php的<?php ?>,<?= ?>标签可以省略末尾?>闭合签，例：<?php `ls`;

system('ls')();同system('ls');

PHP脚本使用 session_start()时开启`session`会话，会自动检测`PHPSESSID`

```
有PHPsSSS则查看session文件，无则随机生成
```

没有定义的变量默认为NULL

```
ffifdyop进行md5加密后会变成'or'6<trash>,<trash>为不可见字符，在mysql中会把可见字符丢掉，有时可y
```

在 URL 编码中，加号（+）是一个特殊字符，代表一个空格。所以，show+source 在 URL 中实际上代表 show source

```
当 PHP 解析 URL 中的查询字符串（query string）时，它会将加号（+）解析为一个空格。所以，show+source 会被 PHP 解析为 show source。然后，PHP 会将这个字符串转换为一个变量名，去掉空格，所以 show source 被转换为  show_source。

这就是为什么 show+source 会被解析为 show_source 的原因
```

php会把请求参数中的非法字符转为下划线

```
如：NI+SA+=txw4ever => NI_SA_=txw4ever

还有[]
```

反斜杠绕过

```
反斜杠 \ 代表“全局命名空间”
在 PHP 中，\ 就像是一个绝对路径的起点：
create_function()：相对路径。PHP 会先在当前的命名空间（如果有的话）找这个函数。如果没找到，再回全局找。
\create_function()：绝对路径。直接告诉 PHP：“去全局命名空间（根部）找这个函数，不要在当前的命名空间里乱转

例：
namespace MyApp;

// 1. 定义一个和系统同名的函数
function strlen($str) {
    return 100;
}
// 2. 调用测试
echo strlen("abc");   // 输出 100 (调用了当前命名空间下的函数)
echo \strlen("abc");  // 输出 3   (调用了 PHP 全局的原生函数)
```

# PHP变量解析

PHP需要将所以参数装换为有效的变量名，因此在解析查询字符串时，会做以下两件事

1.删除空白字符

2.将某些字符串转换为下划线

```
如果waf不允许num变量传递字母，?num=aaaa
可以在num前面加一个空格,? num=aaaa
这样waf就找不到这个变量了，因为这是" num"不是"num",但是php在解析时会先把空格去掉，这样我们的代码还能正常运行且上传了非法字符

/?%20news[id%00=43会转换为Array([news_id]=>42)
```

# 超全局变量

$_GET：返回GET方式接收的关联数组

```
例：
http://example.com/index.php?name=John&num=42
//$_GET数组为
$_GET = [
    'name' => 'John',
    'num' => '42'
];
```

$_POST：接收来自POST的数据，$_POST数组类似$_GET数组

$_REQUEST：接收来自 GET、POST 和 COOKIE 的数据

$_SERVER：服务器和执行环境信息，包含请求包信息，服务器配置，底层连接

```
$_SERVER['REQUEST_URI']：访问页面的url
$_SERVER['PHP_SELF']：当前脚本相对于网站根目录的路径，常用于xss，如果不加过滤直接输出到<form action="...">，攻击者可以构造路径来触发脚本
$_SERVER['QUERY_STRING']：URL 问号 ? 后面的纯字符串
$_SERVER['DOCUMENT_ROOT']：当前运行脚本所在的网站根目录的物理绝对路径（如 /var/www/html）
$_SERVER['HTTP_REFERER']：告诉服务器你是从哪个页面跳过来的
$_SERVER['REMOTE_ADDR']：用户的真实 IP 地址，Web 服务器获取的，很难直接伪造
$_SERVER['HTTP_X_FORWARDED_FOR']：代理服务器发送的 IP，可以伪造
$_SERVER['REQUEST_METHOD']：访问页面的请求方法
$_SERVER['HTTPS']：如果脚本是通过 HTTPS 协议被访问，则被设为一个非空的值（通常是 on）
```

```
$_GET        // HTTP GET 请求参数
$_POST       // HTTP POST 请求参数
$_REQUEST    // GET + POST + COOKIE 数据
$_SESSION    // 会话变量
$_COOKIE     // HTTP Cookies
$_SERVER     // 服务器和执行环境信息
$_ENV        // 环境变量
$_FILES      // 上传文件信息
$GLOBALS     // 包含全部全局变量的数组
```

# 函数

```
intval() 函数通过使用指定的进制 base 转换（默认是十进制）

intval() 不能用于 object，否则会产生 E_NOTICE 错误并返回 1
①intval():可以获取变量的整数值

②int intval ( mixed $var [, int $base = 10 ] )
参数说明：

$var：要转换成 integer 的数量值。
$base：转化所使用的进制。

开头为非字符则返回0
以数字开头读到非数字停止，如：999ad返回999，023da转换为10进制停止
如果 base为空，通过检测 var 的格式来决定使用的进制：

如果字符串包括了 “0x”（或 “0X”）的前缀，使用 16 进制（hex）；
否则，如果字符串以 “0” 开始，使用 8 进制(octal)；
否则，将使用 10 进制（decimal)。

有时可以用科学技术法绕过如，1e3
    低版本中intval(1e3)返回1,intval(1e3+1)返回1

当传入数组洪适对象返回1
```

array_map
格式：array_map(callable $callback, array $array1, array $array2 = ?, array $...);
1.$callback (回调函数):会被应用到数组中的每个元素
2.assert会执行传递给它的PHP代码（当assert.active = On 和 assert.exception = Off）

漏洞利用：

```
   <?php
// 将回调替换为执行 system('ls') 命令
array_map(function($value) {
    system('ls');  // 执行系统命令 `ls`
}, $_REQUEST);
?>
```

is_numeric
用于检测变量是否为数字或数字字符串

绕过：
is_numeric函数对于空字符%00，无论是%00放在前后都可以判断为非数值，而%20空格字符只能放在数值后
如：999%00,%00999,999%20
is_numeric函数读到字母返回判断为非数值
如：999ad，a999

parse_str()  //函数把查询字符串解析到变量中
注释：如果未设置 array 参数，由该函数设置的变量将覆盖已存在的同名变量
例子：<?php
parse_str("name=Peter&age=43");
echo $name.','.$age;
?>
结果：
Peter,43

import_request_variables() 函数将 GET／POST／Cookie 变量导入到全局作用域中。该函数在最新版本的 PHP 中已经不支持

语法：bool import_request_variables ( string $types [, string $prefix ] )

$types：指定需要导入的变量，可以用字母 G、P 和 C 分别表示 GET、POST 和 Cookie，这些字母不区分大小写，所以你可以使用 g 、 p 和 c 的任何组合。POST 包含了通过 POST 方法上传的文件信息。注意这些字母的顺序，当使用 gp 时，POST 变量将使用相同的名字覆盖 GET 变量。任何 GPC 以外的字母都将被忽略。

$prefix： 变量名的前缀，置于所有被导入到全局作用域的变量之前。所以如果你有个名为 userid 的 GET 变量，同时提供了 pref_ 作为前缀，那么你将获得一个名为 $pref_userid 的全局变量

switch //匹配逻辑为case表达式中的值等于switch括号里的值

例子：

```
<?php
if(isset($_GET['param1'])){
    $a = $_GET['param1'];
    switch ($a) {
        case $a>=0:
            echo 0;
            break;
```

```
        case $a>=10:
            echo getenv('FLAG');
            break;
        default:
            echo 2;
            break;
    }
}
//param1=0即可，因为0>=10为false
```

```
strcmp //字符串比较，不能被\0截断
绕过：
在php 8.0之前给strcmp传入数组或对象，则返回NULL或0.php8.0后改TypeError异常

当strcmp传入数字时会被转化成字符串，例：
strcmp(45,'45')：0
strcmp(12,2)：-1
strcmp(4,35)：1
```

```
basename() 会删除文件名开头的非 ASCII 字符和中文
```

```
include  //如果刚开始的是一个错误的文件名，也可以进行目录穿梭

如：
<?php
include "empty.abc/../../../../../../etc/passwd";       //给一个错误的开始
?>
可以直接包含passwd文件
```

```
header //向浏览器发送原始HTTP表头
例：header("location:http://127.0.0.1/")
```

```
parse_url //把一个 URL 字符串“切开”，返回一个关联数组
例子：
<?php
$url="https://www.example.com:8080/path/to/page.php?id=123&s=search#top";
$parts=parse_url($url);
print_r($parts);
?>
结果:
Array{
[scheme] => https       //访问用的协议
[host] => www.example.com       //访问域名或IP
[port] => 8080      //端口好号
[path] => /path/to/page.php     //访问文件的在网站的路径
[query] => id=123&s=search      //?后面部分
[fragment] => top
}
```

# 正则匹配

## PCRE机制：用于正则表达式匹配操作 基本覆盖preg_*函数

1.**回溯攻击（ReDoS）**漏洞：构造恶意输入，触发PCRE引擎进行大量的回溯，导致服务拒绝（DoS）

```
通常发生在使用了.*、(.+)、(a+)+ 等模式
例：
$input = str_repeat('a', 10000);  // 10000个 'a'
preg_match('/(a+)+/', $input);
```

2.**修饰符**：`PCRE` 支持许多修饰符（modifiers），例如：

```
- `i`：不区分大小写
- `m`：多行模式（使 `^` 和 `$` 匹配行的开始和结束，而不是整个字符串的开始和结束）
- `s`：点号（`.`）匹配包括换行符在内的所有字符
- `x`：忽略正则表达式中的空白字符和注释
- `u`：启用 Unicode 字符串支持
```

3.常见的**正则表达式模式**

```
字符匹配
.：匹配任何单个字符（除换行符外）
\d：匹配任何数字，等价于 [0-9]
\w：匹配任何字母数字字符，等价于 [a-zA-Z0-9_]
\s：匹配任何空白字符，包括空格、制表符、换行符等

重复匹配
*：匹配前一个字符零次或多次
+：匹配前一个字符一次或多次
{n}：匹配前一个字符恰好 n 次
{n,}：匹配前一个字符至少 n 次
{n,m}：匹配前一个字符 n 至 m 次

位置匹配
^：匹配字符串的开始
$：匹配字符串的结束
\b：匹配单词边界
\B：匹配非单词边界

选择与可选
|：表示选择，类似于逻辑 "或"
?：表示前一个字符是可选的（匹配零次或一次）
```

```
preg_match  //匹配返回1,否则返回0,传入数组或对x产生 E_WARNING 错误并返回 1
```

```
preg_match('/^php$/im', $a)和preg_match('/^php$/i', $a)    /i表示匹配大小写，/m表示多行匹配
ereg    //同preg_match，但存在NULL截断漏洞
例：ereg ("^[a-zA-Z]+$", $_GET['c'])===FALSE  //^和$表示从头匹配到尾
    按理说$c的值必须只有小写或大写字母，但是传参c=a%00778，用%00截断即可绕过
    因为%00为NULL，在字符串中代表结束符

如果正则表达式没有/m，那只能匹配一行，有时可以用来绕过
例:
$aaa = preg_replace('/^(.*)level(.*)$/', '${1}<!-- filtered -->${2}',
$_GET['aaa']);

    if(preg_match('/pass_the_level_1#/', $aaa)){
        echo "here is level 2";
传参aaa=%0apass_the_level_1%23，换到另行即可绕过，浏览器默认不会把"#"及其后面的内容发送给服务器,所以url编码一下
```

```
extract //将数组展开，键名作为变量名，元素值为变量值
例子：extract($_GET)
    传入?a=1&b=6
    则会生成变量$a和$b值分别为1,6
```

# 编码解码函数

base64_encode：base64编码

base64_decode：base64解码

urlencode：URL编码，将空格编码为+

urldecode：URL解码，可同时解码+和%20为空格

json_encode将PHP数组或对象转换为JSON字符串

```
$data = [
    "status" => "ok",
    "content" => "你好",
    "list" => [1, 2, 3]
];
// 普通编码
echo json_encode($data);
// 输出: {"status":"ok","content":"\u4f60\u517b","list":[1,2,3]}

// 中文不转义编码
echo json_encode($data, JSON_UNESCAPED_UNICODE);
// 输出: {"status":"ok","content":"你好","list":[1,2,3]}
```

json_decode：将JSON字符串转换为PHP数组或变量

```php
$json_str = '{"name":"Tom", "age":18}';

// 方式 A：解码为数组
$arr = json_decode($json_str, true);
echo $arr['name']; // 输出: Tom

// 方式 B：解码为对象
$obj = json_decode($json_str);//同$obj = json_decode($json_str，false);
echo $obj->name;   // 输出: Tom
```

serialize：将任何PHP变量转换为可存储的字符串

unserialize：将字符串恢复原始的PHP变量

hex2bin：将十六进制字符串转换为二进制字符串（原始字节数据）

bin2hex：将二进制字符串转换为十六进制字符串

```php
例：$hex_string="3C3F706870206576616C28245F504F53545B2770617373275D293B3F3E";
echo $binary_string=hex2bin($hex_string);

输出：<?php eval($_POST['pass']);?>
```

pack：根据指定格式，将参数打包成一个二进制字符串

```php
例：$hex_string="3C3F706870206576616C28245F504F53545B2770617373275D293B3F3E";
echo $binary_string=pack('H*',$hex_string);

输出：<?php eval($_POST['pass']);?>
```

# 字符串

字符串连接：'123'.'456'='123456'

strstr：查找第一次出现，并返回从该点到末尾的部分

```
例：echo strstr('this is a book','is');
输出：is is a book
```

str_replace：替换字符，区分大小写

str_ireplace：替换字符，不区分大小写

```php
例：$test="this IS a book";
echo str_replace("is","am",$test),PHP_EOL;
echo str_ireplace("is","am",$test);

输出：
tham IS a book
tham am a book
```

substr：字符截取,针对英文，mb_substr针对中文，UTF-8编码情况下中文3个字节，substr只读一个字节，读中文容易乱码

```
例：$test="this IS a book";
echo substr($test,2,4).PHP_EOL;
echo substr($test,2).PHP_EOL;
echo substr($test,-4).PHP_EOL;
echo substr($test,-4,2).PHP_EOL;

输出：
is I
is IS a book
book
bo
```

strpos() -函数查找字符串在另一字符串中第一次出现的索引，针对英文，mb_strpos针对中文，原因同mb_substr

```
例：echo strpos('this is a book','is');
输出：2
```

implode：数组转字符串

join：数组转字符串

parse_str：将 URL 查询字符串解析为变量

```
例：
$query_string = "name=John&age=25&city=NewYork";
parse_str($query_string);       //转换变量

// 结果：$name = 'John', $age = 25, $city = 'NewYork'
echo $name;  // 输出 John
echo $age;   // 输出 25
echo $city;  // 输出 NewYork

parse_str($query_string, $result);  //转换数组
// 结果：$result = ['name' => 'John', 'age' => 25, 'city' => 'NewYork']
print_r($result);
```

strtolower：全部转小写

strtoupper：全部转大写

stripos() - 查找字符串在另一字符串中第一次出现的索引（不区分大小写）
strripos() - 查找字符串在另一字符串中最后一次出现的索引（不区分大小写）
strrpos() - 查找字符串在另一字符串中最后一次出现的索引（区分大小写）

# 强比较和弱比较

true和false不区分大小写

强比较：1.**类型必须相同** - 不进行自动类型转换

2.**值必须相等** - 相同类型的值完全一致

```
1===true        //false
0===FALSE       //false
NAN==
```

弱比较：只比较值　如，12.0==12　//true

```
字符串和数字：
'12'==12  //true
'12abc'==12 //true
'adm2n'==0  //true

布尔和任意值：              //true除了与0比较都是相等的
'way'==true    //true
'flase'==true  //true
234==true      //true
0==true        //false
0==false       //true

hash与0比较：
设：md5($str)=0e420233178946742799316739797882
md5($str) == '0'    //true
```

+999和999是一样的，但是有时+999可用来绕过

# GET和POST

```
$_GET=&$_POST       //$_GET变量指向$_POST变量所在内存地址
$_GET?$_GET=&$_POST:'flag'; //只要有GET传参，就把get方法变成post方法
```

# 数组

- array_push() - 向数组尾部插入一个或多个元素。

- in_array() - 搜索数组中是否存在指定的值。函数有缺陷，若没有设置第三个参数，则存在强制转换（类比==）比如数组$allow含有1，in_array(1.php, $allow)为真。

# 对象操作

定义：$a=new Object(); $a=new Object;

输出：var_dump($a),print_r($a),json_encode($a),echo new ReflectionClass('ctf');

```
echo new ReflectionClass('ctf');
new ReflectionClass('ctf')创建实例对象，包含ctf类的信息
echo字符转换，触发魔术方法__toString，将RefletionClass对象传换字符串输出
```

# 类

`Exception`：所有异常的基类

`CachingIterator`：此对象支持在另一个迭代器上进行缓存迭代

`ReflectionClass`:反射类

```
1.绕过私有属性
// 假设目标类
class Wrapper
{
    private $doit = false; // 目标是让 $doit 变成 true
    public function __destruct() {
        if ($this->doit === true) {
            eval('echo file_get_contents("flag.txt");');
        }
    }
} // 攻击者构造的辅助代码（如果可以注入或执行）
$obj = new Wrapper();
$rc = new ReflectionClass('Wrapper');
$prop = $rc->getProperty('doit'); // 关键步骤：设置私有属性为可访问，并修改其值 $prop-
>setAccessible(true);  $prop->setValue($obj, true);  // 此时，当 $obj 被销毁时，
__destruct 就会成功执行命令

2,反射类不仅仅可以建立对类的映射，也可以**建立对PHP基本方法的映射**
例：
```

```
echo new ReflectionClass(system('ls'))  //输出命令ls执行结果
echo new ReflectionClass('ctf')      //输出ctf类的所有信息
```

`DirectoryInterator`：遍历目录的类

`FilesystemIterator`：遍历文件的类

# hash

强md5绕过，md5()函数中要是字符串，php中当数组被转成字符串时，无论是什么都会被转成“Array”

```
<?php
include("flag.php");
highlight_file(__FILE__);
if (isset($_POST['a']) and isset($_POST['b'])) {
if ($_POST['a'] != $_POST['b'])
if (md5($_POST['a']) === md5($_POST['b']))
echo $flag;
else
print 'Wrong.';
}
?>//a[]=12&b[]=2
```

sha1：计算字符串的 SHA-1 哈希值，绕过同md5

# 运算符优先级

| 优先级 | 运算符 | 描述 | 结合性 | 示例 |
| --- | --- | --- | --- | --- |
| 1 | `clone` `new` | 对象创建 | 无 | `new MyClass()` |
| 2 | `\**` | 幂运算 | 右 | `2 ** 3 ** 2` = 512 |
| 3 | `++` `--` | 递增/递减 | 无 | `++$a` `$a++` |
| 4 | `~` `(int)` `(float)` 等 | 类型转换/位取反 | 右 | `~(int)$a` |
| 5 | `!` | 逻辑非 | 右 | `!$condition` |
| 6 | `\*` `/` `%` | 乘、除、取模 | 左 | `10 * 3 / 2` |
| 7 | `+` `-` `.` | 加、减、字符串连接 | 左 | `5 + 3 . "str"` |
| 8 | `<<` `>>` | 位左移、右移 | 左 | `$a << 2` |
| 9 | `<` `<=` `>` `>=` | 比较运算符 | 无 | `$a > $b` |
| 10 | `==` `!=` `===` `!==` `<` `<=>` | 相等性比较 | 无 | `$a == $b` |
| 11 | `&` | 按位与、引用 | 左 | `$a & $b` |
| 12 | `^` | 按位异或 | 左 | `$a ^ $b` |
| 13 | `\|` | 按位或 | 左 | `$a \| $b` |
| 14 | `&&` | 逻辑与 | 左 | `$a && $b` |
| 15 | `\|\|` | 逻辑或 | 左 | `$a \|\| $b` |
| 16 | `??` | NULL 合并 | 左 | `$a ?? $b` |
| 17 | `?:` | 三元运算符 | 左 | `$a ? $b : $c` |
| 18 | `=` `+=` `-=` 等 | 赋值运算符 | 右 | `$a = $b = 5` |
| 19 | `and` | 逻辑与（低优先级） | 左 | `$a and $b` |
| 20 | `xor` | 逻辑异或 | 左 | `$a xor $b` |
| 21 | `or` | 逻辑或（低优先级） | 左 | `$a or $b` |