It rather involved being on the other side of this airtight hatchway: Disabling Safe DLL searching

devblogs.microsoft.com/oldnewthing/20130705-00

July 5, 2013

Raymond Chen

The Microsoft Vulnerability Research team discovered a potential current directory attack in a third party program. The vendor, however, turned around and forwarded the report to the Microsoft Security Response Center:

> Our investigation suggests that this issue is due to a bug in Microsoft system DLLs rather than our program. When a process is launched, for example, when the user double-clicks the icon in Explorer, a new process object is created, and the DLLs are loaded by a component known as the Loader. The Loader locates the DLLs, maps them into memory, and then calls the DllMain function for each of the modules. It appears that some Microsoft DLLs obtain DLLs from the current directory and are therefore susceptible to a current directory attack. We created a simple Win32 application which demonstrates the issue:
>
> ```
> #include <windows.h>
> int __cdecl main(int argc, char **argv)
> {
>  return MessageBox(NULL, "Test", "Test", MB_OK);
> }
> ```
>
> If you place a fake copy of `DWMAPI.DLL` in the same directory as the application, then the Loader will use that fake copy instead of the system one.
>
> This technique can be used to attack many popular programs. For example, placing a fake copy of `DWMAPI.DLL` in the `C:\Program Files\Internet Explorer` directory allows it to be injected into Internet Explorer. Placing the file in the `C:\Program Files\Adobe\Reader 9.0\Reader` directory allows it to be injected into Adobe Reader.

(I like how the report begins with some exposition.)

The vendor appears to have confused two directories, the current directory and the application directory. They start out talking about a current directory attack, but when the money sentence arrives, they talk about placing the rogue DLL "in the same directory as the

application," which makes this not a current directory attack but an application directory attack.

We saw some time ago that the directory is the application bundle, and the application bundle can override DLLs in the system directory. Again, this is just another illustration of the importance of securing your application directory.

The specific attacks listed at the end of the report require writing into `C:\Program Files`, but in order to drop your rogue `DWMAPI.DLL` file into that directory, you need to have administrative privileges in the first place.

In other words, in order to attack the system, you first need to get on the other side of the airtight hatchway.

There was one final attempt to salvage this bogus vulnerability report:

> We can also reproduce the problem without requiring write access to the `Program Files` directory by disabling Safe DLL searching.

Nice try. In order to disable Safe DLL searching, you need to have administrator privileges, so you're already on the other side of the airtight hatchway. And if you elevate to administrator and disable safe DLL searching, then is it any surprise that you have unsafe DLL searching? This is just another case of *If you set up an insecure system, don't be surprised that there's a security vulnerability*.

Raymond Chen

Follow