应用开发安全指南 V1.0

中天智慧科技有限公司

2018 年 3 月 19 日

目录

图表目录

第1章 概述

1.1 目标

本指南针对 Web 应用系统应当遵循的应用开发安全标准进行了规范性说明，旨在指导代码开发人员和安全检查管理人员进行应用安全开发的安全配置，以提高应用系统的安全防护能力。

1.2 适用范围

本指南用于所有 Web 应用系统的设计、开发，包括：

- 适用于应用系统需求整理、概要设计中安全相关部分的设计参考；
- 适用于应用系统开发过程序中安全相关功能设计、编码参考；
- 适用于应用系统测试阶段安全性测试的测试参考；
- 适用于安全管理人员应用安全评估参考。

1.3 应用系统安全范围

本指南重点考虑在应用程序易受攻击的重要环节。将重点放在程序设计、输入验证、身份验证和授权、加密及敏感数据处理、应用配置、会话管理、异常处理，并要求适当的审核和记录策略。

第2章 Web 应用安全需求及风险分布

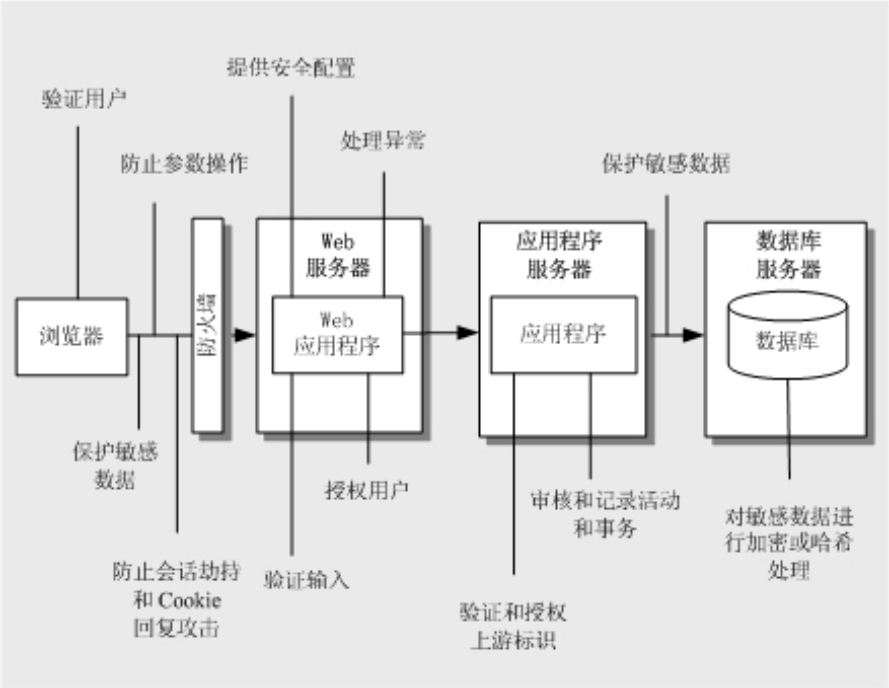
2.1 应用系统安全性基本需求

应用系统的安全性基本需求应包括如下方面：

- 应用系统应满足系统对于身份认证的需求，应明确提出用户身份认证体系的强度，认证失败后的处置方式。
- 应用系统应具备完善的用户会话管理机制，明确用户会话的产生、管理、销毁的条件。
- 应用系统应该在数据的输入、输出前进行安全性检测。
- 应用系统应包含用户权限分配和管理功能，应根据系统所处理的业务数据的保密性和完整性要求，确定系统采用的用户权限访问控制模型和权限的划分，避免权限的过分集中与分散。
- 应用系统应从数据的可用性角度出发，明确考虑数据安全和冗余恢复相关功能。
- 应用系统应包括安全审计功能，应明确对于日志内容的要求。
- 应用系统应当设计基本的数据安全保护功能，包括以下内容：包括数据采集、数据传输、数据处理、数据存储、数据备份和恢复的安全。对重要的、敏感数据应当进行加密和完整性保护。
- 应用系统所在的终端与服务器端之间的、或程序模块之间的敏感数据通

信应有明确而严格的安全加密机制。

2.2 Web 应用程序风险分布



图表 1 Web 应用程序风险分布

第3章 输入输出验证

绝大多数针对应用系统的攻击都从恶意输入开始，因此输入验证是应用研发人员需要首位解决的问题。正确的输入验证是防御目前应用程序攻击的最有效方法之一，多数的针对应用程序的攻击都是从用户的恶意输入开始，正确实现的输入验证是防止 SQL 注入和其他输入攻击的有效对策，而对输出进行转义，则可防止广泛发生的 XSS 攻击。因此合理有效输入输出验证是 Web 应用程序安全开发的重点内容。

3.1 输入来源

在 Web 应用系统中，用户可以向应用程序提交输入的主要有以下方式：

- Get
- Post
- Cookie
- Http Heder,例如：HTTP_REFERER、HTTP_USER_AGENT 等
- Flash 本地缓存

PHP，ASP.net，JAVA 等语言，及各类中间件中均有多种取得上述用户数据的方法函数，这些函数本指南不做一一说明，研发人员需充分了解各种函数所取值的范围，在过滤其取得的内容时方能考虑周全。例如：某函数是否只从 Get 中获取数据，或一个函数同时是可从 get，post，cookie 中获取数据等。

除了用户向应用程序提交的输入外，应用程序还可能从数据库、本地文件中、或者其他程序中获得数据，这些数据我们也认为是应用程序获得的输入，因此对于这种输入也必须进行输入验证。

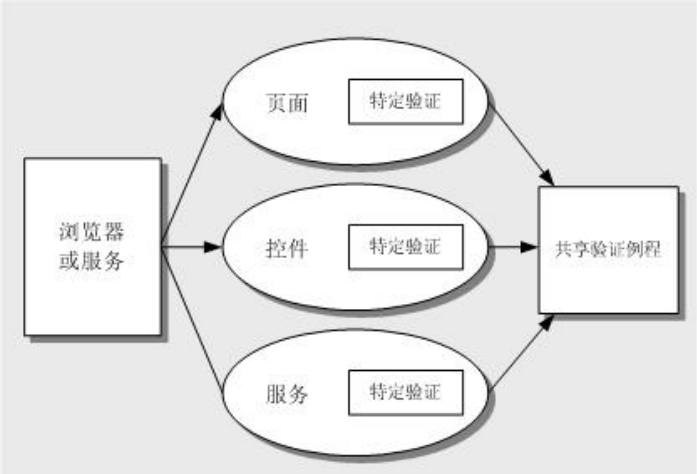
因此在提到验证输入时，要考虑到上述各种可能性，本指南提到的用户输入所包含的范围除非特别说明均包括以上部分。

为了开发出更加安全的程序，我们在接受到任何输入时，都应该认为这是有害的输入。

3.2 集中验证

将输入验证策略作为应用程序设计的核心元素，应考虑集中式验证方法，例如，通过使用共享库中的公共验证和过滤代码。这可确保验证规则应用的一致性，还能减少开发的工作量，且有助于以后的维护工作。

许多情况下，不同的字段要求不同的验证方法，例如，要求使用专门开发的常规表达式。但是，通常可以得到验证常用字段（如电子邮件地址、标题、名称、包括邮政编码在内的通讯地址，等等）的常规方法。下图显示了此验证方法。同时集中方法还利于提升开发效率，及降低维护成本。



图表 2 集中认证

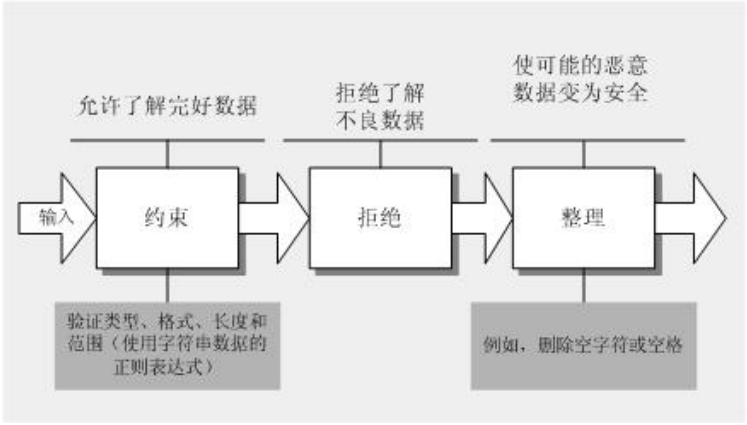
3.3 不要依赖客户端的验证

应使用服务器端代码执行其自身的验证。如果攻击者绕过客户端或关闭客户端脚本例程（例如，通过禁用 JavaScript 脚本），后果如何？使用客户端验证可以减少客户端到服务器端的往返次数，但不要依赖这种方法进行安全验证。

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。为防止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。

3.4 限制和过滤输入输出

输入验证的首选方法是从开始就限制允许输入的内容。按照已知的有效类型、模式和范围验证数据要比通过查找已知有害字符的数据验证方法容易。设计应用程序时，应了解应用程序需要输入什么内容。与潜在的恶意输入相比，有效数据的范围通常是更为有限的集合。然而，为了使防御更为彻底，您可能还希望拒绝已知的有害输入，然后过滤输入。 下图显示了我们推荐的策略。



图表 3 限制和过滤输入输出

3.4.1 限制输入

限制输入与允许输入有益数据类似。这是首选的输入验证方法。其思想是定义一个过滤器，根据类型、长度、格式和范围来过滤输入的数据。定义应用程序字段可以接受的数据输入，并强制应用该定义。拒绝一切有害数据。

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。

3.4.2 验证数据的类型、长度、格式和范围

在适当的地方对输入数据使用强类型检查，例如需要代入数据库查询的用户数据，应该严格限制类型（数字，纯字符等）。应该检查字符串字段的长度，在许多情况下，还应检查字符串的格式是否正确。例如，邮政编码和身份证号码等都具有明确定义的格式，可以使用常规表达式进行验证。严格的检查不仅是很好的编程习惯，还能让攻击者更难利用您的代码。攻击者可能会通过类型检查，但长度检查会加大攻击者实施其所喜欢的攻击方式的难度。

3.4.3 拒绝恶意输入

sql 注入攻击需要依赖“ ‘”，“”“等特殊字符，而 XSS 攻击需要输入“<,>”等 html 标记，因此如果在处理输入之前对这些应用程序本身用不到的字符进行拒绝，则可抵御绝大多数此类攻击。

3.4.4 过滤、转义输出（防止 XSS 漏洞）

过滤是为了使有潜在危害的数据变得安全。如果所允许的输入范围不能保证输入数据的安全性，那么过滤输入是非常有用的。过滤包括从删除用户输入字符串后面的空格到去除值（以便按照文字格式处理该数据）等一切行为。

在 Web 应用程序中，另一个常见的过滤输出示例是使用 URL 编码或 HTML 编码来包装数据，并将其作为文本而不是可执行脚本来处理。HtmlEncode 方法去除 HTML 字符，而 UrlEncode 方法对 URL 进行编码，使其成为有效的 URI 请求。

3.4.5 不要相信 Http Header

HTTP Header 是在 HTTP 请求和响应开始时发送。应确保 Web 应用程序的任何安全决策都不是基于 HTTP 头中包含的信息，因为攻击者很容易操作 HTTP 头。例如，HTTP 头中的“referer”字段包含发出请求的网页的 URL。不要基于“referer”字段值作出任何安全决策，以检查发出请求的页面是否由该 Web 应用程序生成，因为该字段很容易伪造。

3.4.6 过滤规则示例

这里有一些过滤规则的示例，供参考。

- 姓氏字段。这是一个很好的应用限制输入的示例。在这种情况下，可以接受的字符串范围为 ASCII A–Z 和 a–z，以及连字符和波浪线（在 SQL 中，波浪线没有意义），以便处理类似 O'Dell 之类的姓氏。还应限制输入内容的最大长度。

- 数量字段。这是应用输入限制的另一个例子。在此示例中，可以使用简单的类型和范围限制。例如，输入数据应该是介于 0 和 1000 之间的正整数。

- 留言版上的备注字段。在这种情况下，您可能允许输入字母和空格，以及省略符号、逗号和连字符等常用字符。允许输入的字符集只包括符号、括号和大括号。

- 脚本字符。有些应用程序可能允许用户使用一组有限的脚本字符修饰文本，如粗体“<b>”、斜体“<b>”，甚至包含指向他们所喜爱的 URL

的连接。处理 URL 时，验证时应先对所输入的值进行编码，以便将其作为 URL 处理。

3.5 解决 SQL 注入问题

注：本指南仅描述如何避免 Sql 注入漏洞，更多关于 Sql 注入的信息请 Google

Sql 注入是 Web 应用中最常见的高风险漏洞，会导致数据库中数据泄露。我们在获取到用户输入后，如果对输入内容进行过滤，过滤掉其中的“ ‘”及可能存在的 sql 关键字，能从很大程度上降低被攻击的风险，但是由于攻击载荷存在很多种变种及不同的表示方法，因此只采用输入过滤来进行 SQL 注入并不能保证完全杜绝此类攻击。

其实 sql 注入只存在与采用 Statement 方式（“拼接方式”）生成、执行的 sql 语句中，例如：

```
    String query = "SELECT * FROM accounts WHERE
custID='" + request.getParameter("id") +"'";
stm=conn.createStatement();
rs=stm.executeQuery(sql);
```

如果使用 Prepared Statement 方式进行 Sql 查询，即参数化查询的方式则可完全避免 Sql 注入，因为 Prepared Statement 方式下，sql 语句会被预先编译，在参数代入 sql 表达式后，不会改变已经编译过的 sql 语句结构，从而使得攻击者注入的 sql 语句永远不会生效。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。

使用参数化查询的实例代码如下：

```
    String sql="select EmpNo,Ename "  +  "from emp
"  +  "where EmpNo=?";
psmt=conn.prepareStatement(sql);
psmt.setInt(1, 7499);
rs=psmt.executeQuery();
```

结论

- 使用 Prepared Statement 方式进行 Sql 查询即可彻底防止 Sql 注入问题。
- 因此在开发过程中必须彻底摒弃 Sql 语句拼接的方式。
- 同时从程序的执行速度和程序可读性的角度我们也推荐使用 Prepared Statement 来进行 Sql 查询。
- 现今流行的框架中都支持 Prepared Statement 方式进行 sql 查询，在接触一个新的框架前，应当弄清如何正确的使用参数化进行 Sql 查询。

3.6 上传下载

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

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。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。

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。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。

第4章 身份认证

4.1 身份验证考虑因素

身份验证是确定调用户身份的过程。有三方面问题需要考虑：

- 确定应用程序中何处需要身份验证。通常在跨越信任边界时，都需要进行身份验证。信任边界通常包括程序集、过程和主机。
- 确认调用方的身份。用户通常使用用户名和密码证明自己的身份。
- 在后续请求中识别用户，这需要哪种形式的验证令牌。

4.2 用户名密码方式的认证

常见的 web 应用程序多数使用用户名密码的方式来进行身份认证，这里要考虑如下问题：

- 重要的用户名和密码是应当使用安全套接字层（SSL）确保通信通道的安全，防止用户凭据被窃听。
- 用户密码应当加密存储，用户密码在数据库中不应以明文储存，以防止数据库被非法人员访问后得到所有用户的信息。
- 根据安全制度规范及系统密级考虑是否要求强求密码复杂度。
- 错误的用户名和错误的密码应当给出相同的错误提示。

4.3 增强身份认证

4.3.1 区域划分

站点的公共区域允许任何用户进行匿名访问。受限区域只能接受特定用户的访问，而且用户必须通过站点的身份验证。例如一个典型的零售网站。您可以匿名浏览产品分类。当您向购物车中添加物品时，应用程序将使用会话标识符验证您的身份；当您下订单时，即可执行安全的交易，这需要您进行登录，以便通过 SSL 验证交易。

将站点分割为公共访问区域和受限访问区域，可以在该站点的不同区域使用不同的身份验证和授权规则，从而限制对 SSL 的使用。使用 SSL 会导致性能下降，为了避免不必要的系统开销，在设计站点时，应该在要求验证访问的区域限制使用 SSL。

4.3.2 防止暴力破解

登陆页面应有机器无法识别、而人眼便于识别的验证码，验证用户名密码的有效性前，应当验证验证码的有效性。

当用户帐户几次登录尝试失败后，可以暂时锁定该帐户或将事件写入日志，并禁止该用户登陆。

在验证用户名密码有效性前，先验证该用户是否锁定，如果已锁定，则无需代入数据库中进行查询。

4.3.3 密码有效期

密码不应固定不变，而应作为常规密码维护的一部分，通过设置密码有效期对密码进行更改。在应用程序设计阶段，应该考虑提供这种类型的功能，根据实际情况决定是否启用。

4.3.4 不要在客户端存储密码

如果必须存储密码，也不能在客户端实际存储密码。可以存储一个单向哈希值，然后使用用户所提供的密码重新计算哈希值。为减少对用户存储的词典攻击威胁，可以使用强密码，并将随机 salt 值与该密码结合使用。

4.3.5 要求使用强密码

不要让用户使攻击者能轻松破解密码。有很多可用的密码编制指南，但通常的做法是要求输入至少 8 位字符，其中要包含大写字母、小写字母、数字和特殊字符。

无论是使用平台实施密码验证还是开发自己的验证策略，此步骤在对付粗暴攻击时都是必需的。在暴力攻击中，攻击者试图通过系统的试错法来破解密码。可以使用正则表达式协助强密码验证。

4.3.6 不要以明文传输密码

以明文形式在网络上发送的密码容易被窃听。为了解决这一问题，应确保通信通道的安全，例如，使用 SSL 对数据流加密，如无法使用 SSL 则应在在客户端本地采用一些加密算法进行加密后传输，这样可以防止密码在传输过程中轻易被截获。

4.3.7 保护用户 Cookie

身份验证的 cookie 被窃取意味着登录被窃取。可以通过加密和安全的通信通道来保护验证票证。另外，还应限制验证票证的有效期，以防止因重复攻击导致的欺骗威胁。在重复攻击中，攻击者可以捕获 cookie，并使用它来非法访问您的站点。减少 cookie 超时时间虽然不能阻止重复攻击，但确实能限制攻击者利用窃取的 cookie 来访问站点的时间。

4.3.8 防止会话固定漏洞

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。

4.3.9 验证码

验证码必须是单一图片，且只能采用 JPEG、PNG 或 GIF 格式。

验证码内容不能与客户端提交的任何信息相关联。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。

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。

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。

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。

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。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。

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。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。

4.4 认证失败提示

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。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。

对每个错误的登录尝试发出相同的错误消息，不管是哪个字段发生错误，特别是用户名或密码字段错误。以防止攻击者通过反复试验（蛮力攻击技术）来发现应用程序的有效用户名，再继续尝试发现相关联的密码。这样会得到有效用户名和密码的枚举，攻击者可以用来访问帐户。

4.5 敏感交易重新认证

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。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支付口令。

第5章 用户权限控制

5.1 授权

对于一个 Web 系统而言，禁止未经授权的用户非法访问页面或非法使用系统中提供的各项功能是非常重要的。对于那些需要权限控制的 Web 系统，首先要进行用户身份的确认，在用户通过登录验证后，需要将用户个人的状态信息保存起来，以便为访问系统后续页面提供依据，在访问需经授权才能的访问的页面时，可以通过判断用户标识，以检验用户是否已通过登录验证。用户权限控制的原则是："只给能满足最低需求的最小权限"

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。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。

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。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。

5.2 多重权限控制

从应用从设计到编写，部署到发布，有诸多问题需要考虑权限控制。例如数据库服务器网络连接权限控制，应用服务器用户权限控制，到 Web Sever 的用

户权限控制，Web Server 目录访问权限控制，应用程序功能权限控制等。针对不同的应该场景应当组合使用这些权限控制技术。

5.3 限制用户访问系统资源

系统级资源包括文件、文件夹、注册表项、Active Directory 对象、数据库对象、事件日志，等等。限制哪些用户可以访问哪些资源，以及可以执行哪些类型的操作。要特别注意匿名 Interne 用户帐户；使用 ACL 锁定这些匿名帐户，以防止他们访问明确拒绝匿名用户访问的资源。

5.4 Web 应用的权限控制

早期对于权限控制的设计主要针对主机资源进行控制，后来发展到针对网络资源进行控制，最常见的模型有下面三种：

- DAC（Discretionary Access Control，自主访问控制模型）
- MAC（Mandatory Access Control，强制访问控制模型）
- RBAC（Role Based Access Control，基于角色的访问控制）

随着 Web 应用的发展，Web 应用不得不面对访问控制的需求，最常见的实现方式是根据 RBAC 模型即基于角色的访问控制来实现的。常见的做法是根据用户的角色，限制用户可以访问的资源，例如普通用户不能访问“用户管理”的页面。

应用程序中权限控制不当常见的问题为：

- 仅仅通过视图来控制用户权限

- 通过用户 cookie 来识别用户权限，

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。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。

- 关键操作未进行第二次权限验证

- 特权页面可匿名访问

第6章 会话管理

Web 应用程序基于无界限的 HTTP 协议构建，因此，会话管理是应用程序级职责。对于应用程序总体安全来讲，会话安全是关键因素。下面是一些可以增强会话安全的措施。

6.1 通过 SSL 传递 Cookie

不要通过 HTTP 连接传递身份验证 cookie。在授权 cookie 内设置安全的 cookie 属性，以便指示浏览器只通过 HTTPS 连接向服务器传回 cookie。

设置 cookie 的安全属性，可以强制浏览器使用安全的连接。例如：使用 javax.servlet.http.Cookie 下的 void setSecure(boolean flag) 方法，强制该 cookie 使用安全模式传输。

6.2 防止固定会话攻击

Tomcat、WebLogic 等 web 容器，在匿名用户访问时也会设置一个 JSESSIOND，用户登陆成功后，如果应用不采取一些措施，则此 JSESSIONID 不会更改，这就产生了固定会话攻击，攻击者可以将提前准备的 JSESSIONID 发送给用户，用户成功登陆后，攻击者则可利用此有效的会话进行对该用户资源的操作。

为了防止固定会话攻击，应当在用户登陆验证成功后，进行 JSESSIONID 的重置，例如：

```
    protected void doPost(HttpServletRequest request, HttpS
ervletResponse response) throws ServletException, IOExcepti
```

```
on {
        String username=request.getParameter("username");
        String password=request.getParameter("password");
        if("admin".equals(username) && "pass".equals(passwo
rd)){
                //是之前的匿名 session 失效
            request.getSession().invalidate();
            request.getSession().setAttribute("login", true
);
            response.sendRedirect("hello.jsp");
        }else{
            response.sendRedirect("login.jsp");
        }
    }
```

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。

6.3 防止 XSS 盗取 cookie

Cookie 的 httponly 选项是微软为了减缓 XSS 攻击中 cookie 被盗而提出的一项新技术，最早仅有 IE 支持，现在在 FireFox 中也得到了支持，而且越来越多的浏览器开始支持 cookie 的这个选项。Httponly 选项可以禁止客户端脚本访问用户的 Cookie，因此可以使得 XSS 攻击脚本获得用户 cookie 失效。

在 tomcat 中可以使用 tomcat 的 servlet 直接写 Header：

```
response.setHeader( "Set-Cookie", "name=value; HttpOnly");
```

6.4 防止会话信息被篡改

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。以防止会话信息被篡改，如恶意用户通过 URL 篡改手机号码等。

防止遗留在内存中的会话信息被窃取，减少内存占用。对于 JSP 或 java 语言使用如下语句：request.getSession().invalidate();

6.5 限制会话有效期

缩短会话有效期可以降低会话劫持和重复攻击的风险。会话有效期越短，攻击者捕获会话 cookie 并利用它访问应用程序的时间越有限。

设置 Session 的有效期例如：

```
setMaxInactiveInterval(int interval)
```

设置 cookie 的有效期例如：

```
c.setMaxAge(int expiry);
```

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。

6.6 流程安全控制

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。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。

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。

第7章 加密与敏感数据处理

7.1 加密的作用

针对不同的数据对象，加密的目的总共有如下几种：

- **保密性**（机密性）。此服务保持数据的机密性
- **认可**（真实性）。此服务确保用户不能拒绝发送特定消息。
- **防止篡改**（完整性）。此服务防止数据被修改。
- **身份验证**。此服务确认消息发送方的身份。

7.2 安全地使用加密技术

7.2.1 使用公开的算法

成功开发出加密算法和例程是非常难的，因此，，应使用平台提供的经过验证和测试的加密服务。切记不要开发自定义的实现方法，因为这通常会导致保护能力减弱。

应当使用算法公开主流加密算法，并且结合实际场景来进行加密算法的选择。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。

7.2.2 选择正确的加密算法

针对不同的加密内容及需求，我们选择算法时需要考虑到加密的主要目的、加解密速度、破解难易度等信息，然后结合实际情况选择合适的算法。

加密算法的密钥长度决定了加揭密速度和破解难易度，常见的算法密钥长度

如下：

- 数据加密标准（DES) 64 位密钥（8 个字节）
- TripleDES 128 位密钥或 192 位密钥（16 或 24 个字节）
- RijndaelAES 128 – 256 位密钥（16 – 32 个字节）
- RSA 384 – 16,384 位密钥（48 – 2,048 个字节）

建议：

- 对于大量数据加密，应使用对称加密算法，如 Triple DES（3DES)、Rijndael（AES）等。
- 要对将暂时存储的数据加密，可以考虑使用较快但较弱的算法，如 DES。
- 对于数字签名，应使用 RSA 或 DSA 等非对称算法。
- 对于用户密码、会话验证等信息应当使用快速的、不可逆的哈希算法，如（SHA1、SHA2、MD5）等。

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。

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。

7.3 敏感数据存储

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。

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。

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。

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。

禁止在隐藏域中存放明文形式的敏感数据。

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。

7.4 敏感数据传输

带有敏感数据的表单必须使用 HTTP-POST 方法提交。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。

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。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。

禁止在 URL 中携带会话标识（如 jsessionid）。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。

第8章 异常管理

8.1 不要向客户端输出信息

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。

程序发生错误或捕获到异常时，不要暴露将会导致信息泄漏的消息。例如，不要暴露包括函数名以及调试内部版本时出问题的行数（该操作不应在生产服务器上进行）的堆栈跟踪详细信息，这些消息会帮助攻击者进行攻击。应向客户端返回一般性的统一的错误消息。

8.2 记录详细信息

向错误日志发送详细的错误消息。应该向服务或应用程序的客户发送最少量的信息，如一般性错误消息和自定义错误日志 ID，随后可以将这些信息映射到事件日志中的详细消息。确保没有记录密码或其他敏感数据。

8.3 捕获更多的异常

使用结构化异常处理机制，并尽可能捕捉异常现象。这样做可以避免将应用程序置于不协调的状态，这种状态可能会导致信息泄漏。它还有助于保护应用程序免受拒绝服务攻击。确定如何在应用程序内部广播异常现象，并着重考虑在应用程序的边界会发生什么事情。

第9章 审核与记录

9.1 审核并记录跨应用层的访问

审核并记录跨应用层的访问，以便用于认可。可以结合使用应用程序级记录和平台审核功能。

9.2 记录关键事件

应用程序应记录的事件类型包括成功和失败的登录尝试、数据修改、数据检索、网络通信和管理功能，如启用或禁用日志记录。日志应包括事件发生的时间、地点（包括主机名）、当前用户的标识、启动该事件的进程标识以及对该事件的详细描述。

9.3 确保日志安全

应使用 Windows ACL、Linux 用户权限机制确保日志文件的安全，并限制对日志文件的访问。这加大了攻击者篡改日志文件以掩饰其攻击行为的难度。应将有权操作日志文件的个人数量减到最小。只为高度信任的帐户（如管理员）授予访问权限。

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。

9.4 定期备份日志

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。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。

应定期删除生产服务器上的日志文件。删除频率取决于应用程序的活动级别。设计程序时，应考虑检索日志文件和将其移到脱机服务器进行分析的方式。

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。

第10章 版本归档与测试

10.1 版本归档

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。因为恶意用户可以通过 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。

归档的页面程序文件的扩展名必须使用小写字母，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。

归档的程序文件中禁止保留调试用的代码。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。

10.2 测试环境及人员安全要求

测试环境的物理、硬软件环境要求应当可以模拟真实环境。

为确保测试环境的安全，应将测试环境与开发环境、生产环境相隔离，避免测试工作对业务的影响。

测试数据如果选择是真实数据，应限定使用该数据的测试人员的数量，并在测试完成后全部删除。

系统测试和验收通常需要大量的（真实数据）尽可能靠近实际运行数据的测试数据。应避免使用含有个人信息的业务数据库。如果要使用其中信息，在用之前应使其失去个性化。当把运行数据用于测试目的时，应采取以下措施保护运行数据。

- 如果该数据由测试人员使用，则需要相关授权业务人员参与，或者监督。
- 每次把运行信息复制到测试应用系统都应有单独的授权。
- 应记录运行信息的复制和使用，以提供一种检查追踪。

在与其他系统的互操作性测试中，应充分考虑对其他系统的影响，选择适当的时间、方法。

10.3 测试后的安全检查

测试完成后，在正式上线前，应进行安全检查，消除测试用的一切后门、用户名及口令等，包括但不限于：

- 测试用的测试数据，即测试过程中的临时数据，特别注意要删除授权访问的那些重要测试数据。
- 测试用的账号、口令，特别是有高级权限的账号。
- 测试用时临时开通的系统服务、防火墙端口及策略。
- 与外系统测试时，要求外系统上临时开启的服务、账号等。