# Why aren't compatibility workarounds disabled when a debugger is attached?

devblogs.microsoft.com/oldnewthing/20100111-00

January 11, 2010

Raymond Chen

Ken Hagan wonders why compatibility workarounds aren't simply disabled when a debugger is attached.

As I noted earlier, many compatibility workarounds are actually quicker than the code that detects whether the workaround would be needed.

```
BOOL IsZoomed(HWND hwnd)
{
  return GetWindowLong(hwnd, GWL_STYLE) & WS_MAXIMIZED;
}
```

Now suppose you find a compatibility problem with some applications that expect the `IsZoomed` function to return exactly `TRUE` or `FALSE`. You then change the function to something like this:

```
BOOL IsZoomed(HWND hwnd)
{
  return (GetWindowLong(hwnd, GWL_STYLE) & WS_MAXIMIZED) != 0;
}
```

Now, we add code to enable the compatibility workaround only if the application is on the list of known applications which need this workaround:

```
BOOL IsZoomed(HWND hwnd)
{
  if (GetWindowLong(hwnd, GWL_STYLE) & WS_MAXIMIZED) {
      if (IsApplicationCompatibilityWorkaroundRequired(ISZOOMED_TRUEFALSE)) {
          return TRUE;
      } else {
          return WS_MAXIMIZED;
      }
  } else {
    return FALSE;
  }
}
```

What was a simple flag test now includes a check to see whether an application compatibility workaround is required. These checks are not cheap, because the compatibility infrastructure needs to look up the currently-running application in the compatibility database, check that the version of the application that is running is the one the compatibility workaround is needed for (which could involve reading the file version resource or looking for other identifying clues), and then returning either the compatible answer ( `TRUE` ) or the answer that resulted from the original simple one-line function.

So not only is the function slower (having to do a compatibility check), it also looks really stupid.

Oh wait, now we also have to stick in a debugger check:

```
BOOL IsZoomed(HWND hwnd)
{
  if (GetWindowLong(hwnd, GWL_STYLE) & WS_MAXIMIZED) {
      if (!IsDebuggerPresent() &&
         IsApplicationCompatibilityWorkaroundRequired(ISZOOMED_TRUEFALSE)) {
          return TRUE;
      } else {
          return WS_MAXIMIZED;
      }
  } else {
    return FALSE;
  }
}
```

And then people complain that Windows is slow and bloated: A simple one-line function ballooned into ten lines.

Another reason why these compatibility workarounds are left intact when a debugger is running is that changing program behavior based on whether a debugger is attached would prevent application vendors from debugging one problem because all sorts of new problems suddenly got injected.

Suppose you support Program X, and you get a report of a security vulnerability in your program. You run the program under the debugger, and when you run the alleged exploit code, you find that the program doesn't behave the same as it does when the debugger is not attached. Some compatibility workaround that was active when your program is run normally is being suppressed, and the change in behavior changes your program enough that the alleged security exploit doesn't behave quite the same.

When run outside the debugger, the program crashes, but when run under the debugger, the program displays a strange error message but manages to keep from crashing. Congratulations, you introduced a Heisenbug.

And then you say, “There’s something wrong with the debugger. It must be a bug in Windows.”

**Pre-emptive Yuhong Bao comment**: The heap manager switches to an alternate algorithm if it detects a debugger, and the `CloseHandle` function raises an exception if running under the debugger.

Raymond Chen

**Follow**