A single-instance program is its own denial of service

devblogs.microsoft.com/oldnewthing/20060620-13

June 20, 2006

Raymond Chen

There are many ways a program can establish itself as a single-instance program; I won't go into them here. But once people head down the path of a single-instance program, they bump into another issue in their security review: Denial of service.

> We are using a named mutex with a fixed name in order to detect whether another copy of the program is running. But that also means an attacker can create the mutex first, thereby preventing our program from running at all! How can I prevent this type of denial of service attack?

If the attacker is running in the same security context as your program is (or would be) running in, then there is nothing you can do. Whatever "secret handshake" you come up with to determine whether another copy of your program is running, the attacker can mimic it. Since it is running in the correct security context, it can do anything that the "real" program can do. But look at the big picture: A single-instance program is its own denial of service! After all, the first instance of the program is preventing the second instance from running. Your program requirements are themselves a security vulnerability. Consequently, you cannot protect against yourself perfectly against a denial of service since a denial of service is what you want in the first place.

What you can do is to understand and narrow the scope of your vulnerability. Clearly you can't protect yourself from an attacker running at the same security privilege, but you can still protect yourself against unprivileged attackers running at other security privileges. This means using securable objects as part of your handshake. Non-administrative users should not be able to prevent other users from running the program, for example. The worst thing that non-administrative users should be allowed to do is to make their own lives miserable.

Raymond Chen

Follow