DLL_injection>

<https://www.youtube.com/watch?v=8ziQtpbXRsM>

<https://www.youtube.com/watch?v=g_Xx90wyk0c>