When you crash, make sure you crash in the right place

devblogs.microsoft.com/oldnewthing/20060928-04

September 28, 2006

Raymond Chen

Last time, I recommended that functions should just crash when given invalid pointers. There's a subtlety to this advice, however, and that's making sure you crash in the right place. If your function and your function's caller both reside on the same side of a security boundary, then go ahead and crash inside your function. If the caller is a bad guy who is trying to get your function to crash, then there's nothing the caller has accomplished if your function runs in the same security context as the caller. After all, if the caller wanted to make your program do something bad, it could've just done that bad thing itself! If it gave you a pointer to invalid memory and you crashed trying to access it, well the caller could have accomplished the same thing by just accessing the invalid memory directly. If your function resides on the other side of a security boundary, however, then having your function crash or behave erratically gives the malicious caller a power which he did not already have. For example, your function may reside in a service or local server, where the call arrives from another process. A malicious caller can pass intentionally malformed data to you via some form of IPC, causing your service or local server to crash. Or your function might reside in the same process as the caller but under a different security context. For example, it might be impersonating, or it may be operating on untrusted data. Another example of a security boundary is the boundary between user mode and kernel mode. Kernel mode cannot crash on parameters passed from user mode, because kernel mode runs at a higher protection level from user mode. In these cases, you want to make sure you crash in the correct context. In the IPC case, there typically will be a stub on the client side that does the hard work of taking the parameters and packaging them up for IPC. If the stub is given an invalid pointer, it should crash **in the stub**, so that the crash occurs in the same security context as the caller. A caller who passes an invalid pointer by mistake can then debug the crash in a context that is meaningful to the caller. (Of course, a malicious caller won't use your stub but will instead package the data manually and IPC it directly to the server. Your server can't crash on malicious inbound data, since that data came from a different security context.)

If you're feeling really ambitious (and few people do), you can have the server react to malformed data by returning a special error code, which the stub detects and converts to an exception. Again, this doesn't do anything to crash the malicious caller, because the malicious caller is bypassing your stub. But it may help the caller who thought it was passing a valid pointer.

Raymond Chen

Follow