Debugger breakpoints are usually implemented by patching the in-memory copy of the code

devblogs.microsoft.com/oldnewthing/20241111-00

November 11, 2024

In practice, when you set a code breakpoint in the debugger, the debugger replaces the instruction at that location with a breakpoint instruction.[1] When execution reaches that instruction, it will encounter the breakpoint instruction and break into the debugger.

When the program has been stopped in the debugger, what happens next can vary from debugger to debugger. Some debuggers remove all their breakpoints when the program stops, and then restore the breakpoints when the program resumes. Other debuggers leave the breakpoints in place even when the program is stopped.

In both cases, if you inspect the memory in the debugger, you will see the original unpatched code. In the first case, it's because the code really is unpatched; the breakpoint instructions are removed. In the second case, it's because the debugger is *lying to you* and showing you the original bytes even though they aren't what are in memory right now.

Most of the time, this deception is insignificant. Everything looks like no patching has occurred.

But sometimes you will notice.

One case where you will notice is if the program tries to read from its own code bytes. In that case, it will see the patched instructions.

Another case is where you mistakenly set a code breakpoint on data. The debugger replace the "instruction" at the data you specified with a breakpoint instruction, and then resumes execution. Your code then tries to read from that data, and instead of reading the original data, it reads the breakpoint instruction. What happens next depends on what the program tries to do with that data, but it's usually not good.

So take care when you set your code breakpoints. Make sure they really are on code.

[1] The encoding of the breakpoint instruction on x86 is the single byte `0xCC`.