VirtualLock only locks your memory into the working set

devblogs.microsoft.com/oldnewthing/20071106-00

November 6, 2007

Raymond Chen

When you lock memory with `VirtualLock` it locks the memory *into your process's working set*. It doesn't mean that the memory will never be paged out. It just means that the memory won't be paged out *as long as there is a thread executing in your process*, because a process's working set need be present in memory only when the process is actually executing. (Earlier versions of the MSDN documentation used to say this more clearly. At some point, the text changed to say that it locked the memory physically rather than locking it into the working set. I don't know who changed it, but it was a step backwards.) The working set is the set of pages that the memory manager will keep in memory while your program is running, because it is the set of pages that the memory manager predicts your program accesses frequently, so keeping those pages in memory when your program is executing keeps your program running without taking a ridiculous number of page faults. (Of course, if the prediction is wrong, then you get a ridiculous number of page faults anyway.) Now look at the contrapositive: If all the threads in your process are blocked, then the working set rules do not apply since the working set is needed only when your process is executing, which it isn't. If all your threads are blocked, then the entire working set is eligible for being paged out. I've seen people use `VirtualLock` expecting that it prevents the memory from being written to the page file. But as you see from the discussion above, there is no such guarantee. (Besides, if the user hibernates the computer, all the pages that aren't in the page file are going to get written to the hibernation file, so they'll end up on disk one way or another.) Even if you've magically managed to prevent the data from being written to the page file, you're still vulnerable to another process calling `ReadProcessMemory` to suck the data out of your address space. If you really want to lock memory, you can grant your process the SeLockMemoryPrivilege privilege and use the AWE functions to allocate non-pageable memory. Mind you, this is generally considered to be an anti-social thing to do on a paging system. The AWE functions were designed for large database programs that want to manage their paging manually. And they still won't prevent somebody from using `ReadProcessMemory` to suck the data out of your address space. If you have relatively small chunks of sensitive data, the solution I've seen recommended is to use `CryptProtectData` and `CryptUnprotectData`. The encryption keys used by these functions are generated pseudo-randomly at boot time and are kept in kernel mode. (Therefore, nobody can

`ReadProcessMemory` them, and they won't get captured by a user-mode crash dump.) Indeed, this is the mechanism that many components in Windows 2003 Server to reduce the exposure of sensitive information in the page file and hibernation file.

Follow-up: I've been informed by the memory manager folks that the working set interpretation was overly conservative and that in practice, the memory that has been virtually locked won't be written to the pagefile. Of course, the other concerns still apply, so you still have to worry about the hibernation file and another process sucking the data out via `ReadProcessMemory`.

Raymond Chen

Follow