Automating DLL Hijack Discovery

Justin Bui · Follow

Published in Posts By SpecterOps Team Members · 11 min read · Jun 30, 2020

(https://github.com/slyd0g/DLLHijackTest). This post will cover DLL hijack discovery in Slack, Microsoft Teams, and Visual Studio Code.

Lastly, I noticed numerous DLL hijacks that were shared between the different applications, investigated the root cause, and discovered that applications using certain Windows API calls are subject to a DLL hijack when not running out of `C:\Windows\System32\`.

I want to give a big shoutout to my coworker, Josiah Massari (@Airzero24), for initially finding some of these DLL hijacks, explaining his methodology

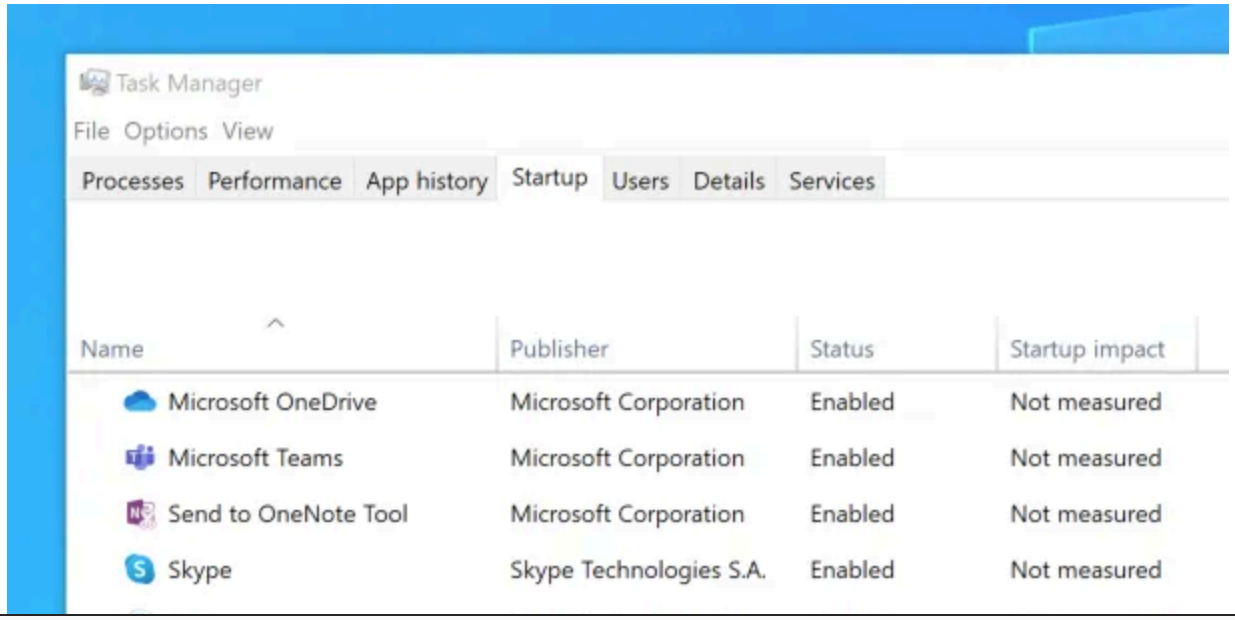
Since DLLs exist as files on disc, you may ask yourself how does an application know where to load DLLs from? Microsoft has documented the DLL search order thoroughly here.

Since Windows XP SP2, safe DLL search mode has been enabled by default (`HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager\SafeDllSearchMode`). With safe DLL search mode enabled, the search order is as follows:

1. The directory from which the application loaded

If an application does not specify where to load a DLL from, Windows will default to the DLL search order shown above. The first location in the DLL search order, the directory from which the application is loaded, is of interest to attackers.

If the application developer wants to load DLLs from `C:\Windows\System32`, but did not explicitly write the application to do so, a malicious DLL planted in the application directory would be loaded before the legitimate DLL in System32. This malicious DLL load is referred to as a DLL hijack and is used by attackers to load malicious code into trusted/signed applications



To begin this process, I started Process Monitor (ProcMon) with the following filters:

- **Process Name** is *slack.exe*
- **Result** contains *NOT FOUND*
- **Path** ends with *.dll*

When this process completes, I would (hopefully) have a list of valid DLL hijacks written to a text file.

The PowerShell script in my DLLHijackTest project does all the magic. It accepts a path to the CSV file generated by ProcMon, a path to your malicious DLL, a path to the process you want to start, and any arguments you want to pass to the process.

```
PS C:Users\John\Desktop> Get-PotentialDLLHijack -CSVPath
.\Logfile.CSV -MaliciousDLLPath .\DLLHijackTest.dll -ProcessPath
"C:\Users\John\AppData\Local\slack\slack.exe"

C:\Users\John\AppData\Local\slack\app-4.6.0\WINSTA.dll
C:\Users\John\AppData\Local\slack\app-4.6.0\LINKINFO.dll
C:\Users\John\AppData\Local\slack\app-4.6.0\ntshrui.dll
C:\Users\John\AppData\Local\slack\app-4.6.0\srvcli.dll
C:\Users\John\AppData\Local\slack\app-4.6.0\cscapi.dll
C:\Users\John\AppData\Local\slack\app-4.6.0\KBDUS.DLL
```

Case Study: Microsoft Teams

Running through the above process again:

```
C:\Users\John\AppData\Local\Microsoft\Teams\current\TextInputFramework.dll
```

Note: I had to make a small modification to the PowerShell script to kill `Teams.exe` since my script attempts to kill the process that it tried to start, which in this case was `Update.exe`.

Case Study: Visual Studio Code

Repeating the process outlined above, I found the follow hijacks for Visual Studio Code:

- `srvcli.dll`
- `cscapi.dll`

I found this interesting and wanted to understand what was causing this behavior.

Methodology: Understanding Shared DLL Hijacks

I observed the stack trace when Slack attempted to load `WINSTA.dll`,

"WINSTA.dll" String in wtsapi32.dll

Right-clicking the location in memory, we are able to find any references to this address

Following the references, we see the `WINSTA.dll` string is being passed to a structure called `ImgDelayDescr`. Looking at documentation on this structure, we can confirm it is related to delay-loaded DLLs.

```
typedef struct ImgDelayDescr {
    DWORD        grAttrs;        // attributes
    RVA          rvaDLLName;     // RVA to dll name
    RVA          rvaHmod;        // RVA of module handle
    RVA          rvaIAT;         // RVA of the IAT
    RVA          rvaINT;         // RVA of the INT
    RVA          rvaBoundIAT;    // RVA of the optional bound IAT
    RVA          rvaUnloadIAT;   // RVA of optional copy of
```

__delayLoadHelper2 and ResolveDelayLoadedAPI in Ghidra

Great! This matches what we saw in our ProcMon stack trace when Slack

This behavior was consistent between `WINSTA.dll`, `LINKINFO.dll`, `ntshrui.dll`, and `srvcli.dll`. The primary difference between each delay-loaded DLL was the "parent" DLL. In all three applications:

- `wtsapi32.dll` delay-loaded `WINSTA.dll`
- `shell32.dll` delay-loaded `LINKINFO.dll`
- `LINKINFO.dll` delay-loaded `ntshrui.dll`
- `ntshrui.dll` delay-loaded `srvcli.dll`

I verified this with PoC programs that call `NetShareEnum` and `NetShareGetInfo`:

```
C:\Users\John\AppData\Local\Microsoft\Teams\current\WINSTA.dll
C:\Users\John\AppData\Local\Microsoft\Teams\current\LINKINFO.dll
C:\Users\John\AppData\Local\Microsoft\Teams\current\ntshrui.dll
C:\Users\John\AppData\Local\Microsoft\Teams\current\srvcli.dll
C:\Users\John\AppData\Local\Microsoft\Teams\current\cscapi.dll
C:\Users\John\AppData\Local\Microsoft\Teams\current\WindowsCodecs.
dll
C:\Users\John\AppData\Local\Microsoft\Teams\current\TextInputFrame
work.dll
```

The following DLL hijacks exist in Visual Studio Code:

I noticed the three applications had overlap with their DLL hijacks and investigated the root cause. I highlighted my methodology for digging into this subject. I learned about delay-loaded DLLs and identified two API calls that introduce DLL hijacks into any program that calls them:

- `NetShareEnum` loads `cscapi.dll`
- `NetShareGetInfo` loads `cscapi.dll`

Thanks for taking the time to read this post, I hope you learned a little about

What is a DLL?

This article describes what a dynamic link library (DLL)...

support.microsoft.com

Dynamic-Link Library Search Order - Win32 apps

Applications can control the location from which a DLL is loaded by specifying a full path or using another mechanism...

docs.microsoft.com

The MSVC linker now supports the delayed loading of DLLs. This relieves you of the need to use the Windows SDK...

docs.microsoft.com

Inside Windows: Win32 Portable Executable File Format in Detail

long time ago, in a galaxy far away, I wrote one of my first articles for Microsoft Systems Journal (now MSDN®...

docs.microsoft.com