BOHOPS

A blog about cybersecurity research, education, and news

WRITTEN BY BOHOPS

JANUARY 7, 2018

QUICK LINKS

- DiskShadow: The Return of VSS Evasion, Persistence, and Active Directory Database Extraction
- Leveraging INF-SCT Fetch & Execute Techniques For Bypass, Evasion, & Persistence (Part 2)
- Abusing .NET Core CLR Diagnostic Features (+ CVE-2023-33127)
- Abusing the COM Registry Structure: CLSID, LocalServer32, & InprocServer32
- Abusing the COM Registry Structure (Part 2): Hijacking & Loading Techniques
- Loading Alternate Data Stream (ADS) DLL/CPL Binaries to Bypass AppLocker
- Vshadow: Abusing the Volume Shadow Service for Evasion, Persistence, and Active Directory Database Extraction
- WS-Management COM: Another Approach for WinRM Lateral Movement
- Unmanaged Code Execution with .NET Dynamic PInvoke
- DotNet Core: A Vector For AWL Bypass & Defense Evasion

EXECUTING COMMANDS AND BYPASSING APPLOCKER WITH POWERSHELL DIAGNOSTIC SCRIPTS

INTRODUCTION

Last week, I was hunting around the Windows Operating System for interesting scripts and binaries that may be useful for future penetration tests and Red Team engagements. With increased client-side security, awareness, and monitoring (e.g. AppLocker, Device Guard, AMSI, Powershell ScriptBlock Logging, PowerShell Constraint Language Mode, User Mode Code Integrity, HIDS/anti-virus, the SOC, etc.), looking for ways to deceive, evade, and/or bypass security solutions have become a significant component of the ethical hacker's playbook.

While hunting, I came across an interesting directory structure that contained diagnostic scripts located at the following 'parent' path:

```
%systemroot%\diagnostics\system\
```

In particular, two subdirectories (\AERO) and (\Audio) contained two very interesting, signed PowerShell Scripts:

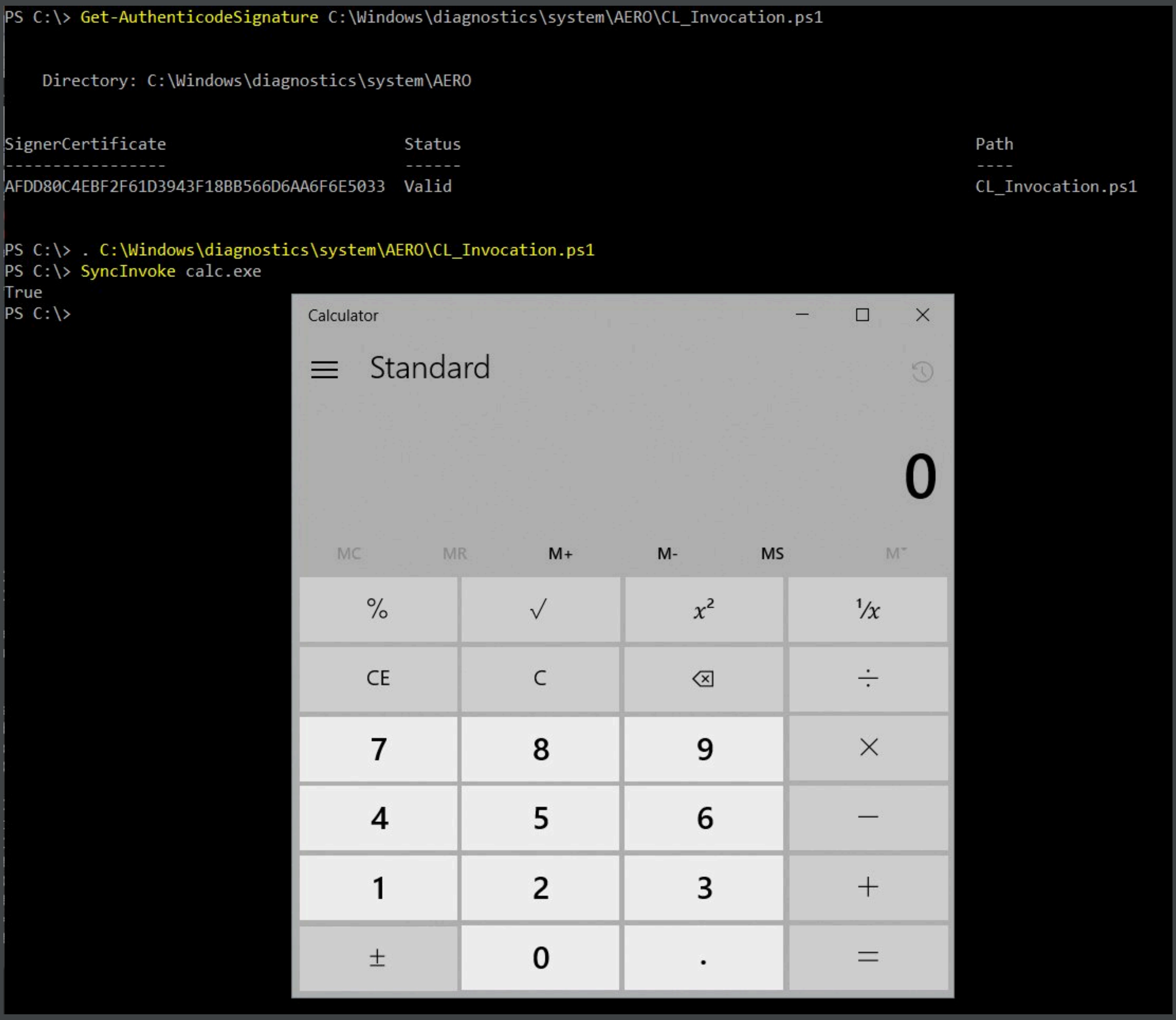
- **CL_Invocation.ps1**
- **CL_LoadAssembly.ps1**

CL_Invocation.ps1 provides a function (SyncInvoke) to execute binaries through System.Diagnostics.Process. and CL_LoadAssembly.ps1 provides two functions (LoadAssemblyFromNS and LoadAssemblyFromPath) for loading .NET/C# assemblies (DLLs/EXEs).

Reblog Subscribe •••

ANALYSIS OF CL_INVOCATION.PS1

While investigating this script, it was quite apparent that executing commands would be very easy, as demonstrated in the following screenshot:



```
PS C:\> Get-AuthenticodeSignature C:\Windows\diagnostics\system\AERO\CL_Invocation.ps1


    Directory: C:\Windows\diagnostics\system\AERO


SignerCertificate                         Status                                                        Path
-----------------                         ------                                                        ----
AFDD80C4EBF2F61D3943F18BB566D6AA6F6E5033  Valid                                                         CL_Invocation.ps1


PS C:\> . C:\Windows\diagnostics\system\AERO\CL_Invocation.ps1
PS C:\> SyncInvoke calc.exe
True
PS C:\>
```

Importing the module and using SyncInvoke is pretty straight forward, and command execution is successfully achieved through:

```
. CL_Invocation.ps1 (or import-module CL_Invocation.ps1)
SyncInvoke <command> <arg...>
```

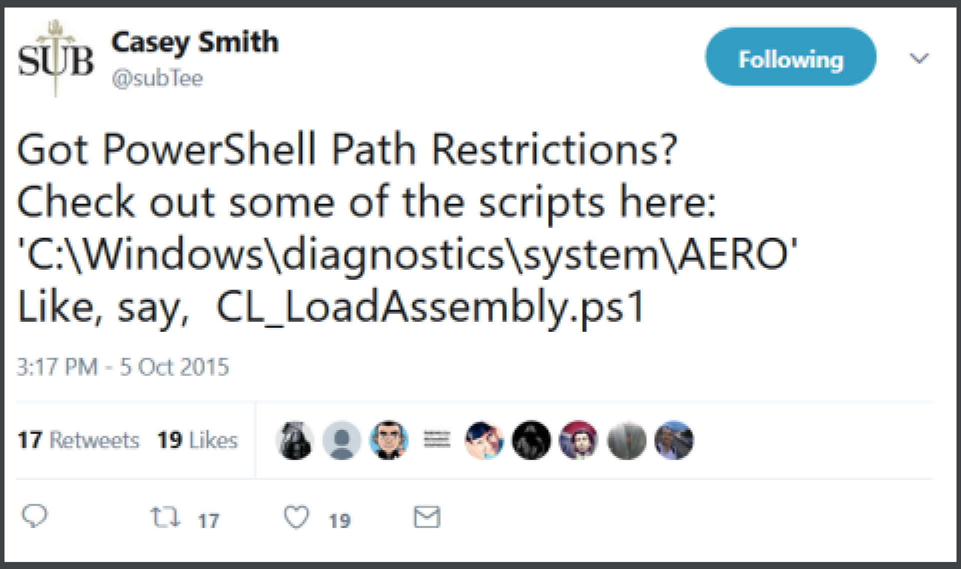
However, further research indicated that this technique **did not** bypass any protections with subsequent testing efforts. PowerShell Contrained Language Mode (in PSv5) prevented the execution of certain PowerShell code/scripts and Default AppLocker policies prevented the execution of unsigned binaries under the context of an unprivileged account. Still, CL_Invocation.ps1 may have merit within trusted execution chains and evading defender analysis when combined with other techniques.

****Big thanks to @Oddvarmoe and @xenosCR for their help and analysis of CL_Invocation**

ANALYSIS OF CL_LOADASSEMBLY.PS1

While investigating CL_LoadAssembly, I found a very interesting write-up (Applocker Bypass-Assembly Load) by @netbiosX that describes research

Smith (@subTee) during a presentation at SchmooCon 2015. He successfully discovered an AppLocker bypass through the use of loading assemblies within PowerShell by URL, file location, and byte code. Additionally, @subTee alluded to a bypass technique with CL_LoadAssembly in a Tweet posted a few years ago:


Casey Smith
@subTee
Following

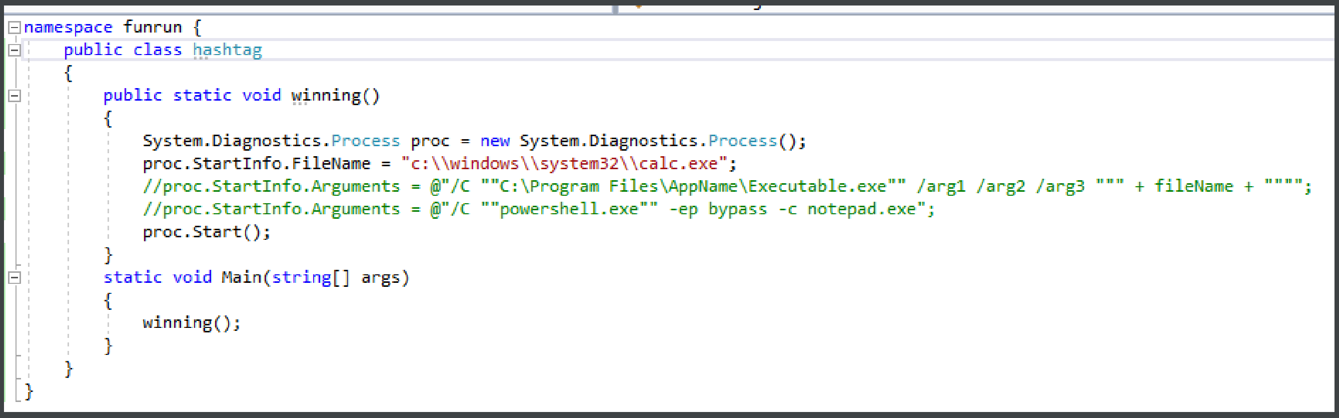
Got PowerShell Path Restrictions?
Check out some of the scripts here:
'C:\Windows\diagnostics\system\AERO'
Like, say, CL_LoadAssembly.ps1

3:17 PM - 5 Oct 2015

17 Retweets 19 Likes

17 19

In order to test this method, I compiled a very basic program (assembly) in C# (Target Framework: .NET 2.0) that I called funrun.exe, which runs calc.exe via proc.start() if (successfully) executed:


```
namespace funrun {
    public class hashtag
    {
        public static void winning()
        {
            System.Diagnostics.Process proc = new System.Diagnostics.Process();
            proc.StartInfo.FileName = "c:\\windows\\system32\\calc.exe";
            //proc.StartInfo.Arguments = @"/C ""C:\Program Files\AppName\Executable.exe"" /arg1 /arg2 /arg3 """ + fileName + """";
            //proc.StartInfo.Arguments = @"/C ""powershell.exe"" -ep bypass -c notepad.exe";
            proc.Start();
        }
        static void Main(string[] args)
        {
            winning();
        }
    }
}
```

Using a Windows 2016 machine with Default AppLocker rules under an unprivileged user context, the user attempted to execute funrun.exe directly. When called on the cmd line and PowerShell (v5), this was prevented by policy as shown in the following screenshot:

Funrun.exe was also prevented by policy when ran under PowerShell version 2:

Using CL_LoadAssembly, the user successfully loads the assembly with a path traversal call to funrun.exe. However, Constrained Language mode prevented the user from calling the method in PowerShell (v5) as indicated in the following screenshot:

To bypass Constrained Language mode, the user invokes PowerShell v2 and successfully loads the assembly with a path traversal call to funrun.exe:

Reblog Subscribe •••

The user calls the funrun assembly method and spawns calc.exe:

Success! As an unprivileged user, we proved that we could bypass Constrained Language mode by invoking PowerShell version 2 (Note: this must be enabled) and bypassed AppLocker by loading an assembly through CL_LoadAssembly.ps1. For completeness, here is the CL sequence:

```
powershell -v 2 -ep bypass
cd C:\windows\diagnostics\system\AERO
import-module .\CL_LoadAssembly.ps1
LoadAssemblyFromPath ..\..\..\..\temp\funrun.exe
[funrun.hashtag]::winning()
```

APPLOCKER BYPASS RESOURCES

For more information about AppLocker bypass techniques, I highly recommend checking out The Ultimate AppLocker Bypass List created and maintained by Oddvar Moe (@Oddvarmoe). Also, these resources were very helpful while drafting this post:

- AppLocker Bypass-Assembly Load – https://pentestlab.blog/tag/assembly-load/
- C# to Windows Meterpreter in 10 min – https://holdmybeersecurity.com/2016/09/11/c-to-windows-meterpreter-in-10mins/

CONCLUSION

Well folks, that covers interesting code execution and AppLocker bypass vectors to incorporate into your red team/pen test engagements. Please feel free to contact me or leave a message if you have any other questions/comments. Thank you for reading!

RELATED

COM XSL Transformation: Bypassing Microsoft Application Control Solutions (CVE-2018-8492)
January 10, 2019
With 3 comments

Loading Alternate Data Stream (ADS) DLL/CPL Binaries to Bypass AppLocker
January 23, 2018

Leveraging INF-SCT Fetch & Execute Techniques For Bypass, Evasion, & Persistence
February 26, 2018
In "ap

4 THOUGHTS ON "EXECUTING COMMANDS AND BYPASSING APPLOCKER WITH POWERSHELL DIAGNOSTIC SCRIPTS"

BOHOPS SEPTEMBER 12, 2018 AT 6:48 PM

I wrote this a while back, and I usually include defensive considerations with my blog posts – It is clear that I did not here, so my apologies for that. Some things to consider – In Win10/2016, PSv2 is not retroactively enabled by default b/c of dependencies on earlier versions of .NET. This is a great preventative measure in itself unless of course, Older .NET (2.0/3.5) is enabled like the assumption is made in the blog post. Surprisingly, this is often the case in many environments. IMO, moving away from PS v2 is advantageous for defenders to take advantage of the security features and optics that are available in later versions of PS(v5). Check out this blog post on ADSecurity.org for more info: https://adsecurity.org/?p=2277

★ Like

MIKE SEPTEMBER 12, 2018 AT 6:20 PM

Next time maybe a brief blurb on recommended prevention strategies for the blue team too?

★ Like

BOHOPS JANUARY 8, 2018 AT 1:03 PM

Hi Pralhad,

Thank you for replying! Yes, PowerShell v2 can be enabled in Win2016 via the Server Manager -> Manage ->Add Roles and Features -> Role-Based or feature-based installation -> .. -> Features –> Install .NET Framework 3.5 which includes .NET 2.0.

On Win 10 go to the Control Panel -> Programs and Features -> Turn Windows Features On/Off -> Toggle PowerShell 2.0 and .Net Framework 3.5

In many environments, various versions of .NET are usually installed which enables the backwards compatibility.

★ Like

PRALHAD JANUARY 8, 2018 AT 8:00 AM

Hello Jimmy,

Good post and Applocker bypass. But I observed Powervshell v2 is not running on Windows 10/2016. May I know how you by got v2 running on ? I get below error.

C:\>powershell -v 2 -ep bypass
Version v2.0.50727 of the .NET Framework is not installed
2 of Windows PowerShell.

Reblog Subscribe •••

★ Like

Comments are closed.

PREVIOUS POST

NEXT POST