Bypassing Application Whitelisting By Using dnx.exe

Published: 2016-11-17 · Archived: 2026-04-05 20:55:09 UTC

Over the past few weeks, I have had the pleasure to work side-by-side with Matt Graeber (@mattifestation) and Casey Smith (@subtee) researching Device Guard user mode code integrity (UMCI) bypasses. If you aren't familiar with Device Guard, you can read more about it here: https://technet.microsoft.com/en-us/itpro/windows/keep-secure/device-guard-deployment-guide.

In short, Device Guard UMCI prevents unsigned binaries from executing, restricts the Windows Scripting Host, and it places PowerShell in Constrained Language mode.

Recently, @mattifestation blogged about a typical Device Guard scenario and using the Microsoft Signed debuggers WinDbg/CDB as shellcode runners.

Soon after, @subtee released a post on using CSI.exe to run unsigned C# code on a Device Guard system.

Taking their lead, I decided to install the Visual Studio Enterprise trial and poke around to see what binaries existed. After much digging, I stumbled across dnx.exe, which is the Microsoft .NET Execution environment. If you are curious, you can read more on dnx.exe here:

https://blogs.msdn.microsoft.com/sujitdmello/2015/04/23/step-by-step-installation-instructions-for-getting-dnx-on-your-windows-machine/

In a Device Guard scenario, dnx.exe is allowed to execute as it is a Microsoft signed binary packaged with Visual Studio Enterprise. In order to execute dnx.exe on a Device Guard system (assuming it isn't already installed), you will need to gather dnx.exe and its required dependencies, and somehow transport everything to your target (this is an exercise left up to the reader).

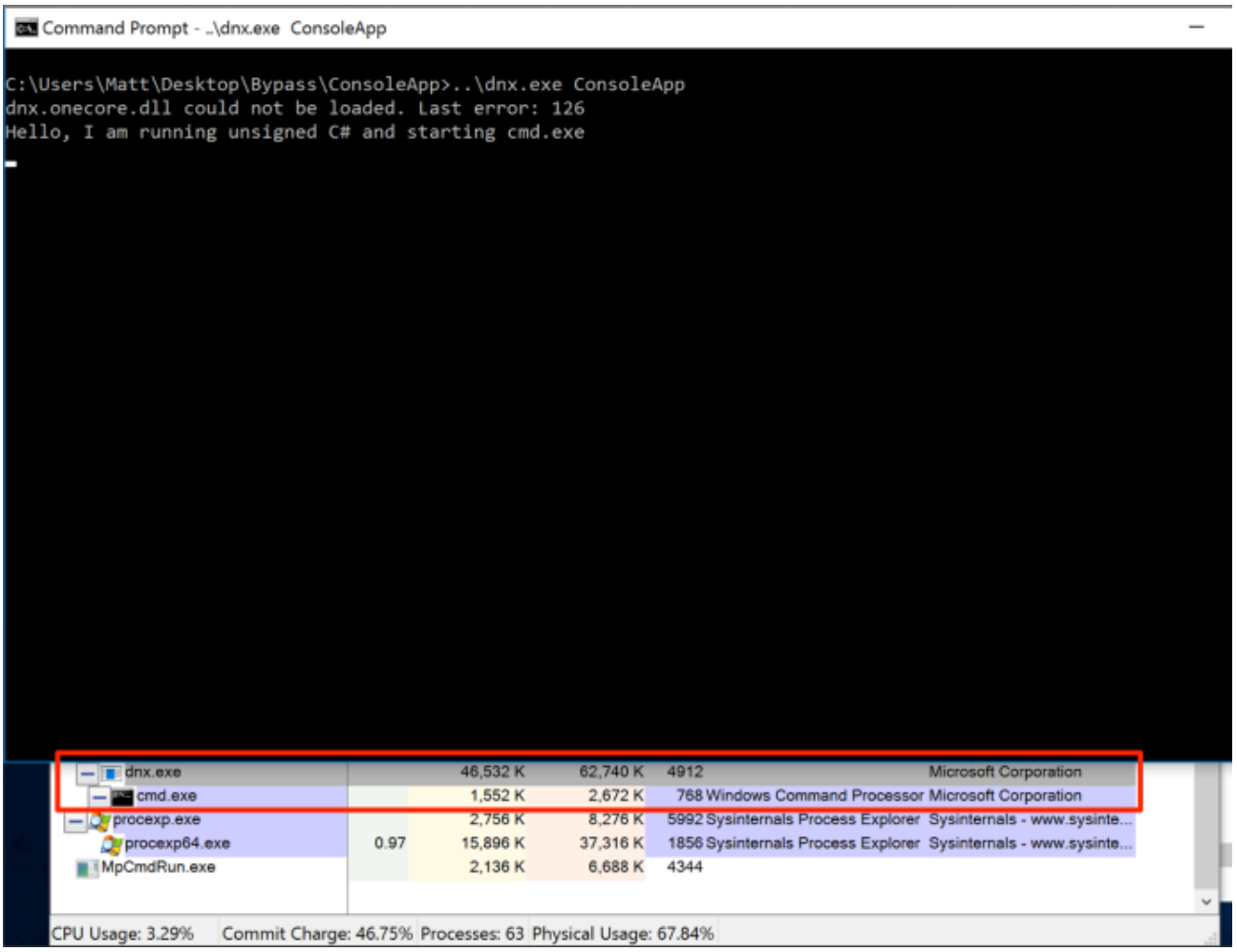
With everything required now on our target host, we can now start down the path of bypassing Device Guard's UMCI. Since dnx.exe allows for executing code in dynamic scenarios, we can use it to execute arbitrary, unsigned C# code. Fortunately, there is a solid example of this on Microsoft's blog above.

For example, we can create a C# file called "Program.cs" and add whatever C# code we want. To demonstrate the execution of unsigned code, we can keep things simple:

```
using System;
public class Program
{
    public static void Main()
    {
        Console.WriteLine("Hello, I am running unsigned C# and starting cmd.exe");
        System.Diagnostics.Process.Start("cmd.exe");
        Console.ReadLine();
    }
}
```

To satisfy the requirements of dnx.exe, a Project.json file is required, which specifies some of the requirements when executing the code. For this PoC, the example "Project.json" file can be used from Microsoft's blog here. As stated in their post, we can execute our C# by placing "Program.cs" and "Project.json" in a folder called "ConsoleApp" (this can obviously be renamed/modified).

Now that we have our files, we can execute our C# using dnx.exe by going into the "ConsoleApp" folder and invoking dnx.exe on it. This is done on a PC running Device Guard:



As you can see above, our unsigned C# successfully executed and is running inside of dnx.exe.

Fortunately, these "misplaced trust" bypasses can be mitigated via code integrity policy FilePublisher file rules. You can read up on creating these mitigation rules here:

http://www.exploit-monday.com/2016/09/using-device-guard-to-mitigate-against.html

You can find a comprehensive bypass mitigation policy here:

https://github.com/mattifestation/DeviceGuardBypassMitigationRules

Cheers!
Matt Nelson

Source: https://enigma0x3.net/2016/11/17/bypassing-application-whitelisting-by-using-dnx-exe/