Why can’t the debugger call GetFinalPathNameByHandle via .call?

devblogs.microsoft.com/oldnewthing/20160407-00

April 7, 2016

Raymond Chen

There were a number of follow-up questions to my demonstration of calling the `GetFinalPathNameByHandleW` function from the debugger.

Mason Wheeler asks, “What debugger is this? Why doesn’t it have the ability to call functions as a standard feature?”

The debugger here is the debugger that comes with the Debugging Tools for Windows. It is the core debugging engine that powers `kd`, `cdb`, `ntsd`, and `windbg`.

This debugging engine operates under very narrow constraints because it is designed to be deployed to a customer machine with minimum overhead or invasiveness. (Indeed, this debugger used to come preinstalled with Windows, but that stopped a while back, presumably to minimize attack surface, and also to give the debugging tools team control over their own release cycle.)

This means that the debugger cannot assume that there is a copy of the Windows SDK installed on the machine. Without the header files, the debugger doesn’t know how to call any particular function. What’s the calling convention? What is the type of each parameter? (Four-byte integers, 8-byte integers, 4-byte floating point values, 8-byte floating point values, and 128-bit SIMD values are all passed differently.) What is the function’s return value? (If the return value is a structure or class, then an extra hidden parameter must be passed. And the debugger needs to know the return value in order to print the result.)

Since the debugger does not have access to a compiler or to header files, it has to make do with the symbolic information embedded in the symbol files. And that’s where the “If this were a function you had written” comes from. If you had written the function, then the information about the calling convention and parameters would be found in the symbol file for the module containing the function you’re trying to call. But if you didn’t write the function, then you don’t necessarily have full symbolic information for the function. In particular, the Windows symbol server does not provide full symbolic information for system functions. It contains only enough information to build stack traces (which basically means it has just the function name and some FPO information). That’s not enough information to simulate a call to the function.

It’s great to have the Visual Studio debugger or some other fancy debugger available. But if you are debugging a customer’s machine via a remote connection, you have to make do with very little. Customers are understandably unhappy when you tell them, “Okay, we can try to debug your problem, but first you need to install Visual Studio onto your domain controller.” Plus there’s the heisenbug effect: The act of installing Visual Studio entails significant changes to the system, one of which may alter the system enough to make the bug no longer occur.

(That’s also why writing a scratch program that does an `OpenProcess`, `DuplicateHandle`, `GetFinalPathByHandle`, and then `MessageBox` was not considered for this problem. If you are debugging remotely, you have to be able to do everything you need from within the debugger. There is nobody sitting at the console to whom you can email your test program, ask to run it, and then have them read the results back to you.)

Raymond Chen

Follow