Adventures in application compatibility: Following a detour off a cliff

devblogs.microsoft.com/oldnewthing/20180604-00

June 4, 2018

Raymond Chen

The application compatibility team reported that a program was crashing in its installer. Debugging the installer revealed that it actually triggered *three* exceptions.

The first exception was due to passing an invalid pointer to the `lstrlen` function. However, this didn't crash the installer because the `lstrlen` function contains an application compatibility mitigation: If you pass a pointer to invalid memory, it handles the access violation exception by reporting that the string length is zero.

It did this twice, so that is the second exception as well.

The third exception is the one that crashed the installer: A system function wanted to report a runtime issue. It did this by checking if a debugger was present, and if so, it printed a diagnostic message to the debugger so that the developer could see that there was a problem and get a chance to investigate it.

The problem is that the call to `IsDebuggerPresent` was crashing.

And it was crashing because the installer used a DLL which detours the `IsDebugger-Present` function when it is loaded, presumably to make it harder to reverse-engineer (though since the source code is freely available, it's not sure why they are so concerned about it).

But notice that the DLL does not un-detour the function when it is unloaded.[1]

Therefore, if you load the DLL and then unload it, you leave the `IsDebuggerPresent` function detoured to a function that no longer exists. And the next person to call the `Is-DebuggerPresent` function will crash.

(Also curious is that the DLL is licensed under the GPL, yet the product that uses it makes no mention of the GPL in its license agreement.)

The fix was to remove the debugging message for the particular case that this program's installer was tripping over.

[1] Not that un-detouring is a perfect solution either, because it runs into the What if two programs did this? problem: Suppose another DLLs also detoured the `IsDebuggerPresent` function. When this DLL unloads, it restores the original function, which also removes the other DLL's detour. And then when the other DLL unloads, it restores what it thought was the original function, but which was actually the detour installed by this DLL. As a result, this DLL's detour gets *reinstalled*.

Raymond Chen

Follow