# OWASP 安全编码规范

## 快速参考指南

Open Web Application Security Project (OWASP)

2022 年 12 月


OWASP 安全编码规范快速参考指南

## 版权与许可



## 中文版本团队成员

王颉 何勇亮 林恒辉

（欢迎大家指正翻译错误。我们将在以后的版本中修正指出的错误。）

# 目录

# 序言

本项与技术无关的文档以清单列表的形式，定义了一套可以集成到软件开发生命周期中的通用软件安全编码规范。采用这些规范将减少最为常见的软件漏洞。

一般来说，开发安全的软件要比在软件包完成以后再纠正安全问题的成本低很多，且还没涉及到因为安全问题而造成的损失。

保护关键软件资源的安全性，比以往任何时候都更为重要，因为攻击者的重点已逐步转向了应用层。2009 年 SANS 的一项研究表明，针对 Web 应用程序的攻击已占据了在互联网上观察到攻击总数的 60％以上。

在使用本指南时，开发团队应该从评估他们的安全软件开发生命周期成熟度和开发人员知识水平着手。由于本指南不涉及如何实现每个编码规范的具体细节，因此，开发人员需要了解相关知识，或者有足够可用资源来提供必要的指导。通过本指南，开发人员可在无需深入了解安全漏洞和攻击的情况下，将编码规范转换成编码要求。当然，开发团队的其他成员应该有责任，通过提供适当的培训、工具和资源，以验证整个系统的设计和开发是安全的。

本文档中使用的重要术语，包括部分标题和文字，都以斜体字标注，并列举在附录 B 的术语列表中。

关于安全软件开发框架的指南不属于本文讨论的范围。但是，我们推荐以下额外的常用规范和资源：

- 明确定义角色和职责。
- 为开发团队提供足够的软件安全培训。
- 采用一个安全软件开发生命周期。
- 建立安全编码标准。
  - OWASP 开发指南项目
- 建立一个可重用的对象库文件。
  - OWASP Enterprise Security API (ESAPI) 项目
- 验证安全控制的有效性。
  - OWASP Application Security Verification Standard (ASVS) 项目
- 建立外包开发安全规范，包括在建议书（RFP）和合同中定义安全需求和验证方法。

# 软件安全与风险原则概览

开发安全的软件需要对安全原则有基本的了解。虽然对于安全原则的全面评估超出了本指南的范围，但是我们还是提供了一个快速的概览。

软件安全的目标是要维护信息资源的保密性，完整性，和可用性，以确保业务的成功运作。该目标通过实施安全控制来实现。本指南重点介绍具体的技术控制，以缓解常见软件漏洞的发生。虽然主要的关注点是Web应用程序及其配套的基础设施，但是本指南的大部分内容可应用于任意软件部署平台。

为了保护业务免受来自与软件相关的不能接受的风险，了解风险的意义是很有帮助的。风险是一组威胁业务成功因素的集合。它可以被定义为：一个威胁代理与一个可能含有漏洞的系统交互，该漏洞可被利用并造成影响。虽然这可能看起来象是一个抽象的概念，但可以这样想象它：一个汽车盗窃犯（威胁代理）来到一个停车场（系统）寻找没有锁车门（漏洞）的车，当找到一个时，他们打开门（利用）并拿走里面任何的东西（影响）。所有这些因素在安全软件开发时都扮演了一个角色。

开发团队采用的方法和攻击者攻击应用程序所采用的方法之间有一个根本区别。开发团队通常采用的方法是基于应用程序的目的行为。换句话说，开发团队根据功能需求文档和用例设计一个应用程序以执行特定的任务。而另一方面，攻击者，基于"没有具体说明应拒绝的行为，则被认为是可行的"原则，对于应用程序可以做什么更感兴趣。为了解决这个问题，一些额外的元素需要被集成到软件生命周期的早期阶段。这些新元素是安全需求和滥用实例。本指南旨在帮助明确高等级的安全需求，并解决许多常见的滥用情况。

Web 开发团队应当明白，基于客户端的输入验证、隐藏字段和界面控件（例如，下拉键和单选按钮）的客户端控制，所带来的安全性收益是有限的，这一点非常重要。攻击者可以使用工具，比如：客户端的 Web 代理（例如，OWASP WebScarab，Burp）或网络数据包捕获工具（例如，Wireshark），进行应用程序流量分析，提交定制的请求，并绕过所有的接口。另外，Flash，Java Applet 和其它客户端对象可以被反编译并进行漏洞分析。

软件的安全漏洞可以在软件开发生命周期的任何阶段被引入，包括：

- 最初没有明确的安全需求；
- 创建有逻辑错误的概念设计；
- 使用糟糕的编码规范，从而带来了技术漏洞；
- 软件部署不当；
- 在维护或者更新过程中引入缺陷。

此外，还有重要的一点需要明白，软件漏洞可以超出软件本身的范围。根据不同的软件、漏洞和配套基础设施的性质，一次成功的攻击会影响下面任何或者所有的方面：

- 软件和其相关的信息；
- 相关服务器的操作系统；
- 后端数据库；
- 在共享环境中的其它应用程序；
- 用户的系统；

- 与用户交互的其它软件。

# 安全编码规范检查列表

## 输入验证

- ☐ 在可信系统（比如：服务器）上执行所有的数据验证。
- ☐ 识别所有的数据源，并将其分为可信的和不可信的。验证所有来自不可信数据源（比如：数据库，文件流，等）的数据。
- ☐ 应当为应用程序应提供一个集中的输入验证规则。
- ☐ 为所有输入明确恰当的字符集，比如：UTF-8。
- ☐ 在输入验证前，将数据按照常用字符进行编码（规范化）。
- ☐ 丢弃任何没有通过输入验证的数据。
- ☐ 确定系统是否支持 UTF-8 扩展字符集，如果支持，在 UTF-8 解码完成以后进行输入验证。
- ☐ 在处理以前，验证所有来自客户端的数据，包括：所有参数、URL、HTTP 头信息（比如： cookie 名字和数据值）。确定包括了来自JavaScript、Flash 或其他嵌入代码的 post back 信息。
- ☐ 验证在请求和响应的报头信息中只含有 ASCII 字符。
- ☐ 核实来自重定向输入的数据（一个攻击者可能向重定向的目标直接提交恶意代码，从而避开应用程序逻辑以及在重定向前执行的任何验证）。
- ☐ 验证正确的数据类型。
- ☐ 验证数据范围。
- ☐ 验证数据长度。
- ☐ 尽可能采用”白名单”形式，验证所有的输入。
- ☐ 如果任何潜在的危险字符必须被作为输入，请确保您执行了额外的控制，比如：输出编码、特定的安全 API、以及在应用程序中使用的原因。部分常见的危险字符包括： < > ” ' % ( ) & + \ \' \” 。
- ☐ 如果您使用的标准验证规则无法验证下面的输入，那么它们需要被单独验证：
  - 验证空字节 (%00)；
  - 验证换行符 (%0d, %0a, \r, \n)；
  - 验证路径替代字符”点-点-斜杠”（../或 ..\）。如果支持 UTF-8 扩展字符集编码，验证替代字符： %c0%ae%c0%ae/ (使用规范化 验证双编码或其他类型的编码攻击)。

## 输出编码

- ☐ 在可信系统（比如：服务器）上执行所有的编码。
- ☐ 为每一种输出编码方法采用一个标准的、已通过测试的规则。

- ☐ 通过语义输出编码方式，对所有返回到客户端的来自于应用程序信任边界之外的数据进行编码。HTML 实体编码是一个例子，但不是在所有的情况下都可用。
- ☐ 除非对目标编译器是安全的，否则请对所有字符进行编码。
- ☐ 针对 SQL、XML 和 LDAP 查询，语义净化所有不可信数据的输出。
- ☐ 对于操作系统命令，净化所有不可信数据输出。

## 身份验证和密码管理

- ☐ 除了那些特定设为"公开"的内容以外，对所有的网页和资源要求身份验证。
- ☐ 所有的身份验证过程必须在可信系统（比如：服务器）上执行。
- ☐ 在任何可能的情况下，建立并使用标准的、已通过测试的身份验证服务。
- ☐ 为所有身份验证控制使用一个集中实现的方法，其中包括利用库文件请求外部身份验证服务。
- ☐ 将身份验证逻辑从被请求的资源中隔离开，并使用重定向到或来自集中的身份验证控制。
- ☐ 所有的身份验证控制应当安全的处理未成功的身份验证。
- ☐ 所有的管理和账户管理功能至少应当具有和主要身份验证机制一样的安全性。
- ☐ 如果您的应用程序管理着凭证的存储，那么应当保证只保存了通过使用强加密单向 salted 哈希算法得到的密码，并且只有应用程序具有对保存密码和密钥的表/文件的写权限（如果可以避免的话，不要使用 MD5 算法）。
- ☐ 密码哈希必须在可信系统（比如：服务器）上执行。
- ☐ 只有当所有的数据输入以后，才进行身份验证数据的验证，特别是对连续身份验证机制。
- ☐ 身份验证的失败提示信息应当避免过于明确。比如：可以使用"用户名和/或密码错误"，而不要使用"用户名错误"或者"密码错误"。错误提示信息在显示和源代码中应保持一致。
- ☐ 为涉及敏感信息或功能的外部系统连接使用身份验证。
- ☐ 用于访问应用程序以外服务的身份验证凭据信息应当加密，并存储在一个可信系统（比如：服务器）中受到保护的地方。源代码不是一个安全的地方。
- ☐ 只使用 HTTP Post 请求传输身份验证的凭据信息。
- ☐ 非临时密码只在加密连接中发送或作为加密的数据（比如，一封加密的邮件）。通过邮件重设临时密码可以是一个例外。
- ☐ 通过政策或规则加强密码复杂度的要求（比如：要求使用字母、数字和/或特殊符号）。身份验证的凭据信息应当足够复杂以对抗在其所部署环境中的各种威胁攻击。
- ☐ 通过政策和规则加强密码长度要求。常用的是 8 个字符长度，但是 16 个字符长度更好，或者考虑使用多单词密码短语。
- ☐ 输入的密码应当在用户的屏幕上模糊显示（比如：在 Web 表单中使用"password"输入类型）。
- ☐ 当连续多次登录失败后（比如：通常情况下是 5 次），应强制锁定账户。账户锁定的时间必须足够长，以阻止暴力攻击猜测登录信息，但是不能长到允许执行一次拒绝服务攻击。
- ☐ 密码重设和更改操作需要类似于账户创建和身份验证的同样控制等级。
- ☐ 密码重设问题应当支持尽可能随机的提问（比如："最喜爱的书"是一个坏的问题，因为《圣经》是最常见的答案）。
- ☐ 如果使用基于邮件的重设，只将临时链接或密码发送到预先注册的邮件地址。

- ☐ 临时密码和链接应当有一个短暂的有效期。
- ☐ 当再次使用临时密码时，强制修改临时密码。
- ☐ 当密码重新设置时，通知用户。
- ☐ 阻止密码重复使用。
- ☐ 密码在被更改前应当至少使用了一天，以阻止密码重用攻击。
- ☐ 根据政策或规则的要求，强制定期更改密码。关键系统可能会要求更频繁的更改。更改时间周期必须进行明确。
- ☐ 为密码填写框禁用"记住密码"功能。
- ☐ 用户账号的上一次使用信息（成功或失败）应当在下一次成功登录时向用户报告。
- ☐ 执行监控以确定针对使用相同密码的多用户帐户攻击。当用户 ID 可以被得到或被猜到时，该攻击模式用来绕开标准的锁死功能。
- ☐ 更改所有厂商提供的默认用户 ID 和密码，或者禁用相关帐号。
- ☐ 在执行关键操作以前，对用户再次进行身份验证。
- ☐ 为高度敏感或重要的交易账户使用多因子身份验证机制。
- ☐ 如果使用了第三方身份验证的代码，仔细检查代码以保证其不会受到任何恶意代码的影响。

## 会话管理

- ☐ 使用服务器或者框架的会话管理控制。应用程序应当只识别有效的会话标识符。
- ☐ 会话标识符必须总是在一个可信系统（比如：服务器）上创建。
- ☐ 会话管理控制应当使用通过审查的算法以保证足够的随机会话标识符。
- ☐ 为包含已验证的会话标识符的 cookie 设置域和路径，以为站点设置一个恰当的限制值。
- ☐ 注销功能应当完全终止相关的会话或连接。
- ☐ 注销功能应当可用于所有受身份验证保护的网页。
- ☐ 在平衡的风险和业务功能需求的基础上，设置一个尽量短的会话超时时间。通常情况下，应当不超过几个小时。
- ☐ 禁止连续的登录并强制执行周期性的会话终止，即使是活动的会话。特别是对于支持富网络连接或连接到关键系统的应用程序。终止时机应当可以根据业务需求调整，并且用户应当收到足够的通知已减少带来的负面影响。
- ☐ 如果一个会话在登录以前就建立，在成功登录以后，关闭该会话并创建一个新的会话。
- ☐ 在任何重新身份验证过程中建立一个新的会话标识符。
- ☐ 不允许同一用户 ID 的并发登录。
- ☐ 不要在 URL、错误信息或日志中暴露会话标识符。会话标识符应当只出现在 HTTP cookie 头信息中。比如，不要将会话标识符以 GET 参数进行传递。
- ☐ 通过在服务器上使用恰当的访问控制，保护服务器端会话数据免受来自服务器其他用户的未授权访问。
- ☐ 生成一个新的会话标识符并周期性地使旧会话标识符失效（这可以缓解那些原标识符被获得的特定会话劫持情况）。
- ☐ 在身份验证的时候，如果连接从 HTTP 变为 HTTPS，则生成一个新的会话标识符。在应用程序中，推荐持续使用 HTTPS，而非在 HTTP 和 HTTPS 之间转换。
- ☐ 为服务器端的操作执行标准的会话管理，比如，通过在每个会话中使用强随机令牌或参数来管理账户。该方法可以用来防止跨站点请求伪造攻击。

- ☐ 通过在每个请求或每个会话中使用强随机令牌或参数，为高度敏感或关键的操作提供标准的会话管理。
- ☐ 为在TLS 连接上传输的cookie 设置"安全"属性。
- ☐ 将 cookie 设置为 HttpOnly 属性，除非在应用程序中明确要求了客户端脚本程序读取或者设置 cookie 的值。

## 访问控制

- ☐ 只使用可信系统对象（比如：服务器端会话对象）以做出访问授权的决定。
- ☐ 使用一个单独的全站点部件以检查访问授权。这包括调用外部授权服务的库文件。
- ☐ 安全的处理访问控制失败的操作。
- ☐ 如果应用程序无法访问其安全配置信息，则拒绝所有的访问。
- ☐ 在每个请求中加强授权控制，包括：服务器端脚本产生的请求，"includes"和来自象 AJAX 和 FLASH 那样的富客户端技术的请求。
- ☐ 将有特权的逻辑从其他应用程序代码中隔离开。
- ☐ 限制只有授权的用户才能访问文件或其他资源，包括那些应用程序外部的直接控制。
- ☐ 限制只有授权的用户才能访问受保护的 URL。
- ☐ 限制只有授权的用户才能访问受保护的功能。
- ☐ 限制只有授权的用户才能访问直接对象引用。
- ☐ 限制只有授权的用户才能访问服务。
- ☐ 限制只有授权的用户才能访问应用程序数据。
- ☐ 限制通过使用访问控制来访问用户、数据属性和策略信息。
- ☐ 限制只有授权的用户才能访问与安全相关的配置信息。
- ☐ 服务器端执行的访问控制规则和表示层实施的访问控制规则必须匹配。
- ☐ 如果状态数据必须存储在客户端，使用加密算法，并在服务器端检查完整性以捕获状态的改变。
- ☐ 强制应用程序逻辑流程遵照业务规则。
- ☐ 限制单一用户或设备在一段时间内可以执行的事务数量。事务数量/时间应当高于实际的业务需求，但也应该足够低以判定自动化攻击。
- ☐ 仅使用"referer"头作为补偿性质的检查，它永远不能被单独用来进行身份验证检查，因为它可以被伪造。
- ☐ 如果长的身份验证会话被允许，周期性的重新验证用户的身份，以确保他们的权限没有改变。如果发生改变，注销该用户，并强制他们重新执行身份验证。
- ☐ 执行帐户审计并将没有使用的帐号强制失效（比如：在用户密码过期后的 30 天以内）。
- ☐ 应用程序必须支持帐户失效，并在帐户停止使用时终止会话（比如：角色、职务状况、业务处理的改变，等等）。
- ☐ 服务帐户，或连接到或来自外部系统的帐号，应当只有尽可能小的权限。
- ☐ 建立一个"访问控制政策"以明确一个应用程序的业务规则、数据类型和身份验证的标准或处理流程，确保访问可以被恰当的提供和控制。这包括了为数据和系统资源确定访问需求。

## 加密规范

- ☐ 所有用于保护来自应用程序用户秘密信息的加密功能都必须在一个可信系统（比如：服务器）上执行。
- ☐ 保护主要秘密信息免受未授权的访问。
- ☐ 安全的处理加密模块失败的操作。
- ☐ 为防范对随机数据的猜测攻击，应当使用加密模块中已验证的随机数生成器生成所有的随机数、随机文件名、随机 GUID 和随机字符串。
- ☐ 应用程序使用的加密模块应当遵从 FIPS 140-2 或其他等同的标准（请见：STM Group）。
- ☐ 建立并使用相关的政策和流程以实现加、解密的密钥管理。

## 错误处理和日志

- ☐ 不要在错误响应中泄露敏感信息，包括：系统的详细信息、会话标识符或者帐号信息。
- ☐ 使用错误处理以避免显示调试或堆栈跟踪信息。
- ☐ 使用通用的错误消息并使用定制的错误页面。
- ☐ 应用程序应当处理应用程序错误，并且不依赖服务器配置。
- ☐ 当错误条件发生时，适当的清空分配的内存。
- ☐ 在默认情况下，应当拒绝访问与安全控制相关联的错误处理逻辑。
- ☐ 所有的日志记录控制应当在可信系统（比如：服务器）上执行。
- ☐ 日志记录控制应当支持记录特定安全事件的成功或者失败操作。
- ☐ 确保日志记录包含了重要的日志事件数据。
- ☐ 确保日志记录中包含的不可信数据，不会在查看界面或者软件时以代码的形式被执行。
- ☐ 限制只有授权的个人才能访问日志。
- ☐ 为所有的日志记录采用一个主要的常规操作。
- ☐ 不要在日志中保存敏感信息，包括：不必要的系统详细信息、会话标识符或密码。
- ☐ 确保一个执行日志查询分析机制的存在。
- ☐ 记录所有失败的输入验证。
- ☐ 记录所有的身份验证尝试，特别是失败的验证。
- ☐ 记录所有失败的访问控制。
- ☐ 记录明显的修改事件，包括对于状态数据非期待的修改。
- ☐ 记录连接无效或者已过期的会话令牌尝试。
- ☐ 记录所有的系统例外。
- ☐ 记录所有的管理功能行为，包括对于安全配置设置的更改。
- ☐ 记录所有失败的后端TLS 链接。
- ☐ 记录加密模块的错误。
- ☐ 使用加密哈希功能以验证日志记录的完整性。

## 数据保护

- ☐ 授予最低权限，以限制用户只能访问为完成任务所需要的功能、数据和系统信息。

- ☐ 保护所有存放在服务器上缓存的或临时拷贝的敏感数据，以避免非授权的访问，并在临时工作文件不再需要时被尽快清除。
- ☐ 即使在服务器端，任然要加密存储的高度机密信息，比如，身份验证的验证数据。总是使用已经被很好验证过的算法，更多指导信息请参见”加密规范”部分。
- ☐ 保护服务器端的源代码不被用户下载。
- ☐ 不要在客户端上以明文形式或其他非加密安全模式保存密码、连接字符串或其他敏感信息。这包括嵌入在不安全的形式中： MS viewstate、 Adobe flash 或者已编译的代码。
- ☐ 删除用户可访问产品中的注释，以防止泄露后台系统或者其他敏感信息。
- ☐ 删除不需要的应用程序和系统文档，因为这些也可能向攻击者泄露有用的信息。
- ☐ 不要在 HTTP GET 请求参数中包含敏感信息。
- ☐ 禁止表单中的自动填充功能，因为表单中可能包含敏感信息，包括身份验证信息。
- ☐ 禁止客户端缓存网页，因为可能包含敏感信息。“Cache-Control: no-store”，可以和 HTTP 报头控制”Pragma: no-cache”一起使用，该控制不是非常有效，但是与 HTTP/1.0 向后兼容。
- ☐ 应用程序应当支持，当数据不再需要的时候，删除敏感信息（比如：个人信息或者特定财务数据）。
- ☐ 为存储在服务器中的敏感信息提供恰当的访问控制。这包括缓存的数据、临时文件以及只允许特定系统用户访问的数据。

## 通讯安全

- ☐ 为所有敏感信息采用加密传输。其中应该包括使用 TLS 对连接的保护，以及支持对敏感文件或非基于 HTTP 连接的不连续加密。
- ☐ TLS 证书应当是有效的，有正确且未过期的域名，并且在需要时，可以和中间证书一起安装。
- ☐ 没有成功的TLS 连接不应当后退成为一个不安全的连接。
- ☐ 为所有要求身份验证的访问内容和所有其他的敏感信息提供 TLS 连接。
- ☐ 为包含敏感信息或功能、且连接到外部系统的连接使用 TLS。
- ☐ 使用配置合理的单一标准 TLS 连接。
- ☐ 为所有的连接明确字符编码。
- ☐ 当链接到外部站点时，过滤来自 HTTP referer 中包含敏感信息的参数。

## 系统配置

- ☐ 确保服务器、框架和系统部件采用了认可的最新版本。
- ☐ 确保服务器、框架和系统部件安装了当前使用版本的所有补丁。
- ☐ 关闭目录列表功能。
- ☐ 将 Web 服务器、进程和服务的账户限制为尽可能低的权限。
- ☐ 当例外发生时，安全的进行错误处理。
- ☐ 移除所有不需要的功能和文件。
- ☐ 在部署前，移除测试代码和产品不需要的功能。
- ☐ 通过将不进行对外检索的路径目录放在一个隔离的父目录里，以防止目录结构在 robots.txt 文档中暴露。然后，在 robots.txt 文档中”禁止”整个父目录，而不是对每个单独目录的”禁止”。

- ☐ 明确应用程序采用哪种 HTTP 方法：GET 或 POST，以及是否需要在应用程序不同网页中以不同的方式进行处理。
- ☐ 禁用不需要的 HTTP 方法，比如WebDAV 扩展。如果需要使用一个扩展的 HTTP 方法以支持文件处理，则使用一个好的经过验证的身份验证机制。
- ☐ 如果 Web 服务器支持 HTTP1.0 和 1.1，确保以相似的方式对它们进行配置，或者确保您理解了它们之间可能存在差异（比如：处理扩展的 HTTP 方法）。
- ☐ 移除在 HTTP 相应报头中有关 OS、Web 服务版本和应用程序框架的无关信息。
- ☐ 应用程序存储的安全配置信息应当可以以可读的形式输出，以支持审计。
- ☐ 使用一个资产管理系统，并将系统部件和软件注册在其中。
- ☐ 将开发环境从生成网络隔离开，并只提供给授权的开发和测试团队访问。开发环境往往没有实际生成环境那么安全，攻击者可以使用这些差别发现共有的弱点或者是可被利用的漏洞。
- ☐ 使用一个软件变更管理系统以管理和记录在开发和产品中代码的变更。

## 数据库安全

- ☐ 使用强类型的参数化查询方法。
- ☐ 使用输入验证和输出编码，并确保处理了元字符。如果失败，则不执行数据库命令。
- ☐ 确保变量是强类型的。
- ☐ 当应用程序访问数据库时，应使用尽可能最低的权限。
- ☐ 为数据库访问使用安全凭证。
- ☐ 连接字符串不应当在应用程序中硬编码。连接字符串应当存储在一个可信服务器的独立配置文件中，并且应当被加密。
- ☐ 使用存储过程以实现抽象访问数据，并允许对数据库中表的删除权限。
- ☐ 尽可能地快速关闭数据库连接。
- ☐ 删除或者修改所有默认的数据库管理员密码。使用强密码、强短语，或者使用多因子身份验证。
- ☐ 关闭所有不必要的数据库功能（比如：不必要的存储过程或服务、应用程序包、仅最小化安装需要的功能和选项（表面范围缩减））。
- ☐ 删除厂商提供的不必要的默认信息（比如：数据库模式示例）。
- ☐ 禁用任何不支持业务需求的默认帐户。
- ☐ 应用程序应当以不同的凭证为每个信任的角色（比如：用户、只读用户、访问用户、管理员）连接数据库。

## 文件管理

- ☐ 不要把用户提交的数据直接传送给任何动态调用功能。
- ☐ 在允许上传一个文档以前进行身份验证。
- ☐ 只允许上传满足业务需要的相关文档类型。
- ☐ 通过检查文件报头信息，验证上传文档是否是所期待的类型。只验证文件类型扩展是不够的。
- ☐ 不要把文件保存在与应用程序相同的 Web 环境中。文件应当保存在内容服务器或者数据库中。

- ☐ 防止或限制上传任意可能被 Web 服务器解析的文件。
- ☐ 关闭在文件上传目录的运行权限。
- ☐ 通过装上目标文件路径作为使用了相关路径或者已变更根目录环境的逻辑盘，在 UNIX 中实现安全的文件上传服务。
- ☐ 当引用已有文件时，使用一个白名单记录允许的文件名和类型。验证传递的参数值，如果与预期的值不匹配，则拒绝使用，或者使用默认的硬编码文件值代替。
- ☐ 不要将用户提交的数据传递到动态重定向中。如果必须允许使用，那么重定向应当只接受通过验证的相对路径 URL。
- ☐ 不要传递目录或文件路径，使用预先设置路径列表中的匹配索引值。
- ☐ 绝对不要将绝对文件路径传递给客户。
- ☐ 确保应用程序文件和资源是只读的。
- ☐ 对用户上传的文件扫描进行病毒和恶意软件。

## 内存管理

- ☐ 对不可信数据进行输入和输出控制。
- ☐ 重复确认缓存空间的大小是否和指定的大小一样。
- ☐ 当使用允许多字节拷贝的函数时，比如 strncpy()，如果目的缓存容量和源缓存容量相等时，需要留意字符串没有 NULL 终止。
- ☐ 如果在循环中调用函数时，检查缓存大小，以确保不会出现超出分配空间大小的危险。
- ☐ 在将输入字符串传递给拷贝和连接函数前，将所有输入的字符串缩短到合理的长度。
- ☐ 关闭资源时要特别注意，不要依赖垃圾回收机制（比如：连接对象、文档处理，等）。
- ☐ 在可能的情况下，使用不可执行的堆栈。
- ☐ 避免使用已知有漏洞的函数（比如：printf，strcat，strcpy，等）。
- ☐ 当方法结束时和在所有的退出节点时，正确地清空所分配的内存。

## 通用编码规范

- ☐ 为常用的任务使用已测试且已认可的托管代码，而不创建新的非托管代码。
- ☐ 使用特定任务的内置 API 以执行操作系统的任务。不允许应用程序直接将代码发送给操作系统，特别是通过使用应用程序初始的命令shell。
- ☐ 使用校验和或哈希值验证编译后的代码、库文件、可执行文件和配置文件的完整性。
- ☐ 使用死锁来防止多个同时发送的请求，或使用一个同步机制防止竞态条件。
- ☐ 在同时发生不恰当的访问时，保护共享的变量和资源。
- ☐ 在声明时或在第一次使用前，明确初始化您的所有变量和其他数据存储。
- ☐ 当应用程序运行发生必须提升权限的情况时，尽量晚点提升权限，并且尽快放弃所提升的权限。
- ☐ 通过了解您使用的编程语言的底层表达式以及它们是如何进行数学计算，从而避免计算错误。密切注意字节大小依赖、精确度、有无符合、截尾操作、转换、字节之间的组合、“not-a-number”计算、以及对于编程语言底层表达式如何处理非常大或者非常小的数。
- ☐ 不要将用户提供的数据传递给任何动态运行的功能。

- ☐ 限制用户生成新代码或更改现有代码。
- ☐ 审核所有从属的应用程序、第三方代码和库文件，以确定业务的需要，并验证功能的安全性，因为它们可能产生新的漏洞。
- ☐ 执行安全更新。如果应用程序采用自动更新，则为您的代码使用加密签名，以确保的您的下载客户端验证这些签名。使用加密的信道传输来自主机服务器的代码。

# 附录 A: 外部的参考资料

## 引用的参考资料

- SANS CIS Controls version 8

  https://www.sans.org/blog/cis-controls-v8/
- Web Application Security Consortium

  http://www.webappsec.org/
- Common Weakness Enumeration (CWE)

  https://cwe.mitre.org/
- Department of Homeland Security: Build Security In Portal

  https://www.cisa.gov/uscert/bsi
- CERT Secure Coding

  http://www.cert.org/secure-coding/
- MSDN Security Developer Center

  http://msdn.microsoft.com/en-us/security/default.aspx

## 安全咨询网站:

检查已知威胁的有用资源，以支持基础设施和框架。

Secunia Citrix Vulnerability List:

- https://www.flexera.com/products/software-vulnerability-research/secunia-research

Security Focus Vulnerability Search (archived):

- https://bugtraq.securityfocus.com/archive

Common Vulnerability Enumeration:

- https://www.cve.org/

# 附录 B: 术语表

**滥用实例**：描述了对软件有意或无意的错误使用。滥用实例应当挑战系统设计的各种假设。

**访问控制**：一组授予或拒绝用户或其他实体对系统资源访问的控制。这通常基于继承的角色和在一个角色中的单独权限，但也包含系统与系统之间的交互。

**身份验证**：一组验证正在与软件交互的用户或其他实体身份的控制

**可用性**：一个系统可访问性和可用性的一种度量。

**规范化**：将数据的各种编码方式和表式转换成为一种单一的简单形式。

**通信安全**：一组帮助确保软件以一种安全的方式处理信息收发的控制。

**保密性**：确保信息只公布给授权的当事人。

**语义输出编码方式**：根据数据如何被应用程序使用，对输出进行编码。具体方法有所不同，取决于输出数据的使用方式。如果数据被包含在返回客户端的响应中，请说明下面的情况：在 JavaScript 中、在 CSS 中或在 URL 中的一个 HTML 文档的主体部分或一个 HTML 属性。您还必须说明其它的用例，比如：SQL 查询、XML 和 LDAP。

**跨站点请求伪造**：一个外部网站或应用程序强迫一个客户端制作一个非意愿的请求到另一个有活跃会话的客户端应用程序。当它们使用已知的、可预见的 URL 和参数时，并且当浏览器对每个发往应用程序的请求自动传输所有需要的会话信息时，应用程序是可被攻击的。（该类攻击是本文中唯一包含并讨论的攻击，因为相关的漏洞非常常见的，但又被知之甚少。）

**密码规范**：一组确保在应用程序中密码的操作被安全处理的控制。

**数据保护**：一组确保软件以一种安全的方式处理所保存的信息的控制。

**数据库安全**：一组确保软件以一种安全的方式与数据库交互，并且数据库被安全配置的控制。

**错误处理和日志记录**：一组确保应用程序安全的处理错误并执行正确的事件记录的实践。

**利用**：使用一个漏洞。通常而言，这是一种有意的行为，旨在利用漏洞危及软件的安全控制。

**文件管理**：一组涵盖代码和其它系统文件之间交互的控制。

**通用编码规范**：一组涵盖不易分入其它组的编码实践。

**危险字符**：任何字符或编码表示的字符，可以影响应用程序预期的操作，或者被解释为具有特殊的意义、预期用途以外的字符。这些字符可用于：

- 修改现有代码或语句的结构；
- 插入新的非预期的代码；
- 修改路径；
- 程序功能或程序导致非预期的结果；
- 导致错误的条件；
- 下游应用程序或系统拥有上述任何效果。

**HTML 实体编码**：用等价的 HTML 实体取代 ASCII 字符的过程。比如：编码过程将用 HTML 等价 “&lt”取代小于号”<“。HTML 实体在大多数的编译器中是”无行动的”，特别是在浏览器中，可以缓解某些客户端攻击。

**影响**：当一个意外事件发生时，也就是一个漏洞被利用时，对业务负面影响的衡量。

**输入验证**：一组验证所有输入数据的属性与应用程序所期待数据的类型、长度、范围、允许的字符集所匹配，但不包含已知危险字符的控制。

**完整性**： 保证信息是准确的、完整的和有效的，并且没有被一个未授权的行为所修改。

**日志事件数据**： 这应当包含以下信息：

1. 来自一个可信系统组件的时间戳；
2. 每个事件的严重程度；
3. 如果与其它日志事件混合在一起的话，标记与安全相关的事件；
4. 明确引发事件的帐户或用户身份；
5. 与请求相关的源 IP 地址；
6. 事件的结果（成功或失败）；
7. 事件的描述。

**内存管理**： 一组解决内存和缓存使用的控制。

**缓解**：为减少一个漏洞的严重程度而采取的步骤。这些步骤可以包括：删除一个漏洞、使漏洞更难被利用、减少漏洞被利用后带来的负面影响。

**多因子身份验证**： 一个要求用户处理多种不同种类凭据的身份验证过程。通常，该过程基于用户拥有的信息（比如：智能卡）、他们知道的信息（比如：个人识别码）、或者他们自身的信息（比如：来自生物方面的数据）。

**输出编码**： 一组解决使用编码技术以确保应用程序输出的数据是安全的控制。

**参数化查询（预处理语句）**： 通过使用占位符，保持查询语句和数据相分离。
查询语句结构由占位符定义，SQL 语句发送给数据库并做好准备，然后准备好的语句与参数值相结合。这样就防止了查询被更改，因为参数值与已编译的语句相结合，
而不是 SQL 字符串。

**净化数据**：通过使用删除数据、字符替代、编码或转义字符，将潜在的有害数据变得安全的处理过程。

**安全控制**：一个缓解潜在的漏洞，并有助于确保软件只以预期的模式进行工作的行为。

**安全需求**：一组有助于确保软件以一种安全的模式建立并部署的设计和功能需求。

**连续身份验证**： 身份验证数据在连续的网页上被请求，而不是在一个单独的网页中一起被要求。

**会话管理**：一组有助于确保 Web 应用程序以安全的模式处理 HTTP 会话的控制。

**状态数据**： 当应用程序或者服务器使用数据或参数模拟持续的连接，或跟踪跨多个请求的处理或交易的客户端状态。

**系统**： 一个涵括操作系统、Web 服务器、应用程序框架和相关基础设施的通用术语。

**系统配置**：一组帮助确保基础设施部件支持软件安全部署的控制。

**威胁代理**：任何可能对系统造成负面影响的实体。这可能是一个想危害系统安全控制的恶意用户。但是，这也可能是一个意外的系统错误使用，或者是物理威胁，如火灾或水灾。

**信任边界**：通常，一个信任边界由您直接控制的系统部件组成。来自您直接控制系统以外的所有连接和数据，包括所有由其它各方管理的客户端和系统，在允许进一步的系统交互之前，应当被认为是不可信的，并在边界被验证。

**漏洞**：一个导致系统易受攻击或损坏的脆弱点。