


# 《软件安全》实验报告

学号: 2313211 姓名: 王众

## 实验名称:

跨站脚本攻击

## 实验要求:

复现课本第十一章实验三, 通过和<script>两类方式实现跨站脚本攻击, 撰写实验报告。有能力者可以自己撰写更安全的过滤程序。

## 实验过程:

### 1 Script方式

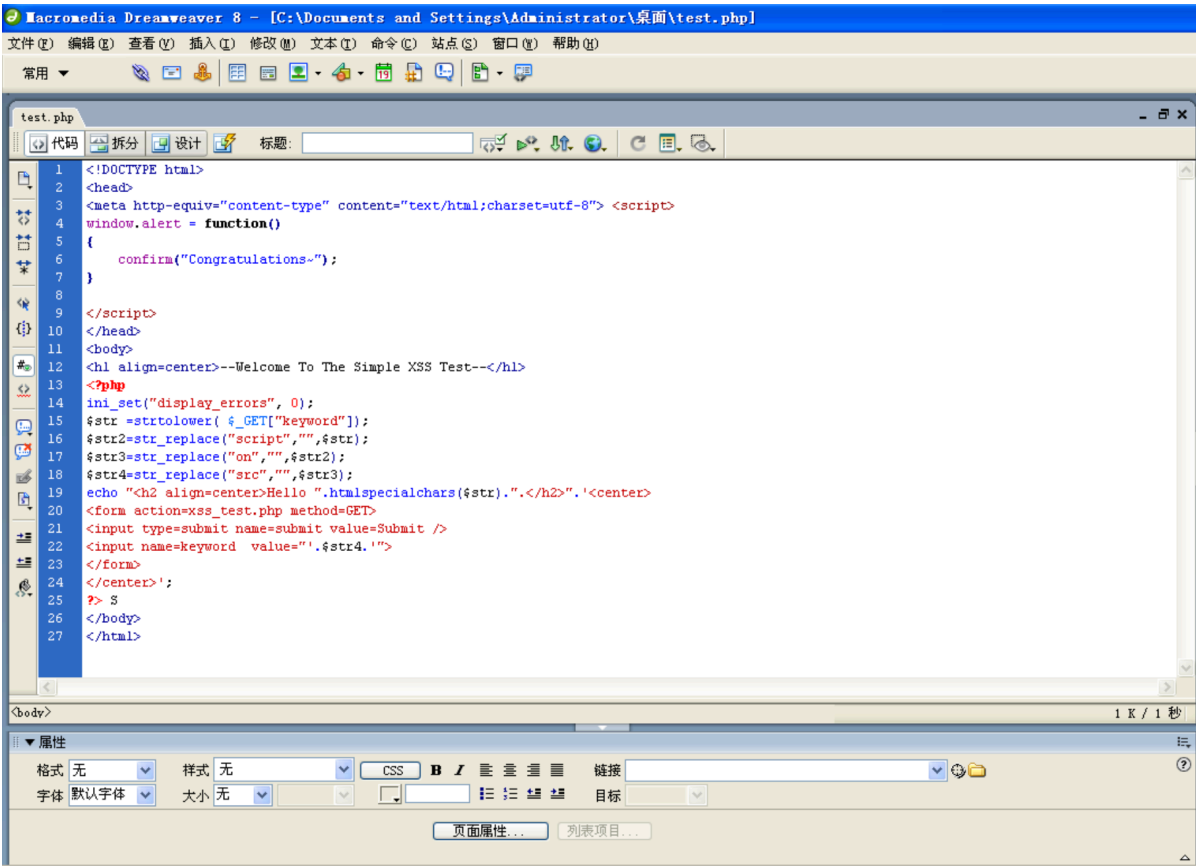
#### 1.1 建立Dreamweaver文件

首先我们输入源代码:

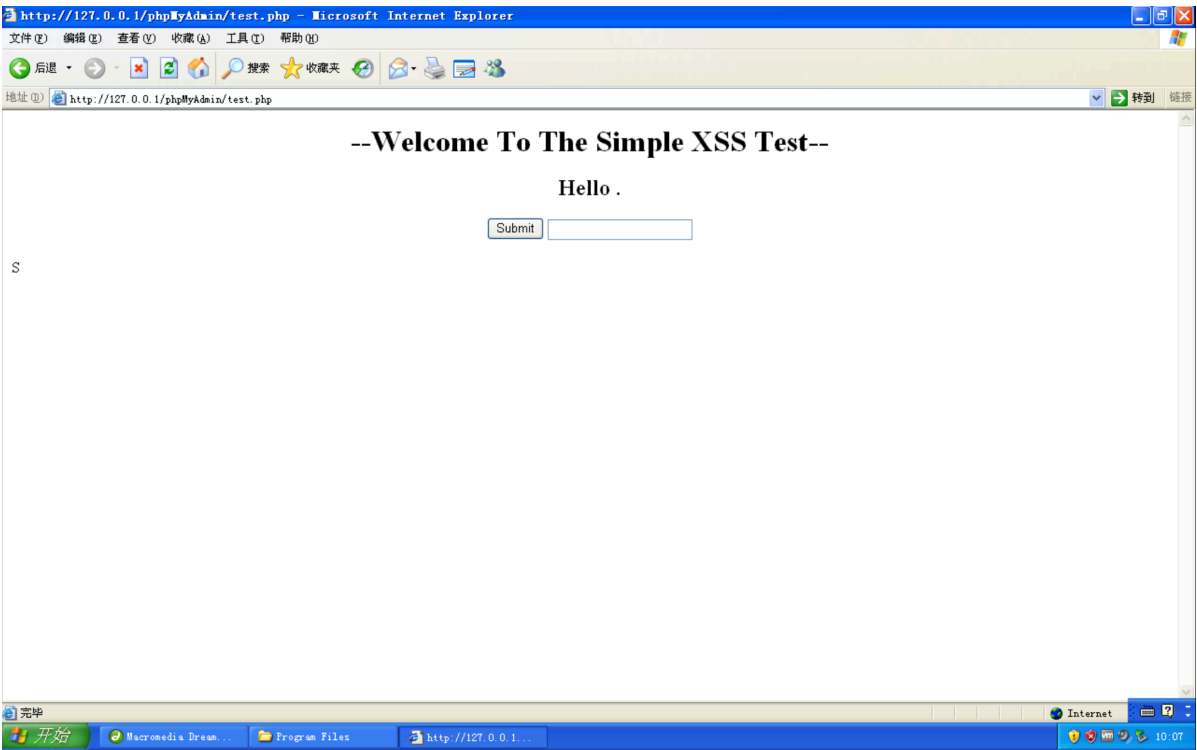
```
<!DOCTYPE html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8"> <script>
window.alert = function()
{
    confirm("Congratulations~");
}

</script>
</head>
<body>
<h1 align=center>--welcome To The Simple XSS Test--</h1>
<?php
ini_set("display_errors", 0);
$str =strtolower( $_GET["keyword"]);
$str2=str_replace("script","", $str);
$str3=str_replace("on","", $str2);
$str4=str_replace("src","", $str3);
echo "<h2 align=center>Hello ".htmlspecialchars($str)."</h2>". '<center>
<form action=xss_test.php method=GET>
<input type=submit name=submit value=Submit />
<input name=keyword value="'. $str4. "'>
</form>
</center>';
?>
</body>
</html>
```

我们将其输入到软件中，显示如下：

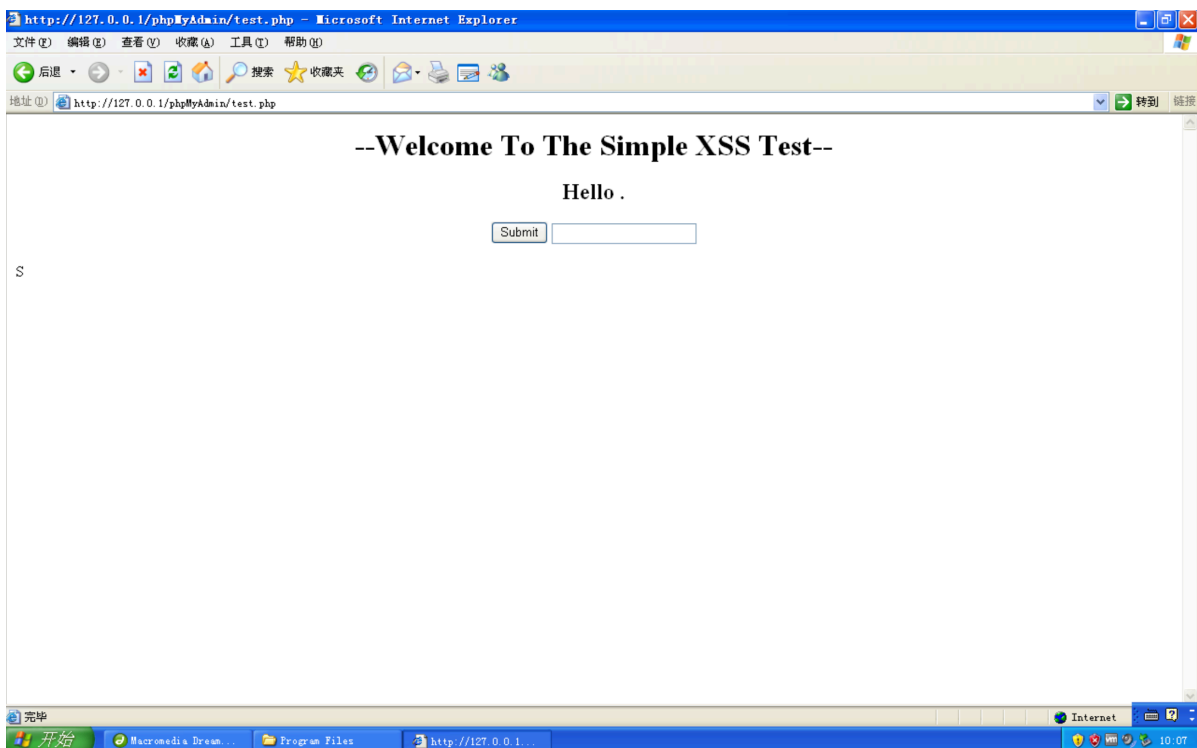


我们打开网页，输入地址：[http://127.0.0.1/xss\\_test.php](http://127.0.0.1/xss_test.php)，看到如下界面，证明代码运行无误，可以开始测试。

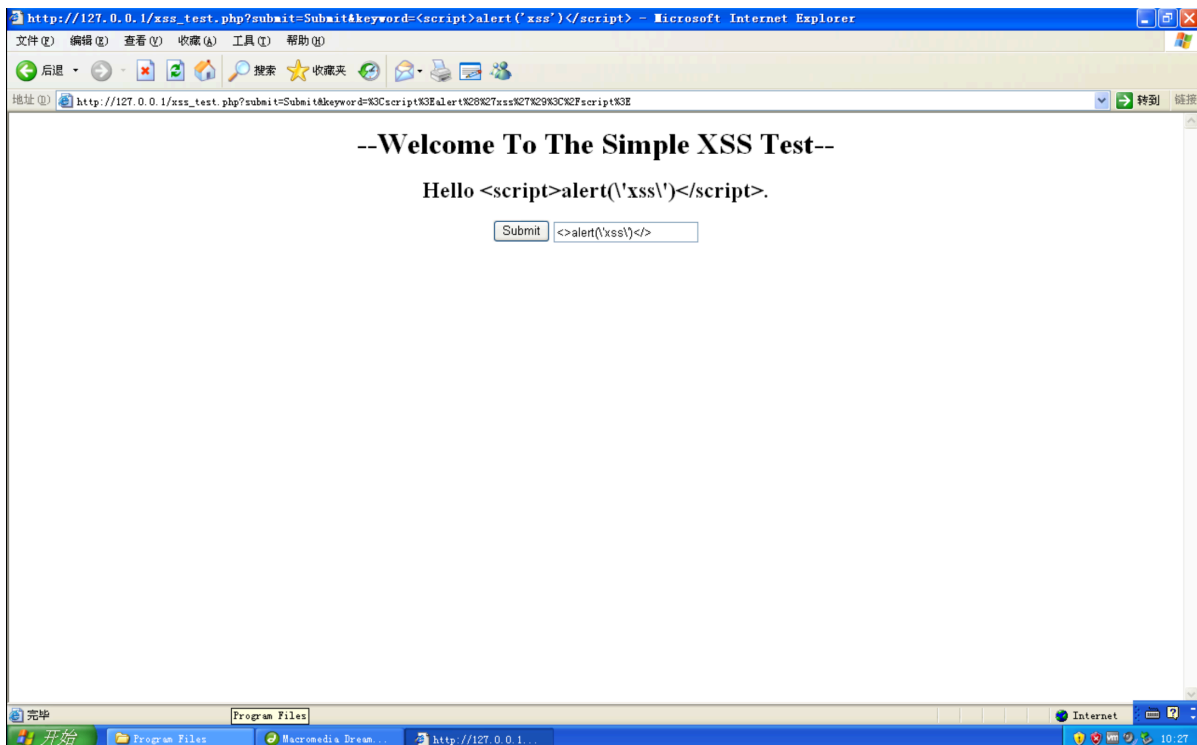


## 1.2 黑盒测试

首先从黑盒测试的角度来进行实验，访问URL：[http://127.0.0.1/xss\\_test.php](http://127.0.0.1/xss_test.php)，页面效果如图所示：



在界面中，我们可以看到一个 `Submit` 按钮和输入框，并且还有标题提示 `xss`。于是输入上面学过最简单的 `xss` 脚本：`<script>alert('xss')</script>` 来进行测试。我们点击 `Submit` 按钮以后，效果如下：

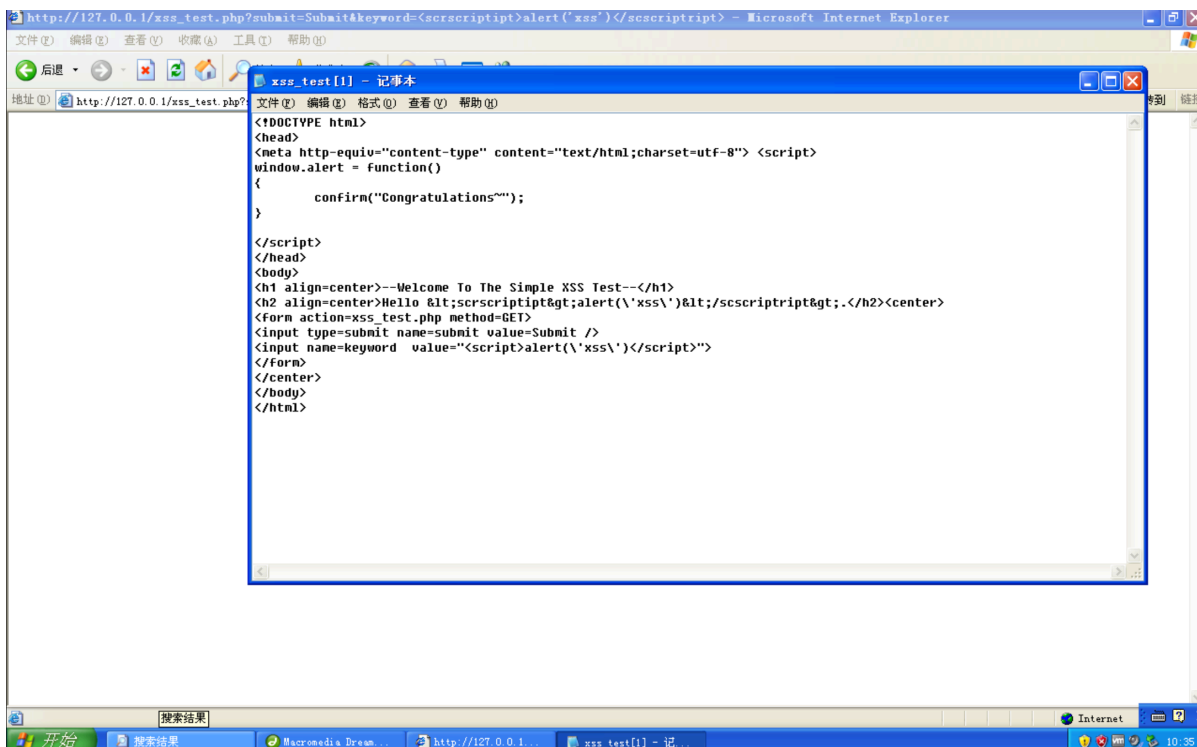


我们发现，发现 `Hello` 后面出现了我们输入的内容，并且输入框中的回显过滤了 `script` 关键字，这个时候考虑后台只是最简单的一次过滤。

我们再利用双写关键字来重新构造脚本：`<scrscripript>alert('xss')</scscripript>`，并再次进行测试，效果如下：



至此，虽然输入框中显示出来的代码确为我们想要运行的攻击脚本，但是这句代码并没有被执行（如果成功执行，会弹窗输出“congratulations”）。所以我们接着寻找问题。我们跳转到该 php 文件的源码，就可以发现：第5行重写的 alert 函数。如果可以成功执行 alert 函数的话，页面将会跳出一个确认框，显示 Congratulations~。这应该是我们 xss 成功攻击的标志。但现在我们并没有看到该弹窗，说明代码确实没有被执行。



我们继续查看源代码，我们发现相比于最先的 php 代码，第十六行发生了改变。

```
<input name=keyword value="<script>alert('xss')</script>">
```

分析这行代码知道，虽然我们成功的插入了 <script></script> 标签组,但是我们并没有跳出 input 的标签，使得我们的脚本仅仅可以回显而不能利用。这个时候的思路就是想办法将前面的 <input> 标签闭合，于是构造如下脚本：

```
"><script>alert('xss')</script><!--
```

分析一下这行代码："> 用来闭合前面的<input> 标签。而 <!-- 其实是为了美观，用来注释掉后面不需要的 "> ,否则页面就会在输入框后面回显 ">。我们输入该代码，进行测试，得到以下结果：



说明我们的攻击还是没有成功。这是因为 php 服务器为了避免一些用户特殊构造的攻击，将双引号等符号转义，于是修改 php-apache2handler.ini，将“magic\_quotes\_gpc = on”设置为“magic\_quotes\_gpc = off”。

我们发现经过调试，我们成功的完成了这次黑盒测试，输出我们想要的结果“Congratulations~”！



## 1.3 白盒测试

我们前往 xss\_test.php 文件中查看页面的核心逻辑。

```
<?php
    ini_set( "display_errors", 0);
    $str=strtolower( $_GET[ "keyword"] );
    $str2=str_replace( "script", "", $str);
    $str3=str_replace( "on", "", $str2);
    $str4=str_replace( "src", "", $str3);
    echo "<h2 align=center>Hello ".htmlspecialchars($str). "</h2>". '<center>
    <form action=xss_test.php method=GET>
    <input type=submit name=submit value=Submit />
    <input name=keyword value="'. $str4. "'>
    </form>
    </center>';
?>
```

分析上述代码可知，这些代码的逻辑与我们第2步中进行的黑盒测试所总结出的逻辑基本相符。但是也有黑盒测试中没测试到的地方。比如，Hello 后面显示的值是经过小写转换的。输入框中回显值的过滤方法是将 script、on、src 等关键字都替换成了空。所以我们如果修改 php 代码，将这些过滤全部取消的话，我们也可以实施攻击。

## 2 img方式

### 2.1 用<img>标签构造脚本

我们构造的脚本如下：

```
<img src=ops! onerror="alert('xss')">
```

<img> 标签是用来定义 HTML 中的图像，src 一般是图像的来源。而 onerror 事件会在文档或图像加载过程中发生错误时被触发。所以上面这个攻击脚本的逻辑是，当 img 加载一个错误的图像来源 ops! 时，会触发 onerror 事件，从而执行 alert 函数。

### 2.2 运行结果



## 心得体会：

通过本次实验，我粗略地学习了两种跨站脚本攻击的方法，一个是通过 script 方式去进行攻击，一个是通过 img 方式去进行攻击，两个方式都是 CTF 的 web 中常规的部分，我在调试的过程中遇到了很多问题，比如说使用 win11 浏览器，php 会过滤大部分敏感的特殊字符，我都努力去解决这些问题。我了解了 script 中过滤关键字的原理，即替换成无效的内容；还学会了如果构造一个可以实现 img 攻击的 Payload。