If you have full trust, then you can do anything, so don't be surprised that you can do bad things, too

devblogs.microsoft.com/oldnewthing/20090121-00

January 21, 2009

Raymond Chen

This is another example of the dubious security vulnerability known as wrapping a simple idea inside layers of obfuscation and then thinking that somehow the obfuscation is the source of the problem.

First of all, consider this: Suppose a program calls one of its own functions but gets the calling convention wrong and ends up corrupting its stack. Is that a security vulnerability in the operating system? No, it's a bug in the program. Now, maybe a bad guy can try to exploit this bug in the program, but if such an exploit could be found, it's naturally a vulnerability in the program, not in the operating system.

Okay, now we add the layers of obfuscation.

First, instead of calling the function incorrectly, we filter it through another function that takes a callback function. For example, let's use `EnumFonts`, which is a function that takes a callback function which is called once for each font. If you pass a callback function that has the wrong calling convention, that's your problem. The `EnumFonts` function doesn't know what calling convention your callback function uses; it's your responsibility to pass a correctly-formed function.

Okay, let's add another layer of obfuscation. Instead of passing an incorrect callback function from unmanaged code, let's use `DllImport` to call the `EnumFonts` function from a VB.NET program. We're still passing an invalid callback function, but now it's being done from a VB.NET program instead of an unmanaged program.

At this point, you announce that you have found a security vulnerability in VB. And since a lot of scripts are written in VB, you add that this vulnerability affects Word, Excel, Web pages, anything that supports script.

But if you take away all the layers of obfuscation, you see that all that is really going on is that the program called a function with the wrong calling convention. Hardly an earth-shattering vulnerability.

Now let's look to see if the layers of obfuscation made the original problem broader in scope. First of all, the caller and the crash take place in the same process, so we haven't crossed a security boundary from, say, one process to another, or from user mode to kernel mode. Furthermore, in order to call a DllImported function, the program needs to have full trust, in which case it already can do bad things to your computer without having to enlist the assistance of the `EnumFont` function. It could just start passing totally bogus parameters to `Marshal.Copy` and corrupt memory all it wants. The only interesting thing would be if a trusted program could somehow be tricked into passing a bogus function pointer to `EnumFonts` or otherwise be tricked into calling a bogus function, but even in that case, it would be a bug in that trusted program, not in VB itself.

Raymond Chen

Follow