

Inside Get-AuthenticodeSignature

mez0.cc/posts/authentifind

Table of Contents

- [Table of Contents](#)
- [Introduction](#)
- [Finding the Code](#)
- [AuthentiFind](#)
- [Conclusion](#)

Introduction

[Get-AuthenticodeSignature](#) in PowerShell is used to extract information from a files authenticode:

Gets information about the Authenticode signature for a file.

Here is an example usage on `Kernel32.dll` :

```
Get-AuthenticodeSignature -FilePath C:\Windows\System32\kernel32.dll|Format-List

SignerCertificate      : [Subject]
                       : CN=Microsoft Windows, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
                       :
                       : [Issuer]
                       : CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
                       :
                       : [Serial Number]
                       : 330000033B655FAEFADB75E9D600000000033B
                       :
                       : [Not Before]
                       : 02/09/2021 19:23:41
                       :
                       : [Not After]
                       : 01/09/2022 19:23:41
                       :
                       : [Thumbprint]
                       : BBD2C438000344F439BDFE5ABAC3223357CD67F

TimeStamperCertificate : [Subject]
                       : CN=Microsoft Time-Stamp Service, OU=Thales TSS ESN:4D2F-E3DD-BEEF, OU=Microsoft Operations Puerto Rico, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
                       :
                       : [Issuer]
                       : CN=Microsoft Time-Stamp PCA 2010, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
                       :
                       : [Serial Number]
                       : 33000001B0A1E38332E80D38C00001000001B0
                       :
                       : [Not Before]
                       : 02/03/2022 18:51:42
                       :
                       : [Not After]
                       : 11/05/2023 19:51:42
                       :
                       : [Thumbprint]
                       : 029E2F90DDDF0F914D05561992565E4BF2453C18

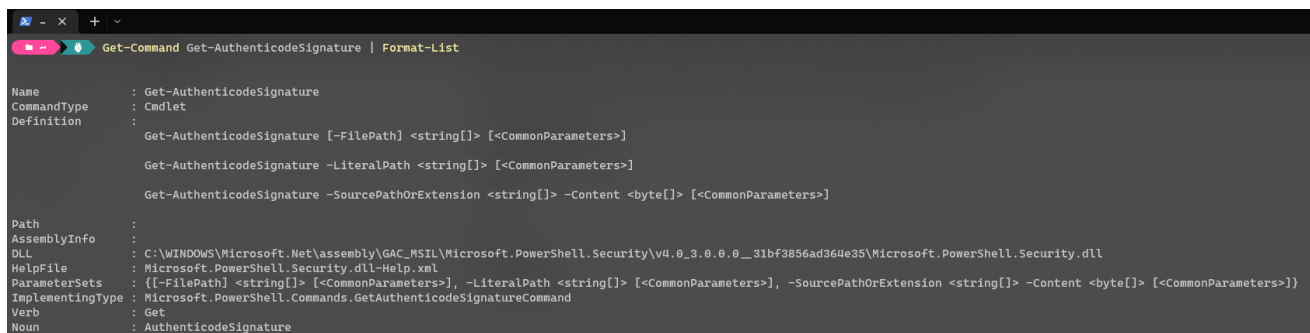
Status                : Valid
StatusMessage         : Signature verified.
Path                  : C:\Windows\System32\kernel32.dll
SignatureType         : Catalog
IsOSBinary            : True
```

My requirement is to identify every signed file on a system from a particular vendor. Its surprisingly easy to replicate...

Finding the Code

As these cmdlets tend to be .NET, they can just be opened with [dotpeek](#). However, the first thing is to find which DLL it is imported from. This is essentially the same thing that [amonsec](#) did in [AppLocker Policy Enumeration in C](#).

The following screenshot shows the command to get the DLL:



```
Get-Command Get-AuthenticodeSignature | Format-List

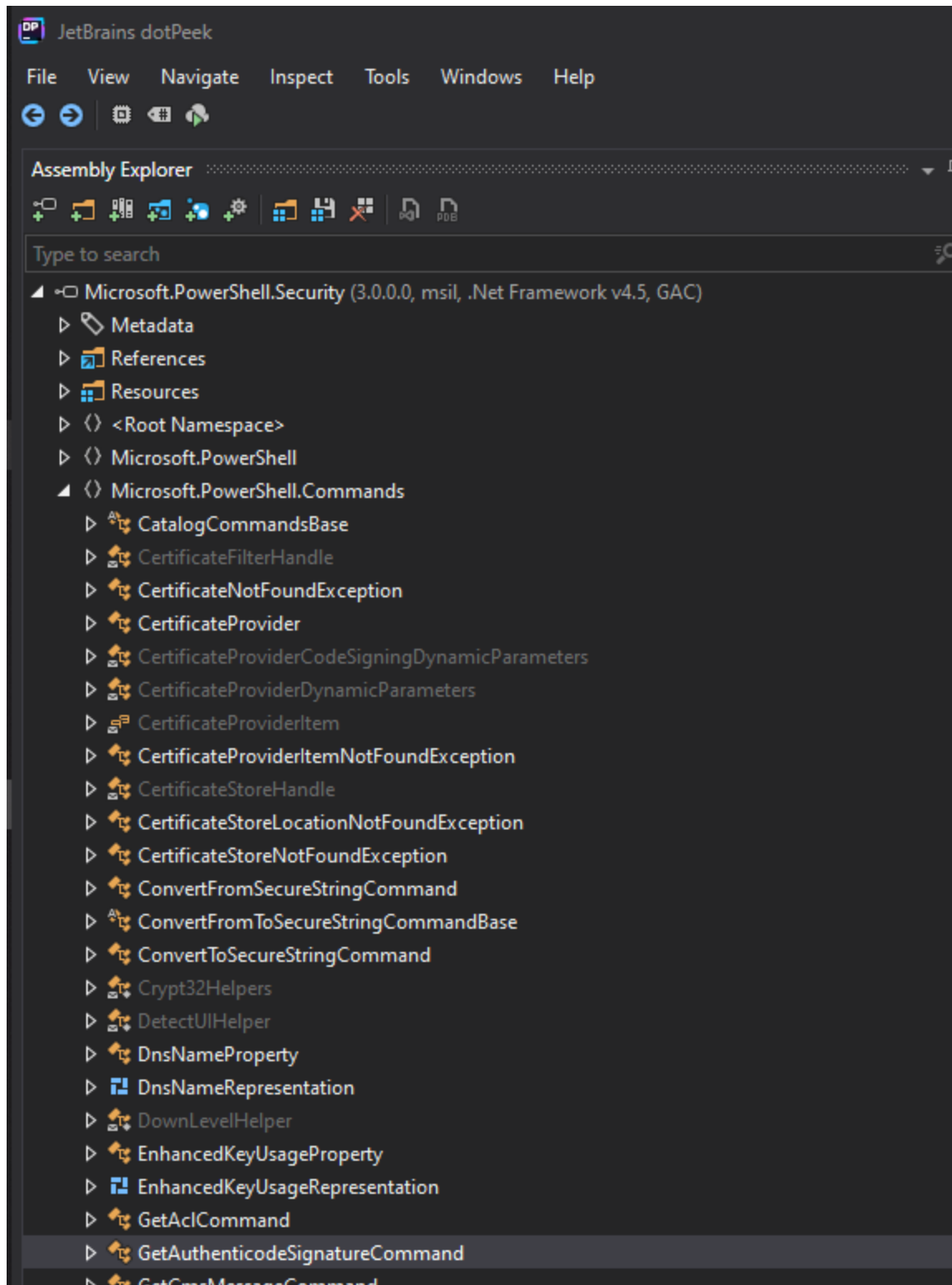
Name           : Get-AuthenticodeSignature
CommandType    : Cmdlet
Definition     : Get-AuthenticodeSignature [-FilePath] <string[]> [<CommonParameters>]
                Get-AuthenticodeSignature -LiteralPath <string[]> [<CommonParameters>]
                Get-AuthenticodeSignature -SourcePathOrExtension <string[]> -Content <byte[]> [<CommonParameters>]

Path           :
AssemblyInfo   :
DLL            : C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\Microsoft.PowerShell.Security\v4.0_3.0.0.0_31bf3856ad364e35\Microsoft.PowerShell.Security.dll
HelpFile       : Microsoft.PowerShell.Security.dll-Help.xml
ParameterSets : {[[-FilePath] <string[]> [<CommonParameters>]}, [-LiteralPath <string[]> [<CommonParameters>]], [-SourcePathOrExtension <string[]> -Content <byte[]> [<CommonParameters>]]}
ImplementingType : Microsoft.PowerShell.Commands.GetAuthenticodeSignatureCommand
Verb           : Get
Noun           : AuthenticodeSignature
```

The DLL location:

C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\Microsoft.PowerShell.Security\v4.0_3.0.0.0_

Opening in dotpeek shows all the classes, namespaces, and so on:



Opening the `GetAuthenticodeSignatureCommand` class:

```

using System.Management.Automation;
using System.Text;

namespace Microsoft.PowerShell.Commands
{
    [Cmdlet("Get", "AuthenticodeSignature", DefaultParameterSetName = "ByPath", HelpUri = "https://go.microsoft.com/fwlink/?LinkID=113307")]
    [OutputType(typeof (Signature))]
    public sealed class GetAuthenticodeSignatureCommand : SignatureCommandsBase
    {
        public GetAuthenticodeSignatureCommand()
            : base("Get-AuthenticodeSignature")
        {
        }

        protected override Signature PerformAction(string filePath) => SignatureHelper.GetSignature(filePath, (string) null);

        protected override Signature PerformAction(
            string sourcePathOrExtension,
            byte[] content)
        {
            return SignatureHelper.GetSignature(sourcePathOrExtension, Encoding.Unicode.GetString(content));
        }
    }
}

```

Cmdlet definitions are quite clean, we are looking for the `GetSignature()` function. Going on this function:

```

[ArchitectureSensitive]
internal static Signature GetSignature(string fileName, string fileContent)
{
    Signature signature = (Signature) null;
    if (fileContent == null)
        signature = SignatureHelper.GetSignatureFromCatalog(fileName);
    if (signature == null || signature.Status != SignatureStatus.Valid)
        signature = SignatureHelper.GetSignatureFromWinVerifyTrust(fileName, fileContent);
    return signature;
}

```

Here, we are interested in:

```
private static Signature GetSignatureFromCatalog(string filename)
```

There's a good chunk of code in here, all we care about is this section:

```

try
{
    using (FileStream fileStream = File.OpenRead(filename))
    {
        System.Management.Automation.Security.NativeMethods.SIGNATURE_INFO psiginfo
= new System.Management.Automation.Security.NativeMethods.SIGNATURE_INFO();
        psiginfo.cbSize = (uint) Marshal.SizeOf((object) psiginfo);
        IntPtr zero1 = IntPtr.Zero;
        IntPtr zero2 = IntPtr.Zero;
        try
        {
            if
(Utills.Succeeded(System.Management.Automation.Security.NativeMethods.WTGetSignatureInf
fileStream.SafeFileHandle.DangerousGetHandle(),
System.Management.Automation.Security.NativeMethods.SIGNATURE_INFO_FLAGS.SIF_AUTHENTIC
|
System.Management.Automation.Security.NativeMethods.SIGNATURE_INFO_FLAGS.SIF_CATALOG_S
|
System.Management.Automation.Security.NativeMethods.SIGNATURE_INFO_FLAGS.SIF_CHECK_OS_
|
System.Management.Automation.Security.NativeMethods.SIGNATURE_INFO_FLAGS.SIF_BASE_VERI
|
System.Management.Automation.Security.NativeMethods.SIGNATURE_INFO_FLAGS.SIF_CATALOG_F
ref psiginfo, ref zero1, ref zero2)))
            {
                uint fromSignatureState =
SignatureHelper.GetErrorFromSignatureState(psiginfo.nSignatureState);
                if (zero1 != IntPtr.Zero)
                {
                    X509Certificate2 signer = new X509Certificate2(zero1);
                    X509Certificate2 timestamperCert;
                    SignatureHelper.TryGetProviderSigner(zero2, out IntPtr _, out
timestamperCert);
                    ...

```

Pretty straight forward. First off, context management with `using` and then reading the file. What makes this easy is that the structs and enums can be looked up in dotkpeek by clicking F12 on the function, enum, or struct. The first one encountered is:

```

System.Management.Automation.Security.NativeMethods.SIGNATURE_INFO psiginfo = new
System.Management.Automation.Security.NativeMethods.SIGNATURE_INFO();

```

Pressing F12 on `SIGNATURE_INFO` goes to `NativeMethods.cs` :

```
GetAuthenticCodeSignatureCommand.cs  SignatureHelper.cs  Signature.cs  NativeMethods.cs ⇨ ✕
}
internal enum SIGNATURE_INFO_AVAILABILITY
{
    SIA_DISPLAYNAME = 1,
    SIA_PUBLISHERNAME = 2,
    SIA_MOREINFOURL = 4,
    SIA_HASH = 8,
}

internal enum SIGNATURE_INFO_TYPE
{
    SIT_UNKNOWN,
    SIT_AUTHENTICODE,
    SIT_CATALOG,
}

internal struct SIGNATURE_INFO
{
    internal uint cbSize;
    internal NativeMethods.SIGNATURE_STATE nSignatureState;
    internal NativeMethods.SIGNATURE_INFO_TYPE nSignatureType;
    internal uint dwSignatureInfoAvailability;
    internal uint dwInfoAvailability;
    [MarshalAs(UnmanagedType.LPWStr)]
    internal string pszDisplayName;
    internal uint cchDisplayName;
    [MarshalAs(UnmanagedType.LPWStr)]
    internal string pszPublisherName;
    internal uint cchPublisherName;
    [MarshalAs(UnmanagedType.LPWStr)]
    internal string pszMoreInfoURL;
    internal uint cchMoreInfoURL;
    internal IntPtr prgbHash;
    internal uint cbHash;
    internal int fOSBinary;
}

internal struct CERT_INFO
{
    internal uint dwVersion;
    internal NativeMethods.CRYPT_ATTR_BLOB SerialNumber;
    internal NativeMethods.CRYPT_ALGORITHM_IDENTIFIER SignatureAlgorithm;
    internal NativeMethods.CRYPT_ATTR_BLOB Issuer;
    internal NativeMethods.FILETIME NotBefore;
    internal NativeMethods.FILETIME NotAfter;
    internal NativeMethods.CRYPT_ATTR_BLOB Subject;
```

The same thing happens with function calls, for example:

```
if
(Utills.Succeeded(System.Management.Automation.Security.NativeMethods.WTGetSignatureInf
fileStream.SafeFileHandle.DangerousGetHandle(),
System.Management.Automation.Security.NativeMethods.SIGNATURE_INFO_FLAGS.SIF_AUTHENTIC
|
System.Management.Automation.Security.NativeMethods.SIGNATURE_INFO_FLAGS.SIF_CATALOG_S
|
System.Management.Automation.Security.NativeMethods.SIGNATURE_INFO_FLAGS.SIF_CHECK_OS_
|
System.Management.Automation.Security.NativeMethods.SIGNATURE_INFO_FLAGS.SIF_BASE_VERI
|
System.Management.Automation.Security.NativeMethods.SIGNATURE_INFO_FLAGS.SIF_CATALOG_F
ref psiginfo, ref zero1, ref zero2)))
```

F12 on `WTGetSignatureInfo` :

```
[DllImport("wintrust.dll", CallingConvention = CallingConvention.StdCall)]
internal static extern int WTGetSignatureInfo(
    [MarshalAs(UnmanagedType.LPWStr), In] string pszFile,
    [In] IntPtr hFile,
    NativeMethods.SIGNATURE_INFO_FLAGS sigInfoFlags,
    ref NativeMethods.SIGNATURE_INFO psiginfo,
    ref IntPtr ppCertContext,
    ref IntPtr phWTStateData);
```

Some F12'ing later, and here are all the DLL Imports:

```
[DllImportAttribute("wintrust.dll", EntryPoint = "WTGetSignatureInfo",
CallingConvention = CallingConvention.StdCall)]
internal static extern int WTGetSignatureInfo
(
    [InAttribute()][MarshalAsAttribute(UnmanagedType.LPWStr)] string pszFile,
    [InAttribute()] IntPtr hFile,
    SIGNATURE_INFO_FLAGS sigInfoFlags,
    ref SIGNATURE_INFO psiginfo,
    ref IntPtr ppCertContext,
    ref IntPtr phWVTStateData
);
```

```
[DllImport("wintrust.dll", CharSet = CharSet.Unicode, SetLastError = true)]
internal static extern IntPtr WTHelperProvDataFromStateData(IntPtr hStateData);
```

```
[DllImport("wintrust.dll", CharSet = CharSet.Unicode, SetLastError = true)]
internal static extern IntPtr WTHelperGetProvSignerFromChain
(
    IntPtr pProvData,
    uint idxSigner,
    uint fCounterSigner,
    uint idxCounterSigner
);
```

```
[DllImport("wintrust.dll", CharSet = CharSet.Unicode, SetLastError = true)]
internal static extern IntPtr WTHelperGetProvCertFromChain(IntPtr pSgnr, uint
idxCert);
```

```
[DllImport("wintrust.dll", CharSet = CharSet.Unicode, SetLastError = true)]
internal static extern uint WinVerifyTrust(IntPtr hWndNotUsed, IntPtr pgActionID,
IntPtr pWinTrustData);
```

```
[DllImport("crypt32.dll", CharSet = CharSet.Unicode, SetLastError = true)]
internal static extern bool CertFreeCertificateContext(IntPtr certContext);
```

All in all, the function requirements:

- [WTGetSignatureInfo](#)
- [WTHelperProvDataFromStateData](#)
- [WTHelperGetProvSignerFromChain](#)
- [WTHelperGetProvCertFromChain](#)
- [WinVerifyTrust](#)
- [CertFreeCertificateContext](#)

Knowing all the information, the next thing is to reimplement it.

AuthentiFind

.NET will be the easiest one first as I have all the data. So, my requirements are a complete recursive directory search (including all subdirectories) and Microsoft explained how to do this in [How to iterate through a directory tree \(C# Programming Guide\)](#). After removing a bunch of junk prints, this is the code:

```

public static void TraverseTree(string root, string issuer)
{
    Stack<string> dirs = new Stack<string>(20);

    if (!Directory.Exists(root))
    {
        throw new ArgumentException();
    }
    dirs.Push(root);

    while (dirs.Count > 0)
    {
        string currentDir = dirs.Pop();
        string[] subDirs;
        try
        {
            subDirs = Directory.GetDirectories(currentDir);
        }
        catch (UnauthorizedAccessException e)
        {
            continue;
        }
        catch (DirectoryNotFoundException e)
        {
            continue;
        }

        string[] files;
        try
        {
            files = Directory.GetFiles(currentDir);
        }
        catch (UnauthorizedAccessException e)
        {
            continue;
        }
        catch (DirectoryNotFoundException e)
        {
            continue;
        }
        foreach (string file in files)
        {
            try
            {
                // Do file check here
            }
            catch (FileNotFoundException e)
            {
                continue;
            }
        }
        foreach (string str in subDirs)
    }
}

```

```
        dirs.Push(str);  
    }  
}
```

Note the two function parameters:

1. A Source Directory (practically this will be `c:\` for me)
2. Issuer: Something to grep for in the Issuer/Subject properties

Using the DotPeek as an outline, this is my parsing function:

```

private static void ParseFileForSignature(string filepath, string needle)
{
    IntPtr ppCertContext = IntPtr.Zero;
    IntPtr phWVTStateData = IntPtr.Zero;

    try
    {
        using (FileStream fs = new FileStream(filepath, FileMode.Open,
        FileAccess.Read, FileShare.Read))
        {
            SIGNATURE_INFO psiginfo = new SIGNATURE_INFO();
            psiginfo.cbSize = (uint)Marshal.SizeOf((object)psiginfo);

            int error = WTGetSignatureInfo(
                filepath,
                fs.SafeFileHandle.DangerousGetHandle(),
                SIGNATURE_INFO_FLAGS.SIF_AUTHENTICODE_SIGNED |
SIGNATURE_INFO_FLAGS.SIF_CATALOG_SIGNED | SIGNATURE_INFO_FLAGS.SIF_CHECK_OS_BINARY |
SIGNATURE_INFO_FLAGS.SIF_BASE_VERIFICATION | SIGNATURE_INFO_FLAGS.SIF_CATALOG_FIRST,
                ref psiginfo,
                ref ppCertContext,
                ref phWVTStateData);

            uint state = GetErrorFromSignatureState(psiginfo.nSignatureState);

            if (ppCertContext != IntPtr.Zero)
            {
                X509Certificate2 signer = new X509Certificate2(ppCertContext);
                X509Certificate2 timestamperCert =
                TryGetProviderSigner(phWVTStateData, out IntPtr _);

                if (signer.Issuer.ToLower().Contains(needle.ToLower()))
                {
                    Console.WriteLine($"{filepath}:{signer.Issuer}");
                }
                else if (timestamperCert.Issuer.ToLower().Contains(needle.ToLower()))
                {
                    Console.WriteLine($"{filepath}:{timestamperCert.Issuer}");
                }
                else if (signer.Subject.ToLower().Contains(needle.ToLower()))
                {
                    Console.WriteLine($"{filepath}:{signer.Subject}");
                }
                else if
                (timestamperCert.Subject.ToLower().Contains(needle.ToLower()))
                {
                    Console.WriteLine($"{filepath}:{timestamperCert.Subject}");
                }
                else
                {
                    return;
                }
            }
        }
    }
}

```

```

    }
  }
}
finally
{
    if (phWVTStateData != IntPtr.Zero)
        FreeWVTStateData(phWVTStateData);
    if (ppCertContext != IntPtr.Zero)
        CertFreeCertificateContext(ppCertContext);
}
return;
}

```

If the grep data is in the subject or the issuer of either the signer or the timestamp, then display it.

Below is an example output of pointing it at `c:\windows\system32\` :

```

G:\Dropbox\GitHub\mez-0\Authentifind\Authentifind\bin\Debug>Authentifind.exe microsoft c:\Windows\System32
c:\Windows\System32\12520437.cpx:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\12520850.cpx:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\@AppHelpToast.png:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\@AudioToastIcon.png:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\@EnrollmentToastIcon.png:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\@VpnToastIcon.png:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\@WirelessDisplayToast.png:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\aaauthhelper.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\aadtb.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\aadwamExtension.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\AarSvc.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\AboveLockAppHost.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\accessibilitypl.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\accountaccessor.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\AccountsRt.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\AcGeneral.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\AcLayers.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\acledit.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\aclui.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\acppage.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\AcSpecfc.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\ActionCenter.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32>ActionCenterCPL.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\ActivationClient.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\ActivationManager.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\activeds.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\activeds.tlb:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\ActiveSyncProvider.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\actxprxy.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\AcWinRT.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\acwow64.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\AcXtrnal.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\AdaptiveCards.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\AddressParser.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\AdmTmpl.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\adpprovider.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\adrclient.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\adslidp.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\adslidpc.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\adsmsext.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\adsmt.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\adtschema.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\advapi32.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\advapi32res.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\advpack.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\aeevts.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\aepic.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\agentactivationruntime.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\agentactivationruntimestarter.exe:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\agentactivationruntimewindows.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
c:\Windows\System32\altspace.dll:CN=Microsoft Windows Production PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US

```

Voila.

Conclusion

This is more of a note to self, and is probably only usable in very few scenarios. For me, I needed to find every signed file from a *particular* vendor and this was my solution. Using the following calls, it could be a potentially useful BOF:

- `WTGetSignatureInfo`
- `WTHelperProvDataFromStateData`
- `WTHelperGetProvSignerFromChain`
- `WTHelperGetProvCertFromChain`
- `WinVerifyTrust`
- `CertFreeCertificateContext`

Code is available on [GitHub](#).