It rather involved being on the other side of this airtight hatchway

devblogs.microsoft.com/oldnewthing/20060508-22

May 8, 2006

Raymond Chen

Not every code injection bug is a security hole.

Yes, a code injection bug is a serious one indeed. But it doesn't become a security hole until it actually allows someone to do something they normally wouldn't be able to.

For example, suppose there's a bug where if you type a really long file name into a particular edit control and click "Save", the program overflows a buffer. With enough work, you might be able to turn this into a code injection bug, by entering a carefully-crafted file name. But that's not a security hole yet. All you've found so far is a serious bug. (Yes, it's odd that I'm underplaying a serious bug, but only because I'm comparing it to a security hole.)

Look at what you were able to do: You were able to get a program to execute code of your choosing. Big deal. You can already do that without having to go through all this effort. If you wanted to execute code of your own choosing, then you can just put it in a program and run it!

The hard way

1. Write the code that you want to inject, compile it to native machine code.
2. Analyze the failure, develop a special string whose binary representation results in the overwriting of a return address, choosing the value so that it points back into the stack.
3. Write an encoder that takes the code you wrote in step 1 and converts it into a string with no embedded zeros. (Because an embedded zero will be treated as a string terminator.)
4. Write a decoder that itself contains no embedded-zeros.
5. Append the encoded result from step 3 to the decoder you wrote in step 4 and combine it with the binary representation you developed in step 2.
6. Type the resulting string into the program.
7. Watch your code run.

The easy way

1. Write the code that you want to inject. (You can use any language, doesn't have to compile to native code.)
2. Run it.

It's like saying that somebody's home windows are insecure because a burglar could get into the house by merely unlocking and opening the windows from the inside. (But if the burglar has to get inside in order to unlock the windows…)

Code injection doesn't become a security hole until you have elevation of privilege. In other words, if attackers gains the ability to do something they normally wouldn't. If the attack vector requires setting a registry key, then the attacker must already have obtained the ability to run enough code to set a registry key, in which case they can just forget about "unlocking the window from the inside" and just replace the code that sets the registry with the full-on exploit. The alleged attack vector is a red herring. The burglar is already inside the house.

Or suppose you found a technique to cause an application to log sensitive information, triggered by a setting that only administrators can enable. Therefore, in order to "exploit" this hole, you need to gain administrator privileges, in which case why stop at logging? Since you have administrator privileges, you can just replace the application with a hacked version that does whatever you want.

Of course, code injection can indeed be a security hole if it permits elevation of privilege. For example, if you can inject code into a program running at a different security level, then you have the opportunity to elevate. This is why extreme care must be taken when writing unix root-setuid programs and Windows services: These programs run with elevated privileges and therefore any code injection bug becomes a fatal security hole.

A common starting point from which to evaluate elevation of privilege is the Internet hacker. If some hacker on the Internet can inject code onto your computer, then they have successfully elevated their privileges, because that hacker didn't have the ability to execute arbitrary code on your machine prior to the exploit. Next time, we'll look at some perhaps-unexpected places your program can become vulnerable to an Internet attack, even if you think your program isn't network-facing.

Raymond Chen

Follow