Defense in depth means that you protect against exploits that don't exist yet

devblogs.microsoft.com/oldnewthing/20090319-00

March 19, 2009

Raymond Chen

Defense in depth is about protecting against threats that are already being protected against, just in case the existing protection fails. This is why there is not merely a lock on your safety deposit box, but also a lock on the door to the safety deposit box room, and then a lock on the doors of the bank itself. This is why you wear your seat belt even though the car is equipped with air bags. This is why factories have multiple safety systems. It's why, when you put away a gun, you set the safety *and* remove the ammunition *and* lock the gun case.

An insistent anonymous commenter refused to believe in this principle and couldn't distinguish between the absence of a known security vulnerability and the potential for one, believing that security is a boolean value, that you're either secure or you're insecure, and that if two systems are identical except that the second system has an additional safety check, this is proof that the first system must have been insecure.

As I described in the comments to the article, there is the potential for bad things to happen if a COM data object is allowed into the process. Even though the CSRSS process never calls any of the potentially dangerous functions in a dangerous way, the potential for some other flaw to result in dangerous behavior creates enough risk that the trade-off tipped toward removing the *potential* for problems, even though the potential is currently (and hopefully will always remain) unrealized.

Remember that one of the guidelines of security is that the more valuable the target, the more effort you put into securing it. In this case, CSRSS runs with System security privileges, which is even higher than Administrator. You want to erect a lot of barriers for this puppy.

It's like a hospital that has the rule "No cell phones allowed in hospital rooms because they may interfere with the equipment." The staff instruct you to leave your phone outside, but you insist that your phone does not pose a problem because it's turned off, and besides, it doesn't use the same radio frequency as the monitoring equipment. Tough. Defense in depth. Even if it's turned off, even if uses a different radio frequency, they won't let it into the room.

The same thing is true with data objects. CSRSS is careful to extract only the information it needs, but that's like walking into a hospital room with a cell phone whose antenna has been switched off. Sure, the antenna is off, but somebody might bump into you and accidentally turn it on, or there may be some software flaw in the phone that causes it to turn on spontaneously. Sure, you might argue that those failures aren't your fault, so you shouldn't be blamed for them, but try telling that to the person whose monitoring equipment failed to notify the hospital staff of an irregular heartbeat.

People who study security vulnerabilities have quite a wide array of tricks available to them once they find even the tiniest crack. Even something as simple as a null pointer fault (in itself just a denial of service and not a source of pwnage) can be combined with other techniques and become a full-fledged exploit.

For example, even though your cell phone antenna is off, its Bluetooth transceiver may still be on, and somebody might be able to hack into your Bluetooth headset and convince it to tell the cell phone, "Hey, I'd like to make a call. Please turn on your antenna." Even though this is a security flaw in the Bluetooth headset, it was used as a stepping stone into hacking your cell phone.

There's also the possibility that you simply forgot that you had set a text message for delayed delivery, causing the phone to turn on its antenna when the delivery time is reached. Oops. You messed up, and now somebody is intensive care.

As of this writing, there is no known exploit for drag and drop into console windows, but since drag and drop uses highly extensible technology (namely COM and data objects), the possibility that one of those extension points may be used as an attack vector was deemed too great a risk compared to the benefit of the feature. The anonymous commenter concludes,

> Now if this is not a security hole, then either Csrss doesn't execute code in OLE objects it receives, or it doesn't accept any OLE objects received, or isn't able to receive OLE objects at all. Which one is it?

CSRSS does not execute untrusted code in OLE objects it receives, but the fact that OLE objects are in the CSRSS process at all give the security folks the heebie-jeebies. Although there is no known security hole, there is great *potential* for a security hole, and that's the reason for removing the potentially dangerous code from CSRSS even though it is (in theory) never executed.

I bet you'd be nervous if somebody pointed a loaded gun at you even though the safety is engaged.

Other discussion of defense in depth, including more examples:

- Larry Osterman explains why defense in depth is so important and responds to an article that rejects the notion.
- Michael Howard gives an example of how a defense in depth precaution protected against a security vulnerability that was not yet known to exist.
- Eric Lippert describes the use of salt as a defense in depth against password theft.
- The RSS Technologies Team describes how Internet Explorer removes all script from feeds before displaying them, and then, as a defense in depth precaution, displays the feed in a zone that forbids scripting just in case there happens to be a bug in the script removal code.

Raymond Chen

Follow