Windows File Confusion: Masquerading Unsigned Binaries as Signed Ones

exploit-monday.com/2013/02/WindowsFileConfusion.html

Could it be? A non-PowerShell related blog post?

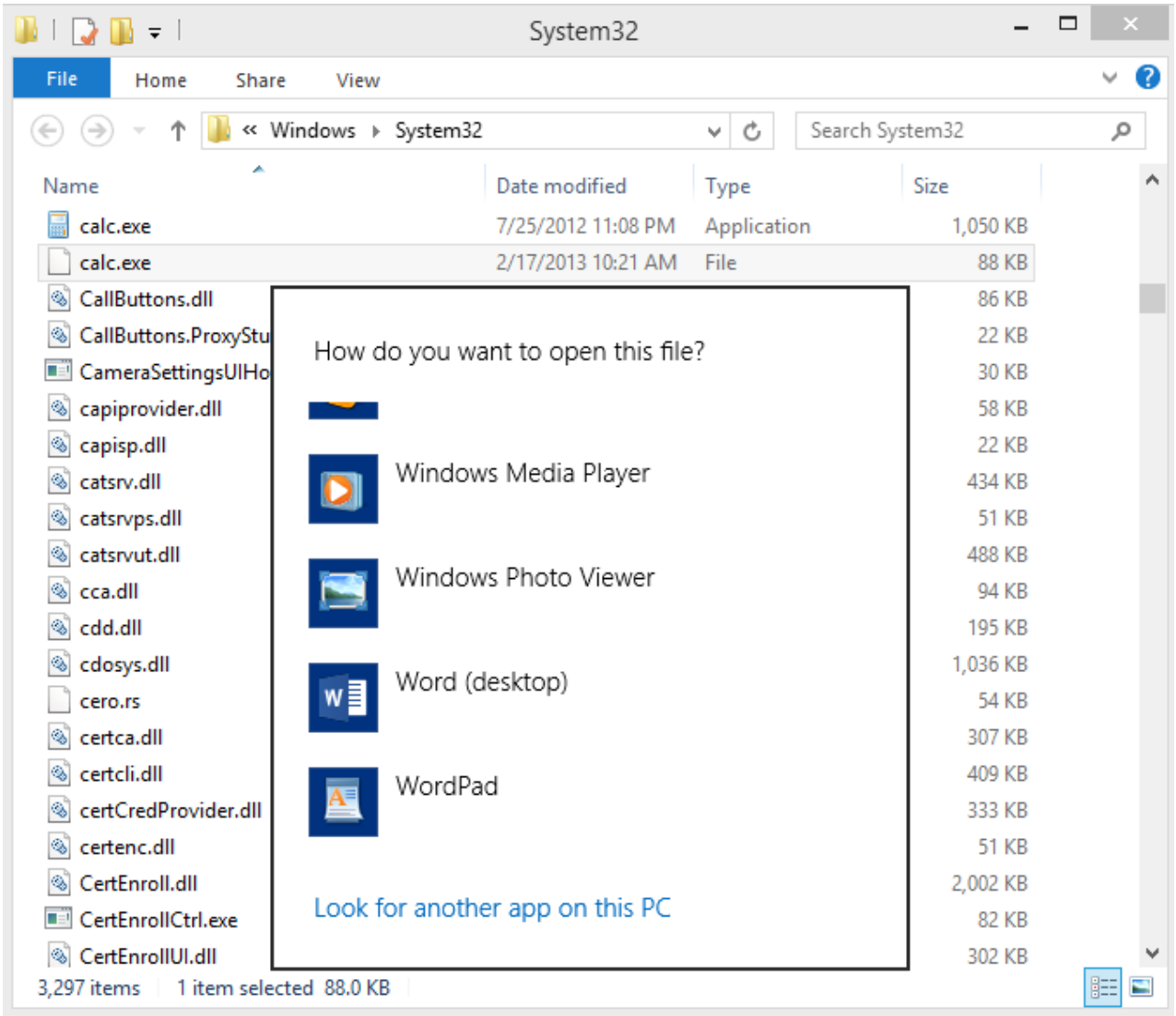
A while ago, Mark Baggett (@MarkBaggett) tipped me off to a technique that he had seen malware using in the wild. He explained that if you could manage to execute an unsigned binary with the same name as a signed binary (but with trailing white space) that it would take on the code signature of the legitimate binary.

I was obviously intrigued. This posed a bit of a challenge though because Windows automatically strips trailing white space from file names. I eventually figured out how to bypass this slight restriction by echoing the contents of my binary using `type` to a path prefixed with \\?\.

For example, to name the file "evil.exe" to "calc.exe   " (note the three trailing spaces), you would do the following:

type evil.exe > "\\?\C:\Windows\System32\calc.exe   "

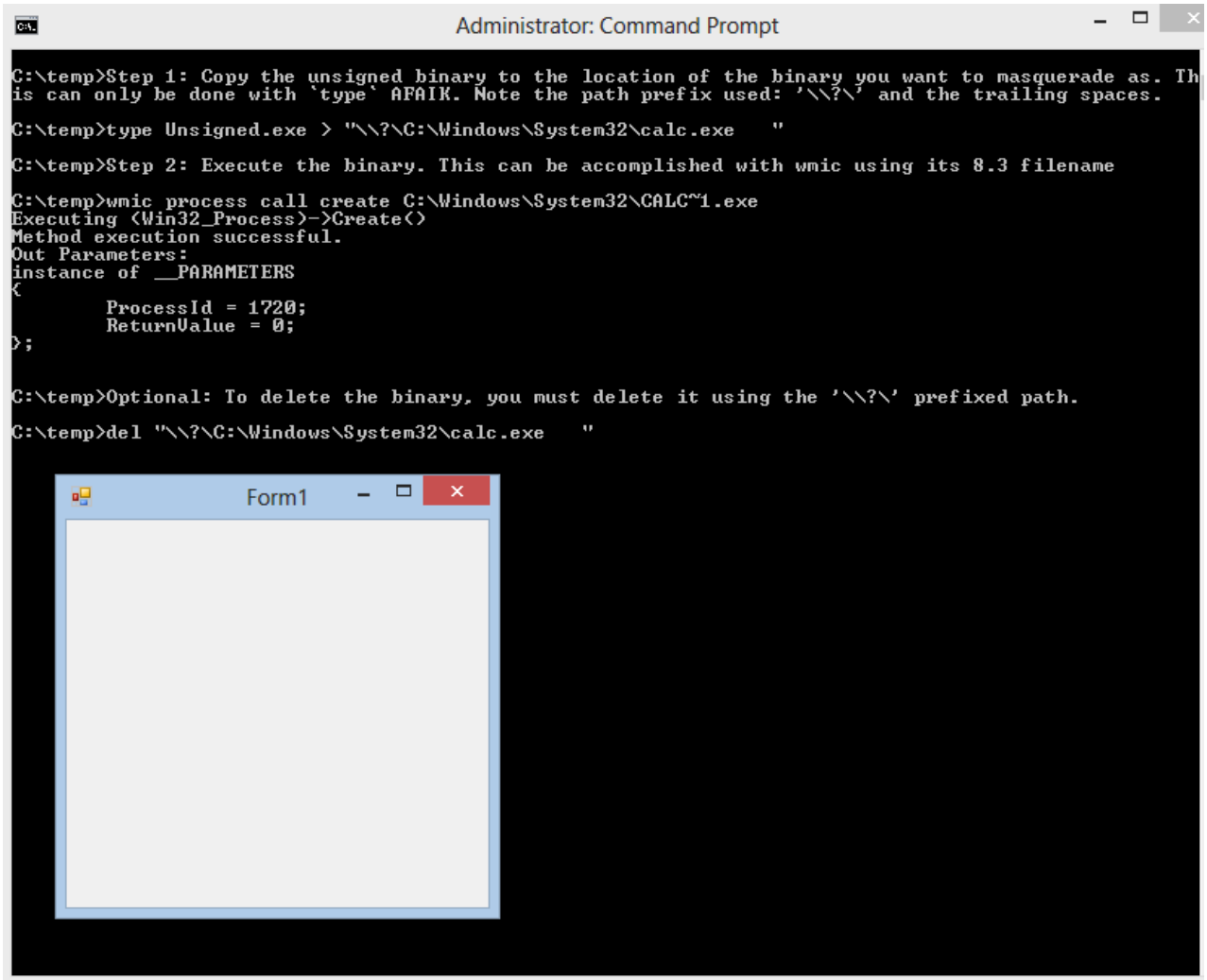
Great. We now have an unsigned binary with trailing white space. Now how does one execute it? As is demonstrated by the following screenshot, there is no longer a file association with the binary and Windows doesn't know how to load it.



I asked some friends for ideas on how to execute it and Chris Campbell (@obscuresec) had the genius idea to execute it via its 8.3 file name. Trying to call it directly from cmd.exe with its 8.3 file name unfortunately opened the original, signed calc. However, executing it with the Create method of the Win32_Process WMI class worked like a champ! This was accomplished with the following WMIC command:

wmic process call create C:\Windows\System32\CALC~1.exe

The following screenshot demonstrates the steps necessary to create a binary with trailing white space and consequently execute it:



Upon executing the unsigned file masquerading as the original calc.exe, you notice something amusing - when viewed in Process Explorer, the signed calc.exe is mistaken for the unsigned one and Process Explorer mistakenly reports the unsigned calc as having a valid code signature both in the process and module listings. It doesn't take long to see that something isn't right though. For example: the following differences are evident:

- The file sizes differ
- The unsigned binary is a .NET executable. The original calc is not.
- The unsigned binary is lacking the company name of "Microsoft Corporation". I tried to replicate the company name by modifying the assembly information of my unsigned binary but it caused Process Explorer to no longer say that it was '(Verified) Microsoft Windows'.

Another peculiarity I witnessed was that depending upon how you referenced the file names, the hash of the unsigned binary differed. Its true hash would only be manifested by referring to its 8.3 file name.

All in all, I would consider this technique to be more of a novelty than a serious vulnerability. For example, sigcheck detects the unsigned binary. Also, if code integrity checks are enforced (as Windows RT does), the kernel will prevent the unsigned binary from being executed. The underlying logic flaw is that Microsoft made the assumption

that an executable would not have any trailing white space. As a result, the file information of the original executable (with no trailing white space) is processed rather than that of the one with trailing white space.

You can download the binary I used in the screenshots here. All it is is a basic Windows Forms .NET executable with the calc.exe icon and assembly information.