How did the scopes for the CryptProtectMemory function end up in a strange order?

devblogs.microsoft.com/oldnewthing/20150423-00

April 23, 2015

Raymond Chen

A few weeks ago, I left an exercise: Propose a theory as to why the names and values of the scopes for the `CryptProtectMemory` function are the way they are.

I didn't know the answer when I posed the exercise, but I went back and dug into it.

The `CryptProtectMemory` function started out as an internal function back in Windows 2000, and when originally introduced, there were only two scopes: Within a process and cross-process. The Flags parameter therefore defined only a single bit, leaving the other bits reserved (must be zero). If the bottom bit was clear, then the memory was protected within a process; if the bottom bit was set, then the memory was protected across processes.

Later, the team realized that they needed to add a third scope, the one that corresponds to `CRYPTPROTECT_SAME_LOGON`. They didn't want to make a breaking change for existing callers, but they saw that they could retarget what used to be a Flags parameter as an Options parameter, and they added the new scope as a third option.

The numeric values remained unchanged, which meant that the new function was backward-compatible with existing callers.

Bonus chatter: Commenter sense is correct that `SAME_LOGON` can be used by a service while impersonating the client, however it is not the case that the scope can be larger when impersonating a remote user. The memory block returned by the `CryptProtectMemory` function can be decrypted only on the same machine that encrypted it, and only as long as the machine has not been rebooted.

Raymond Chen

Follow