If you configure a program to run in Windows 2000 compatibility mode, then it is also vulnerable to Windows 2000 security issues

devblogs.microsoft.com/oldnewthing/20170911-00

September 11, 2017

Raymond Chen

We received a security vulnerability report that said, basically, that if you apply Windows 2000 compatibility mode to an application, then it becomes vulnerable to Windows 2000 security issues.

Well, yeah. Because that's what you asked for.

If you set a program to run in Windows 2000 compatibility mode, then one of the things that happens is that the DLL loading follows the Windows 2000 rules, and Windows 2000 predates the **SafeDllSearchMode** setting, so they always follow the "**SafeDllSearch-Mode** is disabled" rules.

This is intentional, because one of the reasons the program was put into Windows 2000 compatibility mode is that it relies on the Windows 2000 algorithm for DLL loading. In other words, the program relies on bug-for-bug compatibility,[1] and the Windows 2000 compatibility does its best to oblige.[2]

Is this a security vulnerability?

Well, it's a security vulnerability in the program, that it stops working when the more secure DLL loading algorithm is used. On the other hand, good luck getting the vendor to do anything to address this issue. The fact that the program requires Windows 2000 compatibility mode is a strong indication that the vendor is not going to do anything about the matter, given that it's had over fifteen years to do something about it and hasn't.

But what about if a user manually applies the Windows 2000 compatibility mode to a program that doesn't need it? Is it a security vulnerability that Windows allows the user to put a current-day program into a compatibility mode that reintroduces old security vulnerabilities? Or is this a case of "If you configure your system to be insecure, then don't be surprised that you have a security vulnerability"?

Let’s look at the usual questions for evaluating whether something is a security vulnerability: Who is the attacker? Who is the victim? What has the attacker gained?

The attacker is somebody who can set a program into an insecure compatibility mode. The victim is somebody who runs the program thinking they are getting a normal program, but are instead getting an insecure program. The attacker can now compromise the program by using the old security vulnerability.

Okay, but let’s take closer look at the relationship between the the attacker and the victim. If a local user applies an insecure compatibility mode to a program, it affects only that user. The user hasn’t gained anything. They could have just written a program that does whatever they like and run it. No need to pile on the style points by employing DLL injection. In this case, the attacker is attacking himself. This is not particularly interesting.

In order to change what other users experience when they run the program, you need to have administrator privileges in order to modify the system compatibility database or edit system shortcuts. In that case, you’re already on the other side of the airtight hatchway.

Compatibility shims should be applied only to address compatibility issues and not as something you run around applying to anything you see, because some compatibility shims weaken security for compatibility reasons.

[1] And that isn’t even the weirdest “throwback Thursday” compatibility shim. My favorite is EmulateHeap, which replaces the standard heap with an exact copy of the Windows 95 heap manager.

[2] Note that the compatibility shim infrastructure performs only in-process shimming. It can alter the way the process internally behaves (or how in-process components like the DLL loader behave), but it doesn’t alter the security boundaries between the program and the rest of the system. So even though it weakens the security to Windows 2000 levels, it does so only to the extent that the application could have weakened security on its own (say by implementing an insecure algorithm).

Raymond Chen

Follow