《软件安全》实验报告

姓名：冯思程 学号：2112213 班级： 计算机科学与技术二班

**实验名称：**

跨站脚本攻击

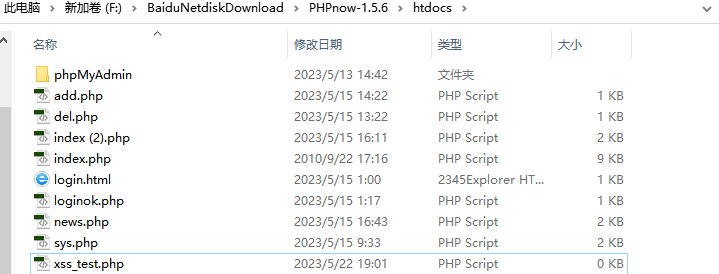
**实验要求：**

复现课本第十一章实验三，通过img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. **首先进行网页代码的编写：**

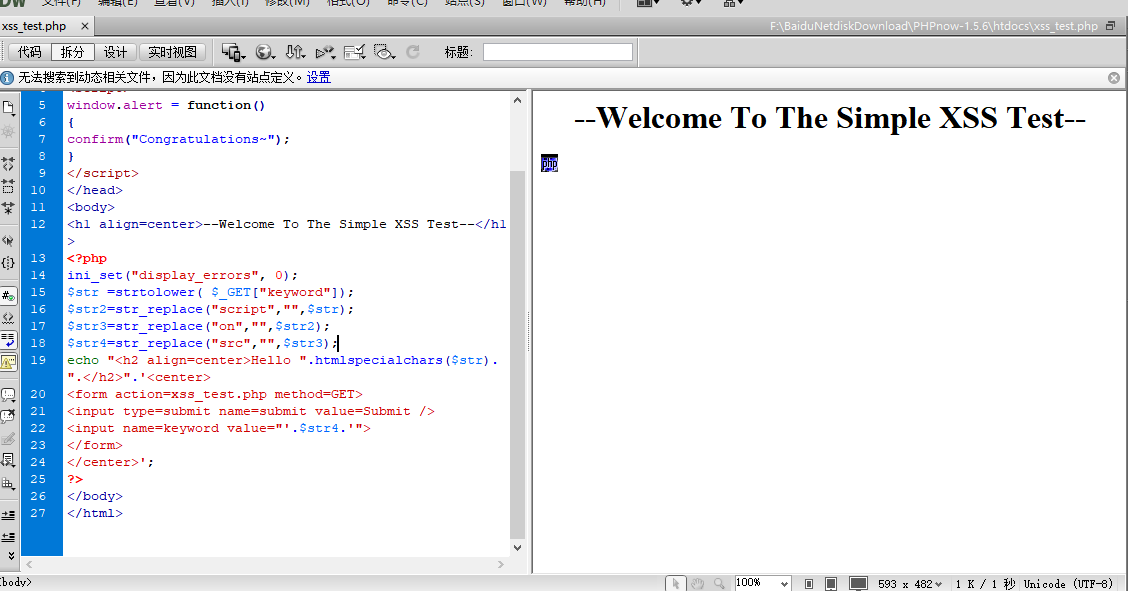
这里我选择在本机windows10操作系统下进行实验，首先找到phpnow安装路径下的htdocs文件夹，新建一个文本文件，然后重命名为xss\_test.php，如下：



然后用记事本打开方式打开文本文档，将教材上给出的示例网页代码粘贴到上面，如下：



保存文件后，用dreamweaver打开，如下：



打开浏览器，登录网址127.0.0.1/xss\_test.php,如下：



现在提前的准备工作已经做好，可以开始进一步的攻击实验部分。

1. **首先进行script方式实现跨站脚本攻击：**

观察上图中127.0.0.1/xss\_test.php的网页内容，可以看到一个 Submit 按钮和输入框，并且还有标题提示 XSS。

于是我首先采用最简单的XSS脚本进行攻击，如下：

脚本代码：<script>alert('xss')</script>



结果发现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。于是可以利用双写关键字绕过，构造脚本，结果如下：

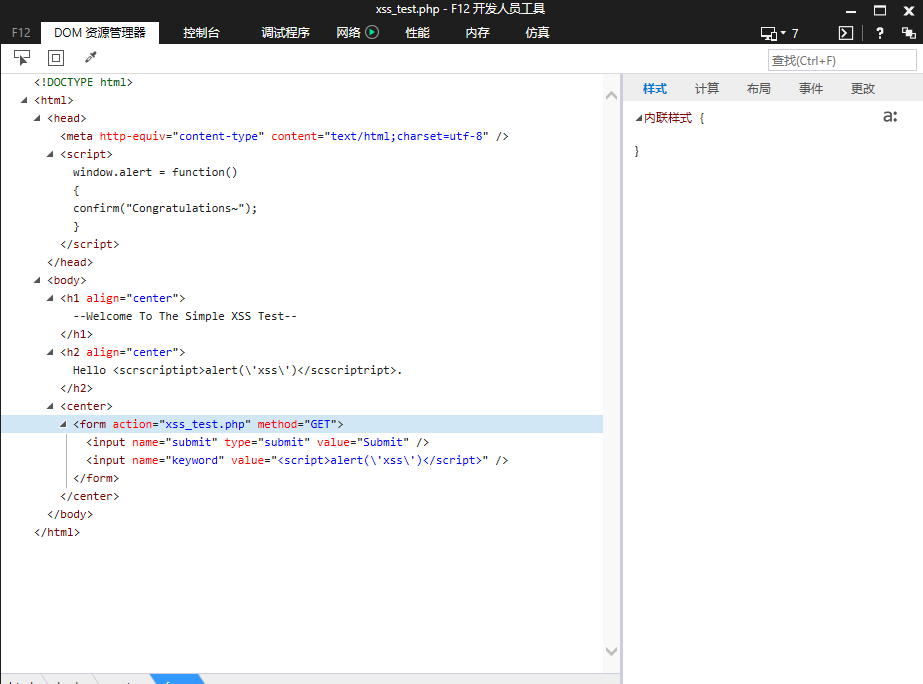
脚本代码：<scrscriptipt>alert('xss')</scscriptript>



发现虽然输入框中的回显确实是我们想要攻击的脚本，但是代码并没有执行。因为在黑

盒测试情况下，我们并不能看到全部代码的整个逻辑，所以无法判断问题到底出在哪里。这

个时候我们可以在页面点击右键查看源码，尝试从源码片段中分析问题。点击鼠标右键，选择查看网页源代码，发现点击查看源码网页无反应，经过多次尝试发现可以点击网页的工具-javascript控制台也可以找到源码展示，如下：



刚开始就会看到第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 成功攻击的的标志。然后接着往下查看21行的<input>标签，我们唯一能输入且有可能控制的地方，如下：

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

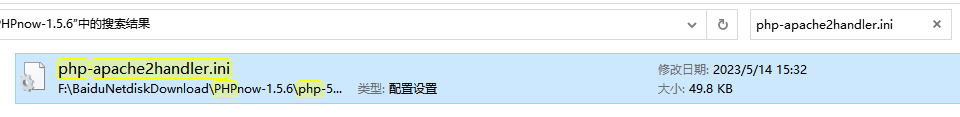
分析这行代码知道，虽然我们成功的插入了<script></script>标签组,但是我们并没有

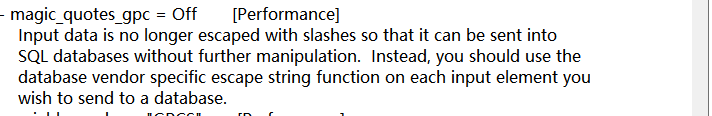
跳出input的标签，使得我们的脚本仅仅可以回显而不能利用。这个时候的思路就是想办法

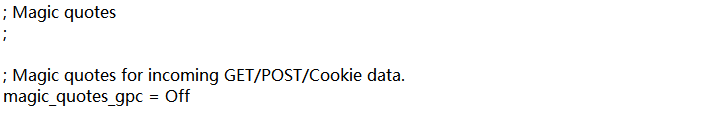
将前面的<input>标签闭合，于是构造如下脚本。

脚本代码："><scrscriptipt>alert(`xss`)</scscriptript><!--（注意xss旁边是`符号，最后面是两个-符号）

又注意到教材上的重要提醒，因为双引号没办法被正常处理，php 服务器自动会对输入的双引号等进行转义，以预防用户构造特殊输入进行攻击，比如本实验所进行的攻击。为了确保实验可以成功运行，请在 phpnow 安装目录下搜索文件php-apache2handler.ini，并将“magic\_quotes\_gpc = On”设置为“magic\_quotes\_gpc = Off”，如下：

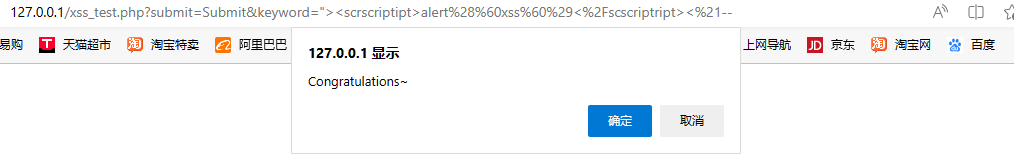








我更改后发现还是没有出现弹出框，发现可能是由于浏览器问题，于是换一个浏览器进行实验，成功弹出了结果，结果如下：



我们再来查看一下页面源码（我还是利用javascript控制器来进行查看），仔细查看一下 16 行代码执行的逻辑，如下：



<input name=keyword value="**"><script>alert('xss')</script><!--**">

如上所示，其中粗体是我成功构造的脚本，"> 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 "> 。上面就是我编写这个脚本的原理。

然后我从网页源码的逻辑分析：

代码：

<?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>';

?>

发现跟在上面进行黑盒测试的情况差不多，但是也有没测试到的地方。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，过滤的内容并不是很多，不算一个很安全的网页过滤代码。

1. **利用img方式实现跨站脚本攻击：**

一种使用<img>标签的脚本构造方法：

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

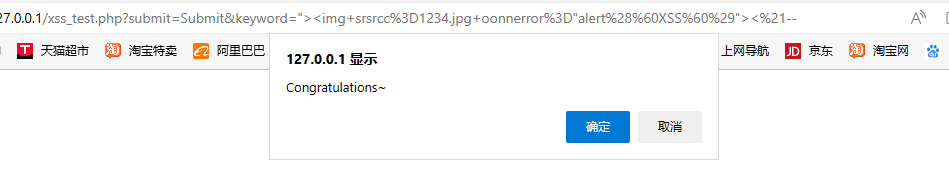
<img>标签是用来定义 HTML 中的图像，src 一般是图像的来源。而 onerror 事件会在文

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。

下面根据本实验源码中过滤的内容将上述 Payload 加工一番，就可以成功弹窗了。 Payload 是利用一些标签和事件组合构造的，本质是不变的，通过代码分析后我构造出下面的脚本：

脚本代码："><img srsrcc=1234.jpg oonnerror="alert(`XSS`)"><!--

结果如下：



构造的原理是会对src和onerror字段进行过滤，所以需要掩盖一层，其他原理在上文已经说明，这里图片只需要随便编一个名字即可。

1. **编写更加安全的过滤程序**

这里由于无论多写出多少过滤字段，都有方法被覆盖过去从而成功攻击，于是放弃通过增加过滤字段的添加来提高安全性，我这里考虑使用filter\_input函数进行输入过滤，利用fliter\_sanitize\_string过滤器来删除掉恶意的内容，只保留字符串形式的用户输入，代码如下：（红色是我新添加的代码）

<?php

ini\_set("display\_errors", 0);

$str =strtolower( $\_GET["keyword"]);

**$str =filter\_input(INPUT\_GET,"keyword",FILTER\_SANITIZE\_STRING);**

$str2=str\_replace("script","",$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>';

?>

保存修改，重新用上面的script和img方法进行攻击，结果如下：





结果都失败了，没能成功攻击，可见提高了过滤程序的安全性。

**心得体会：**

在进行跨站脚本（XSS）攻击实验时，我尝试了两种方法：使用 <script> 标签和 <img> 标签进行攻击，并成功地执行了攻击。这次实验让我对 XSS 攻击的原理和影响有了更深入的理解。

通过在 URL 参数中注入恶意脚本代码，我成功地利用了原始的代码中的漏洞，绕过了简单的过滤措施，让脚本在用户浏览器中执行。这使我意识到仅仅依靠字符替换等简单的过滤机制是不够安全的。

为了加强代码的安全性，我尝试编写了一个更加严格的过滤程序，成功在一定程度上提高了过滤代码的安全性，我希望能够阻止恶意脚本的注入并保护用户的安全。

然而，这次实验也让我认识到，确保系统的安全性是一个复杂的过程，并且没有绝对安全的系统。攻击者可以使用各种技术和方法来绕过过滤机制。因此，在实际开发中，我们需要依靠专业的安全工程师来提供更全面和深入的安全防护措施。

总的来说，这次实验让我意识到 XSS 攻击的严重性，并提醒我在开发过程中始终将安全性放在首要位置。不仅要注意输入过滤和输出编码，还要及时更新和修复漏洞，实施安全头部和会话管理等安全措施，以保护用户的数据和系统的完整性。