Is it a good idea to let WriteProcessMemory manage the page protection for me?

devblogs.microsoft.com/oldnewthing/20190729-00

July 29, 2019

Raymond Chen

Some time ago, I noted that the `WriteProcessMemory` function will make a page read-write if necessary, as a courtesy to debuggers who want to use it to patch code. Is this something you should rely on?

No, it's not something you should rely on. It is a courtesy, not contractual, and courtesies can go away at any time.

As I noted in the article, the courtesy itself can create problems. It creates a race condition where the courtesy page protection change can collide with an actual page protection change in the program, causing one or the other to be lost.

App thread	WriteProcessMemory

	`if (page-is-read-only) {`
	`  make-page-read-write;`
	`  write-the-data;`

`change-protection-to-read-execute;`	
	`  restore-original-protection;` `}`

The `WriteProcessMemory` function noticed that the page was write-protected, so it changed the page to read-write, wrote the data, and then changed it back to read-only. But at the same time, the application changed the protection from read-only to read-execute, Unfortunately, that change was overwritten by the `WriteProcessMemory` function when it tried to restore the original protection. The result: When the app tries to execute code on the page, it gets a no-execute exception.

You can imagine other race conditions. For example, the app thread could change the protection to read-execute one step earlier, after the `WriteProcessMemory` function changed it to read-write, but before it could write the data. (In that case, the `WriteProcess-`

`Memory` function reports that the write operation failed.) Or perhaps the app thread changed the protection to read-execute two steps earlier, afer the `WriteProcessMemory` function realized that the page was read-only but before it could change it to read-write.

These are all bad things, where the `WriteProcessMemory` function tried to be unobstrusive but ended up interfering with the operation of the program beyond simply writing memory.

You should try to avoid bad things.

As noted, the intended audience for the `WriteProcessMemory` function was debuggers, and when debuggers patch process memory, they do so when the process is broken into the debugger, hence no app threads can be running. The race condition doesn't exist in that case.

If you're going to use the `WriteProcessMemory` function to write to memory of a live running process, you need to coordinate with that process to make sure your memory write operation won't collide with the app's own virtual memory operations.

Bonus chatter: As I noted, this behavior of the `WriteProcessMemory` function is a courtesy, not contractual. Windows 95 and Windows CE dealt with the problem differently: Instead of making the page temporarily read-write, they made the page *permanently* read-write. In other words, they didn't bother restoring the original page protections when they were done. They just left the page read-write.

Raymond Chen

Follow