Don't trust the return address, no really

devblogs.microsoft.com/oldnewthing/20060817-17

August 17, 2006

Raymond Chen

In the discussion of how to prevent non-“trusted” DLLs from using private OS resources, more than one person suggested having the `LoadLibrary` or `FindResource` function behave differently depending on who the caller is. But we already saw that you can’t trust the return address and that you definitely shouldn’t use the return address to make a security decision (which is what these people are proposing). All attackers have to do is find some other “trusted” code to do their dirty work. For example, the `LoadString` function internally calls `FindResource` to locate the appropriate string bundle. Therefore, if attackers want to get a string resource from a “trusted” DLL, they could use `LoadString` to do it, since `LoadString` will call `FindResource`, and `FindResource` will say, “Oh, my caller is `USER32.DLL`, which is trusted.” Bingo, they just stole a string resource. “Well, I could add that same check to `LoadString`.” I was just giving `LoadString` as an example of a “middle man” function that you can exploit. Sure, extra code could be added to `LoadString` to check its return address and reject attempts to load strings from “protected” libraries if the caller is “untrusted”, but attackers would just look for some other middle man they could exploit. And even if you were diligent enough to protect all such potential middle-men, you still are vulnerable to the sort of stack-manipulation games that don’t require anything from a “trusted” DLL aside from a return instruction. (And there are plenty of those.) No, you cannot impose security boundaries within a process. Once you let code run unchecked in your process, you have to treat the entire process as compromised. Even the parts that you thought were trustworthy.

Now, you might say, “Oh, we’re not really making a security decision here. We just want to make circumventing the system so much hard work that somebody who goes to that much effort knows that they’re doing something unsupported.” But as commenter Duncan Bayne points out, that applies only to the first person to do it. They then make a library out of their technique, or publish it in a magazine article, and now anybody can use it without a struggle, and consequently without it crossing their mind that “Gosh, maybe this isn’t such a great idea to use in production software.”

Raymond Chen

Follow