《软件安全》实验报告

姓名：蒋枘言　学号：2313546　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、实验过程

（一）源码部分

我们还是使用 Windows XP 虚拟机。在 phpnow\htdocs 路径下创建文件 xss_test.php，PHP 代码的内容如下：

php

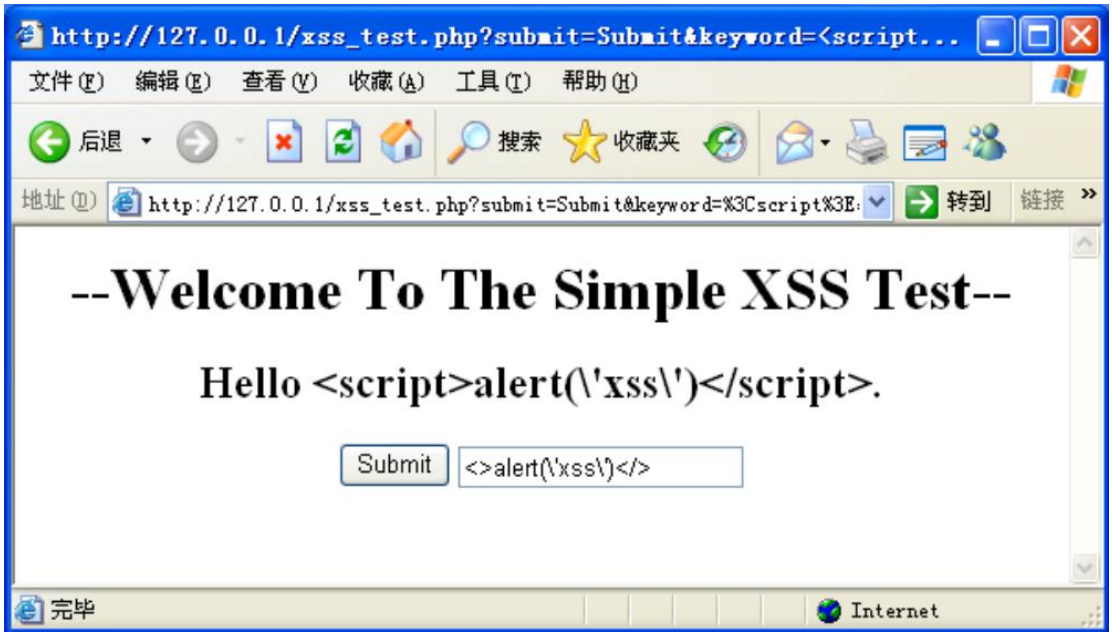
```
<!DOCTYPE html>
<head>
    <!-- 设置网页内容的编码格式为 UTF-8 -->
    <meta http-equiv="content-type" content="text/html;charset=utf-8">

    <script>
        // 重写默认的 alert 函数，当调用 alert 时，弹出一个 confirm 对话框
        window.alert = function() {
            confirm("Congratulations~");
        }
    </script>
</head>
<body>
    <!-- 页面标题，显示在页面中央 -->
    <h1 align=center>--Welcome To The Simple XSS Test--</h1>

    <?php
    // 关闭 PHP 错误显示，防止错误信息泄露
    ini_set("display_errors", 0);

    // 获取用户提交的 GET 参数 keyword 并转换为小写
    $str = strtolower($_GET["keyword"]);

    // 简单过滤：去除 "script" 关键字
    $str2 = str_replace("script", "", $str);
```

```
    // 进一步过滤：去除 "on"（如 onerror、onclick 等 XSS 事件）
    $str3 = str_replace("on", "", $str2);

    // 再过滤：去除 "src"（避免通过资源引入注入）
    $str4 = str_replace("src", "", $str3);

    // 输出结果，原始输入使用 htmlspecialchars 转义显示
    // 输入框中使用经过多次替换后的内容
    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>
```

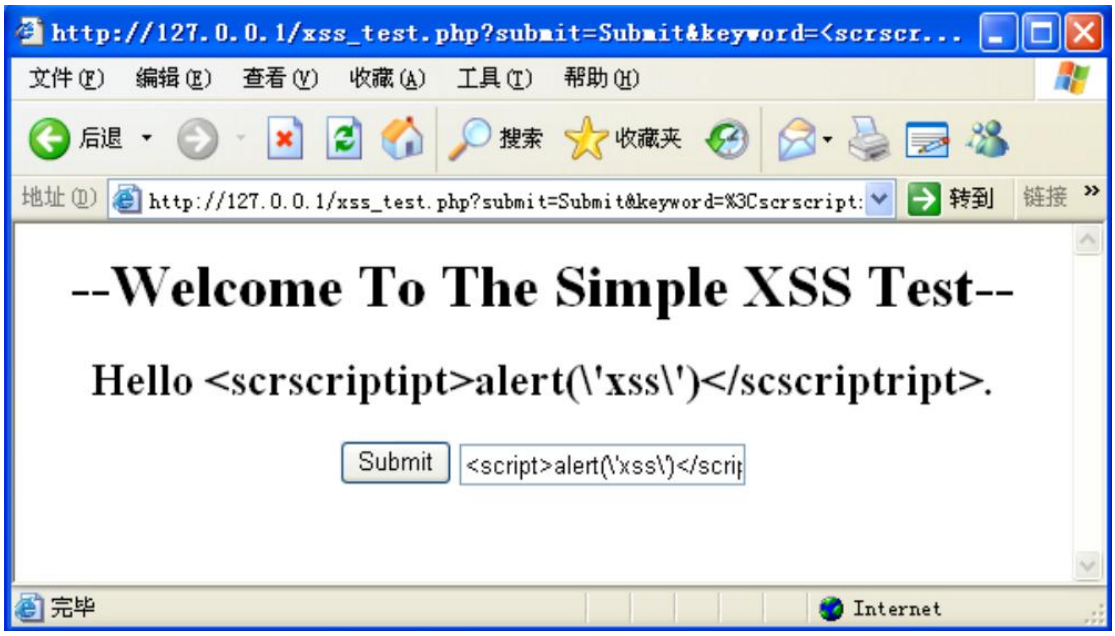
访问 http://127.0.0.1/xss_test.php，得到如下界面，代码运行正常。

（二）通过<script>标签的 XSS 攻击

首先输入 XSS 脚本<script>alert('xss')</script>进行测试。点击“Submit”按钮，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，输入 XSS 脚本<scrscriptipt>alert('xss')</scscriptript>进行测试，结果如下：

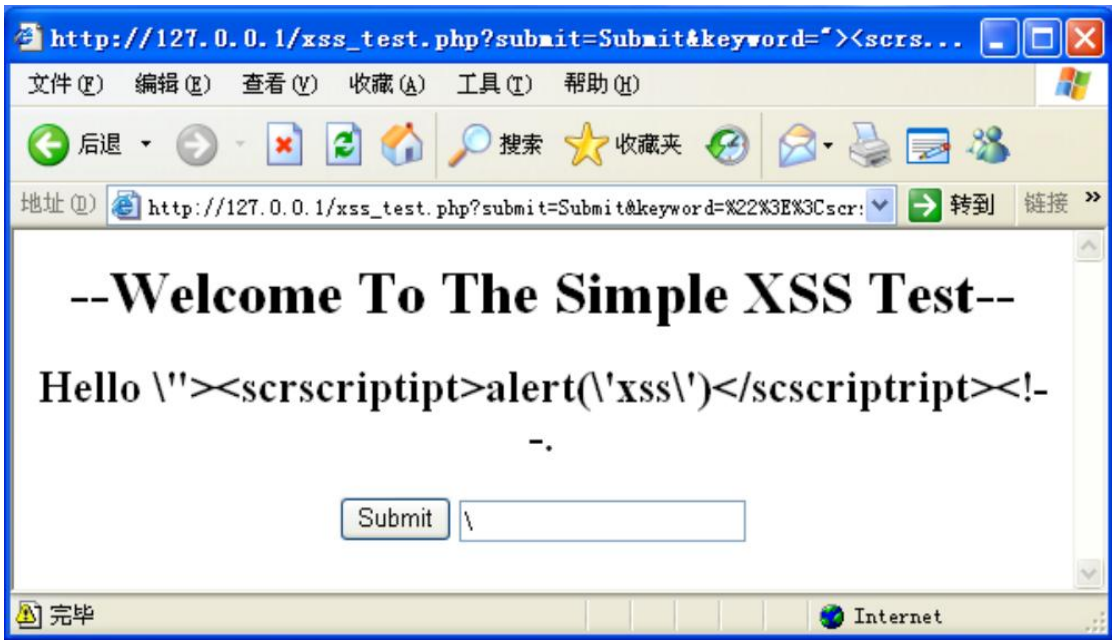


现在发现输入框中的回显确实是我们要攻击的脚本，但是它却没有执行。右键点击页面查看源代码：第 16 行的<input>标签是唯一能输入且有可能控制的地方。分析这行代码可知，虽然成功地插入了<script></script>标签，但是并没有跳出<input>标签，使得脚本仅可以回显而不能利用。这时的思路就是想办法将前面的<input>标签闭合，于是构造 XSS 脚本："><scrscriptipt>alert('XSS')</scscriptript><!--。

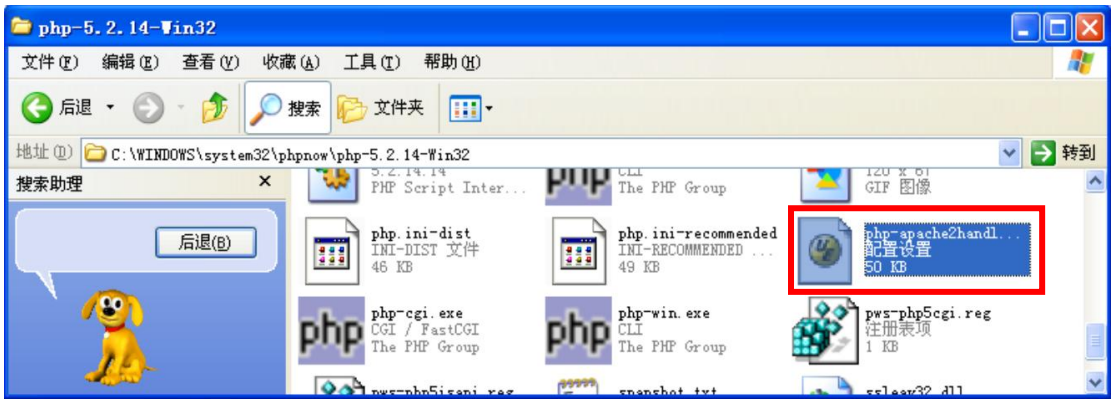
xss_test[1]

```
<!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>
</html>
```

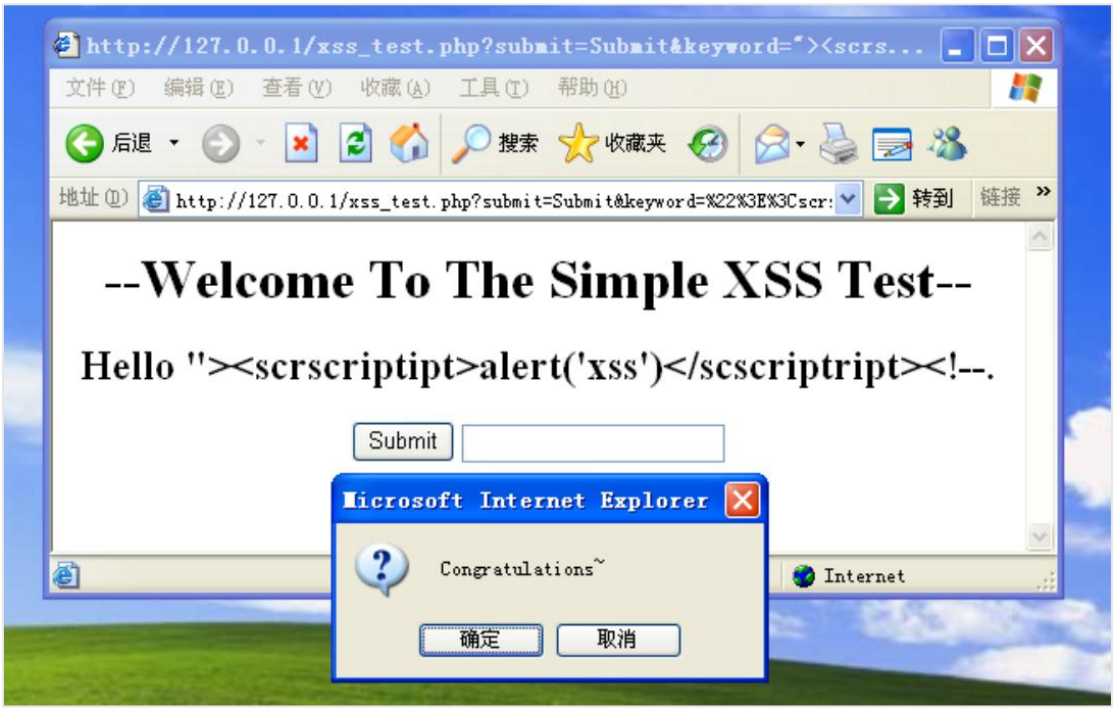
执行脚本"><scrscriptipt>alert('XSS')</scscriptript><!--，发现还是没有成功。这时输入的**双引号没有正常被处理**，是因为 PHP 服务器自动会对输入的双引号等**进行转义**，以预防用户构造特殊输入进行攻击，如本实验所进行的攻击。



为了确保实验可以成功运行，在 PHPnow 的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。（★★★一定要注意的是：php-apache2handler.ini 里面不止一个 magic_quotes_gpc，要修改的是 **Data Handling 部分**的 magic_quotes_gpc，不是 About this file 部分的 magic_quotes_gpc，否则还是会生成转义字符！★★★）



现在再来执行脚本"><scrscriptipt>alert('XSS')</scscriptript><!--，于是成功弹出了显示“Congratulations~”的对话框，说明跨站脚本攻击成功。




```
<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;
<form action=xss_test.php method=GET>
<input type=submit name=submit value=Submit />
<input name=keyword  value=""><script>alert('xss')</script><!--">
</form>
```

（三）通过<img>标签的 XSS 攻击

我们想要通过改造脚本<img src=ops! onerror="alert('XSS')">来进行攻击。其中，<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。仿照<script>标签，改造后的攻击脚本是"><img ssrcrc=ops! oonnerror="alert('XSS')"><!--。

执行脚本，成功弹出了显示“Congratulations~”的对话框，说明跨站脚本攻击成功。


```
<!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;img ssrcrc=ops! oonnerror=&quot;alert('xss')&q
<form action=xss_test.php method=GET>
<input type=submit name=submit value=Submit />
<input name=keyword  value=""><img src=ops! onerror="alert('xss')"><!--">
</form>
</center></body>
</html>
```

四、心得体会

在这次实验中，我初步理解了跨站脚本攻击（XSS）。在实验过程中，我成功地利用了**<script>**标签和**<img>**标签两种方式发起攻击，并通过**双写关键字**等方式绕过了简单的过滤机制。这让我更加意识到，对于**用户输入的处理**，过滤和转义的不足可能会导致严重的安全漏洞。在实际开发中，应该对用户输入进行全面过滤和编码，特别是在 HTML 属性、JavaScript 和 URL 等上下文中，绝对**不能直接信任**$_GET / $_POST 中的内容。

<script>标签和<img>标签两种攻击方式可以有如下对比：

对比项	<script> 标签攻击	<img> 标签攻击
触发条件	直接嵌入脚本，立即执行	需依赖事件触发（如加载失败）
隐蔽性	较低（<script> 易被过滤）	较高（利用合法标签属性）
绕过难度	较难（现代浏览器默认拦截内联脚本）	较易（依赖事件属性和编码）

想要进一步加强安全性，可以对源代码做出如下改进：

php

```php
<?php

// 自定义安全过滤函数
function secure_input($input) {
    // 定义允许的标签
    $allowed_tags =
'<b><i><u><em><strong><a><img><p><br><h1><h2><h3><ul><ol><li><blockquot
e>';

    // 对输入进行 HTML 标签过滤，移除不允许的标签
    $filtered_input = strip_tags($input, $allowed_tags);

    // 进一步转义所有 HTML 特殊字符
    // 转义 <, >, & 等特殊字符为 HTML 实体
    $filtered_input = htmlspecialchars($filtered_input, ENT_QUOTES,
'UTF-8');

    // 对 HTML 中的危险事件属性（如 onerror, onclick 等）进行清理
    // 移除事件属性
    $filtered_input = preg_replace('/on[a-z]+\s*=\s*["\'][^"\']*["\']/',
'', $filtered_input);

    return $filtered_input;
}

ini_set("display_errors", 0);

// 获取用户提交的 GET 参数 keyword
$str = $_GET["keyword"] ?? "";

// 使用更安全的过滤函数
$str_safe = secure_input($str);

// 输出经过安全处理后的内容
```

```
echo "<h2 align=center>Hello " . $str_safe . ".</h2>";
echo '<center>
<form action="xss_test.php" method="GET">
<input type="submit" name="submit" value="Submit" />
<input name="keyword" value="' . $str_safe . '">
</form>
</center>';
?>
```

我们自定义了 secure_input()安全输入函数：这个函数首先去除不允许的 HTML 标签，然后将剩下的 HTML 特殊字符进行转义，最后通过正则表达式去除任何可能的 JavaScript 事件处理属性。

1. 通过使用 strip_tags()函数，我们只允许特定的 HTML 标签存在（如<b>, <i>, <img>等），而对于其他标签（如<script>, <iframe>等）进行移除。

2. htmlspecialchars()将用户输入中的特殊字符（如 <, >, &等）转义为 HTML 实体，防止注入和 XSS 攻击。

3. 通过 preg_replace()删除所有 HTML 元素中的事件属性（如 onclick, onerror, onload 等），因为这些属性可能被用来执行恶意 JavaScript 代码。

这个更安全的过滤程序大大增强了对跨站脚本攻击（XSS）的防范能力。它不仅清除了不安全的 HTML 标签，还转义了所有危险字符并剥离了潜在的恶意事件处理属性。通过这种方法，能够有效避免 XSS 攻击，提高 Web 应用的安全性。