Why is Explorer opted out of Data Execution Prevention and termination on heap corruption, and how effective is the policy to opt it back in?

devblogs.microsoft.com/oldnewthing/20170620-00

June 20, 2017

Raymond Chen

Every so often, somebody will report a security issue with the fact that Explorer is linked with `/NXCOMPAT:NO`, which opts it out of Data Execution Prevention (DEP), also known as no-execute (NX). Explorer is also opted out of termination on heap corruption, but that's harder to detect from a static analysis. Why does Explorer opt out of these things? Doesn't that make it less secure?

Explorer opts out of DEP and termination on heap corruption as a backward-compatibility concession to shell extensions which are not DEP-compatible, or which corrupt the heap and managed to get away with it in earlier versions of Windows.

This compatibility concession applies only to the x86 version of Explorer. Explorer on all other processors (such as x64) enables both data execution prevention and termination on heap corruption.

Furthermore, this compatibility concession is controlled by a pair of group policies, both under Computer Configuration, Administrative Templates, Windows Components, File Explorer:

- Turn off Data Execution Prevention for Explorer
- Turn off heap termination on corruption[1]

This means that even though x86 Explorer does not have DEP or termination on heap corruption enabled in its file header, it will manually enable them as soon as it verifies that neither policy is in effect. This is one of the first things Explorer does, so the window of opportunity is relatively small. (And, as noted earlier, this situation exists only on x86. All other processor architectures enable DEP and termination on heap corruption from the get-go.)

Windows defaults to the safe setting of the policy, namely to re-enable DEP and termination on heap corruption. The policy is there so that organizations can selectively enable the policies if they have a shell extension that is not DEP-compliant or which contains heap corruption bugs.

Explorer is a single-instance application, and this means that if you are running the x64 version of Windows, and you manually run the x86 version of Explorer, that copy of Explorer doesn't do much other than hand the request to the existing 64-bit copy of Explorer. So the 32-bit version is there, but it doesn't hang around for long.

So go ahead and disable those policies in your organization. The x64 version of Explorer already has DEP and termination on heap corruption enabled. Disabling these policies (or leaving them unconfigured) will make sure you cover the x86 systems as well.

Bonus chatter: But wait, if I run an x86 application on an x64 version of Windows, and I use the File.Open dialog, then doesn't that give me an x86 Explorer? And wouldn't the policies take effect in that case?

No, the policies don't take effect in that case because the File.Open dialog is not running inside the Explorer process. It's running inside Notepad, or whatever your x86 process is. The DEP and termination on heap corruption policies of the host process are the ones that control behavior.

[1] You might have noticed the incorrect wording of the policy. The policy text says "heap termination on corruption", but that doesn't make sense because it's not the heap that is terminating; it's the process that is terminating. The phrase should be "termination on heap corruption". The reason for the messed up wording is that the heap feature is known as "terminate on corruption", and people lazily glue the word "heap" onto the phrase "terminate on corruption".

Raymond Chen

Follow