浅析白盒审计中的字符编码及 SQL 注入

by phithon

http://www.leavesongs.com

尽管现在呼吁所有的程序都使用 unicode 编码，所有的网站都使用 utf-8 编码，来一个统一的国际规范。但仍然有很多，包括国内及国外（特别是非英语国家）的一些 cms，仍然使用着自己国家的一套编码，比如 gbk，作为自己默认的编码类型。也有一些 cms 为了考虑老用户，所以出了 gbk 和 utf-8 两个版本。

我们就以 gbk 字符编码为示范，拉开帷幕。gbk 是一种多字符编码，具体定义自行百度。但有一个地方尤其要注意：

通常来说，一个 gbk 编码汉字，占用 2 个字节。一个 utf-8 编码的汉字，占用 3 个字节。在 php 中，我们可以通过输出

echo strlen('和');

来测试。当将页面编码保存为 gbk 时输出 2，utf-8 时输出 3。

除了 gbk 以外，所有 ANSI 编码都是 2 个字节。ansi 只是一个标准，在不用的电脑上它代表的编码可能不相同，比如简体中文系统中 ANSI 就代表是 GBK。

以上是一点关于多字节编码的小知识，只有我们足够了解它的组成及特性以后，才能更好地去分析它身上存在的问题。

说了这么多废话，现在来研究一下在 SQL 注入中，字符编码带来的各种问题。

0x01 MYSQL 中的宽字符注入

这是一个老话题了，也被人玩过无数遍。但作为我们这篇文章的序幕，也是基础，是必须要提的。

我们先搭建一个实验环境。暂且称之为 phithon 内容管理系统 v1.0，首先先新建一个数据库，把如下压缩包中的 sql 文件导入：

测试代码及数据库：http://pan.baidu.com/s/1eQmUArw 提取密码:75tu

之后的 phithon 内容管理系统会逐步完善，但会一直使用这个数据表。

源码很简单（**注意先关闭自己 php 环境的 magic_quotes_gpc**）：

```
<?php
//连接数据库部分，注意使用了 gbk 编码，把数据库信息填写进去
$conn = mysql_connect('localhost', 'root', 'toor!@#$') or die('bad!');
mysql_query("SET NAMES 'gbk'");
mysql_select_db('test', $conn) OR emMsg("连接数据库失败，未找到您填写的数据库");
//执行 sql 语句
$id = isset($_GET['id']) ? addslashes($_GET['id']) : 1;
$sql = "SELECT * FROM news WHERE tid='{$id}'";
$result = mysql_query($sql, $conn) or die(mysql_error()); //sql 出错会报错，方便观察
?>
<!DOCTYPE html>
<html>
<head>
<meta charset="gbk" />
<title>新闻</title>
</head>
<body>
<?php
$row = mysql_fetch_array($result, MYSQL_ASSOC);
echo "<h2>{$row['title']}</h2><p>{$row['content']}<p>\n";
mysql_free_result($result);
?>
</body>
</html>
```

SQL 语句是 SELECT * FROM news WHERE tid='{$id}'，就是根据文章的 id 把文章从 news 表中取出来。

在这个 sql 语句前面，我们使用了一个 addslashes 函数，将$id 的值转义。这是通常 cms 中对 sql 注入进行的操作，只要我们的输入参数在单引号中，就逃逸不出单引号的限制，无法注入，如下图：

新闻1

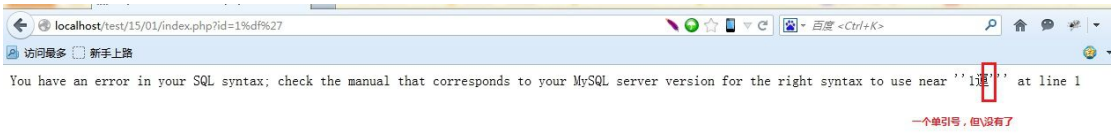
这是第一篇文章

那么怎么逃过 addslashes 的限制？众所周知 addslashes 函数产生的效果就是，让’变成\’，让引号变得不再是“单引号”，只是一撇而已。一般绕过方式就是，想办法处理\’前面的\：

1.想办法给\前面再加一个\（或单数个即可），变成\\’，这样\被转义了，’逃出了限制

2.想办法把\弄没有。

我们这里的宽字节注入是利用 mysql 的一个特性，mysql 在使用 GBK 编码的时候，会认为两个字符是一个汉字（前一个 ascii 码要大于 128，才到汉字的范围）。如果我们输入%df’看会怎样：



我们可以看到，已经报错了。我们看到报错，说明 sql 语句出错，看到出错说明可以注入了。

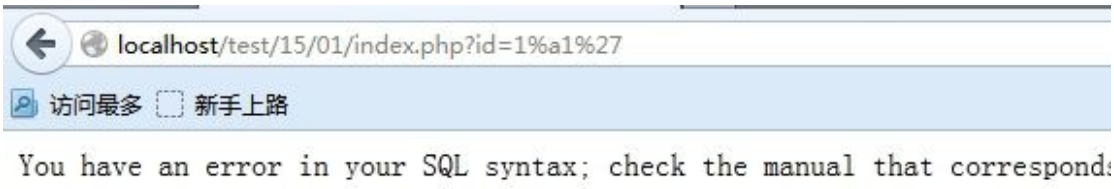
为什么从刚才到现在，只是在’也就是%27 前面加了一个%df 就报错了？而且从图中可以看到，报错的原因就是多了一个单引号，而单引号前面的反斜杠不见了。

这就是 mysql 的特性，因为 gbk 是多字节编码，他认为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。

因为两个字节代表一个汉字，所以我们可以试试“%df%df%27”：

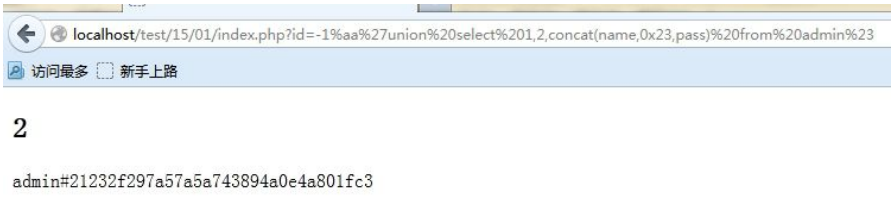
不报错了。因为%df%df 是一个汉字，%5c%27 不是汉字，仍然是\'。

那么 mysql 怎么判断一个字符是不是汉字，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：



%a1%5c 他可能不是汉字，但一定会被 mysql 认为是一个宽字符，就能够让后面的%27 逃逸了出来。

于是我可以构造一个 exp 出来，查询管理员账号密码：



0x02 GB2312 与 GBK 的不同

曾经有一个问题一直困扰我很久。

gb2312 和 gbk 应该都是宽字节家族的一员。但我们来做个小实验。把 phithon 内容管理系统中 set names 修改成 gb2312:

```php
<?php
//连接数据库部分，注意使用了gbk编码
$conn = mysql_connect('localhost', 'root', 'toor!@#$') or die('bad!');
mysql_query("SET NAMES 'gb2312'")
mysql_select_db('test', $conn) OR emMsg("连接数据库失败，未找到您填写的数据库");
//执行sql语句
$id = isset($_GET['id']) ? addslashes($_GET['id']) : 1;
$sql = "SELECT * FROM news WHERE tid='{$id}'";
$result = mysql_query($sql, $conn) or die(mysql_error());
?>
<!DOCTYPE html>
```

结果就是不能注入了：

新闻1

这是第一篇文章

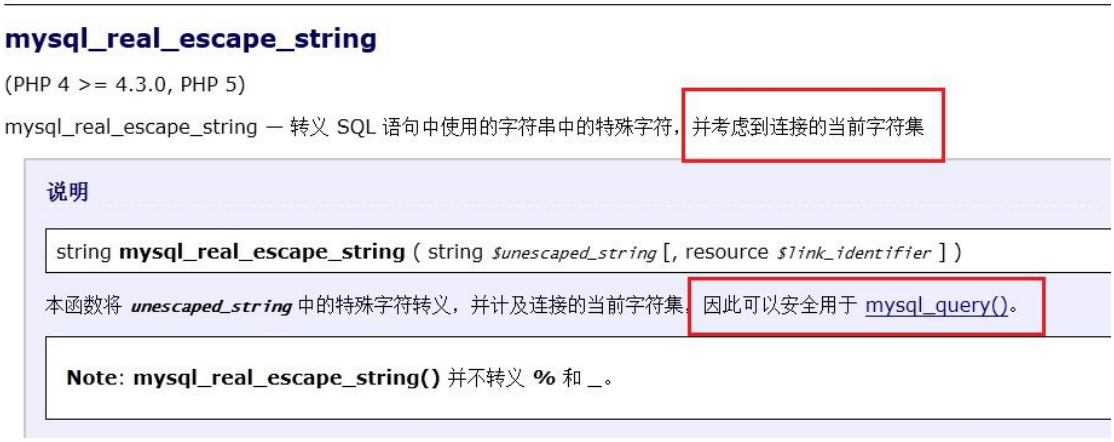
有些同学不信的话，也可以把数据库编码也改成 gb2312，也是不成功的。

为什么，这归结于 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，所以自然也是不会被吃掉的。

所以，把这个思路扩展到世界上所有多字节编码，我们可以这样认为：只要低位的范围中含有 0x5c 的编码，就可以进行宽字符注入。

0x03 mysql_real_escape_string 解决问题？

部分 cms 对宽字节注入有所了解，于是寻求解决方案。在 php 文档中，大家会发现一个函数，mysql_real_escape_string，文档里说了，考虑到连接的当前字符集。



mysql_real_escape_string

(PHP 4 >= 4.3.0, PHP 5)

mysql_real_escape_string — 转义 SQL 语句中使用的字符串中的特殊字符，并考虑到连接的当前字符集

说明

string **mysql_real_escape_string** (string *$unescaped_string* [, resource *$link_identifier*])

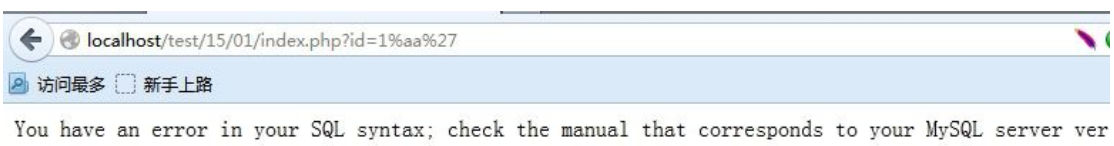
本函数将 ***unescaped_string*** 中的特殊字符转义，并计及连接的当前字符集，因此可以安全用于 mysql_query()。

Note: **mysql_real_escape_string()** 并不转义 **%** 和 _。

于是，有的 cms 就把 addslashes 替换成 mysql_real_escape_string，来抵御宽字符注入。我们继续做试验，phithon 内容管理系统 v1.2:，就用 mysql_real_escape_string 来过滤输入：

```php
<?php
//连接数据库部分，注意使用了gbk编码
$conn = mysql_connect('localhost', 'root', 'toor!@#$') or die('bad!');
mysql_query("SET NAMES 'gbk'");
mysql_select_db('test', $conn) OR emMsg("连接数据库失败，未找到您填写的数据库");
//执行sql语句
$id = isset($_GET['id']) ? mysql_real_escape_string($_GET['id']) : 1;
$sql = "SELECT * FROM news WHERE tid='{$id}'";
$result = mysql_query($sql, $conn) or die(mysql_error());
?>
<!DOCTYPE html>
<html>
<head>
<meta charset="gbk" />
<title>新闻</title>
</head>
<body>
<?php
$row = mysql_fetch_array($result, MYSQL_ASSOC);
echo "<h2>{$row['title']}</h2><p>{$row['content']}<p>\n";
mysql_free_result($result);
?>
</body>
</html>
```

我们来试试能不能注入：



一样没压力注入。为什么，明明我用了 mysql_real_escape_string，但却仍然不能抵御宽字符注入。

原因就是，你没有指定 php 连接 mysql 的字符集。我们需要在执行 sql 语句之前调用一下 mysql_set_charset 函数，设置当前连接的字符集为 gbk。

```php
<?php
//连接数据库部分，注意使用了gbk编码
$conn = mysql_connect('localhost', 'root', 'toor!@#$') or die('bad!');
mysql_query("SET NAMES 'gbk'");
mysql_select_db('test', $conn) OR emMsg("连接数据库失败，未找到您填写的数据库");
//执行sql语句
mysql_set_charset('gbk', $conn);
$id = isset($_GET['id']) ? mysql_real_escape_string($_GET['id']) : 1;
$sql = "SELECT * FROM news WHERE tid='{$id}'";
$result = mysql_query($sql, $conn) or die(mysql_error());
?>
<!DOCTYPE html>
<html>
<head>
<meta charset="gbk" />
<title>新闻</title>
</head>
<body>
<?php
$row = mysql_fetch_array($result, MYSQL_ASSOC);
echo "<h2>{$row['title']}</h2><p>{$row['content']}<p>\n";
mysql_free_result($result);
?>
</body>
</html>
```

就可以避免这个问题了：

新闻1

这是第一篇文章

0x04 宽字符注入的修复

在 3 中我们说到了一种修复方法，就是先调用 mysql_set_charset 函数设置连接所使用的字符集为 gbk，再调用 mysql_real_escape_string 来过滤用户输入。

这个方式是可行的，但有部分老的 cms，在多处使用 addslashes 来过滤字符串，我们不可能去一个一个把 addslashes 都修改成 mysql_real_escape_string。我们第二个解决方案就是，将 character_set_client 设置为 binary（二进制）。

只需在所有 sql 语句前指定一下连接的形式是二进制：

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

这几个变量是什么意思？

当我们的 mysql 接受到客户端的数据后，会认为他的编码是 character_set_client，然后会将之将换成 character_set_connection 的编码，然后进入具体表和字段后，再转换成字段对应的编码。

然后，当查询结果产生后，会从表和字段的编码，转换成 character_set_results 编码，返回给客户端。

所以，我们将 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。

比如，我们的 phithon 内容管理系统 v2.0 版本更新如下：

```php
<?php
//连接数据库部分，注意使用了gbk编码
$conn = mysql_connect('localhost', 'root', 'toor!@#$') or die('bad!');
mysql_query("SET NAMES 'gbk'");
mysql_select_db('test', $conn) OR emMsg("连接数据库失败，未找到您填写的数据库");
//执行sql语句
mysql_query("SET character_set_connection=gbk, character_set_results=gbk,character_set_client=binary", $conn);
$id = isset($_GET['id']) ? addslashes($_GET['id']) : 1;
$sql = "SELECT * FROM news WHERE tid='{$id}'";
$result = mysql_query($sql, $conn) or die(mysql_error());
?>
<!DOCTYPE html>
<html>
<head>
<meta charset="gbk" />
<title>新闻</title>
</head>
<body>
<?php
$row = mysql_fetch_array($result, MYSQL_ASSOC);
echo "<h2>{$row['title']}</h2><p>{$row['content']}<p>\n";
mysql_free_result($result);
?>
</body>
</html>
```

已经不能够注入了：

在我审计过的代码中，大部分 cms 是以这样的方式来避免宽字符注入的。这个方法可以说是有效的，但如果开发者画蛇添足地增加一些东西，会让之前的努力前功尽弃。

0x05 iconv 导致的致命后果

很多 cms，不止一个，我就不提名字了，他们的 gbk 版本都存在因为字符编码造成的注入。但有的同学说，自己测试了这些 cms 的宽字符注入，没有效果呢，难道是自己姿势不对？

当然不是。实际上，这一章说的已经不再是宽字符注入了，因为问题并不是出在 mysql 上，而是出在 php 中了。

很多 cms（真的很多哦，不信大家自己网上找找）会将接收到数据，调用这样一个函数，转换其编码：

iconv('utf-8', 'gbk', $_GET['word']);

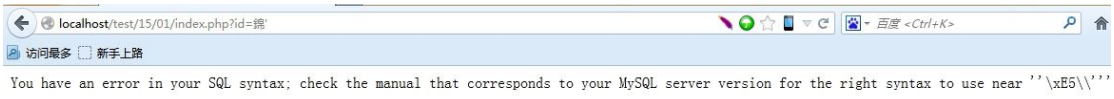
目的一般是为了避免乱码，特别是在搜索框的位置。

比如我们的 phithon 内容管理系统 v3.0

```php
<?php
//连接数据库部分，注意使用了gbk编码
$conn = mysql_connect('localhost', 'root', 'toor!@#$') or die('bad!');
mysql_query("SET NAMES 'gbk'");
mysql_select_db('test', $conn) OR emMsg("连接数据库失败，未找到您填写的数据库");
//执行sql语句
mysql_query("SET character_set_connection=gbk, character_set_results=gbk,character_set_client=binary", $conn);
$id = isset($_GET['id']) ? addslashes($_GET['id']) : 1;
$id = iconv('utf-8', 'gbk', $id);
$sql = "SELECT * FROM news WHERE tid='{$id}'";
$result = mysql_query($sql, $conn) or die(mysql_error());
?>
<!DOCTYPE html>
<html>
<head>
<meta charset="gbk" />
<title>新闻</title>
</head>
<body>
<?php
$row = mysql_fetch_array($result, MYSQL_ASSOC);
echo "<h2>{$row['title']}</h2><p>{$row['content']}<p>\n";
mysql_free_result($result);
?>
</body>
</html>
```

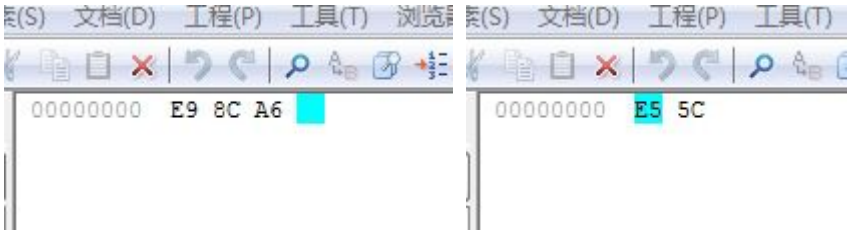
我们可以看到，它在 sql 语句执行前，将 character_set_client 设置成了 binary，所以可以避免宽字符注入的问题。但之后其调用了 iconv 将已经过滤过的参数$id 给转换了一下。

那我们来试试此时能不能注入：



居然报错了。说明可以注入。而我只是输入了一个“錦'”。这是什么原因？

我们来分析一下。“錦”这个字，它的 utf-8 编码是 0xe98ca6，它的 gbk 编码是 0xe55c。



有的同学可能就领悟了。\的 ascii 码正是 5c。那么，当我们的錦被 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。

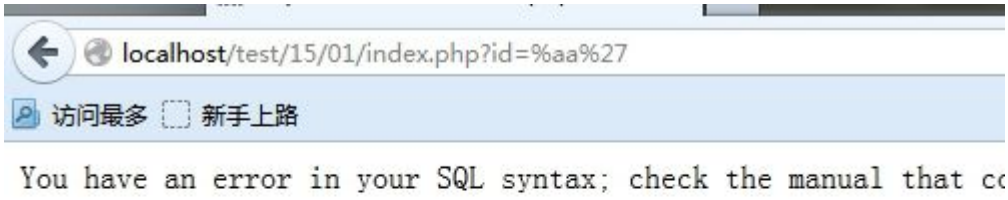
这正利用了我之前说的，绕过 addslashes 的两种方式的第一种：将\转义掉。

那么，如果我是用 iconv 将 gbk 转换成 utf-8 呢？

```php
<?php
//连接数据库部分，注意使用了gbk编码
$conn = mysql_connect('localhost', 'root', 'toor!@#$') or die('bad!');
mysql_query("SET NAMES 'gbk'");
mysql_select_db('test', $conn) OR emMsg("连接数据库失败，未找到您填写的数据库");
//执行sql语句
mysql_query("SET character_set_connection=gbk, character_set_results=gbk,character_set_client=binary", $conn);
$id = isset($_GET['id']) ? addslashes($_GET['id']) : 1;
$id = iconv('gbk', 'utf-8', $id);
$sql = "SELECT * FROM news WHERE tid='{$id}'";
$result = mysql_query($sql, $conn) or die(mysql_error());
?>
<!DOCTYPE html>
<html>
<head>
<meta charset="gbk" />
<title>新闻</title>
</head>
<body>
<?php
$row = mysql_fetch_array($result, MYSQL_ASSOC);
echo "<h2>{$row['title']}</h2><p>{$row['content']}<p>\n";
mysql_free_result($result);
?>
</body>
</html>
```

将gbk转换成utf-8

我们来试试：



果然又成功了。这次直接用宽字符注入的姿势来的，但实际上问题出在 php 而不是 mysql。我们知道一个 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。

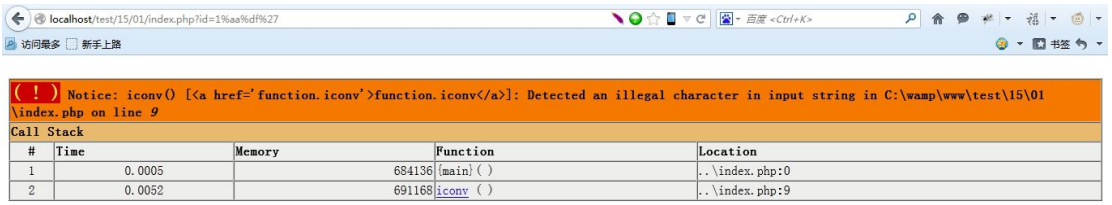
那么为什么之前 utf-8 转换成 gbk 的时候，没有使用这个姿势？

这跟 utf-8 的规则有关，UTF-8 的编码规则很简单，只有二条：

1）对于单字节的符号，字节的第一位设为 0，后面 7 位为这个符号的 unicode 码。因此对于英语字母，UTF-8 编码和 ASCII 码是相同的。

2）对于 n 字节的符号（n>1），第一个字节的前 n 位都设为 1，第 n+1 位设为 0，后面字节的前两位一律设为 10。剩下的没有提及的二进制位，全部为这个符号的 unicode 码。

从 2 我们可以看到，对于多字节的符号，其第 2、3、4 字节的前两位都是 10，也就是说，\（0x0000005c）不会出现在 utf-8 编码中，所以 utf-8 转换成 gbk 时，如果有\则 php 会报错：



但因为 gbk 编码中包含了\，所以仍然可以利用，只是利用方式不同罢了。

总而言之，在我们处理了 mysql 的宽字符注入以后，也别认为就可以高枕无忧了。调用 iconv 时千万要小心，避免出现不必要的麻烦。

0x06 总结

在逐渐国际化的今天，推行 utf-8 编码是大趋势。如果就安全性来说的话，我也觉得使用 utf-8 编码能够避免很多多字节造成的问题。

不光是 gbk，我只是习惯性地把 gbk 作为一个典型的例子在文中与大家说明。世界上的多字节编码有很多，特别是韩国、日本及一些非英语国家的 cms，都可能存在由字符编码造成的安全问题，大家应该有扩展性的思维。

总结一下全文中提到的由字符编码引发的安全问题及其解决方案：

1.gbk 编码造成的宽字符注入问题，解决方法是设置 character_set_client=binary。

2.矫正人们对于 mysql_real_escape_string 的误解，单独调用 set name=gbk 和 mysql_real_escape_string 是无法避免宽字符注入问题的。还得调用 mysql_set_charset 来设置一下字符集。

3.谨慎使用 iconv 来转换字符串编码，很容易出现问题。只要我们把前端 html/js/css 所有编码设置成 gbk，mysql/php 编码设置成 gbk，就不会出现乱码问题。不用画蛇添足地去调用 iconv 转换编码，造成不必要的麻烦。

这篇文章是我对于自己白盒审计经验的一点小总结，但自己确实在很多方面存在欠缺，文中所提到的姿势难免存在纰漏和错误，希望有相同爱好的同学能与我指出，共同进步。

这篇文章不像上篇 xss 的，能够举出很多 0day 实例来论证宽字符造成的危害。原因有二：

1.宽字符问题确实不如富文本 xss 那么普遍，gbk 编码的 cms 所占的比例也比较小，怪我才疏学浅，并不能每一章都找到相应的实例。

2.注入的危害比 xss 大得多，如果作为 0day 发出来，影响很坏。但我确实在写文章以及以前的审计过程中找到不少 cms 存在的编码问题。

所以我用实验的形式，自己写了一个 php 小文件，给大家作为例子，希望不会因为例证的不足，影响大家学习的效果。

例子 php 文件和 sql 文件打包下载：

链接:http://pan.baidu.com/s/1eQmUArw 提取密码:75tu