部分参考

简介

解析漏洞主要说的是一些特殊文件被IIS、apache、nginx或其他 Web服务器在某种情况下解释称脚本文件格式的漏洞。

比如网站管理员配置不当，导致php2、phtml、ascx等等这些文件也被当成脚本文件执行了。甚至某些情况下管理员错误的服务器配置导致.html、.xml等静态页面后缀的文件也被当成脚本文件执行。

但是，大部分的解析漏洞还是由于web服务器自身的漏洞，导致特殊文件被当成脚本文件执行了。

常见的容器

> iis
>
> 6.0：(win server 03服务器)
> 7.0、7.5：(08服务器)
> 8.0、8.5：(2012)
> 10.0：(2016)

> apache
> linux+php+apache
> nginx
>
> nginx+linux

文件解析漏洞

iis

iis5.x/6.0

解析利用方法有两种

目录解析

`/xx.asp/xx.jpg`

- 在 **IIS5.x/6.0** 中，在网站下建立文件夹的名字为 `*.asp、*.asa、*.cer、*.cdx` 的文件夹，那么其目录内的任何扩展名的文件都会被IIS当做asp文件来解释并执行。
- 例如创建目录 test.asp，那么 /test.asp/1.jpg 将被当做asp文件来执行。
- 假设黑客可以控制上传文件夹路径，就可以不管上传后你的图片改不改名都能拿shell了

文件名解析漏洞

`xx.asp;.jpg`

- 在**IIS5.x/6.0**中， 分号后面的不被解析
- 也就是说 `xie.asp;.jpg` 会被服务器看成是 `xie.asp`。
- 还有IIS6.0默认的可执行文件除了asp还包含这两种 `.asa、.cer` 。而有些网站对用户上传的文件进行校验，只是校验其后缀名。

- 所以我们只要上传`*.asp;.jpg`、`*.asa;.jpg`、`*.cer;.jpg`后缀的文件，就可以通过服务器校验，并且服务器会把它当成asp文件执行。

IIS 7.0/iis7.5/ nginx<8.03

畸形文件名解析漏洞

```
/text.asa       在iis7.0/7.5依然存在这样的漏洞
/text.cer
/text.cdx   <asp脚本解析>
```

检测网站能不能被asp或php解析

> 上传一张图片后，然后将图片打开一个新的标签，在图片后缀名后添加php看能不能运行

```
图片地址为：xx.com/update/1.jpg
xx.com/update/1.jpg/.asp     测试能不能运行asp文件
xx.com/update/1.jpg/.php
```

在默认Fast-CGI开启状况下，我们往图片里面写入下面的代码

```
<?php fputs(fopen('shell.php','w'),'<?php @eval($_POST[x])?>')?>
```

- 将文件保存成test.jpg格式，上传到服务器，假设上传路径为/upload，上传成功后，直接访问/upload/test.jpg/x.php，此时神奇的畸形解析开始发挥作用啦。
- test.jpg将会被服务器当成php文件执行，所以图片里面的代码就会被执行。我们会神奇的发现在/upload 目录下创建了一个一句话木马文件 shell.php 。
- 临时解决办法：设置 cgi.fix_pathinfo为0

这个解析漏洞和下面讲的Nginx的解析漏洞是一样的。

注意：确切的来说这种漏洞不属于容器的漏洞，而是属于php版本上的漏洞
所以只要是IIS的网站支持php脚本，都可以尝试`a.jpg/php`

其他解析漏洞

操作系统解析漏洞

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。

Apache

文件名解析漏洞

- 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。
- 比如`text.php.owf.rar`，`.owf`和`.rar`这两种后缀是apache不可识别的解析，apache就会把`text.php.owf.rar`解析成`text.php`。

- 如何判断是不是合法的后缀就是这个漏洞的利用关键，测试时可以尝试上传一个`xie.php.rara.jpg.png..`(把你知道的后缀都写上去)去测试是否是合法后缀。任意不识别的后缀，逐级向上识别。
- apache一般会定义黑名单，来限制一些文件类型的上传

  如禁止上传`php、asp、aspx、asa`类的文件

  这时候可以在后面添加一些允许的后缀进行突破黑名单，如`xx.php.jpg、xx.php._、xx.php.`等

.htaccess文件

- .htaccess文件是Apache服务器中的一个配置文件，它负责相关目录下的网页配置。
- 通过 .htaccess文件，可以实现：`网页301重定向、自定义404错误页面、改变文件扩展名、允许/阻止特定的用户或者目录的访问、禁止目录列表、配置默认文档`等功能
- IIS平台上不存在该文件，该文件默认开启，启用和关闭在 httpd.conf 文件中配置。

.htaccess 文件生效前提条件为

```
mod_rewrite 模块开启
    AllowOverride All

#1：这个.htaccess的意思就是把所有名字里面含有shell的文件当成php脚本来执行(这个shell也可以换成shell.jpg或者其他类型)
<FilesMatch   "shell">
SetHandler  application/x-httpd-php
</FilesMatchc>

#2：这里代码的意思可以让 .jpg后缀名文件格式的文件名以php格式解析
AddType application/x-httpd-php .jpg
```

Nginx

畸形解析漏洞

漏洞原因：

- php的配置文件 php.ini 文件中开启了 cgi.fix_pathinfo
- /etc/php5/fpm/pool.d/www.conf中不正确的配置security.limit_extensions，导致允许将其他格式文件作为php解析执行

实验分析：

- 在nginx<0.8.03环境中，我们新建一个文件，内容为： ，然后将其名字修改为：test.jpg
- 在浏览器中访问`http://192.168.10.139/test.jpg` 显示图片解析错误。在浏览器中访问`http://192.168.10.139/test.jpg/test.php`，显示：Access denied. 。这就奇怪了，test.jpg是文件不是目录，test.php更是根本就不存在的文件，访问/test.jpg/test.php没有报404，而是显示 Access denied. 。这是到底为啥?

原因在于：

Nginx拿到文件路径（更专业的说法是URI）`/test.jpg/test.php` 后，一看后缀是.php，便认为该文件是php文件，于是转交给php去处理。php一看`/test.jpg/test.php` 不存在，便删去最后的/test.php，又看/test.jpg存在，便把/test.jpg当成要执行的文件了，又因为后缀为.jpg，php认为这不是php文件，于是返回 Access denied. 。

这其中涉及到php的一个选项：`cgi.fix_pathinfo`，该值默认为1，表示开启。开启这一选项有什么用呢？看名字就知道是对文件路径进行处理。举个例子，当 php 遇到文件路径 `/aaa.xxx/bbb.yyy/ccc.zzz` 时，若 `/aaa.xxx/bbb.yyy/ccc.zzz` 不存在，则会去掉最后的 `/ccc.zzz` ，然后判断 `/aaa.xxx/bbb.yyy` 是否存在，若存在，则把 `/aaa.xxx/bbb.yyy` 当做文件 `/aaa.xxx/bbb.yyy/ccc.zzz` ，若 `/aaa.xxx/bbb.yyy` 仍不存在，则继续去掉 `/bbb.yyy` ，以此类推。

该选项在配置文件 php.ini 中。若是关闭该选项，访问`http://127.0.0.1/test.jpg/test.php`只会返回找不到文件。但关闭该选项很可能会导致一些其他错误，所以一般默认是开启的。

但是目前我们还没能成功执行代码，test.jpg 没有当成php文件执行，只是返回了 Access denied ，因为新版本的php引入了security.limit_extensions ，限制了可执行文件的后缀，默认只允许执行.php文件。

这一漏洞是由于Nginx中php配置不当而造成的，与Nginx版本无关，但在高版本的php中，由于`security.limit_extensions`的引入，使得该漏洞难以被成功利用。

为何是Nginx中的php才会有这一问题呢？因为Nginx只要一看URL中路径名以.php结尾，便不管该文件是否存在，直接交给php处理。而如Apache等，会先看该文件是否存在，若存在则再决定该如何处理。cgi.fix_pathinfo是php具有的，若在php前便已正确判断了文件是否存在，cgi.fix_pathinfo便派不上用场了，这一问题自然也就不存在了。（IIS在这一点和Nginx是一样的，同样存在这一问题）

<8.03?空字节代码执行漏洞

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: `0.5。*、0.6.*、0.7<=0.7.65、0.8<=0.8.37`
- 使用：Nginx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

在正常情况下这种方法是不能用的，但在`插马或burp抓包改包`时可以使用下

什么时候会自动截断

> get提交会自动截断
> post中输入使用%00然后全选`ctrl+shift+u`进行手动截断
>
> 在url提交当中会自动截断

上传文件漏洞

上传检测流程

将一个post包提交到web服务器，服务器检测允不允许，同意的话就上传，不同意就返回错误信息

服务器明明规则

第一种类型：上传文件名和服务器命名一致

第二种类型：上传文件名和服务器命名不一致(啊ui见，时间，日期等等)

常见的上传检测方式

> 1.客户端JavaScript检测（通常为检测文件扩展名）
> 2.服务端MIME类型检测（检测Content-Type内容）
> 3.服务端目录路径检测（检测跟path参数相关的内容）
> 4.服务端文件扩展名检测（检测跟文件extension相关的内容）
> 5.服务端文件内容检测（检测内容是否合法或含有恶意2代码）

一般绕过都是修改下面两种参数

```
Content-Disposition: form-data; name="upload_file"; filename="Koala.jpg"
Content-Type: image/jpeg
```

客户端javascript检测与绕过

首先判断js本地验证，通常可以根据它的验证警告弹框可以判断。

用burp抓包，在点击提交的时候burp没有抓到包就已经弹框，那么说明这个就是本地js验证。

绕过方法：

> 1.使用burp抓包改名
>
> 2.使用firebug直接删除掉本地验证的js代码
>
> 3.添加js验证的白名单如将php的格式添加进去

以upload-labs Pass-01的源码为例做一个绕过测试

```
function checkFile() {
    var file = document.getElementsByName('upload_file')[0].value;
    if (file == null || file == "") {
        alert("请选择要上传的文件!");
        return false;
    }
    //定义允许上传的文件类型
    var allow_ext = ".jpg|.png|.gif";
    //提取上传文件的类型
    var ext_name = file.substring(file.lastIndexOf("."));
    //判断上传文件类型是否允许上传
    if (allow_ext.indexOf(ext_name + "|") == -1) {
        var errMsg = "该文件不允许上传，请上传" + allow_ext + "类型的文件,当前文件类型
为: " + ext_name;
        alert(errMsg);
        return false;
    }
}
```

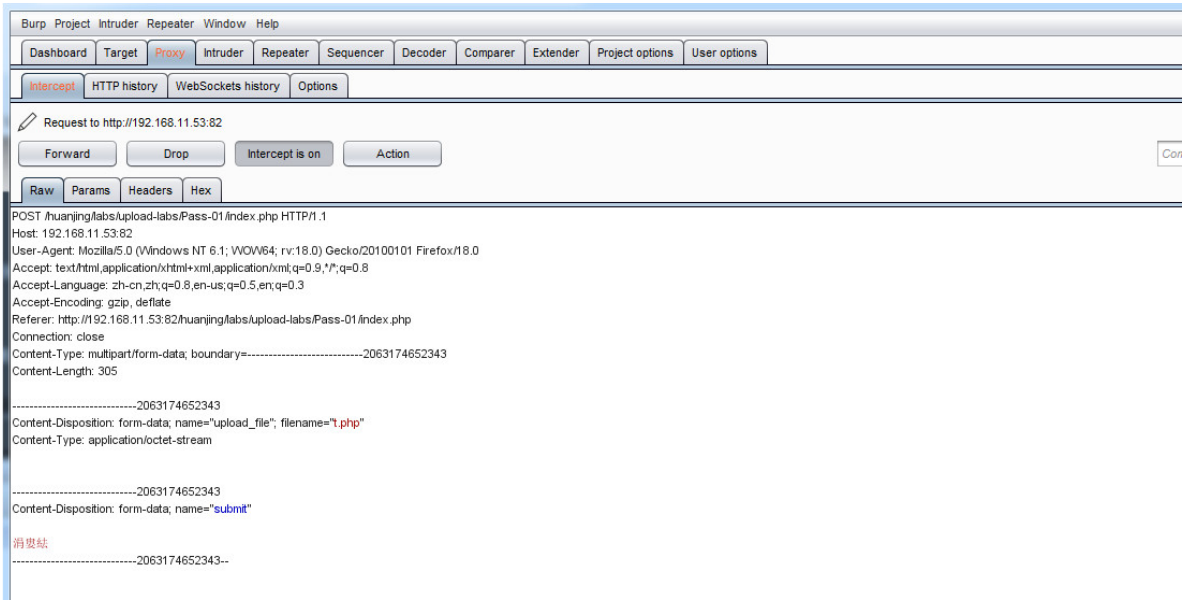
正常情况测试，开启burp

```
上传一张jpg的图片，能够正常抓包
上传一个php文件，弹警告框，burp没有抓到包
```

绕过测试：修改允许上传的类型--->上传--->抓包

```
 var allow_ext = ".php|.jpg|.png|.gif";
```

可以正常抓包，说明成功绕过了js本地验证



另外两种就不写了

上传服务器端验证绕过

服务端检测绕过（MIME类型检测）

- MIME的作用：使客服端软件识别区分不同类型的数据。
- 例如web浏览器就是通过MIME类型来判断使GIF图片，还是可打印的postscript文件。
- web服务器使用MIME来说明发送数据的种类，web客户端使用MIME来说明希望接收到的数据种类。Tomcat的安装目录\conf\web.xml中就定义了大量MIME类型
- MIME检测其实就是对HTTP数据包的`Content-Type`内容检测

绕过方法：

> 直接使用burp抓包，得到post上传数据后，直接修改为系统允许上传的类型
>
> 将`Content-type: text/plain`改成`Content-Type: image/gif`

可以用upload-labs Pass-01的源码进行尝试练习

服务端目录路径检测（检测跟path参数相关的内容）

- 目录路径检测，一般就检测路径是否合法，但稍微特殊一点的都没有防御。

实例

比如比较新的`fckeditor php <=2.6.4`任意文件上传漏洞

> 漏洞成因是因为`对目录路径的检测不够严谨`而导致可以用0x00 截断进行攻击

当post下面的url的时候

```
/fckeditor264/filemanager/connectors/php/connector.php?
Command=FileUpload&Type=Image&

CurrentFolder=fuck.php%00file.jpg HTTP/1.0
```

CurrentFolder这个变量的值会传到`ServerMapfolder($resourceType,$folderPath,$sCommand)`中的形参`$folder`里,而`$folder`在这个函数中并没做任何检测，就被`CombinePaths()`了

绕过方法

1、主要是利用 %00 截断攻击

`%00`的作用就是在url中删掉后面的内容

- 当 POST 下面的 URL 的时候

> CurrentFolder=fuck.php%00file.jpg HTTP/1.0

- 服务器端的 php 代码未进行检测就被直接写入文件系统了
- 此时的 content-type 还是 image/jpeg，但是文件名已经被截断了，最终上传的文件是 fuck.php

%00截断的两种利用方式：

> 1、更改filename，xx.php .jpg，在burpsuit中将空格对应的hex 20改为00
>
> 2、更改filename，xx.php%00.jpg，在burpsuit中将%00进行右键转换-url-urldecoder

2、修改上传目录绕过

- 如果 html 代码中有一个隐藏标签 ,这是文件上传时默认的文件夹，而我们对这个参数是可控的。
- 使用 burpsuite 将 value 值改为 pentest.php ，并提交上传一句话木马文件，如果服务器不存在此目录，则会创建此目录，然后将一句话木马写入该目录，如果是 IIS 6.0 ，则会解析网页木马

filepath漏洞

- filepath漏洞用来突破自动命名规则，主要有以下两种利用方式：

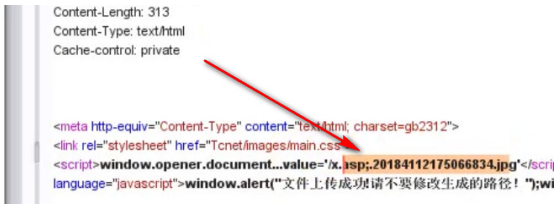
```
 2.1、改变文件上传后路径(filepath)，可以结合`IIS6.0目录解析漏洞`,但需要一定的创建权限

使用burp抓到包后放到Repeater，

修改路径为：/x.asp/
此时上传的xxx.gif的路径就变成了：/x.asp/xxx.gif

 2.2、直接改变文件名称(都是在filepath下进行修改)

修改路径为：/x.asp;.
修改后为：/x.asp;.xxx.gif
```



3、如果upload没有权限的话，可以将文件上传到上一级目录

> 通过burp抓包后将上传的文件名1.jpg修改为`../1.jpg`

4、文件头欺骗漏洞

- 在一句话木马前面加入GIF89a，然后将木马保存为图片格式，可以欺骗简单的waf。

5、filetype漏洞

- filetype漏洞主要是针对content-type字段，主要有两种利用方式：

```
1、先上传一个图片，然后将content-type:images/jpeg改为content-type:text/asp，然后对filename进行00截断，将图片内容替换为一句话木马。

2、直接使用burp抓包，得到post上传数据后，将Content-Type: text/plain改成 Content- Type: image/gif。
```

6、iconv函数限制上传

- 如果某天你上传文件发现，不敢你上传什么文件，上传后的文件都会自动添加一个.jpg的后缀，那么我们可以怀疑是否是使用iconv这个函数进行了上传的限制，
- 此时我们可以使用类似00截断的方法，但是这里不是00截断，二是80-EF截断，也就是说我们可以修改HEX为80到EF中的某一个来进行截断，如果真是使用这个函数的话，那么恭喜你上传任意文件成功！如上传一个xx.php ，然后截断抓包将后面的空格对应的十六进制改为80到EF中的任意一个！

7、双文件上传

南方、良精、动易...都有可能出现双文件上传

- 在一个文件上传的地方，右键审查元素，首先修改action为完整路径，然后复制粘贴上传浏览文件（<input ……），这样就会出现两个上传框，第一个上传正常文件，第二个选择一句话木马，然后提交。

测试

找到一个上传点，打开burp抓取上传包，然后将数据包发送到Repeater进行改包

服务端文件扩展名检测（检测跟文件extension相关的内容）

黑名单检测

- 黑名单的安全性比白名单的安全性低很多，攻击手法自然也比白名单多，一般有个专门的blacklist文件，里面会包含常见的危险脚本文件例如fckeditor 2.4.3或之前版本的黑名单

白名单检测

- 白名单相对来说比黑名单安全一些，但也不见得就绝对安全了

绕过方式

绕过黑名单检测

1、文件名大小写绕过

- 用像 AsP，pHp 之类的文件名绕过黑名单检测

2、名单列表绕过

- 用黑名单里没有的名单进行攻击，比如黑名单里没有 asa 或 cer 之类

3、特殊文件名绕过

- 比如发送的 http 包里把文件名改成 test.asp. 或 test.asp_(下划线为空格)，这种命名方式在 windows 系统里是不被允许的，所以需要在burp 之类里进行修改，然后绕过验证后，会被

windows 系统自动去掉后面的点和空格，但要注意 Unix/Linux 系统没有这个特性。

4、0x00 截断绕过

- 在扩展名检测这一块目前我只遇到过 asp 的程序有这种漏洞，给个简单的伪代码

```
 name = getname(http request) //假如这时候获取到的文件名是test.asp .jpg(asp 后面为
0x00)

 type = gettype(name) //而在gettype()函数里处理方式是从后往前扫描扩展名，所以判断为jpg

 if (type == jpg)

    SaveFileToPath(UploadPath.name, name) //但在这里却是以0x00 作为文件名截断
 //最后以test.asp 存入路径里
```

5、.htaccess 文件攻击

- 配合名单列表绕过，上传一个自定义的.htaccess，就可以轻松绕过各种检测

6、解析调用/漏洞绕过

- 这类漏洞直接配合上传一个代码注入过的非黑名单文件即可，再利用解析调用/漏洞

绕过白名单检测

1、0x00 截断绕过

- 用像test.asp%00.jpg 的方式进行截断，属于白名单文件，再利用服务端代码的检测逻辑漏洞进行攻击，目前我只遇到过asp 的程序有这种漏洞

2、解析调用/漏洞绕过

- 这类漏洞直接配合上传一个代码注入过的白名单文件即可，再利用解析调用/漏洞

3、详解.htaccess文件攻击

一般留后门还可以，但想通过修改这个文件突破上传是不太现实的，因为第一次拿shell是没有权限修改的

3.1 简介

- 无论是黑名单还是白名单
- 再直接点就是直接攻击.htaccess 文件
- (其实目前我只见过结合黑名单攻击的，在后面的攻击分类里，我会把它归到黑名单绕过攻击里。但网上是把这个单独分类出来的，可能别人有一些我不知道的方式和技巧吧，所以在这里我也暂时保留这个单独分类)

在PHP manual 中提到了下面一段话

```
move_uploaded_file section, there is a warning which states
‘If the destination file already exists, it will be overwritten.’
如果PHP 安全没配置好
就可以通过move_uploaded_file 函数把自己写的.htaccess 文件覆盖掉服务器上的
这样就能任意定义解析名单了
```

3.2 测试：

通过一个.htaccess 文件调用php 的解析器去解析一个文件名中只要包含"haha"这个字符串的任意文件，所以无论文件名是什么样子，只要包含"haha"这个字符串，都可以被以php 的方式来解析，是不是相当邪恶，一个自定义的.htaccess 文件就可以以各种各样的方式去绕过很多上传验证机制

- 建一个 .htaccess 文件，里面的内容如下

```
<FilesMatch "haha">
SetHandler application/x-httpd-php
</FilesMatch>
```

- 同目录有个我们上传一个只有文件名并包含字符串"haha"，但是却无任何扩展名的文件里面的内容是php 一句话木马
- 然后我们用中国菜刀去连接测试，结果如我们预期的一样

- 所以一个可以由 hacker 掌控的 .htaccess 文件是非常邪恶的，基本上可以秒杀各种市面上的上传验证检测 (内容检测除外)。
- 从实际环境来说，我个人接触过的，一般是配合黑名单攻击。比如黑名单里有漏网之鱼，不够完整，漏掉了 .htaccess 扩展名

服务端文件内容检测（检测内容是否合法或含有恶意代码）

如果文件内容检测设置得比较严格，那么上传攻击将变得非常困难，也可以说它是在代码层检测的最后一道关卡，如果它被突破了，就算没有代码层的漏洞，也给后面利用应用层的解析漏洞带来了机会

1、绕过检测文件头

- 主要是检测文件内容开始处的文件幻数，比如图片类型的文件幻数如下

要绕过jpg 文件幻数检测就要在文件开头写上下图的值

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	FF	D8	FF	E0	00	10	4A	46	49	46	00	01	01	00	00	01	ÿØÿà JFIF

Value = FF D8 FF E0 00 10 4A 46 49 46

要绕过gif 文件幻数检测就要在文件开头写上下图的值

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	47	49	46	38	39	61	0A	00	0A	00	D5	00	00	00	00	00	GIF89a

Value = 47 49 46 38 39 61

要绕过png 文件幻数检测就要在文件开头写上下面的值

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	89	50	4E	47	0D	0A	1A	0A	00	00	00	0D	49	48	44	52	‰PNG

2、文件相关信息检测

- 图像文件相关信息检测常用的就是getimagesize()函数

只需要把文件头部分伪造好就ok 了，就是在幻数的基础上还加了一些文件信息

有点像下面的结构

```
GIF89a
(...some binary data for image...)
<?php phpinfo(); ?>
(... skipping the rest of binary data ...)
```

3、文件加载检测

简介

- 这个是最变态的检测了，一般是调用API 或函数去进行文件加载测试常见的是图像渲染测试，再变态点的甚至是进行二次渲染(后面会提到)
- 对渲染/加载测试的攻击方式是代码注入绕过，对二次渲染的攻击方式是攻击文件加载器自身
- 对渲染/加载测试攻击- 代码注入绕过：

  可以用图像处理软件对一张图片进行代码注入，用winhex 看数据可以分析出这类工具的原理是
- 在不破坏文件本身的渲染情况下找一个空白区进行填充代码，一般会是图片的注释区

  对于渲染测试基本上都能绕过，毕竟本身的文件结构是完整的

但如果碰到变态的二次渲染
基本上就没法绕过了，估计就只能对文件加载器进行攻击了

4、绕过二次渲染

什么是二次渲染

- 虽然上传上去一个图片马，在进行数据库备份拿shell时也能成功，但这个图片里的代码不能执行，为什么?
- 是因为服务器把你上传的图片里的代码给去掉又重新渲染一遍。

  这就是二次渲染，只保留正常图片的内容，把图片之外的代码都给去掉了。

怎么看有没有进行二次渲染。

上传一个图片马，然后下载到本地，用winhex查看图片里面的代码还在不在。

绕过方式：

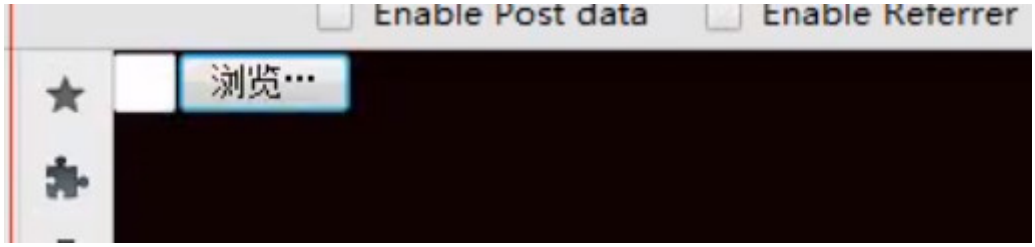
- 攻击二次渲染函数本身

  通过上传不完整的图片让其渲染函数暴漏，然后攻击

  第二种方法：对文件加载器进行溢出攻击

表单提交按钮

一些网站有上传点却没有提交按钮



这时候可以打开firebug审查元素，然后编写一个提交按钮

写入表单

```
<input type="submit" value="提交" name="bb">
```

例如`slblog.upload.com/sleditor/upload.asp`