# ctf PHP弱类型认证,回首再看CTF中的那些PHP弱类型

转载

weixin_39719732 于 2021-03-13 00:45:37 发布 46 收藏
文章标签： ctf PHP弱类型认证
原文来自SecIN社区—作者：湿巾不湿

0x00 无话可说的类型转换

众所周知PHP中各数据类型转整数型与取整函数intval()关系密切，intval()函数就是类型转换产生的PHP弱类型问题的关键因素。<?php

var_dump("abc1"==0); //Ture

var_dump("2abc"==2); //Ture

var_dump(intval("123")); //int(123)

var_dump(intval("abc")); //int(0)

var_dump(intval("2abc")); //int(2)

?>

上面的测试代码简单呈现出字符类型转整数型产生问题，除了字符串转整数之外，还有数组转整数：<?php

var_dump(intval(array())); //int(0)

var_dump(intval(array(2,3,4))); //int(1)

var_dump(intval(array('aa','bb','cc'))); //int(1)

?>

另外，还有十六进制：<?php

var_dump(intval('0x3A')); //int(0)

var_dump(intval(0x3A)); //int(58)

?>

0x01 老生常谈的一些函数

strcmp()

我们知道strcmp()函数的功能是比较两个字符串(区分大小写)，如果str1< str2 则返回< 0，如果str1大于str2函数返回>0，如果str1= str2 则函数返回 0。<?php

var_dump(strcmp("str1", "str2")); //int(-1)

var_dump(strcmp("str3", "str2")); //int(1)

var_dump(strcmp("str1", "str1")); //int(0)

var_dump(strcmp(array(123),"str2")); //NULL

?>

但是两个参数中只要有其中一个传入的值为数组，函数就会返回NULL，而NULL又可以利用在PHP松散比较中。看下这个简单的例子：<?php

include('flag.php');

highlight_file(__FILE__);

$password="aaaaaaaa";

if (isset($_GET['password'])){

$pw = $_GET['password'];

if ($pw != $password) {

if (strcmp($pw, $password) == 0){

echo $flag;

}else{

echo "NO,NO,NO";

}

}else{

echo "What are you doing!?";

}

}

?>

该示例中得到flag的途径是给password传值，通过第二个和第三个if的判断。传入数组既可以使第二个if判断成立，又可以造成strcmp()函数返回NULL，进而利用第三个if中的松散比较满足判断条件。payload: ?password[]=xxxxx

另外附上一张PHP官方文档的松散比较图：

松散比较 ==

| | TRUE | FALSE | 1 | 0 | -1 | "1" | "0" | "-1" | NULL | array() | "php" | "" |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TRUE | TRUE | FALSE | TRUE | FALSE | TRUE | TRUE | FALSE | TRUE | FALSE | FALSE | TRUE | FALSE |
| FALSE | FALSE | TRUE | FALSE | TRUE | FALSE | FALSE | TRUE | FALSE | TRUE | TRUE | FALSE | TRUE |
| 1 | TRUE | FALSE | TRUE | FALSE | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| 0 | FALSE | TRUE | FALSE | TRUE | FALSE | FALSE | TRUE | FALSE | TRUE | FALSE | TRUE | TRUE |
| -1 | TRUE | FALSE | FALSE | FALSE | TRUE | FALSE | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE |
| "1" | TRUE | FALSE | TRUE | FALSE | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| "0" | FALSE | TRUE | FALSE | TRUE | FALSE | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE | FALSE |
| "-1" | TRUE | FALSE | FALSE | FALSE | TRUE | FALSE | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE |
| NULL | FALSE | TRUE | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE | FALSE | TRUE |
| array() | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE | TRUE | FALSE | FALSE |
| "php" | TRUE | FALSE | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE | FALSE |
| "" | FALSE | TRUE | FALSE | TRUE | FALSE | FALSE | FALSE | FALSE | TRUE | FALSE | FALSE | TRUE |

in_array()

in_array()的松散性基本可以理解为“==”。<?php

var_dump(in_array('abc', array(0,1,2))); //bool(true)

var_dump(in_array('abc', array('0',1,2))); //bool(false)

var_dump(in_array('1abc', array(0,1,2))); //bool(ture)

?>

array_search()

array_search()的问题与in_array()一样，皆会对类型进行强制转换。绕过同理。

之前看Mrsm1th师傅的博客时见过一道这样的题目：<?php

if(!is_array($_GET['test'])){exit();}

$test=$_GET['test'];

for($i=0;$i

if($test[$i]==="admin"){

echo "error";

exit();

}

$test[$i]=intval($test[$i]);

}

if(array_search("admin",$test)===0){

echo "flag";

}

else{

echo "false";

}

?>

三个if条件很是苛刻，前两个if分别要求参数test传入的值必须是数组且数组内不能有“admin”，然后第三个条件就要求通过array_search(“admin”,$test)判断。

而我们知道，array_search()与in_array()一样，会类型进行强制转换，那么当我们传入test[]=0
时，array_search("admin",$test)中的判断就相当于"admin"==0，最终等式成立返回匹配成功的数组元素的下标0，满足“===”，得到flag。

switch()

switch()函数常用作条件选择，但函数内的参数与case的类型不同时也会进行类型转换。<?php

include('flag.php');

```
highlight_file(__FILE__);

$password = $_GET['password'];

if ($password != 1) {

switch ($password) {

case 0:

echo "green,green";

break;

case 1:

echo $flag;

break;

default:

echo "awsl";

break;

}

}else{

echo "NO";

}

?>
```

上示例的绕过方式还是一样原理，传入?password=1abc经过类型转换与case 1匹配。

0x02 那些年我们一起绕过的MD5

最近的比赛似乎常有PHP弱类型和md5组合的绕过题目，什么$a==md5($a);啊，什么$a!=$b;md5($a)==md5($b);啊，诸如此类，绕过方法除了利用数组外就是利用0e215962017、QNKCDZO、s878926199a这些md5加密后的值为0e\d+(0e开头，0e后全为数字)的字符串。

原理也很简单，就是当php在经行==松散比较时，在等号前后的值都为0e\d+时，就会判断为科学计数法0的n次方，结果都为0，等式成立。如下例：

```
<?php

var_dump(md5('QNKCDZO')); //string(32) "0e830400451993494058024219903391"

echo '
';

var_dump(md5('QNKCDZO')==0); // bool(true)

?>
```

简单的0e绕过

md5加密后值为0e\d+的字符串：

```
<?php

$a = 'QNKCDZO';
```

```
$b = 's878926199a';

$c = 's214587387a';

$d = '0e215962017';

var_dump(md5($a)); //string(32) "0e830400451993494058024219903391"

echo '
';

var_dump(md5($b)); //string(32) "0e545993274517709034328855841020"

echo '
';

var_dump(md5($c)); //string(32) "0e848240448830537924465865611904"

echo '
';

var_dump(md5($d)); //string(32) "0e291242476940776845150308577824"

?>
```

看下面这个简单例子：

```
<?php

include('flag.php');

highlight_file(__FILE__);

if ($_GET['a']==md5($_GET['a'])) {

echo "$flag";

}

?>
```

为了得到flag需要满足传入的值与其自身的MD5值松散比较相等，我们只需要传入一个0e\d+并且MD5加密后仍然是0e\d+的字符串，使得在进行松散比较时两边的值都被解析为零的n次方即可。传入0e215962017。

常规数组绕过

数组绕过利用的是PHP中的md5()函数的其中一个特性，就是当给md5()传参为数组时会返回NULL：<?php

$a = array('aaa','ccc');

$b = array(1,2,3);

var_dump(md5($a)); // NULL

var_dump(md5($b)); // NULL

?>

而NULL在PHP松散比较中利用简直不要太爽。再看下这个利用特性满足严比较的例子：<?php

include('flag.php');

highlight_file(__FILE__);

if ($_GET['a']!==$_GET['b'] && md5($_GET['a'])===md5($_GET['b'])) {

echo "$flag";

}else{

echo "NONONO";

}

?>

利用php中md5()函数传入数组后返回NULL的特性，我们仅需传入两个不同的数组使其md5()加密后等式两边都为NULL，两个严比较同时成立输出flag。


强类型绕过

所谓MD5强类型绕过，其实就是MD5强碰撞产生的异类。

先拿上个例子来说，假如修改题目源码后无法再通过传入数组绕过严比较，我们该怎么办？比如这样：<?php

include('flag.php');

highlight_file(__FILE__);

if ((string)$_GET['a']!==(string)$_GET['b'] && md5($_GET['a'])===md5($_GET['b'])) {

echo "$flag";

}else{

echo "NONONO";

}

?>

在严比较两边GET方法传参的位置加了类型强制转换，此时我们再传入数组会发现失败了：


先看一下测试的代码：<?php

highlight_file(__FILE__);

var_dump((string)$_GET['a']);

echo '
';

var_dump((string)$_GET['b']);

echo '
';

var_dump(md5((string)$_GET['a']));

echo '
';

var_dump(md5((string)$_GET['b']));

echo '
';

if ((string)$_GET['a']!==(string)$_GET['b']) {

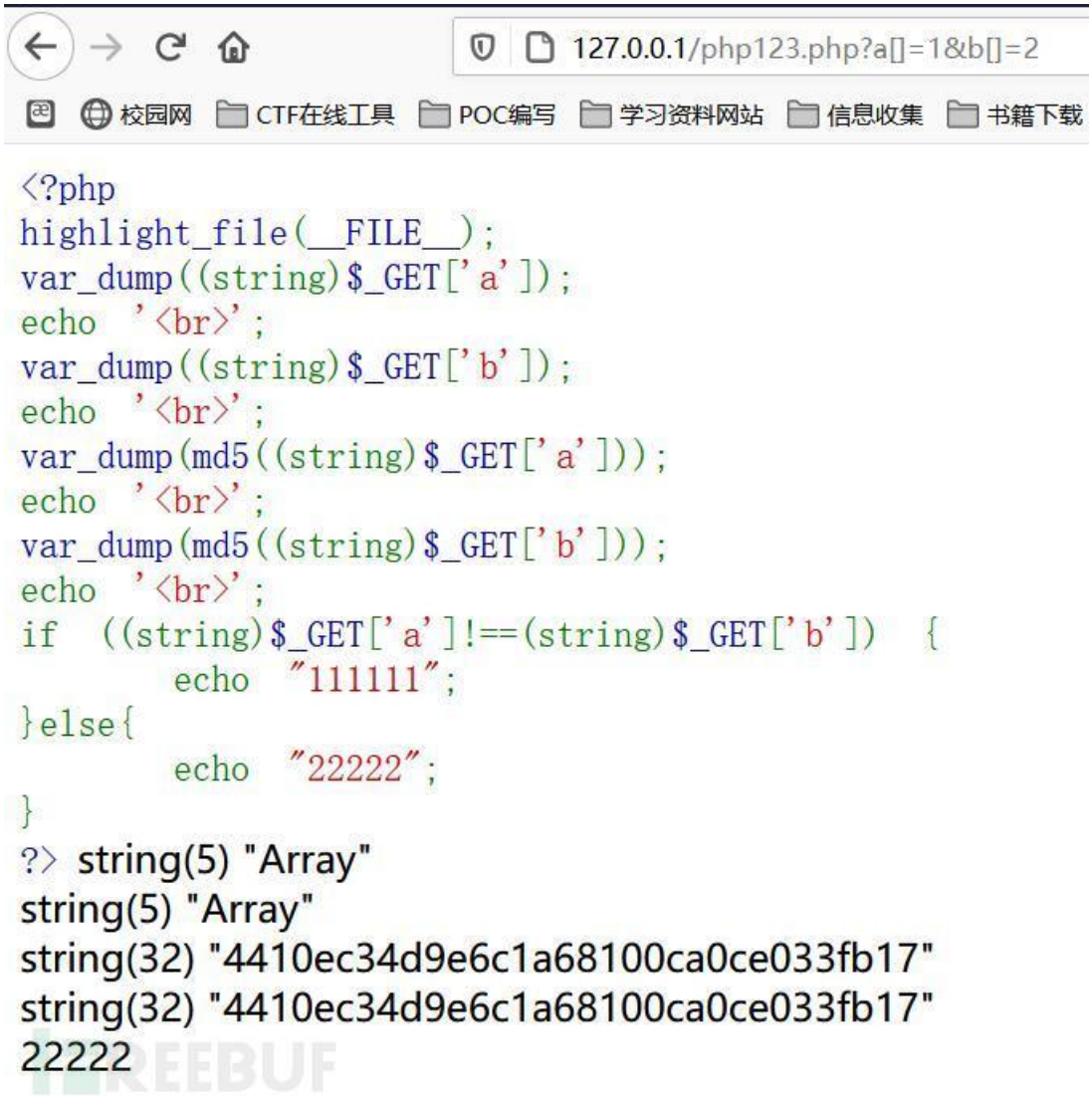
echo "111111";

}else{

echo "22222";

}

?>



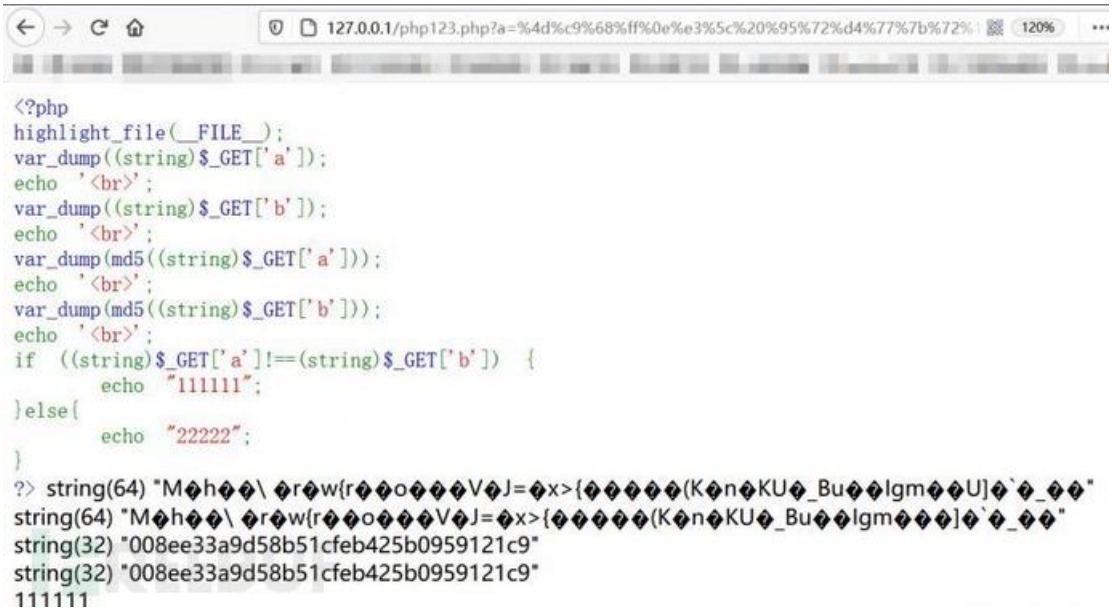

通过GET方法传入的数组在经过(string)类型转后后都变成了字符串“Array”，所以自然无法满足(string)$_GET['a']!==(string)$_GET['b']这个条件。

那么，我们应该怎样才能同时满足(string)$_GET['a']!==(string)$_GET['b']和md5($_GET['a'])===md5($_GET['b'])这两个严判断呢？


嗯，就是强类型绕过。第一次见的时候很懵，是19年安洵杯的一道题，BUU链接：[安洵杯 2019]easy_web，参考该题的Writeup:$data_1=%4d%c9%68%ff%0e%e3%5c%20%95%72%d4%77%7b%72%15%87%d3%6f%a7%b2%1b

$data_2=%4d%c9%68%ff%0e%e3%5c%20%95%72%d4%77%7b%72%15%87%d3%6f%a7%b2%1b%dc%5

这一对payload并不完全一样，是MD5强碰撞产生的异类。。。。直接测试：




通过测试输出的结果，我们可以很直观的看到这对payload经过MD5加密后的值一模一样，同时还满足(string)$_GET['a']!==(string)$_GET['b']

成功绕过强类型严比较：


0x03 结语

本篇参考：

https://www.cnblogs.com/Mrsm1th/p/6745532.html

https://www.cnblogs.com/wangtanzhi/p/12244096.html

如若文中有哪里分析的不够到位，还请海涵并指出错误。