

软件安全实验报告

跨站脚本攻击

姓名：郭子涵 学号：2312145 班级：信息安全、法学双学位班

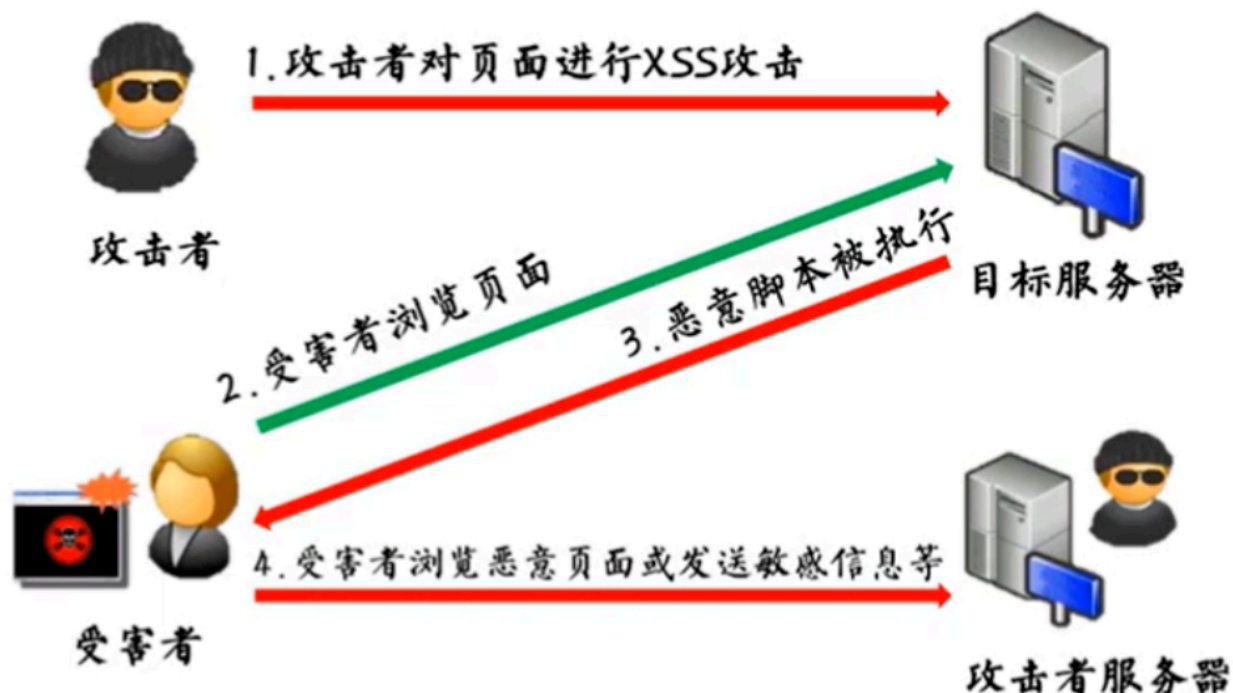
1 实验要求

复现课本第十一章实验三，通过img和script两类方法实现跨站脚本攻击，撰写实现报告。

2 实验背景

XSS(Cross-Site Scripting,跨站脚本攻击)是一种代码注入攻击。攻击者在目标网站上注入恶意代码，当用户(被攻击者)登录网站时就会执行这些恶意代码，通过这些脚本可以读取cookie,session tokens，或者网站其他敏感的网站信息，对用户进行钓鱼欺诈。

XSS攻击的原理是指攻击者在网页中嵌入客户端脚本，通常是JavaScript 编写的恶意代码，也有使用其他客户端脚本语言编写的。当用户使用浏览器浏览被嵌入恶意代码的网页时，恶意代码将会在用户的浏览器上执行。Javascript 可以用来获取用户的 Cookie、改变网页内容、URL 跳转，攻击者可以在 script 标签中输入 Javascript 代码，如 `alert(/xss/)`，实现一些“特殊效果”。其攻击过程可通过下图形象展示：



XSS分为反射型、存储型和DOM型:

1. 反射型也称为非持久型，这种类型的脚本是最常见的，也是使用最为广泛的一种，主要用于将恶意的脚本附加到URL地址的参数中。
2. 存储型：攻击者将已经构造完成的恶意页面发送给用户，用户访问看似正常的页面后收到攻击，这类XSS通常无法直接在URL中看到恶意代码，具有较强的持久性和隐蔽性。

3. DOM型XSS无需和后端交互，而是基于JavaScript上，JS解析URL中恶意参数导致执行JS代码

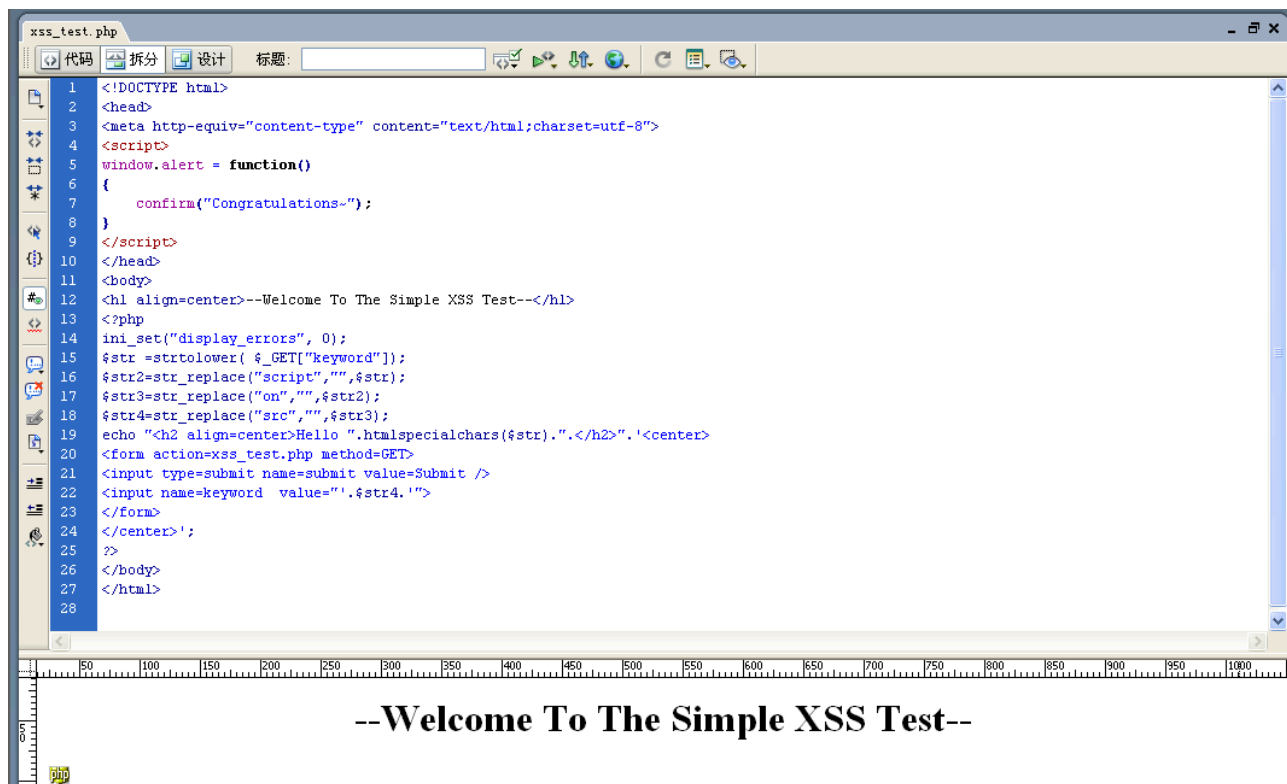
本次实验使用<scripts>和标签的两种方式实现简单的XSS攻击。

3 实验内容

3.1 scripts方法

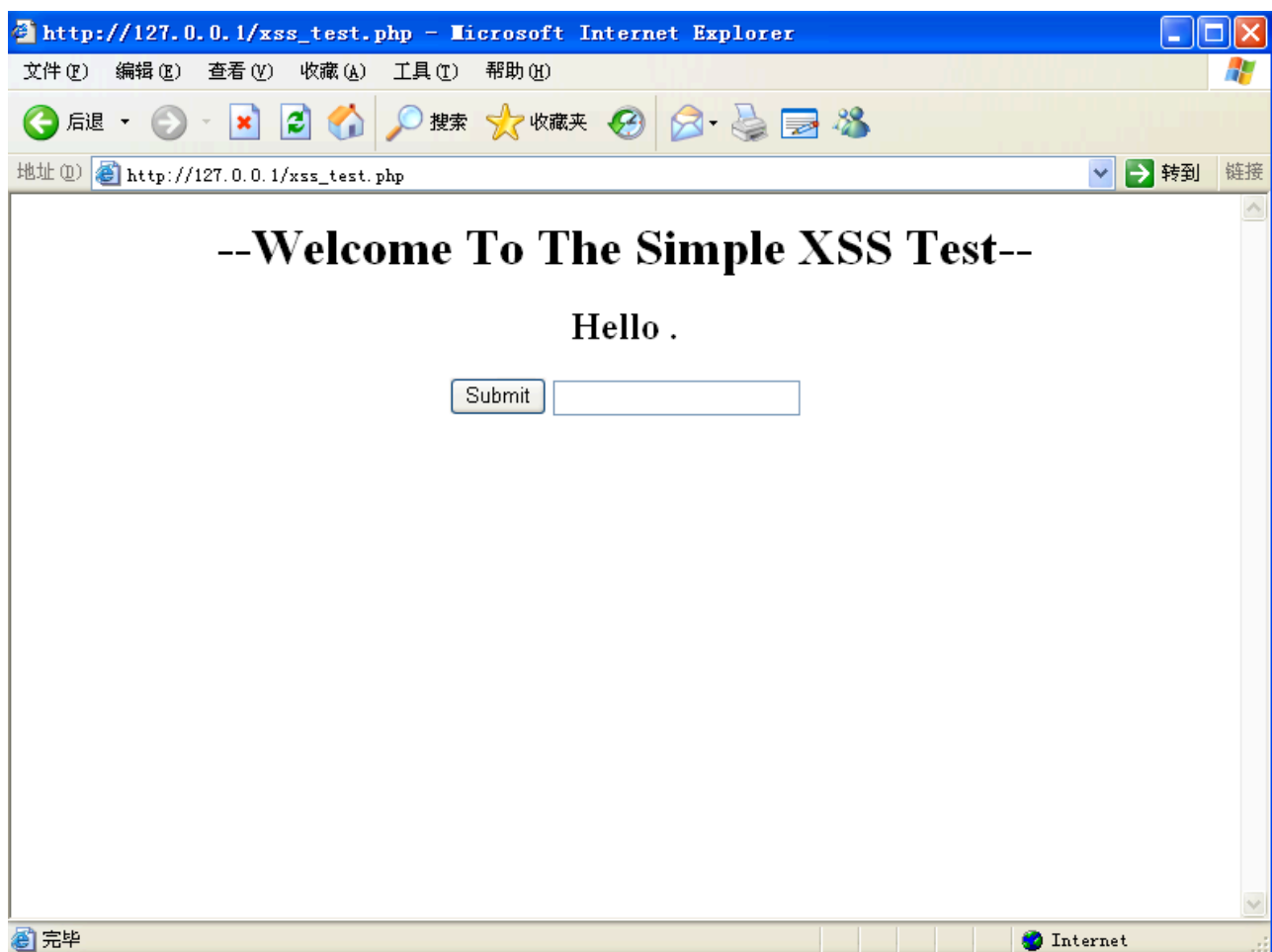
插入源代码：

```
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>". '<center>
20 <form action=xss_test.php method=GET>
21 <input type=submit name=submit value=Submit />
22 <input name=keyword value="'. $str4. '">
23 </form>
24 </center>';
25 ?>
26 </body>
27 </html>
```



```
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>". ' <center>
20 <form action=xss_test.php method=GET>
21 <input type=submit name=submit value=Submit />
22 <input name=keyword value="'. $str4. "'>
23 </form>
24 </center>';
25 ?>
26 </body>
27 </html>
28
```

连接数据库，进入该网页：



使用简单的XSS脚本<script>alert('xss')</script>来进行测试。点击Submit按钮以后，效果如下图所示，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：



这个时候我们考虑后台只是最简单的一次过滤。于是利用双写关键字绕过，构造脚本：
`<scrsriptipt>alert('xss')</scscriptript>`测试。执行效果如下图所示，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。



由于在黑盒测试情况下，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。我们右键查看源码如下：

```

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 <h2 align=center>Hello
    &quot;&gt;&lt;&lt;scrsriptipt&gt;alert('xss')&lt;/scscriptript&gt;&lt;!--</h2><center>
14 <form action=xss_test.php method=GET>
15 <input type=submit name=submit value=Submit />
16 <input name=keyword value=""><script>alert('xss')</script><!-->
17 </form>

```



```

; This directive tells PHP whether to declare the argv&argc variables (that
; would contain the GET information). If you don't use these variables, you
; should turn it off for increased performance.
register_argc_argv = Off

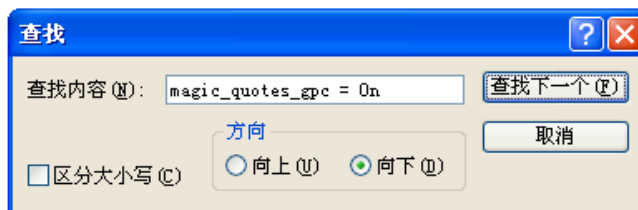
; When enabled, the SERVER and ENV variables are created when they're first
; used (Just In Time) instead of when the script starts. If these variables
; are not used within a script, having this directive on will result in a
; performance gain. The PHP directives register_globals, register_long_arrays,
; and register_argc_argv must be disabled for this directive to have any affect.
auto_globals_jit = On

; Maximum size of POST data
post_max_size = 32M

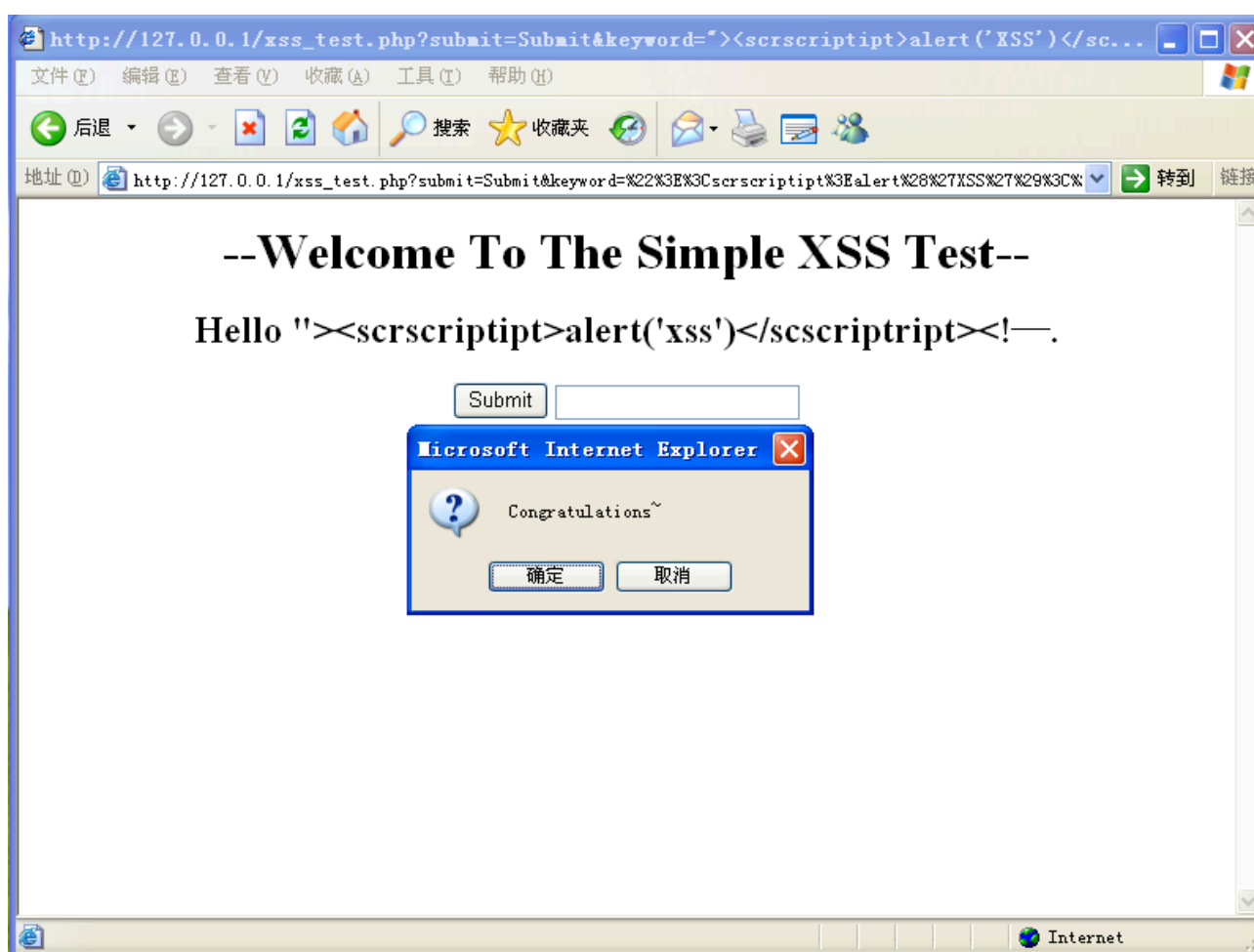
; Magic quotes
;

; Magic quotes for incoming
magic_quotes_gpc = On

```



再次执行脚本"><script>alert('XSS')</script><!--", 成功！



3.2 img方法

使用标签的脚本构造方法,源码如下:

```

1 <!DOCTYPE html>
2 <head>
3 <meta http-equiv="content-type" content="text/html; charset=utf-8">
4 <script>

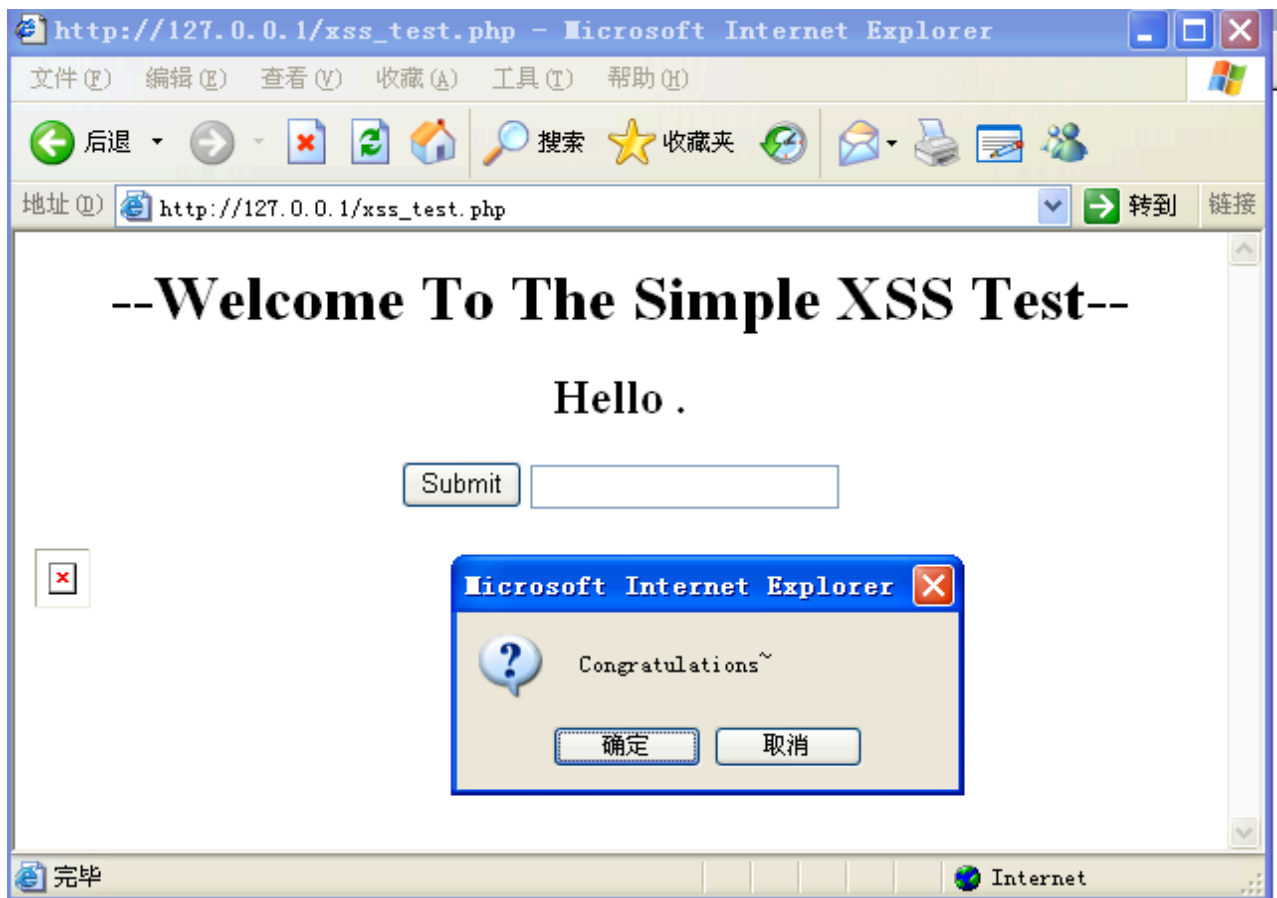
```

```

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><center>
20 <form action=xss_test.php method=GET>
21 <input type=submit name=submit value=Submit />
22 <input name=keyword value='" . htmlspecialchars($str4) . "'>
23 </form>
24 </center>";
25 ?>
26 
27
28 </body>
29 </html>

```

`` 是XSS攻击的payload，
`` 是图片标签，`src="nonexistent.jpg"` 指定了一个**错误的路径**（服务器上并没有这个文件）因此，浏览器尝试加载图片失败，触发onerror，执行
 JavaScript `onerror="alert('XSS')"` alert被重写为 `confirm("Congratulations~")`，浏览器
 显示弹窗，表示代码执行成功。实验成功截图如下：



4 心得体会

本次实验使用<script>和两种标签，实现简单的XSS攻击，其中<script>属于直接注入；属于隐式事件触发。通过本次实验理解了跨站脚本攻击的基本原理、攻击方式、实现条件等。理解了XSS本质上是HTML注入+JavaScript执行。浏览器在解析HTML内容时，若遇到可以执行脚本的位置（如<script>、事件属性），就会运行这些代码。

XSS攻击的核心在于：浏览器信任页面中的脚本，无论它是静态代码还是来自用户输入。执行流程大致总结如下：

1. 用户在输入框中提交了某段看似“合法”的代码；
2. 服务端未能充分过滤并对输入进行HTML编码，输出到了页面中；
3. 浏览器在解析HTML时，将某些标签或属性（如）当作脚本执行环境；
4. 最终执行恶意JavaScript代码，实现攻击行为（如弹窗、盗取Cookie、劫持会话等）

综上，本次实验收获颇丰，理解了浏览器的一些行为，尝试如何思考攻击面。