The DLL Search Order And Hijacking It

malwareandstuff.com/the-dll-search-order-and-hijacking-it

November 10, 2021

If you ever used Process Monitor to track activity of a process, you might have encountered the following pattern:

Figure 1: Example of dnsapi.dll not being found in the application directory

The image above is a snippet from events captured by Process Monitor during the execution of `x32dbg.exe` on Windows 7. `DNSAPI.DLL` and `IPHLPPAPI.DLL` are persisted in the `System` directory, so you might question yourself:

Why would Windows try to search for either of these DLLs in the application directory first?

Operating Systems are very complex and so is the challenge of implementing an error-fault system to search for dependencies, like dynamic linked libraries. Today, we'll talk about `DLL Search Order` and `DLL Search Order Hijacking`, in particular how it works and how adversaries can abuse it.

DLL Search Order

First, we have to talk about what happens when a PE File is executed on the Windows system.

The majority of native binaries you encounter on Windows are linked dynamically. Linked dynamically means that upon start of the execution, it uses information which are embedded inside the binary to locate DLLs that are essential for this process. In comparison with statically linked binaries, when linked dynamically the executable will use the libraries provided by the OS instead of having them compiled into the executable itself.

Before the dynamically linked executable can use or load these libraries, it will have to know where these dependencies are persisted on disk or if they are already in memory. This is where the `DLL Search Order` makes its appearance. To keep it simple, we will focus only on Windows Desktop Applications.

Pre-Checks and In-Memory Search

Before the Windows OS starts searching for the needed DLL on disk, it will first attempt to find the needed module in memory. If a DLL is already in memory, it will not loaded it again. Now this part is a little bit complicated and out of context for this blog article, we would have to define what “loaded” even means. If you are more interested in the first check, I advise you to look up the official Microsoft documentation[1].

If the memory check fails, Windows can fall back to using a list of known DLLs. if the needed library is part of that list, it will use the copy of the known DLL. The list of known DLLs are persisted in the Windows Registry.

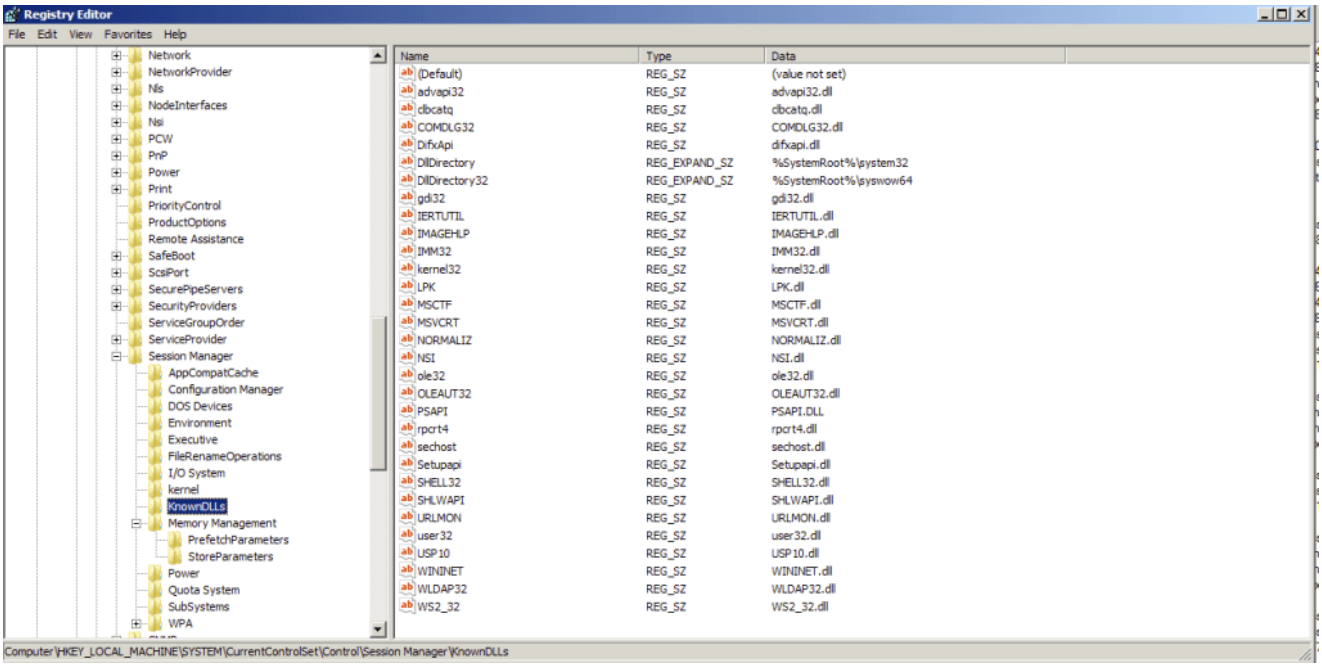


Figure 2: List of KnownDlls on Windows 7

On-Disk Search

If the first two checks fail, the OS will have to search for the DLL on disk. Depending on the OS Settings, Windows will use a different search order. Per default, Windows enables the `DLL Search Mode` feature to harden the system and prevent DLL Search Order Hijacking attacks, a technique we will explain in the upcoming section.

The key to the feature is as follows:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session
Manager\SafeDllSearchMode
```

Let’s take a look at the differences of the search order depending whether `SafeDllSearchMode` is enabled or not.

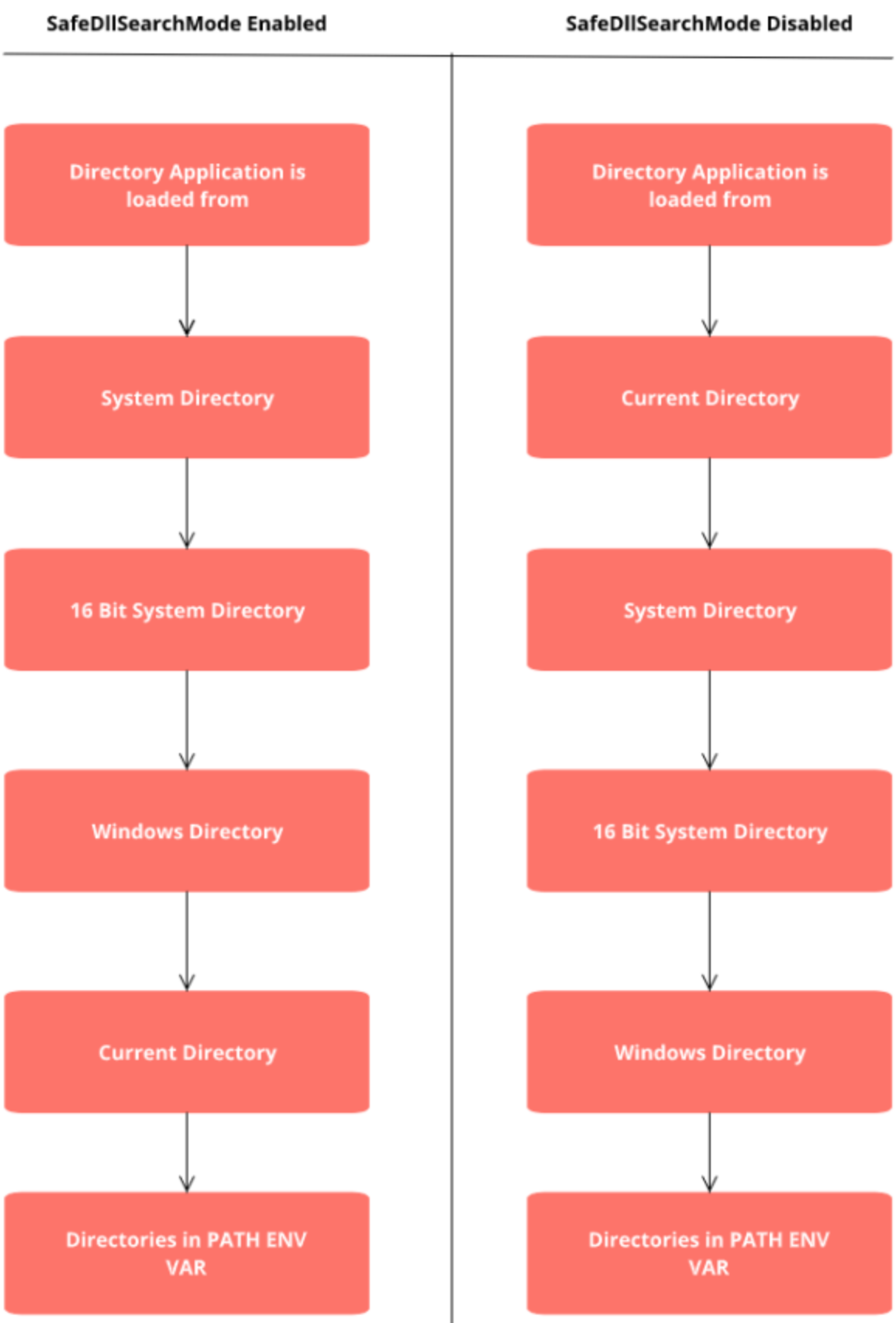

SafeDllSearchMode Enabled
Directory Application is loaded from
System Directory
16 Bit System Directory
Windows Directory
Current Directory
Directories in PATH ENV VAR
SafeDllSearchMode Disabled
Directory Application is loaded from
Current Directory
System Directory
16 Bit System Directory
Windows Directory
Directories in PATH ENV VAR

Figure 3: DLL Search Order flow

We clearly see that the current directory is prioritised if `SafeDllSearchMode` is disabled and this can be abused by adversaries. The art of abusing this search order flow is called `DLL Search Order Hijacking`.

DLL Search Order Hijacking

Adversaries can abuse the search order flow displayed above to load their own malicious DLLs instead of the legitimate ones into memory. There are many ways this technique can be used. However, it is more effective in achieving persistence on the target system then initial execution.

Let's take a step back and revisit our example from above:

- `x32dbg.exe` tries to load `DNSAPI.DLL`
- `DNSAPI.DLL` is not in the list of known DLLs and is also not loaded into memory.
- Since `SafeDllSearchMode` is enabled, it will fall back to the system directory if not found in the application directory

What would happen, if we craft and place a malicious DLL, named `DNSAPI.DLL` into the application directory?

We would be able to hijack the search order flow and force a legitimate application to load our malicious code into memory.

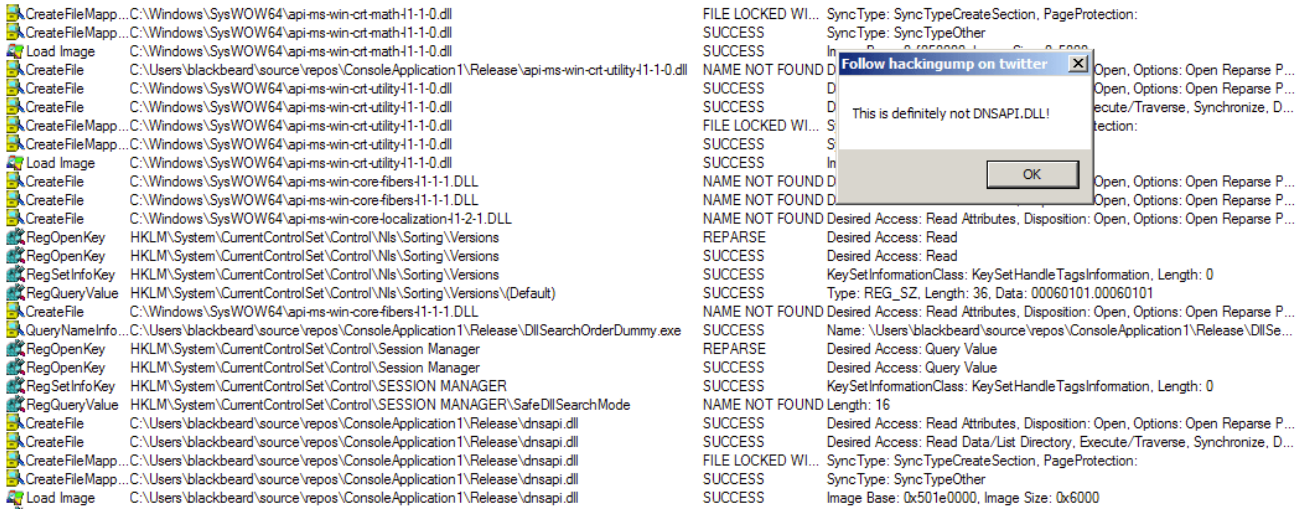
Practical Use Case

Let's take a look at a simple practical example. Our application calls `LoadLibraryA` and tries to load `dnsapi.dll` like in our example from above. Next we craft a small DLL file, which does nothing else but create a message box in the `DLLMain` function. Once the DLL is loaded into memory, the main function will be triggered.

In the first run, we do not place the crafted DLL in the application directory. As expected, Windows will load `dnsapi.dll` from the `system` directory:

DllSearchOrder...	4504	CreateFile	C:\Users\blackbeard\source\repos\ConsoleApplication1\Release\dnsapi.dll	NAME NOT FOUND
DllSearchOrder...	4504	CreateFile	C:\Windows\SysWOW64\dnsapi.dll	SUCCESS
DllSearchOrder...	4504	CreateFile	C:\Windows\SysWOW64\dnsapi.dll	SUCCESS
DllSearchOrder...	4504	CreateFileMapp...	C:\Windows\SysWOW64\dnsapi.dll	FILE LOCKED WI...
DllSearchOrder...	4504	CreateFileMapp...	C:\Windows\SysWOW64\dnsapi.dll	SUCCESS
DllSearchOrder...	4504	Load Image	C:\Windows\SysWOW64\dnsapi.dll	SUCCESS

Next, we will now name our crafted DLL `dnsapi.dll` and place it in the application directory:



Whoops! I think we can all think of a couple use cases of how APT groups and malware can abuse this technique to achieve persistence on the victim's system.

Real world examples and APTs

For the sake of keeping it simple and explaining the core principles behind this persistence technique, we've build a very simple use case here. Of course, the real world looks a little bit different and usually attackers have to take into account:

- Endpoint Security solutions with behaviour based detections, preventing such attacks with signatures
- Programmatic dependencies, which won't allow you to just replace a DLL in an application directory and hope that it will work just fine
- and many more

However, if you never heard about this technique, I hope I was able to create some awareness for it!