If relocated DLLs cannot share pages, then doesn’t ASLR cause all pages to be non-shared?

devblogs.microsoft.com/oldnewthing/20160413-00

April 13, 2016

Raymond Chen

Commenter Medinoc wonders whether it’s still the case that relocated DLLs can’t be shared in memory. If so, then doesn’t ASLR cause all pages to become non-sharable?

There are multiple things in play here. We’ll take them up in historical order, but I’ll start with Windows NT 3.1 instead of Windows 95 because I already discussed Windows 95 a while back.

Windows NT 3.1 tried to load DLLs at their preferred address. If that happened, then the pages were demand-paged from the executable on disk, and if multiple processes loaded the DLL at the preferred address, then the memory was physically shared.[1] On the other hand, if the DLL could not load at its preferred address, then fixups were applied to the entire DLL to relocate it, and the relocated DLL was dumped into the pagefile, and not only did further demand paging come from there, but that relocated copy was not shared between processes.

In other words, if two processes both loaded a DLL, and the DLL got relocated in both of the processes, and it got relocated to the same address in each process, there would nonetheless be two copies of the DLL in the page file, not one copy that was shared between the two processes.

The reason for not sharing the pages in this case is that the likelihood of all the stars aligning is relatively low. Under the Windows NT 3.1 model, each process did its own relocating, and each process chose where the DLL would get relocated to. The likelihood that two processes would both load the same DLL, and have the same virtual memory layout so that they would choose the same relocation destination were relatively low, so the benefit of getting the processes to coordinate among themselves was not worth the effort.

And then ASLR showed up and changed the cost/benefit calculations. With ASLR, DLLs are being relocated constantly, and if the old rules were followed, there would be as many copies of a DLL in the page file as there were processes that used the DLL. This was clearly not a good thing.

The solution is that when a DLL is loaded, ASLR chooses a random destination address, but it then remembers that address for future use, and if another process loads the DLL, the kernel will try to use the same destination address for the DLL in that other process. This means that if two processes load a DLL, that DLL will probably get the same destination address in both processes, which establishes one of the prerequisites for sharing.

ASLR goes further. The kernel doesn't even bother fixing up the entire DLL and dumping it into the page file. Instead, it fixes up the DLL on the fly as it is loaded (stealing a trick from Windows 95), and shares the fixed-up pages.

Another way of looking at this is that the kernel is pretending that the preferred address of the file on disk happens to have matched the ASLR-chosen address all along. It carries out this ruse by patching the bytes of the file as they are read off the disk.

[1] For simplicity of exposition, let's assume that nobody changes page protection. If you are smart enough to ask, "What if somebody changes the page protection?" then you are smart enough to know the answer.

Raymond Chen

Follow