# How is it that WriteProcessMemory succeeds in writing to read-only memory?

devblogs.microsoft.com/oldnewthing/20181206-00

December 6, 2018

Raymond Chen

When you call `WriteProcessMemory` and tell it to write to memory that is read-only, the `WriteProcessMemory` succeeds. How can that be?

Because `WriteProcessMemory` tries really hard to please you.

As I noted some time ago, the primary audience for functions like `CreateRemoteThread` and `WriteProcessMemory` is debuggers. And when debuggers try to patch memory, it's often for things like patching in a breakpoint instruction or doing some edit-and-continue magic. So the `WriteProcessMemory` tries really hard to get those bytes written. If the page is read-only, `WriteProcessMemory` temporarily changes the permission to read-write, updates the memory, and then restores the original permission.

"No need to thank me, just trying to help."

There is a race condition if the target process happens to be manipulating the page protection at the same time that `WriteProcessMemory` is. But that's okay, because the intended audience is debuggers, and debuggers will freeze the target process before trying to edit its memory.

There is no security hole here, because the way the `WriteProcessMemory` function changes the page protection is basically `VirtualProtectEx`, so it will succeed only if you already could have modified the protections yourself anyway. If you didn't have permission to change the protections, then `WriteProcessMemory`'s attempt to change the protections would fail too.

Raymond Chen

**Follow**