DKBA

Web 应用安全开发规范

本规范的相关系列规范或文件：

《C&C++语言安全编程规范》《Java 语言安全编程规范》

相关国际规范或文件一致性：

无

目 录 Table of Contents

Web 应用安全开发规范

1 概述

1.1 背景简介

在 Internet 大众化及 Web 技术飞速演变的今天，Web 安全所面临的挑战日益严峻。黑客攻击技术越来越成熟和大众化，针对 Web 的攻击和破坏不断增长，Web 安全风险达到了前所未有的高度。

许多程序员不知道如何开发安全的应用程序，开发出来的 Web 应用存在较多的安全漏洞，这些安全漏洞一旦被黑客利用将导致严重甚至是灾难性的后果。这并非危言耸听，类似的网上事故举不胜举，公司的 Web 产品也曾多次遭黑客攻击，甚至有黑客利用公司 Web 产品的漏洞敲诈运营商，造成极其恶劣的影响。

本规范就是提供一套完善的、系统化的、实用的 Web 安全开发方法供 Web 研发人员使用，以期达到提高 Web 安全的目的。本规范主要包括三大内容：Web 设计安全、Web 编程安全、Web 配置安全，配套 CBB，多管齐下，实现 Web 应用的整体安全性；本规范主要以 JSP/Java 编程语言为例。

1.2 技术框架

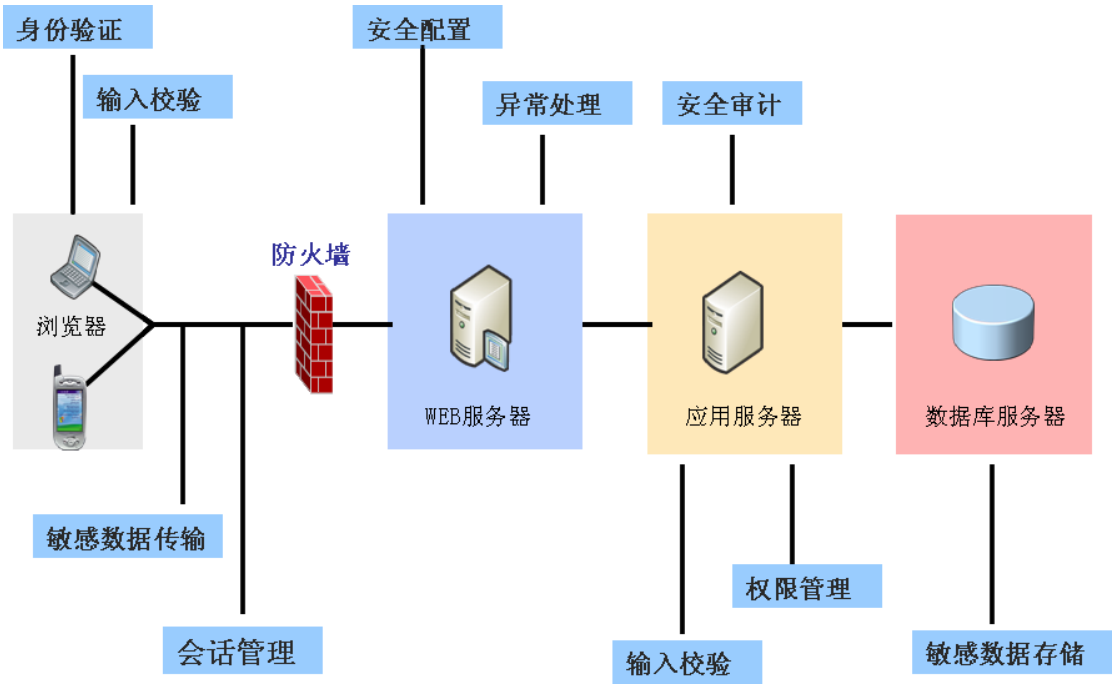

图1 典型的 Web 安全技术框架

图 1 显示了典型的 Web 安全的技术框架和安全技术点，这些安全技术点，贯穿整个 Web 设计开发过程。上图各个区域中存在任何一点薄弱环节，都容易导致安全漏洞。

由于 HTTP 的开放性，Web 应用程序必须能够通过某种形式的身份验证来识别用户，并确保身份验证过程是安全的，同样必须很好地保护用于跟踪已验证用户的会话处理机制。为了防止一些恶意输入，还要对输入的数据和参数进行校验。另外还要考虑 Web 系统的安全配置，敏感数据的保护和用户的权限管理，以及所有操作的安全审计。当然还要考虑代码安全，以及其他方面的威胁。

表 1 列出了一些 Web 缺陷类别，并针对每类缺陷列出了由于设计不当可能会导致的潜在问题。针对这些潜在的问题，本规范中有相应的解决措施。

表1 Web 应用程序缺陷和由于不良设计可能导致的问题

缺陷类别	由于不良设计可能导致的问题
身份验证	身份伪造、口令破解、权限提升和未授权访问。
会话管理	通过捕获导致会话劫持和会话伪造。
权限管理	访问机密或受限数据、篡改和执行未授权操作。
配置管理	未授权访问管理界面、更新配置数据、访问用户帐户和帐户配置文件。
敏感数据	机密信息泄漏和数据篡改。
加密技术	未授权访问机密数据或帐户信息。
安全审计	未能识别入侵征兆、无法证明用户的操作，以及在问题诊断中存在困难。
输入检验	通过嵌入查询字符串、窗体字段、Cookie 和 HTTP 标头中的恶意字符串所执行的攻击。包括命令执行、跨站点脚本编写 (XSS)、SQL 注入和缓冲区溢出攻击等。
参数操作	路径遍历攻击、命令执行、此外还有跳过访问控制机制、导致信息泄露、权限提升和拒绝服务。
异常管理	拒绝服务和敏感的系统级详细信息泄露。

1.3 使用对象

本规范的读者及使用对象主要为 Web 相关的需求分析人员、设计人员、开发人员、测试人员等。

1.4 适用范围

本规范的制定考虑了公司各种 Web 应用开发的共性 ,适合于公司绝大部分 Web 产品，

要求 Web 产品开发必须遵循。

对于嵌入式系统(如 ADSL Modem、硬件防火墙)中的 Web 应用 ,由于其特殊性(CPU、内存、磁盘容量有限，没有成熟的 Web 容器），不强制遵循本规范的所有内容，只需遵循以下章节的规则要求：

3.2 身份验证

3.3 会话管理

3.5 敏感数据保护

4.1 输入校验

4.2 输出编码

4.3 上传下载

4.5 代码注释

4.6 归档要求

1.5 用词约定

- ✧ **规则：**强制必须遵守的原则
- ✧ **建议：**需要加以考虑的原则
- ✧ **说明：**对此规则或建议进行相应的解释
- ✧ **实施指导：**对此规则或建议的实施进行相应的指导

2 常见 Web 安全漏洞

Web 应用的安全漏洞有很多，无法穷举。针对众多的 Web 漏洞，OWASP 的专家们结合各自在各领域的应用安全工作经验及智慧，提出了十大 Web 应用程序安全漏洞，帮助

人们关注最严重的漏洞。

注：OWASP 即开放 Web 应用安全项目，是一个旨在帮助人们理解和提高 Web 应用及服务安全性的项目组织。

十大 Web 应用程序安全漏洞列表

序号	漏洞名称	漏洞描述
1	注入	注入攻击漏洞，例如 SQL、OS 命令以及 LDAP(Lightweight Directory Access Protocol，轻量目录访问协议）注入。这些攻击发生在当不可信的数据作为命令或者查询语句的一部分，被发送给解释器的时候。攻击者发送的恶意数据可以欺骗解释器，以执行计划外的命令或者访问未被授权的数据。
2	跨站脚本	当应用程序收到含有不可信的数据，在没有进行适当的验证和转义的情况下，就将它发送给一个网页浏览器，这就会产生跨站脚本攻击（简称 XSS）。XSS 允许攻击者在受害者的浏览器上执行脚本，从而劫持用户会话、危害网站、或者将用户转向至恶意网站。
3	失效的身份认证和会话管理	与身份认证和会话管理相关的应用程序功能往往得不到正确的实现，这就导致了攻击者破坏密码、密匙、会话令牌或攻击其他的漏洞去冒充其他用户的身份。
4	不安全的直接对象引用	当开发人员暴露一个对内部实现对象的引用时，例如，一个文件、目录或者数据库密匙，就会产生一个不安全的直接对象引用。在没有访问控制检测或其他保护时，攻击者会操控这些引用去访问未授权数据。
5	跨站请求伪造	一个跨站请求伪造攻击迫使登录用户的浏览器将伪造的 HTTP 请求，包括该用户的会话 cookie 和其他认证信息，发送到一个存在漏洞的 Web 应用程序。这就允许了攻击者迫使用户浏览器向存在漏洞的应用程序发送请求，而这些请求会被应用程序认为是用户的合法请求。
6	安全配置错误	好的安全需要对应用程序、框架、应用程序服务器、Web 服务器、数据库服务器和平台，定义和执行安全配置。由于许多设置的默认值并不是安全的，因此，必须定义、实施和维护所有这些设置。这包括了对所有的软件保护及时地更新，包括所有应用程序的库文件。
7	失败的 URL 访问	许多 Web 应用程序在显示受保护的链接和按钮之前会检测 URL 访问权限。

	权限限制	但是，当这些页面被访问时，应用程序也需要执行类似的访问控制检测，否则攻击者将可以伪装这些 URL 去访问隐藏的网页。
8	未经验证的重定向和前转	Web 应用程序经常将用户重定向和前转到其他网页和网站，并且利用不可信的数据去判定目的页面。如果没有得到适当验证，攻击者可以将受害用户重定向到钓鱼网站或恶意网站，或者使用前转去访问未经授权的网页。
9	不安全的加密存储	许多 Web 应用程序并没有使用恰当的加密措施或 Hash 算法保护敏感数据，比如信用卡、社会安全号码（SSN）、认证凭据等。攻击者可能利用这种弱保护数据实行身份盗窃、信用卡欺骗或或其他犯罪。
10	传输层保护不足	应用程序时常没有身份认证、加密措施，甚至没有保护敏感网络数据的保密性和完整性。而当进行保护时，应用程序有时采用弱算法、使用过期或无效的证书，或不正确地使用这些技术。

3 Web 设计安全规范

3.1 Web 部署要求

规则 3.1.1：如果 Web 应用对 Internet 开放，Web 服务器应当置于 DMZ 区，在 Web 服务器与 Internet 之间，Web 服务器与内网之间应当有防火墙隔离，并设置合理的策略。

规则 3.1.2：如果 Web 应用对 Internet 开放，Web 服务器应该部署在其专用的服务器上，应避免将数据库服务器或其他核心应用与 Web 服务器部署在同一台主机上。

说明：Web 服务器比较容易被攻击，如果数据库或核心应用与 Web 服务器部署在同一台主机，一旦 Web 服务器被攻陷，那么数据库和核心应用也就被攻击者掌控了。

规则 3.1.3：Web 站点的根目录必须安装在非系统卷中。

说明：Web 站点根目录安装在非系统卷，如单独创建一个目录/home/Web 作为 Web 站点根目录，能够防止攻击者使用目录遍历攻击访问系统工具和可执行文件。

建议 3.1.1：Web 服务器与应用服务器需物理分离（即安装在不同的主机上），以提高应

用的安全性。

建议 3.1.2 :如果 Web 应用系统存在不同的访问等级(如个人帐号使用、客户服务、管理)，那么应该通过不同的 Web 服务器来处理来自不同访问等级的请求，而且 Web 应用应该鉴别请求是否来自正确的 Web 服务器。

说明：这样便于通过防火墙的访问控制策略和 Web 应用来控制不同访问等级的访问，比如通过防火墙策略控制，只允许内网访问管理 Portal。

建议 3.1.3 : 对于“客户服务”和“管理”类的访问，除了普通的认证，还应该增加额外的访问限制。

说明：额外的访问限制，可以限制请求来自企业内网，可以建立 VPN，或采用双向认证的 SSL；或采用更简单的办法，通过 IP 地址白名单对客户端的 IP 地址进行过滤判断，实施参考《附件 3 客户端 IP 鉴权实施指导》。

3.2 身份验证

3.2.1 口令

关于 Web 应用及容器涉及到的口令，请遵循《产品网络安全红线》的口令安全要求(详细内容请见：*附件 4 口令安全要求.xlsx*)。

3.2.2 认证

规则 3.2.2.1 : 对用户的最终认证处理过程必须放到应用服务器进行。

说明：不允许仅仅通过脚本或其他形式在客户端进行验证，必须在应用服务器进行最终认证处理（如果采用集中认证，那么对用户的最终认证就是放在集中认证服务器进行）。

规则 3.2.2.2：网页上的登录/认证表单必须加入验证码。

说明：使用验证码的目的是为了阻止攻击者使用自动登录工具连续尝试登录，从而降低被暴力破解的可能。如果觉得验证码影响用户体验，那么可以在前 3 次登录尝试中不使用验证码，3 次登录失败后必须使用验证码。验证码在设计上必须要考虑到一些安全因素，以免能被轻易地破解。具体实现细节请查看 3.2.3 验证码。如图：

图2 验证码

实施指导：

建议使用电信软件与核心网网络安全工程部提供的验证码 CBB。

备注：对于嵌入式系统，如果实现验证码比较困难，可以通过多次认证失败锁定客户端 IP 的方式来防止暴力破解。

规则 3.2.2.3：用户名、密码和验证码必须在同一个请求中提交给服务器，必须先判断验证码是否正确，只有当验证码检验通过后才进行用户名和密码的检验，否则直接提示验证码错误。

说明：如果验证码和用户名、密码分开提交，攻击者就可以绕过验证码校验（如：先手工提交正确的验证码，再通过程序暴力破解），验证码就形同虚设，攻击者依然可以暴力破解用户名及口令。

规则 3.2.2.4：所有登录页面的认证处理模块必须统一。

说明：可以存在多个登录页面，但是不允许存在多个可用于处理登录认证请求的模块，防止

不一致的认证方式。

规则 3.2.2.5：所有针对其他第三方开放接口的认证处理模块必须统一。

规则 3.2.2.6：认证处理模块必须对提交的参数进行合法性检查。

说明：具体输入校验部分请查看 4.1 输入校验。

规则 3.2.2.7：认证失败后，不能提示给用户详细以及明确的错误原因，只能给出一般性的提示。

说明：可以提示：“用户名或者口令错误，登录失败”；不能提示：“用户名不存在”、“口令必须是 6 位”等等。

规则 3.2.2.8：最终用户 portal 和管理 portal 分离。

说明：最终用户 portal 和管理 portal 分离，防止相互影响，防止来自用户面的攻击影响管理面。

实施指导：

将最终用户 portal 和管理 portal 分别部署在不同的物理服务器 如果为了解决成本合设(部署在同一台物理服务器上），那么，必须做到端口分离（通过不同的端口提供 Web 服务），一般的 Web 容器（如 tomcat）支持为不同的 Web 应用创建不同的端口。

规则 3.2.2.9：禁止在系统中预留任何的后门帐号或特殊的访问机制。

规则 3.2.2.10：对于重要的管理事务或重要的交易事务要进行重新认证，以防范会话劫持和跨站请求伪造给用户带来损失。

说明：重要的管理事务，比如重新启动业务模块；重要的交易事务，比如转账、余额转移、充值等。重新认证，比如让用户重新输入口令。

规则 3.2.2.11：用户名和密码认证通过后，必须更换会话标识，以防止会话固定（session

fixation）漏洞。

实施指导：

场景一：对于从 HTTP 转到 HTTPS 再转到 HTTP（也就是仅在认证过程采用 HTTPS，认证成功后又转到 HTTP）的，在用户名和密码认证通过后增加以下行代码：

```
request.getSession().invalidate();
HttpSession newSession=request.getSession(true);
Cookie cookie = new Cookie("JSESSIONID",newSession.getId());
cookie.setMaxAge(-1);
cookie.setSecure(false);
cookie.setPath(request.getContextPath());
response.addCookie(cookie);
```

场景二：对于全程采用 HTTPS 协议，或者全程采用 HTTP 协议的，在用户名和密码认证通过后增加以下行代码：

```
request.getSession().invalidate();
request.getSession(true);
```

建议 3.2.2.1：管理页面建议实施强身份认证。

说明：如双因素认证、SSL 双向证书认证、生物认证等；还可以通过应用程序限制只允许某些特定的 IP 地址访问管理页面，并且这些特定的 IP 地址可配置。

建议 3.2.2.2：同一客户端在多次连续尝试登录失败后，服务端需要进行用户帐号或者是客户端所在机器的 IP 地址的锁定策略，且该锁定策略必须设置解锁时长，超时后自动解锁。

说明：

登录失败应该提示用户：如果重试多少次不成功系统将会锁定。在锁定期间不允许该用户帐号（或者客户端所在机器的 IP 地址）登录。

允许连续失败的次数(指从最后一次成功以来失败次数的累计值)可配置，取值范围为：0-99 次，0 表示不执行锁定策略，建议默认：5 次。

锁定时长的取值范围为：0-999 分钟，建议默认：30 分钟，当取值为 0 时，表示无限期锁定，只能通过管理员手动解锁（需要提供管理员对服务器锁定其它用户帐号/IP 进行解锁的功能界面）。建议优先使用帐号锁定策略。

注意：应用程序的超级用户帐号不能被锁定，只能锁定操作的客户端所在的 IP，这是为了防止系统不可用。特别说明：锁客户端 IP 策略存在缺陷，当用户使用 proxy 上网时，那么锁定客户端 IP 会导致使用该 proxy 上网的所有用户在 IP 锁定期间都不能使用该 Web 应用；锁定用户帐户的策略也存在缺陷，当攻击者不断尝试某帐户的口令，就给该帐户带来拒绝服务攻击，使该帐户不可用。

3.2.3 验证码

规则 3.2.3.1：验证码必须是单一图片，且只能采用 JPEG、PNG 或 GIF 格式。

说明：验证码不能使用文本格式，不允许多图片组合（如用四个图片拼成的验证码）。

规则 3.2.3.2：验证码内容不能与客户端提交的任何信息相关联。

说明：在使用验证码生成模块时不允许接收来自客户端的任何参数，例如：禁止通过 getcode.jsp?code=1234 的 URL 请求，将 1234 作为验证码随机数。

规则 3.2.3.3：验证码模块生成的随机数不能在客户端的静态页面中的网页源代码里出现。

说明：在客户端网页上点击鼠标右键、选择“查看源文件”时，必须看不到验证码模块生成的随机数。

规则 3.2.3.4：验证码字符串要求是随机生成，生成的随机数必须是安全的。

说明：对于 java 语言可以使用类 java.security.SecureRandom 来生成安全的随机数。

规则 3.2.3.5：验证码要求有背景干扰，背景干扰元素的颜色、位置、数量要求随机变化。

规则 3.2.3.6：验证码在一次使用后要求立即失效，新的请求需要重新生成验证码。

说明：进行验证码校验后，立即将会话中的验证码信息清空，而不是等到生成新的验证码时再去覆盖旧的验证码，防止验证码多次有效；注意：当客户端提交的验证码为空，验证不通过。

说明：

以上规则可以通过使用电信软件与核心网网络安全工程部提供的验证码 CBB 来实现。

3.3 会话管理

规则 3.3.1：使用会话 cookie 维持会话。

说明 :目前主流的 Web 容器通过以下几种方式维持会话 隐藏域、URL 重写、持久性 cookie、会话 cookie，但通过隐藏域、URL 重写或持久性 cookie 方式维持的会话容易被窃取，所以要求使用会话 cookie 维持会话。如果条件限制必须通过持久性 cookie 维持会话的话，那么 cookie 信息中的重要数据部分如身份信息、计费信息等都必须进行加密。（cookie 有两种：会话 cookie 和持久性 cookie；会话 cookie，也就是非持久性 cookie，不设置过期时间，其生命期为浏览器会话期间，只要关闭浏览器窗口，cookie 就消失了；会话 cookie 一般不存储在硬盘上而是保存在内存里。持久性 cookie，设置了过期时间，被浏览器保存到硬盘上，关闭后再次打开浏览器，持久性 cookie 仍然有效直到超过设定的过期时间。）

备注：对于嵌入式系统的 Web，不适合本条规则，按“**规则 3.3.9**”实施。

规则 3.3.2：会话过程中不允许修改的信息，必须作为会话状态的一部分在服务器端存储和维护。

说明：会话过程中不允许修改的信息，例如，当用户通过认证后，其用户标识在整个会话过程中不能被篡改。禁止通过隐藏域或 URL 重写等不安全的方式存储和维护。对 JSP 语言，就是应该通过 session 对象进行存储和维护。

规则 3.3.3：当 Web 应用跟踪到非法会话，则必须记录日志、清除会话并返回到认证界面。

说明： 非法会话的概念就是通过一系列的服务端合法性检测（包括访问未授权资源，缺少必要参数等情况），最终发现的不是正常请求产生的会话。

规则 3.3.4：禁止使用客户端提交的未经审核的信息来给会话信息赋值。

说明：防止会话信息被篡改，如恶意用户通过 URL 篡改手机号码等。

规则 3.3.5：当用户退出时，必须清除该用户的会话信息。

说明：防止遗留在内存中的会话信息被窃取，减少内存占用。

实施指导：对于 JSP 或 java 语言使用如下语句：request.getSession().invalidate();

规则 3.3.6：必须设置会话超时机制，在超时过后必须要清除该会话信息。

说明：建议默认会话超时时间为 10 分钟（备注：对于嵌入式系统中的 Web，建议默认超时时间为 5 分钟，以减少系统资源占用）。如果没有特殊需求，禁止使用自动发起请求的机制来阻止 session 超时。

规则 3.3.7：在服务器端对业务流程进行必要的流程安全控制，保证流程衔接正确，防止关键鉴别步骤被绕过、重复、乱序。

说明：客户端流程控制很容易被旁路（绕过），因此流程控制必须在服务器端实现。

实施指导：可以通过在 session 对象中创建一个表示流程当前状态的标识位，用 0、1、2、3、…、N 分别表示不同的处理步骤，标识位的初始值为 0，当接收到步骤 N 的处理请求时，判断该标识位是否为 N-1，如果不为 N-1，则表示步骤被绕过（或重复或乱序），拒绝受理，否则受理，受理完成后更改标识位为 N。

规则 3.3.8：所有登录后才能访问的页面都必须有明显的“注销（或退出）”的按钮或菜单，如果该按钮或菜单被点击，则必须使对应的会话立即失效。

说明：这样做是为了让用户能够方便地、安全地注销或退出，减小会话劫持的风险。

规则 3.3.9：如果产品（如嵌入式系统）无法使用通用的 Web 容器，只能自己实现 Web 服务，那么必须自己实现会话管理，并满足以下要求：

- 采用会话 cookie 维持会话。
- 生成会话标识（session ID）要保证足够的随机、离散，以便不能被猜测、枚举，要求 session ID 要至少要 32 字节，要支持字母和数字字符集。
- 服务端必须对客户端提交的 session ID 的有效性进行校验。

说明：在嵌入式系统中部署 Web 应用，由于软硬件资源所限，往往无法使用通用的 Web 容器及容器的会话管理功能，只能自己实现。另外，为了节省内存，嵌入式 webserver 进程往往是动态启动，为了使 session 更快的超时，建议增加心跳机制，对客户端浏览器是否关闭进行探测，5s 一个心跳，30s 没有心跳则 session 超时，关闭该 session。

3.4 权限管理

规则 3.4.1：对于每一个需要授权访问的页面或 servlet 的请求都必须核实用户的会话标识是否合法、用户是否被授权执行这个操作。

说明：防止用户通过直接输入 URL，越权请求并执行一些页面或 servlet；建议通过过滤器实现。

实施指导： 请参考“*附件 5 Web 权限管理设计规格说明书.docx*”。

规则 3.4.2：授权和用户角色数据必须存放在服务器端，不能存放在客户端，鉴权处理也必须在服务器端完成。

说明：禁止将授权和角色数据存放在客户端中（比如 cookie 或隐藏域中），以防止被篡改。

规则 3.4.3：一个帐号只能拥有必需的角色和必需的权限。一个组只能拥有必需的角色和必需的权限。一个角色只能拥有必需的权限。

说明：做到权限最小化和职责分离（职责分离就是分清帐号角色，系统管理帐号只用于系统管理，审计帐号只用于审计，操作员帐号只用于业务维护操作，普通用户帐号只能使用业务。）这样即使帐号被攻击者盗取，也能把安全损失控制在最小的限度。

规则 3.4.4：对于运行应用程序的操作系统帐号，不应使用“root”、“administrator”、“supervisor”等特权帐号或高级别权限帐号，应该尽可能地使用低级别权限的操作系统帐号。

规则 3.4.5：对于应用程序连接数据库服务器的数据库帐号，在满足业务需求的前提下，必须使用最低级别权限的数据库帐号。

说明：根据业务系统要求，创建相应的数据库帐号，并授予必需的数据库权限。不能使用“sa”、“sysman”等管理帐号或高级别权限帐号。

3.5 敏感数据保护

3.5.1 敏感数据定义

敏感数据包括但不限于：口令、密钥、证书、会话标识、License、隐私数据（如短消息的内容）、授权凭据、个人数据（如姓名、住址、电话等）等，在程序文件、配置文件、日志文件、备份文件及数据库中都有可能包含敏感数据。

3.5.2 敏感数据存储

规则 3.5.2.1：禁止在代码中存储敏感数据。

说明：禁止在代码中存储如数据库连接字符串、口令和密钥之类的敏感数据，这样容易导致泄密。用于加密密钥的密钥可以硬编码在代码中。

规则 3.5.2.2：禁止密钥或帐号的口令以明文形式存储在数据库或者文件中。

说明：密钥或帐号的口令必须经过加密存储。例外情况，如果 Web 容器的配置文件中只能以明文方式配置连接数据库的用户名和口令，那么就不用强制遵循该规则，将该配置文件的属性改为只有属主可读写。

规则 3.5.2.3：禁止在 cookie 中以明文形式存储敏感数据。

说明：cookie 信息容易被窃取，尽量不要在 cookie 中存储敏感数据；如果条件限制必须使用 cookie 存储敏感信息时，必须先对敏感信息加密再存储到 cookie。

规则 3.5.2.4：禁止在隐藏域中存放明文形式的敏感数据。

规则 3.5.2.5：禁止用自己开发的加密算法，必须使用公开、安全的标准加密算法。

实施指导：

场景 1：后台服务端保存数据库的登录口令

后台服务器登录数据库需要使用登录数据库的明文口令，此时后台服务器加密保存该口令后，下次登录时需要还原成明文，因此，在这种情况下，不可用不可逆的加密算法，而需要使用对称加密算法或者非对称加密算法，一般也不建议采用非对称加密算法。

推荐的对称加密算法：AES128、AES192、AES256。

场景 2：后台服务端保存用户的登录口令

在该场景下，一般情况是：客户端提交用户名及用户口令，后台服务端对用户名及用户口令进行验证，然后返回验证的结果。此时，在后台服务端，用户口令可以不需要还原，因此建议使用不可逆的加密算法，对“用户名+口令”字符串进行加密。

推荐的不可逆加密算法： SHA256、SHA384、SHA512，HMAC-SHA256、HMAC-SHA384、HMAC-SHA512。

规则 3.5.2.6：禁止在日志中记录明文的敏感数据。

说明：禁止在日志中记录明文的敏感数据（如口令、会话标识 jsessionid 等）， 防止敏感信息泄漏。

规则 3.5.2.7：禁止带有敏感数据的 Web 页面缓存。

说明：带有敏感数据的 Web 页面都应该禁止缓存，以防止敏感信息泄漏或通过代理服务器上网的用户数据互窜问题。

实施指导：

在 HTML 页面的<HEAD>标签内加入如下代码：

<HEAD>

```
        <META HTTP-EQUIV="Expires" CONTENT="0">
        <META HTTP-EQUIV="Pragma" CONTENT="no-cache">
        <META HTTP-EQUIV="Cache-control" CONTENT="no-cache">
        <META HTTP-EQUIV="Cache" CONTENT="no-cache">
</HEAD>
```

在 JSP 页面的最前面加入如下代码：

```
<%
    response.setHeader("Cache-Control","no-cache");
    response.setHeader("Pragma","no-cache");
    response.setDateHeader("Expires",0);
%>
```

注意 :以上代码对于采用强制缓存策略的代理服务器不生效(代理服务器默认是不缓存的），要防止代理服务器缓存页面，可以在链接后加入一个随机数 pageid,此时链接变成：http://localhost:8080/query.do?a=2&pageid=2245562, 其中 2245562 数字是随机生成的，每次请求此页面时，随机数都不同，IE 始终认为此为一个新请求，并重新解析，生成新的响应页面。

3.5.3 敏感数据传输

规则 3.5.3.1：带有敏感数据的表单必须使用 HTTP-POST 方法提交。

说明：禁止使用 HTTP-GET 方法提交带有敏感数据的表单（form），因为该方法使用查询字符串传递表单数据，易被查看、篡改。如果是使用 servlet 处理提交的表单数据，那么

不在 doGet 方法中处理，只在 doPost 方法处理。

实施指导：

```
1. 对于 JSP 页面，将表单的属性 method 赋值为"post"，如下
    <form name="form1" method="post" action="switch.jsp">
2. 如果是使用 servlet 处理提交的表单数据，那么只在 doPost 方法中处理，参考代码如下
public class ValidationServlet extends HttpServlet
{
     public void doPost(HttpServletRequest request, HttpServletResponse
response)
                throws IOException, ServletException
    {
        //对提交的表单数据进行校验
    }
}
```

规则 3.5.3.2：在客户端和服务器间传递明文的敏感数据时，必须使用带服务器端证书的 SSL。

说明：如果在客户端和服务器间传递如帐号、口令等明文的敏感数据，必须使用带服务器端证书的 SSL。由于 SSL 对服务端的 CPU 资源消耗很大，实施时必须考虑服务器的承受能力。

实施指导：

```
1. SSL 的配置请参考《附件 1 Tomcat 配置 SSL 指导》。
2. Web 应用中，从 https 切换到 http 过程中会丢失 session，无法保持会话的连续。解决
```

的办法就是用 http-https-http 过程代替 https-http 过程，保证会话的连续性。原因：当 https 请求转为 http 请求的时候，因为原先的 session 的 secure 属性值是 true，无法再 http 协议中传输，因此，系统生成新的 session，且新的 session 没有继承旧 session 的属性和值，因此，无法保持会话连续。而 http-https-http 这个过程，session 始终不变，因此，可以保持会话连续。

规则 3.5.3.3：禁止在 URL 中携带会话标识（如 jsessionid）。

说明：由于浏览器会保存 URL 历史记录，如果 URL 中携带会话标识，则在多人共用的 PC 上会话标识容易被其他人看到，一旦该会话标识还在其生命有效期，则恶意用户可以冒充受害用户访问 Web 应用系统。

规则 3.5.3.4：禁止将对用户保密的信息传送到客户端。

说明：这些信息一旦传送到客户端，那么用户也就可以获取到了。

3.6 安全审计

本节的安全审计是针对 Web 业务应用，不包括对操作系统、Web 容器的安全审计。对于操作系统和 Web 容器的安全审计，可以参考对应的操作系统安全基线和 Web 安全配置规范。

规则 3.6.1：应用服务器必须对安全事件及操作事件进行日志记录。

说明：安全事件包括登录、注销、添加、删除、修改用户、授权、取消权限、鉴权、修改用户口令等；操作事件包括对业务系统配置参数的修改，对重要业务数据的创建、删除、修改、查询等；对于上述事件的结果，不管是成功还是失败，都需要记录日志。

规则 3.6.2：安全日志必须包括但不限于如下内容：事件发生的时间、事件类型、客户端 IP、

客户端机器名、当前用户的标识、受影响的个体（数据、资源）、成功或失败标识、启动该事件的进程标识以及对该事件的详细描述。

规则 3.6.3：严格限制对安全日志的访问。

说明：只有 Web 应用程序的管理员才能查询数据库表形式或文件形式的安全日志；除数据库超级管理员外，只有应用程序连接数据库的帐号可以查询（select）及插入（insert）安全日志表；除操作系统超级管理员外，只有应用程序的运行帐户才能读、写文件形式的安全日志（但不允许删除）。确保日志的安全，限制对日志的访问，这加大了攻击者篡改日志文件以掩饰其攻击行为的难度。

规则 3.6.4：对日志模块占用资源必须有相应的限制机制。

说明：限制日志模块占用的资源，以防止如自动的恶意登陆尝试导致的资源枯竭类 DOS 攻击；比如限制日志记录占用的磁盘空间。

规则 3.6.5：禁止日志文件和操作系统存储在同一个分区中，同时，应使用转储、滚动、轮循机制，来防止存储日志的分区写满。

说明：所需空间和具体业务、局点容量、日志保存周期相关，要根据实际情况估算。

建议 3.6.1：安全日志应该有备份及清理机制。

说明 :备份及清理机制包括定期备份及清理安全日志和监控用于存放安全日志的磁盘空间的使用情况。可以配置定期备份及清理的时间，可以配置以用于存放安全日志的磁盘空间使用率达到多少时进行备份及清理。

建议 3.6.2：通过网络形式保存安全日志。

说明：在生成安全日志时，即时将日志保存到网络上其他主机，而且生成安全日志的应用程序不能再访问存放在其他主机的日志。

3.7 Web Service

规则 3.7.1：对 Web Service 接口的调用必须进行认证。

说明：认证就是确定谁在调用 Web Service，并且证实调用者身份。

实施指导：

可以通过在消息头中增加用户名和口令，作为认证凭据；

对于安全性要求不高、只向同一信任域内其他主机开放的 Web Service 接口，可以通过简单的 IP 认证来实现接口的认证（只有服务器端指定 IP 地址的客户端才允许调用，IP 地址可配置）。

规则 3.7.2：如果调用者的权限各不相同，那么必须对 Web Service 接口的调用进行鉴权。

说明：鉴权就是判断调用者是否有权限调用该 Web Service 接口。

实施指导：

可以通过 Axis 的 handler 对调用进行鉴权。

规则 3.7.3：通过 Web Service 接口传递敏感数据时，必须保障其机密性。

实施指导：

方案 1：请参考《附件 2 Web Service 安全接入开发指导》。

方案 2：采用 https 安全协议。

规则 3.7.4：通过 Web Service 接口传递重要的交易数据时，必须保障其完整性和不可抵赖性。

说明：重要的交易数据，如转账时涉及的“转入账号”、“转出账号”、“金额”等。

实施指导：

请参考《附件 2 Web Service 安全接入开发指导》。

规则 3.7.5：如果 Web Service 只对特定的 IP 开放，那么必须对调用 Web Service 接口的客户端 IP 进行鉴权，只有在 IP 地址白名单中的客户端才允许调用，IP 地址白名单可配置。

实施指导：请参考《附件 3 客户端 IP 鉴权实施指导》。

规则 3.7.6：对 Web Service 接口调用进行日志记录。

说明：日志内容包括但不限于如下内容：调用时间、操作类型、调用接口名称、详细的接口参数、客户端 IP、客户端机器名、调用者的用户标识、受影响的个体（数据、资源）、成功或失败标识。

规则 3.7.7：必须对 Web Service 提交的参数进行输入校验。

说明：具体输入校验部分请查看 4.1 输入校验。

3.8 RESTful Web Service

RESTful Web Service（也称为 RESTful Web API）是一个使用 HTTP 并遵循 REST 原则的 Web 服务。

规则 3.8.1：对 RESTful Web Service 的调用必须进行认证。

说明：认证就是确定谁在调用 RESTful Web Service，并且证实调用者身份。

实施指导：

客户端发起的 Restful 请求需要在消息头带 Authorization 字段，内容填 Basic Base64(user:pass)，服务端对 user 和 passwd 进行认证。

注意：user 和 pass 必须加密保存在配置文件或数据库中，不能写死在代码中；传输时采用 https 安全协议。

对于安全性要求不高、只向同一信任域内其他主机开放的 Web Service 接口，可以通过简

单的 IP 认证来实现接口的认证（只有服务器端指定 IP 地址的客户端才允许调用，IP 地址可配置）。

规则 3.8.2：如果调用者的权限各不相同，那么必须对 RESTful Web Service 的调用进行鉴权。

说明：鉴权就是判断调用者是否有权限调用该 RESTful Web Service。

规则 3.8.3：通过 RESTful Web Service 传递敏感数据时，必须保障其机密性。

实施指导：

采用 https 安全协议。

规则 3.8.4：如果 RESTful Web Service 只对特定的 IP 开放，那么必须对调用 RESTful Web Service 的客户端 IP 进行鉴权，只有在 IP 地址白名单中的客户端才允许调用，IP 地址白名单可配置。

实施指导：

请参考《附件 3 客户端 IP 鉴权实施指导》。

规则 3.8.5：对 RESTful Web Service 调用进行日志记录。

说明：日志内容包括但不限于如下内容：调用时间、操作类型、调用接口名称、详细的接口参数、客户端 IP、客户端机器名、调用者的用户标识、受影响的个体（数据、资源）、成功或失败标识。

规则 3.8.6：必须对 RESTful Web Service 提交的参数进行输入校验。

说明：具体输入校验部分请查看 4.1 输入校验。

3.9 DWR

DWR（Direct Web Remoting）是一种 Java 和 JavaScript 相结合的开源框架，可以帮助开发人员更容易地完成应用 Ajax 技术的 Web 应用程序，让浏览器上的 JavaScript 方法调用运行在 Web 服务器上的 Java 方法。

规则 3.9.1：关闭 DWR 调试功能。

说明：如果开启了 DWR 调试功能，那么攻击者可以轻易查看和调用系统提供的所有 DWR 接口，所以，版本发布时，一定要关闭 DWR 调试功能。

实施指导：

修改对应的 web.xml 文件中的 debug 参数值为 false：

```
<servlet>
  <servlet-name>dwr-invoker</servlet-name>
  <servlet-class>org.directwebremoting.servlet.DwrServlet</servlet-class>
  <init-param>
    <param-name>debug</param-name>
    <param-value>false</param-value>
  </init-param>
.....
```

规则 3.9.2：对 DWR 接口的调用必须进行认证。

说明：认证就是确定谁在调用 DWR 接口，并且证实调用者身份。

实施指导：

对于 DWR 接口的认证直接沿用 3.2.2 认证机制，不用单独再做认证。

规则 3.9.3：对 DWR 接口的调用必须进行鉴权。

说明：鉴权就是判断调用者是否有权限调用该 DWR 接口。

实施指导：

DWR 的请求和普通的 Web 请求一样，都可以通过过滤器来鉴权，对于 DWR 接口的鉴权直接沿用**规则 3.4.1** 的鉴权机制，具体实现参照**规则 3.4.1** 的实施指导。

规则 3.9.4：必须对 DWR 提交的参数进行输入校验。

说明：具体输入校验部分请查看 4.1 输入校验。

4 Web 编程安全规范

4.1 输入校验

规则 4.1.1：必须对所有用户产生的输入进行校验，一旦数据不合法，应该告知用户输入非法并且建议用户纠正输入。

说明：用户产生的输入是指来自 text、password、textareas 或 file 表单域的数据；必须假定所有用户产生的输入都是不可信的，并对它们进行合法性校验。

规则 4.1.2：必须对所有服务器产生的输入进行校验，一旦数据不合法，必须使会话失效，并记录告警日志。

说明：服务器产生的输入是指除用户产生的输入以外的输入，例如来自 hidden fields、selection boxes、check boxes、radio buttons、cookies、HTTP headers、热点链接包含的 URL 参数的数据或客户端脚本等；必须假定所有服务器产生的输入都是被篡改过的、恶意的，并对它们进行合法性校验，如果不合法，说明有人恶意篡改数据。举例：假如用户资料填写表单中的“性别”为必填项，用 radio button（ ‘男’和‘女’对应实际值分别

为‘1’和‘0’）来限制用户的输入，如果应用程序收到的“性别”值为‘2’，那么可以断定有人恶意篡改数据。

规则 4.1.3：禁止将 HTTP 标题头中的任何未加密信息作为安全决策依据。

说明：HTTP 标题头是在 HTTP 请求和 HTTP 响应的开始阶段发送的。Web 应用程序必须确保不以 HTTP 标题头中的任何未加密信息作为安全决策依据，因为攻击者要操作这一标题头是很容易的。例如，标题头中的 referer 字段包含来自请求源端的 Web 页面的 URL。不要根据 referer 字段的值做出任何安全决策（如检查请求是否来源于 Web 应用程序生成的页面），因为该字段是很容易被伪造的。

规则 4.1.4：不能依赖于客户端校验，必须使用服务端代码对输入数据进行最终校验。

说明：客户端的校验只能作为辅助手段，减少客户端和服务端的信息交互次数。

规则 4.1.5：对于在客户端已经做了输入校验，在服务器端再次以相同的规则进行校验时，一旦数据不合法，必须使会话失效，并记录告警日志。

说明：肯定存在攻击行为，攻击者绕过了客户端的输入校验，因此必须使会话失效，并记入告警日志。

规则 4.1.6：如果输入为数字参数则必须进行数字型判断。

说明：这里的数字参数指的是完全由数字组成的数据。

实施指导：

```
String mobileno = request.getParameter("mobileno");
String characterPattern = "^\\d+$";    //正则表达式表示是否全为数字
if (!mobileno.matches (characterPattern))
{
```

```
    out.println (“Invalid Input”);
}
```

规则 4.1.7：如果输入只允许包含某些特定的字符或字符的组合，则使用白名单进行输入校验。

说明：对于一些有规则可循的输入，如 email 地址、日期、小数等，使用正则表达式进行白名单校验，这样比使用黑名单进行校验更有效。

实施指导：

```
email 地址校验的方法：
String emailAddress = request.getParameter("emailAddress");
String characterPattern =
"^([a-z0-9A-Z]+[_-]?)+[a-z0-9A-Z]@(([a-z0-9A-Z]+[_-]?)+(-[a-z0-9A-Z]+)?\\.)+[a-z
A-Z]{2,4}$"; //email 正则表达式
if (!emailAddress.matches(characterPattern))
{
    out.println (“Invalid Email Address”);
}
```

规则 4.1.8：如果输入为字符串参数则必须进行字符型合法性判断。

说明：可定义一个合法字符集。

实施指导：

```
String text = request.getParameter("text");
String characterPattern = "^[A-Za-z]*$"; //开发者自行定义字符规则(方括号内的字符
```

```
集)
if (!text.matches (characterPattern))
{
      out.println ( “Invalid Input” );
}
```

规则 4.1.9：校验输入数据的长度。

说明：如果输入数据是字符串，必须校验字符串的长度是否符合要求，长度校验会加大攻击者实施攻击的难度。

规则 4.1.10：校验输入数据的范围。

说明：如果输入数据是数值，必须校验数值的范围是否正确，如年龄应该为 0～150 之间的正整数。

规则 4.1.11：禁止通过字符串串联直接使用用户输入构造可执行 SQL 语句。

说明：禁止通过字符串串联直接使用用户输入构造可执行 SQL 语句，如：string sql = "select status from Users where UserName='" + txtUserName.Text + "'";这样很容易被 SQL 注入攻击。

规则 4.1.12：对于 java/JSP 语言，使用预编译语句 PreparedStatement 代替直接的语句执行 Statement。

说明：使用预编译语句 PreparedStatement，类型化 SQL 参数将检查输入的类型，确保输入值在数据库中当作字符串、数字、日期或 boolean 等值而不是可执行代码进行处理，从而防止 SQL 注入攻击。而且，由于 PreparedStatement 对象已预编译过，所以其执行速度要快于 Statement 对象。因此，多次执行的 SQL 语句经常创建为

PreparedStatement 对象，还可以提高效率。

实施指导：

参考如下代码：

```
String inssql = "insert into buy(empid, name, age, birthday) values (?,?,?,?)";

PreparedStatement stmt = null;

stmt = conn.prepareStatement(inssql);

stmt.setString(1, empid);

stmt.setString(2, name);

stmt.setInt(3, age);

stmt.setDate(4, birthday);

stmt.execute();
```

备注：使用 like 进行模糊查询时，如果直接用"select * from table where comment like %?%"，程序会报错，必须采用如下方法

```
String express = "select * from table where comment like ?";

pstmt = con.prepareStatement(express);

String c="hello";

pstmt.setString(1, "%"+c+"%");

//参数自动添加单引号，最后的 SQL 语句为：select * from  table where comment like '%hello%'

pstmt.execute();
```

规则 4.1.13：禁止动态构建 XPath 语句。

说明：和动态构建 SQL 一样，动态构建 XPath 语句也会导致注入漏洞（XPath 注入）。动态构建 XPath 语句的例子：public boolean doLogin(String loginID, String password){.....

```
    XPathExpression expr =
xpath.compile("//users/user[loginID/text()='"+loginID+"' and
password/text()='"+password+"' ]/firstname/text()");
    ......}
```

规则 4.1.14：在 JavaBean 中禁止使用 property="*"进行参数赋值。

说明：property="*"这表明用户在可见的 JSP 页面中输入的，或是直接通过 Query String 提交的参数值，将存储到与参数名相匹配的 bean 属性中。例如，网上购物程序，一般，用户是这样提交请求的：http://www.somesite.com /addToBasket.jsp?newItem=ITEM0105342 ,如果用户提交 :http://www.somesite.com /addToBasket.jsp?newItem=ITEM0105342&balance=0，这样，balance=0 的信息就被在存储到了 JavaBean 中了，而 balance 是整个会话中用来存储总费用的，当他们这时点击“chekout”结账的时候，费用就全免了。

规则 4.1.15：用于重定向的输入参数不能包含回车和换行字符，以防止 HTTP 响应拆分攻击。

说明：注意，“回车”字符有多种表示方式（CR = %0d = \r ），“换行”字符有多种表示方式（LF = %0a = \n）。

规则 4.1.16：如果服务端代码执行操作系统命令，禁止从客户端获取命令。

说明：如果服务端代码中使用 Runtime.getRuntime().exec(cmd)或 ProcessBuilder 等执行操作系统命令，那么禁止从客户端获取命令；而且最好不要从客户端获取命令的参数，如果必须从客户获取命令的参数，那么必须采用正则表达式对命令参数进行严格的校验，以防止命令注入（因为，一旦从客户端获取命令或参数，通过;&|<>符号，非常容易构造命令注入，危害系统）。

4.2 输出编码

规则 4.2.1：对于不可信的数据，输出到客户端前必须先进行 HTML 编码。

说明：不可信的数据（也就是其他业务系统生成的未经本应用程序验证的表数据或文件数据），通过对输出到客户端的数据进行编码，可以防止浏览器将 HTML 视为可执行脚本，从而防止跨站脚本攻击。

实施指导：

JSP 语言可以通过替换输出数据的特殊字符【& < > " ' ( )%+-】为其他表示形式后再输出给客户端，例如：

```
<%
    String OutStr = "<script>alert('XSS')</script>";
    OutStr = OutStr.replaceAll("&","&amp;");
    OutStr = OutStr.replaceAll("<","&lt;");
    OutStr = OutStr.replaceAll(">","&gt;");
    OutStr = OutStr.replaceAll("\"","&quot;");
    OutStr = OutStr.replaceAll("\'","&#39;");
```

```
        OutStr = OutStr.replaceAll("\\(","&#40;");
        OutStr = OutStr.replaceAll("\\)","&#41;");
        out.println(OutStr);
    %>
```

ASP.NET 语言可以通过 HtmlEncode 方法对 HTML 的输出进行编码。

PHP 语言可以通过 htmlentities 或 htmlspecialchars 方法对 HTML 输出进行编码。

4.3 上传下载

规则 4.3.1：必须在服务器端采用白名单方式对上传或下载的文件类型、大小进行严格的限制。

规则 4.3.2：禁止以用户提交的数据作为读/写/上传/下载文件的路径或文件名，以防止目录跨越和不安全直接对象引用攻击。

说明：建议对写/上传文件的路径或文件名采用随机方式生成，或将写/上传文件放置在有适当访问许可的专门目录。对读/下载文件采用映射表（例如，用户提交的读文件参数为 1，则读取 file1，参数为 2，则读取 file2）。防止恶意用户构造路径和文件名，实施目录跨越和不安全直接对象引用攻击。

规则 4.3.3：禁止将敏感文件（如日志文件、配置文件、数据库文件等）存放在 Web 内容目录下。

说明：Web 内容目录指的是：通过 Web 可以直接浏览、访问的目录，存放在 Web 内容目录下的文件容易被攻击者直接下载。

4.4 异常处理

规则 4.4.1：应用程序出现异常时，禁止向客户端暴露不必要的信息，只能向客户端返回一般性的错误提示消息。

说明：应用程序出现异常时，禁止将数据库版本、数据库结构、操作系统版本、堆栈跟踪、文件名和路径信息、SQL 查询字符串等对攻击者有用的信息返回给客户端。建议重定向到一个统一、默认的错误提示页面，进行信息过滤。

规则 4.4.2：应用程序捕获异常，并在日志中记录详细的错误信息。

说明：记录详细的错误消息，可供入侵检测及问题定位。

4.5 代码注释

规则 4.5.1：在注释信息中禁止包含物理路径信息。

规则 4.5.2：在注释信息中禁止包含数据库连接信息。

规则 4.5.3：在注释信息中禁止包含 SQL 语句信息。

规则 4.5.4：对于静态页面，在注释信息中禁止包含源代码信息。

规则 4.5.5：对于动态页面不使用普通注释，只使用隐藏注释。

说明：动态页面包括 ASP、PHP、JSP、CGI 等由动态语言生成的页面。通过浏览器查看源码的功能，能够查看动态页面中的普通注释信息，但看不到隐藏注释（隐藏注释不会发送给客户端）。因此，为了减少信息泄漏，建议只使用隐藏注释。

实施指导：

```
<form action=h.jsp>
    <%--隐藏注释 1--%>
```

```
    <textarea name=a length=200></textarea>
    <input type=submit value=test>
</form>
<%
    //隐藏注释 2
    java.lang.String str=(String)request.getParameter("a");
    /*隐藏注释 3*/
    str = str.replaceAll("<","&lt;");
    out.println(str);
%>
```

4.6 归档要求

规则 4.6.1：版本归档时，必须删除开发过程（包括现场定制）中的临时文件、备份文件、无用目录等。

说明：恶意用户可以通过 URL 请求诸如.bak 之类的文件，Web 服务器会将这些文件以文本方式呈现给恶意用户，造成代码的泄漏，严重威胁 Web 应用的安全。

实施指导：

在 web 应用的根目录下执行以下命令：

```
find ./ -name "*.old" -o -name "*.OLD" -o -name "*.bak" -o -name "*.BAK" -o -name "*.temp" -o -name "*.tmp" -o -name "*.save" -o -name "*.backup" -o -name "*.orig" -o -name "*.000" -o -name "*~" -o -name "*~1" -o -name "*.dwt" -o -name "*.tpl"
```

```
-o -name "*.zip" -o -name "*.7z" -o -name "*.rar" -o -name "*.gz" -o -name "*.tgz"
-o -name "*.tar" -o -name "*.bz2"
```

分析查找到的文件是否临时文件、备份文件、无用文件，如果是则删除。

规则 4.6.2：归档的页面程序文件的扩展名必须使用小写字母。

说明：很多 Web server 对大小写是敏感的，但对后缀的大小写映像并没有做正确的处理。攻击者只要在 URL 中将 JSP 文件后缀从小写变成大写，Web 服务器就不能正确处理这个文件后缀，而将其当作纯文本显示。攻击者可以通过查看源码获得这些程序的源代码。因此，归档的页面程序文件的扩展名必须使用小写字母，如 jsp、html、htm、asp 等页面程序文件的扩展名分别为 jsp、html、htm、asp。

规则 4.6.3：归档的程序文件中禁止保留调试用的代码。

说明：这里的“调试用的代码”是指开发过程中进行临时调试所用的、在 Web 应用运行过程中不需要使用到的 Web 页面代码或 servlet 代码。例如：在代码开发过程中为了测试一个添加帐号的功能，开发人员临时编写了一个 JSP 页面进行测试，那么在归档时，该 JSP 页面必须删除，以免被攻击者利用。

4.7 其他

规则 4.7.1：对于 JSP 语言，所有 servlet 必须进行静态映射，不允许通过绝对路径访问。

说明：在 web.xml 文件中为 servlet 配置 URI 映射，使用 servlet 时，引用它的 URI 映射，而不允许通过绝对路径访问。

规则 4.7.2：对客户端提交的表单请求进行合法性校验，防止跨站请求伪造攻击。

说明：跨站请求伪造（CSRF）是一种挟制终端用户在当前已登录的 Web 应用程序上执行

非本意的操作的攻击方法。攻击者可以迫使用户去执行攻击者预先设置的操作，例如，如果用户登录网络银行去查看其存款余额，他没有退出网络银行系统就去了自己喜欢的论坛去灌水，如果攻击者在论坛中精心构造了一个恶意的链接并诱使该用户点击了该链接，那么该用户在网络银行帐户中的资金就有可能被转移到攻击者指定的帐户中。当 CSRF 针对普通用户发动攻击时，将对终端用户的数据和操作指令构成严重的威胁；当受攻击的终端用户具有管理员帐户的时候，CSRF 攻击将危及整个 Web 应用程序。

实施指导：

方法一：为每个session创建唯一的随机字符串，并在受理请求时验证

```
<form   action="/transfer.do" method="post">
 <input type="hidden" name="randomStr"
value=<%=request.getSession().getAttribute("randomStr")%>>
 ......
</form>
//判断客户端提交的随机字符串是否正确
String randomStr = (String)request.getParameter("randomStr");
if(randomStr == null) randomStr="";
if(randomStr.equals(request.getSession().getAttribute("randomStr")))
{//处理请求}
else{
//跨站请求攻击，注销会话
}
```

方法二：受理重要操作请求时，在相应的表单页面增加图片验证码，用户提交操作请求的同时提交验证码，在服务器端先判断用户提交的验证码是否正确，验证码正确再受理操作请求。

方法三：向电信软件与核心网网络安全工程部申请WAF CBB，并部署到应用中，启用AntiCSRF功能，具体方法参考WAF CBB的用户手册。

规则 4.7.3：使用.innerHtml 时，如果只是要显示文本内容，必须在 innerHTML 取得内容后，再用正则表达式去除 HTML 标签，以预防跨站脚本。

说明：使用.innerHtml 会将内容以 HTML 显示，容易被利用，导致跨站脚本。

实施指导：

<a

href="javascript:alert(document.getElementById('test').innerHTML.replace(/<.+?>/gim,''))">无 HTML,符合 W3C 标准

备注：还可以使用.innerText 代替.innerHtml，.innerText 只显示文本内容不显示 HTML 标签，但.innerText 不是 W3C 标准的属性，不能适用于所有浏览器（但适用于 IE 浏览器）。

规则 4.7.4：禁止使用 eval()函数来处理用户提交的字符串。

说明：eval()函数存在安全隐患，该函数可以把输入的字符串当作 JavaScript 表达式执行，容易被恶意用户利用。

建议 4.7.1：关闭登录窗体表单中的自动填充功能，以防止浏览器记录用户名和口令。

说明：浏览器都具有自动保存用户输入数据和自动填充数据的能力。为了保障用户名和口令的安全，必须关闭自动填充选项，指示浏览器不要存储登录窗口中用户名、口令等敏感信息。

实施指导：

在 form 表单头中增加选项（autocomplete="off"），例如：

<form action="Login.jsp" name=login method=post autocomplete="off">

建议 4.7.2：防止网页被框架盗链或者点击劫持。

说明：框架盗链和点击劫持（ClickJacking）都利用到框架技术，防范措施就是防止网页被框架。

实施指导：

方法一：在每个网页上增加如下脚本来禁止iframe嵌套：

```
< <script>
      if(top != self) top.location.href = location.href;
</script>
```

方法二：

向电信软件与核心网网络安全工程部申请 WAF CBB，并部署到应用中，启用 AntiClickjackMode 功能，具体方法参考 WAF CBB 的用户手册。

4.8 PHP

规则 4.8.1：禁止使用 phpinfo()。

说明：phpinfo()函数提供了详细的服务器 PHP 环境配置信息，是 PHP 提供的一个比较方便的排错工具。但是往往因为开发人员的一时疏忽，把带有 phpinfo()的页面部署到生产环境上，

造成严重的信息泄露，给潜在攻击者提供了很大的便利。因此在生产环境中应禁止使用 phpinfo()函数，以避免不必要的安全隐患。

实施指导：

修改 php.ini 文件中的 disable_functions 配置项，把“phpinfo”添加到清单中：

disable_functions=phpinfo

备注：如果要禁用多个函数，函数名之间用逗号隔开。

规则 4.8.2：避免使用 eval()、exec()、passthru()、popen()、proc_open()、system()、shell_exec()、pcntl_exec()，如非得使用，必须对输入参数进行严格的输入校验（如：长度、范围、类型校验），并使用 escapeshellcmd() 、escapeshellarg()函数对其参数进行转义处理。

说明：这些是 PHP 用于调用底层系统命令的函数，如对其参数过滤不严，将容易导致“系统命令执行”漏洞，使黑客轻易地执行系统命令并获得服务器的 shell 权限。另外所谓的 webshell 会通过这些函数跟底层系统进行交互，因此关闭这些函数可有效地降低 webshell

的安全威胁。

实施指导：

1、当不使用这些函数时，需通过 disable_functions 配置项禁止这些函数的使用：

修改php.ini文件中的disable_functions配置项，把要禁用的函数添加到disable_functions参数值中，多个函数，用逗号分开，例如：

```
disable_functions=phpinfo,get_cfg_var，
eval,exec,passthru,popen,proc_open,system,shell_exec,pcntl_exec
```

2、当使用这些函数时，应使用 escapeshellcmd()或 escapeshellarg()对其参数进行过滤。

escapeshellcmd() :

```
<?php
$command = './configure '.$_POST['configure_options'];
$escaped_command = escapeshellcmd($command);
system($escaped_command);
?>
```

escapeshellarg() :

```
<?php
system('ls ' . escapeshellarg($dir));
?>
```

规则 4.8.3 在使用 include()、include_once()、require()、require_once()、show_source()、highlight_file()、readfile()、file_get_contents()、fopen()、file()等文件读取函数时，应对参数进行严格的合规性检查，避免直接使用客户端输入作为参数，必要时应进行白名单限

制。

说明：对这些函数的参数检查不严会很容易导致远程命令执行漏洞，给系统带来不必要的安全隐患。

实施指导：

```
$lang = preg_replace('/[^a-zA-Z0-9_]/', '', $_GET['lang']);
switch ($lang):
    case “en” :
        include( “en.php” );
    case “cn” :
        include( “cn.php” );
    default:
        include( “cn.php” );
endswitch;
```

建议 4.8.4：避免使用 preg_replace()函数。当使用/e 修饰符，preg_replace 会将 replacement 参数当作 PHP 代码执行，如使用不当将容易引入漏洞，建议使用 preg_match()替代。

规则 4.8.5 :明确使用$_GET、$_POST、$_COOKIE 而不是$_REQUEST 获取用户请求数据，这有助于降低漏洞被成功利用的概率。

5 Web 安全配置规范

根据 Web 应用所选用的 Web 容器，按照相应的配置规范进行安全配置，当前可供使

用的安全配置规范如下：

表2 Web 容器安全配置规范列表

规范名称
IIS 安全配置规范
Apache 安全配置规范
Tomcat 安全配置规范
JBoss 安全配置规范
Resin 安全配置规范
WebLogic 安全配置规范

6 配套 CBB 介绍

6.1 WAF CBB

- **提供功能：**

1) 防范 CSRF（跨站请求伪造）攻击

2) 防范 XSS 攻击

3) 防范 MML 注入

4) 防范目录遍历/路径遍历

5) 防范字符串型的 SQL 注入攻击（绝大部分的 SQL 注入为字符串型的）

6) 防范空字符注入

7) 防范点击劫持

8) 防范 HTTP 会话劫持

9) 防范功能可配置（也就是可以通过配置项开启或关闭对应的防范功能）

10) URI 白名单功能，允许配置免检的 URI。

11）输入校验：可以为各输入参数配置输入校验规则（正则表达式），在服务器端实现严格的输入校验。

- **获取方法：**

  WAF CBB

6.2 验证码 CBB

- **提供功能：**

  实现了“3.2.3 验证码”的要求。

- **获取方法：**

  验证码 CBB

7 附件

7.1 附件 1 Tomcat 配置 SSL 指导

附件1
Tomcat配置SSL指导

7.2 附件 2 Web Service 安全接入开发指导

附件2 Web Service
安全接入开发指导.

7.3 附件 3 客户端 IP 鉴权实施指导

附件3
客户端IP鉴权实施指

7.4 附件 4 口令安全要求

附件4
口令安全要求.xlsx

7.5 附件 5 Web 权限管理设计规格说明书

附件5
Web权限管理设计规