PHP代码安全杂谈

转载

代码技巧 于 2018-02-21 00:00:00 发布 787 收藏 2

小提示 点击上方蓝色字体即可一键关注哦~

> 来自：FreeBuf.COM，作者：国光
>
> http://www.freebuf.com/articles/rookie/161474.html

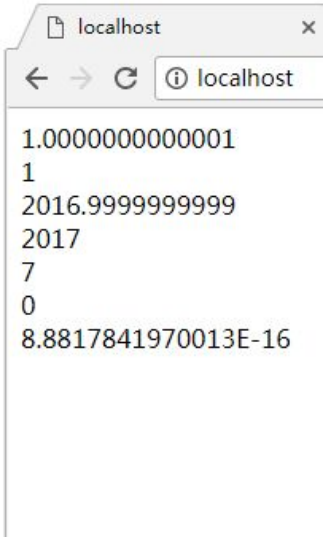
虽然PHP是世界上最好的语言,但是也有一些因为弱类型语言的安全性问题出现。WordPress历史上就出现过由于PHP本身的缺陷而造成的一些安全性问题，如CVE-2014-0166 中的cookie伪造就是利用了PHP Hash比较的缺陷。当然一般这种情况实战中用到的不是很多，但是在CTF竞赛中却是一个值得去考察的一个知识点，特此记录总结之。

一、精度绕过缺陷

理论

在用PHP进行浮点数的运算中,经常会出现一些和预期结果不一样的值，这是由于浮点数的精度有限。尽管取决于系统，PHP 通常使用 `IEEE 754` 双精度格式，则由于取整而导致的最大相对误差为 `1.11e-16` 。非基本数学运算可能会给出更大误差，并且要考虑到进行复合运算时的误差传递。下面看一个有趣的例子:

```php
<?php
    echo(1.0000000000001); //13位小数
    echo"<br>";
    echo(1.00000000000001);//15位小数
    echo"<br>";
    echo(2016.9999999999); //10位小数
    echo"<br>";
    echo(2016.99999999999);//11位小数
    echo"<br>";
    echo floor((0.1+0.7)*10);
    echo"<br>";
    echo((0.1+0.7)-0.7999999999999999);
    echo"<br>";
    echo((0.1+0.7)-0.799999999999999);
?>
```

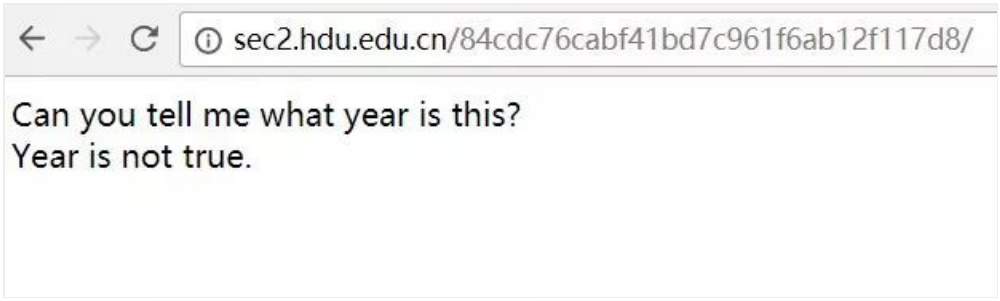


以十进制能够精确表示的有理数如 `0.1` 或 `0.7`，无论有多少尾数都不能被内部所使用的二进制精确表示，因此不能在不丢失一点点精度的情况下转换为二进制的格式。这就会造成混乱的结果：例如，`floor((0.1+0.7)*10)` 通常会返回 `7` 而不是预期中的 `8`，因为该结果内部的表示其实是类似 `7.9999999999999991118...` 。

实践

问鼎杯2017 老眼昏花网上很多write-up感觉就像是看着答案写write-up，个人感觉真正的write-up中应该体现自己的思考在里面。

题目描述



题目言简意赅，让我们把 `2017` 这个值传递给服务器。

考察点

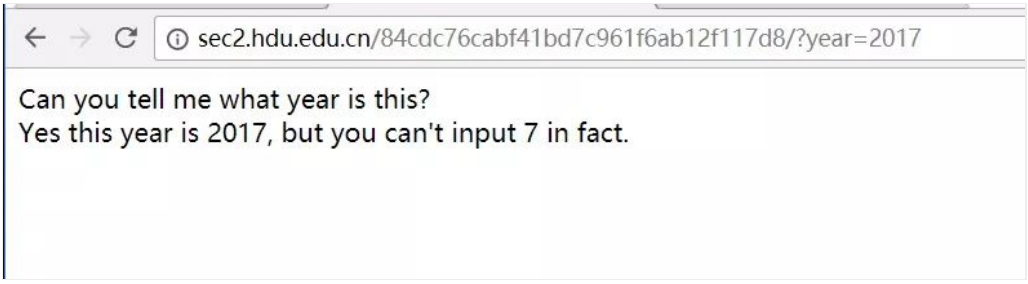
PHP浮点精确度

write-up

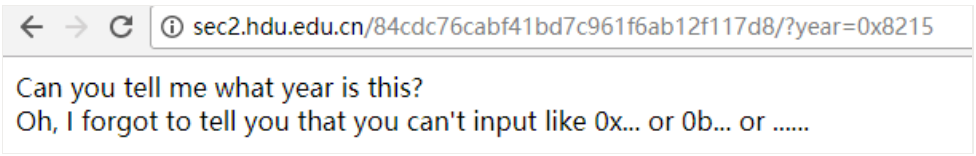
`what year is this?` 所以第一反应是直接给 `year` 参数赋值为 `2017`：

```
?year=2017
```

然而结果如下：



有提示了，说明 year 这个参数是对的，但是 2017 中不可以出现 7 ，这里如果不了解php精度的话，肯定是对 2017 进行各种编码绕过,但是这里对编码也进行过滤了：



所以最后一种可能就是利用PHP精度来绕过:

```
?year=2016.99999999999
```

二、类型转换的缺陷

理论

PHP提供了 is_numeric 函数，用来变量判断是否为数字。PHP弱类型语言的一个特性，当一个整形和一个其他类型行比较的时候，会先把其他类型intval数字化再比。

实践

is_numeric() 用于判断是否是数字，通常配合数值判断。

案例代码

```php
<?php
    error_reporting(0);
    $flag = 'flag{1S_numer1c_Not_S4fe}';
    $id = $_GET['id'];
    is_numeric($id)?die("Sorry...."):NULL;
    if($id>665){
        echo $flag;
    }
?>
```

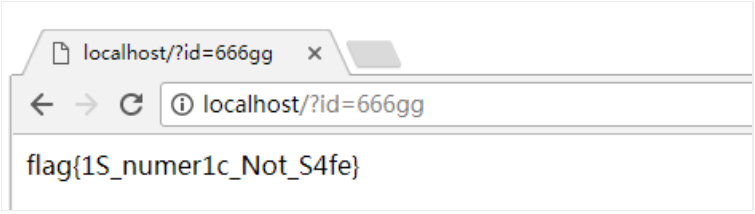
考察点

PHP类型转换缺陷

write-up

分析下代码:首先对GET方式提交的参数 id 的值进行检验。 id 通过 is_numeric 函数来判断是否为数字，如果为数字的话，GG。如果不是数字的话，和 665 进行比较， id 的值大于 665 的时候输出 flag 。
乍看上去又好像不可能这里，但是如果知道 PHP弱类型语言的一个特性，当一个整形和一个其他类型行比较的时候，会先把其他类型intval数字化再比 。这个特性的话就可以很好的绕过。

```
http://localhost/?id=666gg
```



三、松散比较符的缺陷

理论

php比较相等性的运算符有两种，一种是严格比较，另一种是松散比较。

> 如果比较一个数字和字符串或者比较涉及到数字内容的字符串，则字符串会被转换成数值并且比较按照数值来进行

比较运算符

例子	名称	结果
$a == $b	等于	**TRUE**，如果类型转换后 *$a* 等于 *$b*。
$a === $b	全等	**TRUE**，如果 *$a* 等于 *$b*，并且它们的类型也相同。
$a != $b	不等	**TRUE**，如果类型转换后 *$a* 不等于 *$b*。
$a <> $b	不等	**TRUE**，如果类型转换后 *$a* 不等于 *$b*。
$a !== $b	不全等	**TRUE**，如果 *$a* 不等于 *$b*，或者它们的类型不同。
$a < $b	小与	**TRUE**，如果 *$a* 严格小于 *$b*。
$a > $b	大于	**TRUE**，如果 *$a* 严格大于 *$b*。
$a <= $b	小于等于	**TRUE**，如果 *$a* 小于或者等于 *$b*。
$a >= $b	大于等于	**TRUE**，如果 *$a* 大于或者等于 *$b*。
$a <=> $b	太空船运算符（组合比较符）	当*$a*小于、等于、大于*$b*时 分别返回一个小于、等于、大于0的integer 值。PHP7开始提供.
$a ?? $b ?? $c	NULL 合并操作符	从左往右第一个存在且不为 **NULL** 的操作数。如果都没有定义且不为 **NULL**，则返回 **NULL**。PHP7开始提供。

严格比较符严格比较符，会先判断两种字符串的类型是否相等，再比较。

```
===  //全等
!==  //不全等
```

严格比较 ===

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

松散比较符松散比较符，会先将字符串类型转换成相同，再比较。

```
==  //等于
!=  //不等
```

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

PHP 会根据变量的值，自动把变量转换为正确的数据类型。这一点和C 和 C++ 以及 Java 之类的语言明显不同。虽然这样PHP方便了程序员，但是随之而来却会带来一些安全性的问题。

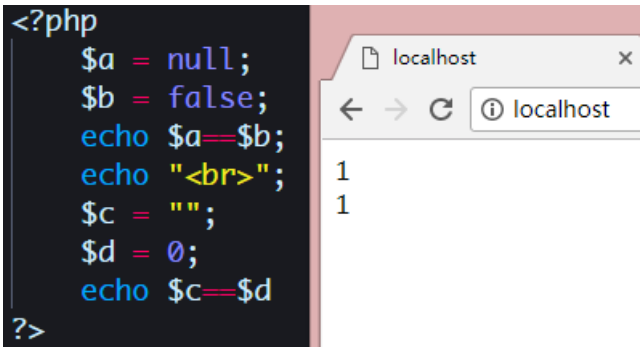
一个简单的例子

```php
<?php
    $a = null;
    $b = false;
    echo $a==$b;
    echo "<br>";
    $c = "";
    $d = 0;
    echo $c==$d
?>
```

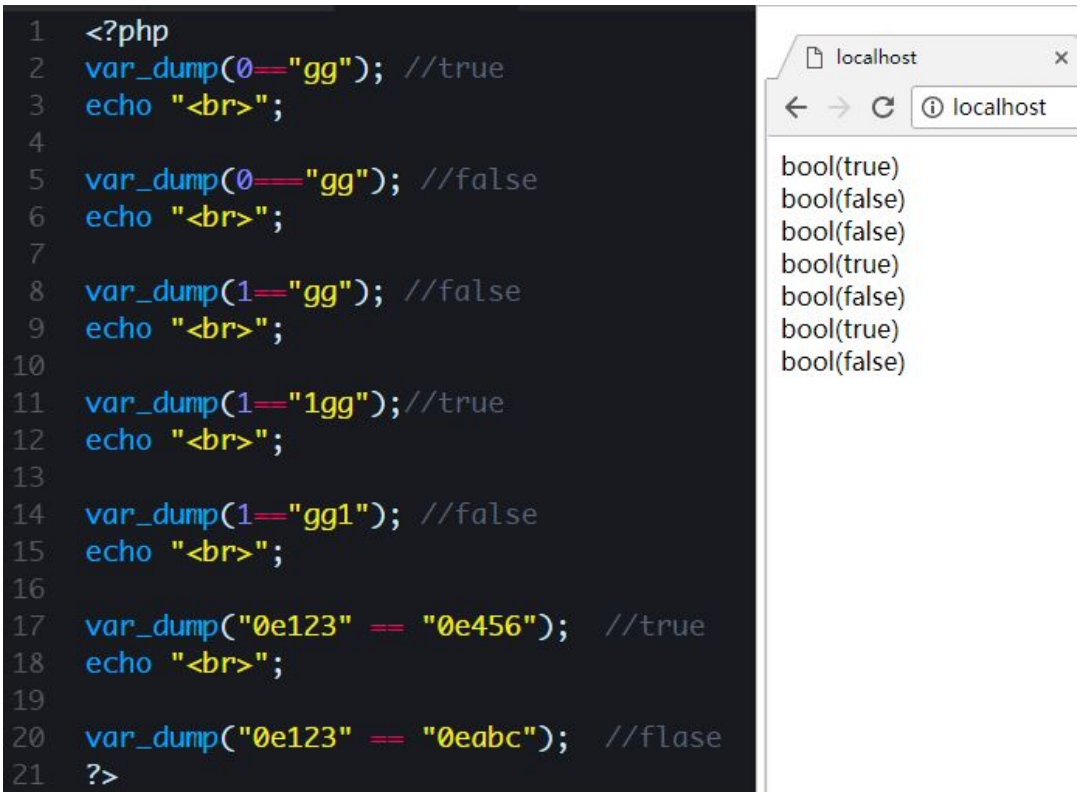
由于php对变量自动转换的特性，这里面的

```
$a==$b 与 $c==$d 均为真
```

所以页面输出的结果为:



一个深入的例子



下面结合PHP 相等性比较缺陷再解释下会好懂一点:

```
var_dump(0=="gg");  //true
var_dump(0==="gg"); //false
var_dump(1=="gg");  //false
```

> `0` 与 `gg` 进行松散性质的不严格比较，会将 `gg` 转换为数值，强制转换，由于 `gg` 是字符串，转化的结果是 `0` ，所以 输出 `true`
>
> `0` 与 `gg` 进行严格 性质的严格比较，这里的 `gg` 是字符串类型，和int类型的 `0` 不相等，所以输出 `false`
>
> `0` 与 `gg` 进行松散性质的不严格比较，会将 `gg` 转换为数值，强制转换，由于 `gg` 是字符串，转化的结果是 `0` ，不等于 `1` ，所以输出 `false`

```
var_dump(1=="1gg"); //true
var_dump(1=="gg1"); //false
```

> `1` 与 `1gg` 进行松散性质的不严格比较，这里 `1gg` 被强制转换为int类型的时候会从字符串的第一位开始做判断进行转换，这里的 `1gg` 第一位是 `1` ，所以这里 `1gg` 被转换为 `1` ，所以输出 `true`
>
> `1` 与 `gg1` 进行严格 性质的严格比较，字符串 `gg1` 的第一位不是数字，所以它被强制转换为 `0` ，所以输出 `false`

```
var_dump("0e123" == "0e456");  //true
var_dump("0e123" == "0eabc");  //flase
```

这里比较特殊，字符串中出现了 `0e` ，PHP手册介绍如下:

当一个字符串欲当作一个数值来取值，其结果和类型如下:如果该字符串没有包含'.','e','E'并且其数值值在整形的范围之内该字符串被当作int来取值，其他所有情况下都被作为float来取值， 该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，否则其值为0。

实践

md5绕过(Hash比较缺陷)南京邮电大学网络攻防训练平台中一道比较经典的 `md5 collision` 题，关于这道题目的WriteUp网上很多，但是真正深入分析的少之又少~~

题目描述

md5 collision**源码**

```php
<?php
    $md51 = md5('QNKCDZO');
    $a = @$_GET['a'];
    $md52 = @md5($a);
    if(isset($a)){
        if ($a != 'QNKCDZO' && $md51 == $md52) {
            echo "nctf{*****************}";
        } else {
            echo "false!!!";
        }
    }
    else{
        echo "please input a";
    }
?>
```

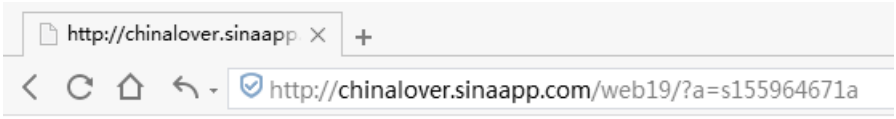
考察点

简单的PHP代码审计

PHP弱类型的Hash比较缺陷

write-up

从源码中可以得输入一个a的参数的变量，a首先不等于 `QNKCDZO` 并且a得md5值必须等于 `QNKCDZO` 加密后的md5值。乍一看好像不可能存在这样的值，但是这里 `QNKCDZO` 加密后的md5值为 `0e830400451993494058024219903391` 这里是 `0e` 开头的，在进行等于比较的时候，PHP把它当作科学计数法，0的无论多少次方都是零。 所以这里利用上面的弱类型的比较的缺陷来进行解题： `?a=s155964671a`



nctf{md5_collision_is_easy}

字符串加密后 md5 为0exxxx的字符串(x必须是10进制数字)列表

| 字符串
| md5
|
| —- | —- |
| QNKCDZO
| 0e830400451993494058024219903391
|
| 240610708
| 0e462097431906509019562988736854
|
aabg7XSs	0e087386482136013740957780965295
aabC9RqS	0e041022518165728065344349536299
s878926199a	0e545993274517709034328855841020

四、sha1() md5()加密函数漏洞缺陷

理论

`md5()` 和 `sha1()` 对一个数组进行加密将返回 NULL

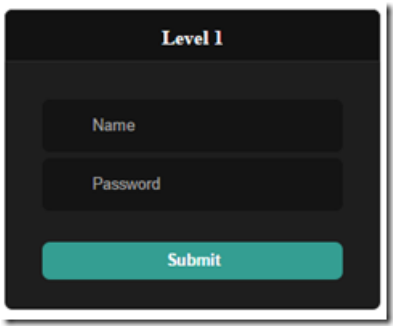
实践

Boston Key Party CTF 2015: Prudential

题目描述

> I dont think sha1 isbroken.Prove me wrong.

题目给了一个登陆框:



考察点

sha1()函数漏洞缺陷

write-up

源代码给出如下:

```
<html>
<head>
    <title>level1</title>
    <link rel='stylesheet' href='style.css' type='text/css'>
</head>
<body>

<?php
require 'flag.php';
if (isset($_GET['name']) and isset($_GET['password'])) {
    if ($_GET['name'] == $_GET['password'])
        print 'Your password can not be your name.';
    else if (sha1($_GET['name']) === sha1($_GET['password']))
      die('Flag: '.$flag);
    else
        print '<p class="alert">Invalid password.</p>';
}
?>

<section class="login">
    <div class="title">
        <a href="./index.txt">Level 1</a>
    </div>

    <form method="get">
        <input type="text" required name="name" placeholder="Name"/><br/>
        <input type="text" required name="password" placeholder="Password" /><br/>
        <input type="submit"/>
    </form>
</section>
</body>
</html>
```

分析一下核心登录代码如下:

```
if ($_GET['name'] == $_GET['password'])
    print 'Your password can not be your name.';
else if (sha1($_GET['name']) === sha1($_GET['password']))
    die('Flag: '.$flag);
```

`GET` 类型提交了两个字段 `name` 和 `password` ，获得flag要求的条件是：

name != password

sha1(name) == sha1(password)

这个乍看起来这是不可能的，但是这里利用 `sha1()` 函数在处理数组的时候由于无法处理将返回 `NULL` 可以绕过if语句的验证，if条件成立将获得 `flag` 。 构造语句如下:

```
?name[]=a&password[]=b
```

这里符合了2个拿到flag的条件：

a不等于b

name和password由于是数组，经过sha1()函数嫁给后都返回 `NULL`

拿到flag： `I_think_that_I_just_broke_sha1`

拓展总结

经过验证，不仅 `sha1()` 函数无法处理数组，这里 `md5()` 函数也有同样的问题，在处理数组的时候，都将返回 `NULL`
测试代码如下:

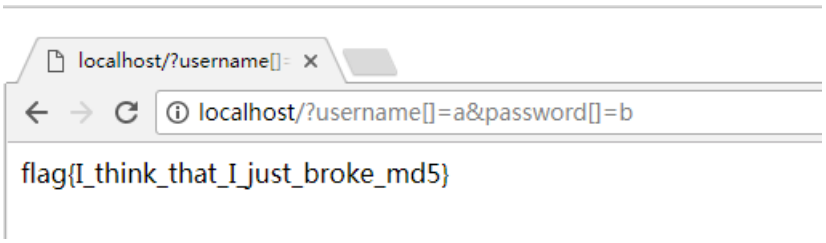
```php
<?php
error_reporting(0);
$flag = 'flag{I_think_that_I_just_broke_md5}';
if (isset($_GET['username']) and isset($_GET['password'])) {
    if ($_GET['username'] == $_GET['password'])
        print 'Your password can not be your username.';
    else if (md5($_GET['username']) === sha1($_GET['password']))
        die($flag);
    else
        print 'Invalid password';
}
?>
```

这里面的核心代码如下:

```
if ($_GET['username'] == $_GET['password'])
并且得满足:
if (md5($_GET['username']) === sha1($_GET['password']))
```

同样利用 `md5()` 函数无法处理数组的这个漏洞，构造get请求拿到flag:

> ?username[]=a&password[]=b



五、字符串处理函数漏洞缺陷

理论

`strcmp()` 函数:比较两个字符串（区分大小写).

用法如下:

```
int strcmp ( string $str1 , string $str2 )
```

具体的用法解释如下:

```
参数 `str1`第一个字符串。
参数 `str2`第二个字符串。
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。
```

这个函数接受到了不符合的类型，例如 `数组` 类型,函数将发生错误。但是在 `5.3` 之前的php中，显示了报错的警告信息后，将 `return 0` !!!! 也就是虽然报了错，但却判定其相等了。

`ereg()` 函数：字符串正则匹配。

`strpos()` 函数：查找字符串在另一字符串中第一次出现的位置，对大小写敏感。

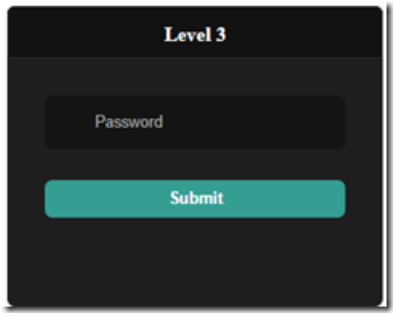
这2个函数都是用来处理字符串的，但是在传入数组参数后都将返回 `NULL` 。

实践

Boston Key Party CTF 2015: Northeastern Univ

题目描述

Of course, a timing attack might be the answer, but Im quite sure that you can do better than that. 题目给了一个登陆框:



考察点

字符串处理函数漏洞缺陷

write-up

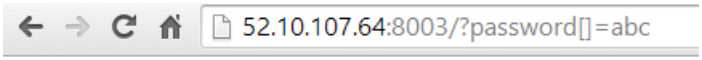
给出源代码如下:

```
<html>
<head>
    <title>level3</title>
    <link rel='stylesheet' href='style.css' type='text/css'>
</head>
<body>

<?php
require 'flag.php';

if (isset($_GET['password'])) {
    if (strcmp($_GET['password'], $flag) == 0)
        die('Flag: '.$flag);
    else
        print '<p class="alert">Invalid password.</p>';
}
?>

<section class="login">
        <div class="title">
                <a href="./index.txt">Level 3</a>
        </div>

        <form method="get">
                <input type="text" required name="password" placeholder="Password" /><br/>
                <input type="submit"/>
        </form>
</section>
</body>
</html>
```

分析一下核心登录代码如下:

```
if (strcmp($_GET['password'], $flag) == 0)
```

这里使用了 `==` 松散比较了 `$flag` 和通过GET方式提交的 `password` 的值，如果想等的话，拿到flag。这里用的是 `==` 松散性质的比较，再利用字符串处理数组时将会报错，在 `5.3` 之前的php中，显示了报错的警告信息后，将 `return 0` 。所有这里将 `password` 参数指定为数组,利用函数漏洞拿到 `flag` :



Flag: Still_better_than_the_d0uble_equals

拓展总结

除了 `strcmp()` 函数外， `ereg()` 和 `strpos()` 函数在处理数组的时候也会异常，返回 `NULL` 。测试代码如下:

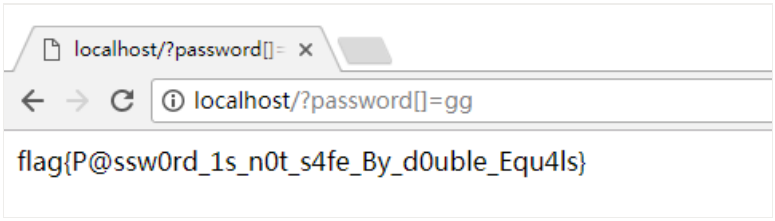
```
<?php
    error_reporting(0);
    $flag = 'flag{P@ssw0rd_1s_n0t_s4fe_By_d0uble_Equ4ls}';
    if (isset ($_GET['password'])) {
        if (ereg ("^[a-zA-Z0-9]+$", $_GET['password']) === FALSE)
            echo 'You password must be alphanumeric';
        else if (strpos ($_GET['password'], '--') !== FALSE)
            die($flag);
        else
            echo 'Invalid password';
    }
?>
```

将参数password赋值一个数组传递进去：

```
http://localhost/?password[]=gg
```

ereg()函数 是处理字符串的，传入数组后返回 NULL， NULL 和 FALSE，是不恒等（===）的，满足第一个 if 条件；

而 strpos()函数 也是处理字符串的，传入数组后返回 NUL L， NULL!==FALSE，满足条件，拿到flag:



六、parse_str函数变量覆盖缺陷

理论

parse_str 函数的作用就是解析字符串并注册成变量，在注册变量之前不会验证当前变量是否存在，所以直接覆盖掉已有变量。

```
void parse_str ( string $str [, array &$arr ] )
```

str 输入的字符串。

arr 如果设置了第二个变量 arr，变量将会以数组元素的形式存入到这个数组，作为替代。

实践

测试代码:

```php
<?php
error_reporting(0);
$flag = 'flag{V4ri4ble_M4y_Be_C0verEd}';
if (empty($_GET['b'])) {
    show_source(__FILE__);
    die();
}else{
    $a = "www.sqlsec.com";
    $b = $_GET['b'];
    @parse_str($b);
    if ($a[0] != 'QNKCDZO' && md5($a[0]) == md5('QNKCDZO')) {
        echo $flag;
    }else{
        exit('your answer is wrong~');
    }
}
?>
```

考察点

parse_str变量覆盖缺陷

write-up

找到核心代码:

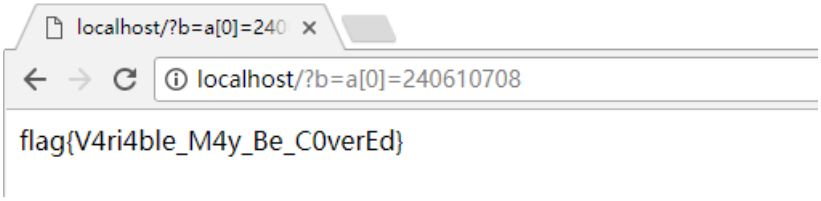
```
@parse_str($b);
这里使用了parse_str函数来传递b的变量值
if ($a[0] != 'QNKCDZO' && md5($a[0]) == md5('QNKCDZO'))
这里用到的是文章上面的知识点md5()函数缺陷
```

因为这里用到了 `parse_str` 函数来传递 `b` ，if的语句的条件是拿 `$a[0]` 来比较的，有因为这里的变量a的值已经三是固定的了:

```
$a = "www.sqlsec.com";
```

这里其实是我博客的地址~~ 不过不重要。 整体代码乍看起来又不可能，但是利用变量覆盖函数的缺陷这里可以对 `a` 的变量进行重新赋值，后面的的if语句再利用本文前面提到的 `md5()` 比较缺陷进行绕过:

```
http://localhost/?b=a[0]=240610708
```



参考文献

PHP 比较运算符

PHP Float 浮点型

PHP 类型比较表

PHP 弱类型总结

PHP Hash比较存在缺陷，影响大量Web网站登录认证、忘记密码等关键业务

PHP代码审计片段讲解（入门代码审计、CTF必备）

浅谈PHP弱类型安全

NJCTF2017 线上赛 web 题解

CTF之PHP黑魔法总结

Some features of PHP in CTF

PHP浮点数运算精度的问题

php strcmp()漏洞

危险的is_numeric——PHPYun 2015-06-26 二次注入漏洞分析

【代码审计】变量覆盖漏洞详解

编程小技巧

bian cheng xiao ji qiao

代码技巧