Subvert CLR Process Listing With .NET Profilers

Archived: 2026-04-05 15:27:54 UTC

I recently stumbled onto an interesting capability of the CLR.

"A profiler is a tool that monitors the execution of another application. A common language runtime (CLR) profiler is a dynamic link library (DLL) that consists of functions that receive messages from, and send messages to, the CLR by using the profiling API. The profiler DLL is loaded by the CLR at run time."

https://msdn.microsoft.com/en-us/library/bb384493(v=vs.110).aspx

So. whats the big deal, really?

Turns out in .NET 4 allows for Registry-Free Profiler Startup and Attach. This can lead to some unintended consequences.

https://msdn.microsoft.com/en-us/library/ee471451(v=vs.100).aspx

In order for this work, you need to set 3 environment variables.

Again from MSDN:

> ## Startup-Load Profilers
>
> A startup-load profiler is loaded when the application to be profiled starts. The profiler is registered through the value of the following environment variable:
>
> - COR_ENABLE_PROFILING=1
>
> Starting with the .NET Framework 4, you use either the COR_PROFILER or the COR_PROFILER_PATH environment variable to specify the location of the profiler. (Only COR_PROFILER is available in earlier versions of the .NET Framework.)
>
> - COR_PROFILER={CLSID of profiler}
> - COR_PROFILER_PATH=full path of the profiler DLL
>
> > If COR_PROFILER_PATH is not present, the common language runtime (CLR) uses the CLSID from COR_PROFILER to locate the profiler in the HKEY_CLASSES_ROOT of the registry. If COR_PROFILER_PATH is present, the CLR uses its value to locate the profiler **and skips registry lookup**. (However, you still have to set COR_PROFILER, as discussed in the following list of rules.)

So, if our objective is to hijack a .NET process, like say PowerShell, we don't really want a Profiler to load, we just want to be able to manipulate the process. It turns out you can get a dll to load into the .NET process that is not even a Profiler. This was interesting to me. The CLSID is just random for this purpose.

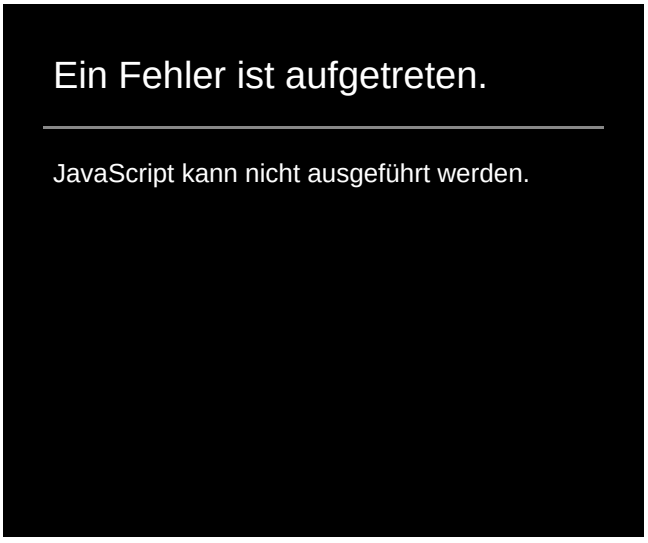
So, I had this idea, I could write quick POC DLL that hides a process from PowerShell. Well, short story is this. If you load a Profiler, and don't properly setup the Profiler structures, then the .NET CLR will promptly eject your dll.

For details of how we hook and hide see this article.

Thats ok. ;-) So what I did was create a DLL that loads another DLL from memory, and then when my profiler gets evicted, my hooking dll will stay resident. So the Profiler just becomes a bootstrap.

The result seen in this clip below. We enumerate processes with Get-Process in a "non-profiled" PowerShell process. We get the details just fine. Then we set our environment variables, load our PowerShell process, and now, the processes are not seen.

Video:



Why does this matter. As PowerShell become the window through which many sysadmins poll and interrogate the operating system. By using attaching a malicious profiler, we can mold the output so to speak to be what we want.

This was just a very basic example. I leave it up to you to explore further capabilities of tampering with the CLR/.NET applications through profilers.

Hope that was helpful.

Thats all for today.

Casey
@subTee
