Why doesn't HeapValidate detect corruption in the managed heap?

devblogs.microsoft.com/oldnewthing/20130130-00

January 30, 2013

Raymond Chen

A customer had a program that was corrupting the managed heap by p/invoking incorrectly. The problem didn't show up until the next garbage collection pass, at which point the CLR got all freaked-out-like. "According to Knowledge Base article 286470, the `GFlags` tool is supposed to catch heap corruption, but it doesn't catch squat."

Depending on your point of view, this is either a case of the customer not understanding what things mean in context or of the KB article author looking at the world through kernel-colored glasses.

The `GFlags` tool, pageheap, full pageheap, and the `HeapValidate` function all operate on heaps, but the sense of the word *heap* here is "heaps created by the `HeapCreate` function." If your program does a `VirtualAlloc` and then carves out sub-allocations from it, well, it's not like `GFlags` and `HeapValidate` are psychic and can magically reverse-engineer your code in order to understand your custom heap implementation and be able to determine whether your custom heap is corrupted.

Clearly no such function could be written, because that's even harder than the Halting Problem! One property of a non-corrupted heap is that it will not send the heap manager into an infinite loop. Therefore, proving that the heap is not corrupted, given no information about the heap implementation other than the code itself, would require proving that the next heap call will return. And that's just *one* of the things the imaginary *ValidateAnyHeap* function would have to do. (We try to limit ourselves to one impossible thing at a time.)

The `HeapValidate` function only knows how to validate heaps created by the `HeapCreate` function. It does not have magic insight into custom heap implementations. The `GFlags` program modifies the behavior of heaps created by the `HeapCreate` function, because it naturally does not know what debugging features you've added to your custom heap implementation, so it doesn't know what it needs to do to turn them on and off.

As far as the kernel folks are concerned, "heap" means "something created by the `HeapCreate` function." Anything else is just an imposter.

If you are looking for corruption in a custom heap implementation, then you need to go ask the authors of that custom heap implementation if they provided any debugging facilities for that heap.