Why don't elevated processes inherit their environment variables from their non-elevated parent?

devblogs.microsoft.com/oldnewthing/20130703-00

July 3, 2013

Raymond Chen

As a general rule, child processes inherit the environment of their parent. But if the parent is non-elevated and the child is elevated, then this inheritance does not happen. Why not? There are two answers to this question. For the kernel-color glasses answer, I defer to Chris Jackson, the App Compat Guy. It's interesting to see how it all works, but it doesn't explain *why* the mechanism was designed to block environment variable inheritance. The reason for the design is that allowing an elevated process to inherit the `PATH` from a non-elevated process creates an attack vector. The non-elevated process sets its `PATH` to put some attacker-controlled directories ahead of the directories the elevated application actually expects. For example, suppose the elevated application links to `C:\Program Files\Common Files\Contoso\ContosoGridControl.dll`. It arranges for this by setting the system `PATH` to include the `C:\Program Files\Common Files\Contoso` directory. Or maybe the program calls `LoadLibrary` on a DLL that might not exist, and it handles the case that the call fails by disabling some optional feature. (Whether this is a good idea or not is beside the point.) The attacker changes the `PATH` to read `\\rogue\server;C:\Program Files\Common Files\Contoso`, so that the library search finds the evil copy on the rogue server before finding the expected version in the `Common Files` directory (or in the case of a DLL that may not exist, it finds the evil copy on the rogue server instead of failing outright). Bingo, the attacker has injected arbitrary code into an elevated process. Game over. For similar reasons, the current directory is reset to the system directory when a non-elevated program launches an elevated program. If the environment and current directory were inherited, then malware could ask to elevate Program X with a custom current directory or environment. The user will merely be asked if they want to run Program X elevated, unaware that it is being run in a nonstandard manner, using an execution environment that did not receive administrator approval. As a result, the malware would be able to sneak into the administrator account under sheep's clothing (the sheep being Program X). What if you want to run another program elevated, and with a custom current directory or environment?

Write a wrapper program which sets the current directory and environment, then launches the desired target process. Then ask the user for permission to run the wrapper elevated.

Raymond Chen

Follow