

0x01 普通注入

SQL参数拼接，未做任何过滤

```
<?php
$con = mysql_connect("localhost","root","root");
if (!$con){die('Could not connect: ' . mysql_error());}
mysql_select_db("test", $con);
$id = stripslashes($_REQUEST[ 'id' ]);
$query = "SELECT * FROM users WHERE id = $id ";
$result = mysql_query($query)or die('<pre>'.mysql_error().'</pre>');
while($row = mysql_fetch_array($result))
{
    echo $row['0'] . " " . $row['1'];
    echo "<br />";
}
echo "<br/>";
echo $query;
mysql_close($con);
?>
```

测试语句：id=1 UNION SELECT user(),2,3,4 from users

0x02 宽字节注入

A、MYSQL中的宽字符注入

示例代码：

```
<?php
$con = mysql_connect("localhost","root","root");
mysql_query("SET NAMES 'gbk'");
mysql_select_db("test", $con);
$id = isset($_GET['id']) ? addslashes($_GET['id']) : 1;
$query = "SELECT * FROM users WHERE id ='{ $id }' ";
$result = mysql_query($query)or die('<pre>'.mysql_error().'</pre>');
while($row = mysql_fetch_array($result))
{
    echo $row['0'] . " " . $row['1'];
    echo "<br />";
}
echo "<br/>";
echo $query;
mysql_close($con);
?>
```

测试语句：%df%27

mysql的特性，因为gbk是多字节编码，两个字节代表一个汉字，所以%df和后面的\也就是%5c变成了一个汉字“運”，而逃逸了出来。

根据gbk编码，第一个字节ascii码大于128，基本上就可以了。比如我们不用%df，用%a1也可以。

gb2312编码的取值范围。它的高位范围是0xA1~0xF7，低位范围是0xA1~0xFE，而\是0x5c，是不在低位范围中的。所以，0x5c根本不是gb2312中的编码，所以不会造成宽字节注入。扩展到世界上所有多字节编码，只要低位的范围中含有0x5c的编码，就可以进行宽字符注入。

宽字符注入的修复：

方案一：指定php连接mysql的字符集

```
mysql_set_charset('gbk',$conn);
```

```
$id =mysql_real_escape_string($_GET['id']);
```

方案二：将character_set_client设置为binary（二进制）

```
mysql_query("SET character_set_connection=gbk, character_set_results=gbk,character_set_client=binary",  
$conn);
```

SET NAMES 'gbk'
SET character_set_connection=gbk, character_set_results=gbk, character_set_client=binary
SELECT * FROM users WHERE id = '\xDF\' union select 1,2,3,4 #'

将character_set_client设置成binary，就不存在宽字节或多字节的问题了，所有数据以二进制的形式传递，就能有效避免宽字符注入。

B、PHP 编码转换

示例代码：

```
<?php  
$con = mysql_connect("localhost","root","root");  
mysql_query("SET NAMES 'gbk'");  
mysql_select_db("test", $con);  
mysql_query("SET character_set_connection=gbk,  
character_set_results=gbk,character_set_client=binary", $con);  
$id = isset($_GET['id']) ? addslashes($_GET['id']) : 1;  
$id=iconv('utf-8','gbk',$id);  
$query = "SELECT * FROM users WHERE id='{ $id}' ";  
$result = mysql_query($query)or die('<pre>'.mysql_error().'</pre>');  
  
while($row = mysql_fetch_array($result))  
{  
    echo $row['0'] . " " . $row['1'];  
    echo "<br />";  
}  
echo "<br/>";  
echo $query;  
  
mysql_close($con);  
  
?>
```

测试语句： 锦'

锦这个字：它的utf-8编码是%e9%8c%a6，它的gbk编码是%e5%5c

锦被iconv从utf-8转换成gbk后，变成了%e5%5c，而后面的'被addslashes变成了%5c%27，这样组合起来就是%e5%5c%5c%27，两个%5c就是\\，正好把反斜杠转义了，导致'逃逸出单引号，产生注入。

```
$id=iconv('gbk','utf-8',$id); //使用%df%27来测试
```

一个gbk汉字2字节，utf-8汉字3字节，如果我们把gbk转换成utf-8，则php会每两个字节一转换。所以，如果'前面的字符是奇数的话，势必会吞掉\\，'逃出限制。

其他函数：

```
mb_convert_encoding($id,'utf-8','gbk') //GBK To UTF-8与 iconv('gbk','utf-8',$id)一样
```

参考文章：

PHP字符编码绕过漏洞总结 <http://www.cnblogs.com/Safe3/archive/2008/08/22/1274095.html>

浅析白盒审计中的字符编码及SQL注入 <http://www.freebuf.com/articles/web/31537.html>

0x03 编码解码

找一些编码解码的函数来绕过防护，如urldecode()、rawurldecode()、base64_decode()

```
<?php
$con = mysql_connect("localhost","root","root");
mysql_select_db("test", $con);
$id = addslashes($_REQUEST['id']);
$id = urldecode($id);//$id = base64_decode($id);
$query = "SELECT * FROM users WHERE id = '{$id}'";
$result = mysql_query($query)or die('<pre>'.mysql_error().'</pre>');

while($row = mysql_fetch_array($result))
{
    echo $row['0'] . " " . $row['1'];
    echo "<br />";
}
echo "<br />";
echo $query;
mysql_close($con);
?>
```

测试语句：

1'union select 1,2,3,4%23 单引号Urlencode 1%2527union select 1,2,3,4%23

1'union select 1,2,3,4# Base64 Encode MSd1bmlvbiBzZWxlY3QgMSwyLDMsNCM=

0x04 二次注入

入库后转义符就会消失，变成hack',查询出库的就是hack'，如果拼接到SQL语句，成功引入了单引号闭合前面字符，导致注入。

测试：

```
CREATE TABLE test (user VARCHAR(20) NOT NULL);
```

```
INSERT INTO test values('hack');
```

```
mysql> INSERT INTO test values('hack\''');
Query OK, 1 row affected (0.06 sec)

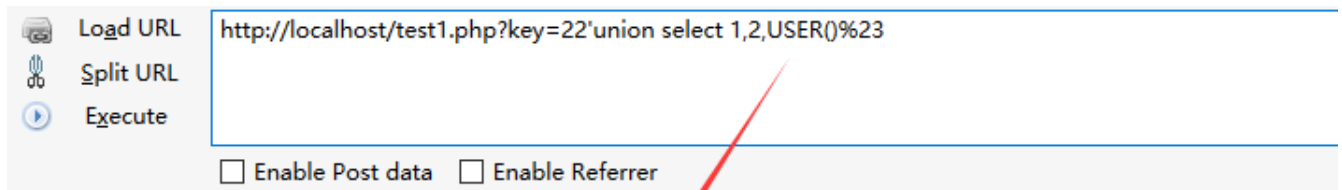
mysql> select * from test;
+-----+
| user  |
+-----+
| hack' |
+-----+
1 row in set (0.00 sec)
```

示例代码：

```
//测试数据
create table test(
id INT NOT NULL,
user VARCHAR(100) NOT NULL,
pass VARCHAR(100) NOT NULL
)
INSERT INTO test values(1,'hack','hack');

//测试代码
<?php
$con = mysql_connect("localhost","root","root");
mysql_select_db("test", $con);
mysql_query("SET character_set_connection=gbk,
character_set_results=gbk,character_set_client=binary", $con);
//update入库
if (isset($_GET['key'])){
    $key=addslashes($_REQUEST['key']);
    $query ="update test set user='{$key}' where id=1";
    echo "INSERT SQL: ".$query."<br/>";
    $result = mysql_query($query);
}
//select 出库，并带入查询
$query = "SELECT * FROM test WHERE id = 1";
$result = mysql_query($query);
$row = mysql_fetch_row($result);
echo "<br/>";
$query = "SELECT * FROM test WHERE user = '{$row[1]}'";
print_r('SELECT SQL: '.$query."<br/>");
$result = mysql_query($query)or die('<pre>'.mysql_error()."</pre>");
echo "<br/>";
print_r(mysql_fetch_row($result));
mysql_close($con);
?>
```

测试截图：



Load URL

Split URL

Execute

☐ Enable Post data ☐ Enable Referrer

INSERT SQL: update test set user='22\'union select 1,2,USER()#' where id=1

SELECT SQL: SELECT * FROM test WHERE user = '22\'union select 1,2,USER()#'

Array ([0] => 1 [1] => 2 [2] => root@localhost)

0x05 全局防护盲点

1、str_replace函数 过滤单引号等，可能造成注入；

2、stripslashes() 函数删除由 addslashes() 函数添加的反斜杠。stripslashes函数使用不当，可能造成注入；

①注入点类似id=1这种整型的参数就会完全无视GPC的过滤；②注入点包含键值对的，那么这里只检测了value，对key的过滤就没有防护；③有时候全局的过滤只过滤掉GET、POST和COOKIE，但是没过滤SERVER。

①FILES注入，全局只转义掉GET、POST等传来的参数，遗漏了FILES；②变量覆盖，危险函数：extract()、parse_str()、\$\$。

0x06 漏洞防护

基本思路：输入（解决数字型注入）-----转义处理（解决字符型注入）-----输出（解决数据库报错）

1、检查输入的数据是否具有所期望的数据格式。PHP 有很多可以用于检查输入的函数，从简单的变量函数和字符类型函数（比如 is_numeric(), ctype_digit()）到复杂的 Perl 兼容正则表达式函数都可以完成这个工作。如果程序等待输入一个数字，可以考虑使用 is_numeric() 来检查，或者直接使用 settype() 来转换它的类型，也可以用 sprintf() 把它格式化为数字。

2、PHP内置转义函数

Addslashes() <http://php.net/manual/zh/function.addslashes.php>

magic_quote_gpc <http://php.net/manual/zh/info.configuration.php#ini.magic-quotes-gpc>

mysql_real_escape_string() <http://php.net/manual/zh/function.mysql-real-escape-string.php>

mysql_escape_string() <http://php.net/manual/zh/function.mysql-escape-string.php>

3、数据库报错信息泄露防范：

把php.ini文件display_errors = Off

数据库查询函数前面加一个@字符

最有效可预防SQL注入攻击的防御方式：预处理技术进行数据库查询：

代码示例：

```
<?php
$mysqli = new mysqli("localhost","root","root","test");
```

```
if(!$mysqli){
    die($mysqli->error);
}
$sql = "select id,username,password from users where id=?";////创建一个预定义的对象 ?占位
$mysqli_stmt = $mysqli->prepare($sql);
$id=$_REQUEST['id'];
$mysqli_stmt->bind_param("i",$id);////绑定参数
$mysqli_stmt->bind_result($id,$username,$password);////绑定结果集
$mysqli_stmt->execute();//执行
while($mysqli_stmt->fetch()){    //取出绑定的结果集
    echo $id." ".$username ." ". $password;
}
echo "<br/>";
echo $sql;
$mysqli_stmt->free_result(); ////关闭结果集
$mysqli_stmt->close();
$mysqli->close();
?>
```

新文章将同步更新到我的个人公众号上，欢迎各位朋友扫描我的公众号二维码关注一下我，随时获取最新动态。

