

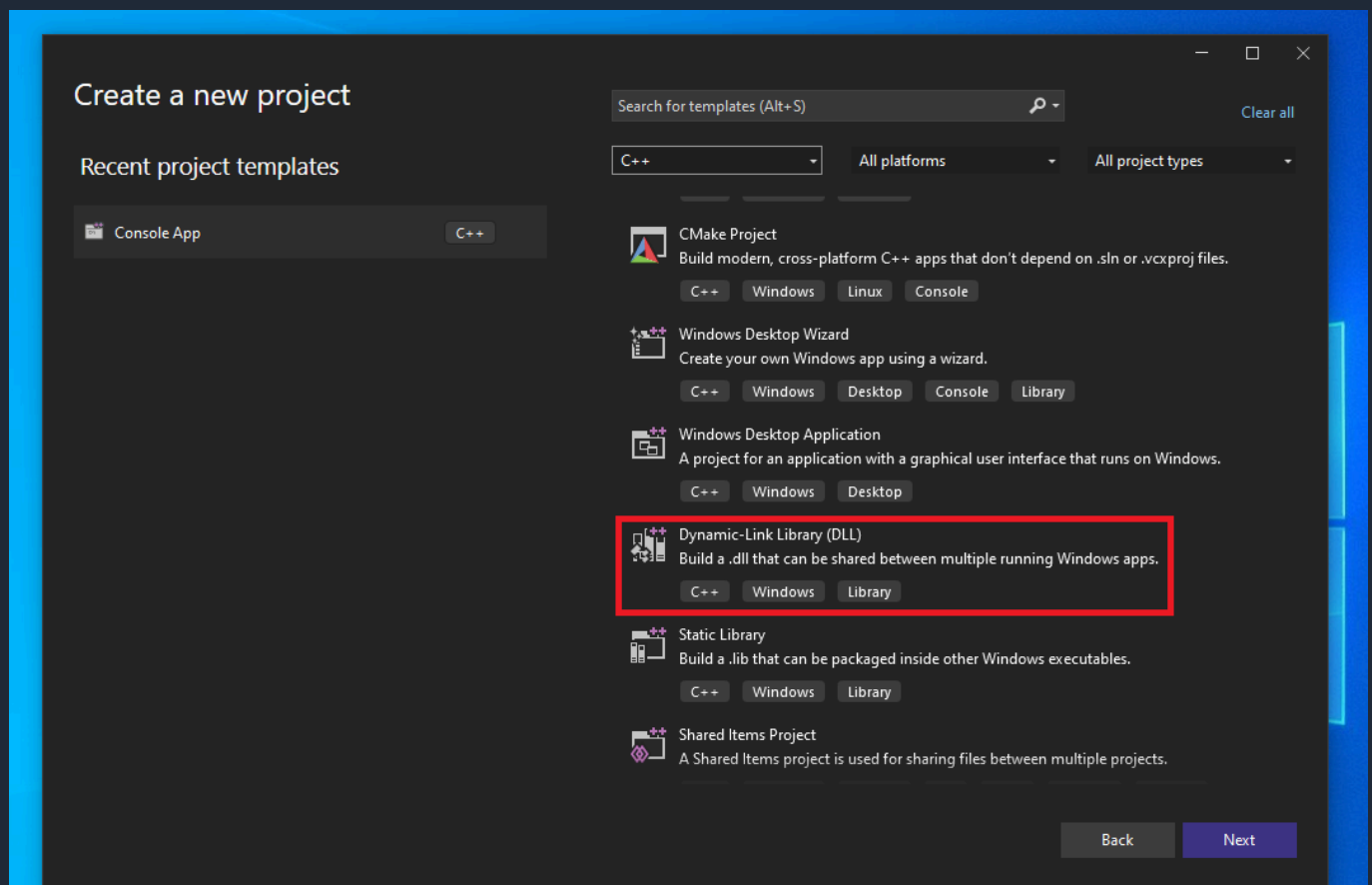
Local Payload Execution - DLL

Introduction

This module explores the usage of Dynamic Link Libraries (DLLs) as payloads and demonstrates how to load a malicious DLL file in the current process.

Creating a DLL

Creating a DLL is simple and can be done using Visual Studio. Create a new project, set the programming language to C++, and finally select Dynamic-Link Library (DLL). This will create a DLL skeleton code that will be modified throughout the remainder of this module. For a refresher as to how DLLs work, feel free to review the introductory DLL module.



DLL Setup

This demo will utilize a message box that appears when the DLL is successfully loaded. Creating a message box can be easily done with the MessageBox WinAPI. The code snippet below will run `MsgBoxPayload` whenever the DLL is loaded into a process. Note

that the precompiled headers were removed from the project's C/C++ settings as shown in the introductory *Dynamic-Link Library* module.

```
#include <Windows.h>
#include <stdio.h>

VOID MsgBoxPayload() {
    MessageBoxA(NULL, "Hacking With MaldevAcademy", "Wow !", MB_OK |
    MB_ICONINFORMATION);
}

BOOL APIENTRY DllMain (HMODULE hModule, DWORD dwReason, LPVOID
lpReserved){

    switch (dwReason){
        case DLL_PROCESS_ATTACH: {
            MsgBoxPayload();
            break;
        };
        case DLL_THREAD_ATTACH:
        case DLL_THREAD_DETACH:
        case DLL_PROCESS_DETACH:
            break;
    }

    return TRUE;
}
```

Local Injection

Recall that the `LoadLibrary` WinAPI is used to load a DLL. The function takes a DLL path on disk and loads it into the address space of the calling process, which in our case will be the current process. Loading the DLL will run its entry point, and thus run the `MsgBoxPayload` function, making the message box appear. Although the concept is simple, it will become useful in later modules to understand more complex techniques.

The code below will take the DLL's name as a command line argument, load it using `LoadLibraryA`, and perform some error checking to ensure the DLL loaded successfully.

```
#include <Windows.h>
#include <stdio.h>
```

```
int main(int argc, char* argv[]) {

    if (argc < 2){
        printf("[!] Missing Argument; Dll Payload To Run \n");
        return -1;
    }

    printf("[i] Injecting \"%s\" To The Local Process Of Pid: %d \n",
argv[1], GetCurrentProcessId());

    printf("[+] Loading Dll... ");
    if (LoadLibraryA(argv[1]) == NULL) {
        printf("[!] LoadLibraryA Failed With Error : %d \n",
GetLastError());
        return -1;
    }
    printf("[+] DONE ! \n");

    printf("[#] Press <Enter> To Quit ... ");
    getchar();

    return 0;
}
```

Output

As expected, the message box successfully appears after injecting the DLL.

```

nt argc, char* argv[])
{
    if (argc < 2){
        printf("[!] Missing Argument\n");
        return -1;
    }

    printf("[i] Injecting \"%s\" \n", argv[1]);

    printf("[+] Loading DLL ...");
    LoadLibraryA(argv[1]) == 0 ?
        printf("[!] LoadLibraryA failed\n") :
        printf("[+] DONE ! \n");

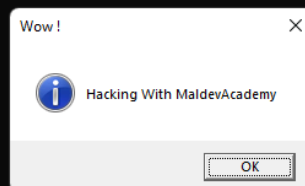
    printf("[#] Press <Enter> To Continue\n");
    getch();
}

```

```

Windows PowerShell
PS C:\Users\User\source\repos\Lesson3\x64\Debug> .\LocalDllInjection.exe .\Dll.dll
[i] Injecting ".\Dll.dll" To The Local Process Of Pid: 8720
[+] Loading DLL ...

```



Process Analysis

To further verify that the DLL is loaded in the process, run Process Hacker, double-click the process which loaded the DLL and head to the "Modules" tab. The DLL's name should appear in the list of modules. Clicking on the DLL's name will retrieve additional information about it such as imports, whether it's signed and section names.

LocalDllInjection.exe (8720) Properties				
General Statistics Performance Threads Token Modules Memory Environment Handles GPU Comment				
Name	Base address	Size	Description	
win32u.dll	0x7ffe36210000	152 kB	Win32u	
vcruntime140d.dll	0x7ffe297e0000	172 kB	Microsoft® C Runtime Library	
uxtheme.dll	0x7ffe33290000	688 kB	Microsoft UxTheme Library	
user32.dll	0x7ffe38880000	1.68 MB	Multi-User Windows USER A...	
ucrtbased.dll	0x7ffd6fcd0000	1.78 MB	Microsoft® C Runtime Library	
ucrtbase.dll	0x7ffe36a20000	1.07 MB	Microsoft® C Runtime Library	
TextShaping.dll	0x7ffe29260000	696 kB		
StaticCache.dat	0x1f1ae190000	17.63 MB		
SortDefault.nls	0x1f1af4e0000	3.23 MB		
rpcrt4.dll	0x7ffe37ad0000	1.13 MB	Remote Procedure Call Runt...	

Previous

Modules

Complete

Next

Name	Base address	Size	Description
l_intl.nls	0x1f1abd30000	12 kB	
locale.nls	0x1f1abee0000	824 kB	
LocalDllInjection.exe	0x7ff629850000	148 kB	
KernelBase.dll	0x7ffe366a0000	3.48 MB	Windows NT BASE API Clie...
kernel32.dll	0x7ffe377f0000	760 kB	Windows NT BASE API Clie...
imm32.dll	0x7ffe377b0000	200 kB	Multi-User Windows IMM32 ...
gdi32full.dll	0x7ffe36240000	1.09 MB	GDI Client DLL
gdi32.dll	0x7ffe37aa0000	164 kB	GDI Client DLL
Dll.dll	0x7ffe1d490000	148 kB	
C_437.NLS	0x1f1abd10000	68 kB	
C_437.NLS	0x1f1abd10000	68 kB	
C_1252.NLS	0x1f1abfb0000	68 kB	
C_1252.NLS	0x1f1abcf0000	68 kB	
combase.dll	0x7ffe36b50000	3.47 MB	Microsoft COM for Windows