十大 Web 应用程序安全漏洞

2004 更新

二零零四年一月二十七日

翻译者：钟卫林，于振东，乐维

评论

“几乎每周都有新的安全漏洞被公布，很多企业可能会觉得跟不上这样的步伐。但是，有大家公认的漏洞和防御方法的列表可以帮助你。

“开放 Web 应用安全计划提供了一个针对 Web 应用和数据库安全的十大漏洞的列表，以及对付他们的最有效的方法。应用程序的漏洞常常被忽略，但是它们却是和网络安全一样重要。如果每个公司都消除了这些漏洞，他们尽管不能完全保证安全，但是他们和整个因特网都会安全得多。”

- ***J.霍华德 比尔斯 三世 J. Howard Beales, III，联邦贸易委员会消费者保护局局长，在美国信息技术联合会的因特网政策委员会上的发言，2003 年 12 月 12 日，星期五***

“错误的配置，不够谨慎，和有漏洞的软件可以在因特网上造成灾难。其中一个最主要的漏洞就是 Web 连接。从设计上说，WWW 服务是开放的，常常作为对有价值资源访问的接口。因而，保证这些服务的安全是至关重要的。但是，由于存在数以百计的漏洞，决定从何处防守便成了一个问题。OWASP 的十大提供了一个公认的在 Web 里最严重的可能出现的漏洞列表。各个机构应该使用这个列表用于集中他们的精力，让他们有信心处理有最大影响的区域。”

- ***尤金 H 思帕福特，普度大学计算机科学教授，普度大学信息保证和安全教育研究中心执行指导***

“这个十大列表准确地描述了巨大冰山露出来的一角。下面的现实是可耻的：大多数的系统和 Web 应用软件都忽略了安全原则，软件工程，操作影响，和真正的常识。”

- ***彼得 G 纽曼博士，SRI 国际计算机实验室首席科学家；ACM 风险论坛主席，“计算机相关的风险”作者***

“该列表对于消费者和厂家而言是很重要的发展。它会教育厂家来避免在其他的 web 应用里出现了无数多次的错误。但是它也给消费者一个方法，来要求厂家遵守一个最小集的对 web 应用安全的期望，同时，也找出没有到达期望的厂家。”

- ***史蒂夫 M 克里思泰，Mitre 首席信息安全工程师，CVE 编辑，***

“OWASP 十大直接指出了最严重的，常常被忽略的政府和商业组织面对的风险。这些风险的根源不是有漏洞的软件，而是不考虑安全的软件开发过程。在你的组织里建立有安全意识的文化最有效的第一步是立刻采用十大作为 web 应用安全的最小化标准。”

- ***杰夫 R 威廉姆斯 Aspect Security CEO，OWASP 十大负责人***

“尽管有些技术公司让你相信有对付 web 安全的万灵药，但是真的没有。解决这个复杂的有挑战性的问题是有好的人力，好的知识，好的教育，好的商业过程，使用好的技术。OWASP 十大创建了一个考虑周到的列表，从里面你可以开始理解你的安全姿态（或者你的服务提供商的），和计划使用你有限的资源”

- ***马克 克菲，OWASP 创建人，Foundstone 战略安全应用安全咨询主任***

“从网页到办公室，几乎所有的组织都购买应用代码，很多写应用代码，每个人都使用应用代码。但是，有了几乎 50 年的编程经验之后，应用程序里的漏洞还是继续被发现。更糟糕的是，同样的漏洞一遍又一遍的出现。没有做到从我们自己的错误中，也从我们的父母那一代的错误中学习，造成了太多的漏洞。这就是为什么针对应用的攻击越来越多。

在编辑十大最严重的应用代码漏洞的过程中，OWASP 侧重常见的弱点和相关对策，给开发人员和用户提供了真正的服务。现在是软件开发组织，程序员，和用户来使用这个考虑周全的指南的时候了”

- ***查尔斯 P 福利格 CISSP，Cable & Wireless 安全结构师，“安全计算”作者***

“新的投资回报（ROI）是安全。公司必须能够提供可信任的 web 应用和 web 服务给贸易伙伴，他们需要安全的技术和传统的金融上的标记，比如保险。它不只停留在这里；坏的安全将一个公司的数据和应用置于险地，因此其商业实体的实行性也陷于险地。让你的客户失望是一件事，从你自己系统的损失和中断的状态中恢复是另一件事。”

■ ***罗伯特 A 帕里斯， AIG 电子商务风险解决方案高级副总裁，首席担保官***

“Web 开发人员应该知道，他们的商业应用和客户数据受保护的程度是由他们的代码安全度决定的。OWASP 十大列表是一个很好的方法来理解如何保护性的写代码，和避免 web 应用的安全漏洞”

■ ***克里斯 维索包，@Stake Inc 研究发展主任***

“医疗工业需要提供安全的 web 应用来保护用户的隐私。OWASP 十大将帮助医疗组织评估 web 应用和解决方案的安全。任何使用包含这些漏洞的应用的医疗组织可能有困难达到 HIPAA 规定。”

■ ***丽沙 高拉格，URAC 信息和技术认证高级副总裁***

“随着越来越多的公司将 web 应用开发外包出去，使用安全的代码必须成为管理团队的优先列表的首条。OWASP 十大计划准确的描述了十大应用漏洞，给执行人员一个金矿，来帮助设立 Web 应用的安全策略。”

■ ***司徒亚特 麦克鲁， Foundstone Inc 总裁，首席技术官***

“应用安全的考量常常被当作专家的领域，在代码完成后其他商业需求满足后来实施。在 Oracle，我们发现安全化应用的人物从开始最有效。开发人员必须对安全漏洞和如何应付有基本的了解，发布经理在产品出厂前必须明白地检查安全需求被满足了。对于公司在开发人员中建立安全的意识，准备应用部署安全标准，OWASP 十大是一个很好的起始点。”

■ ***约翰 黑曼 Oracle 安全主任***

“很多人都看过描述‘最大的网络安全风险’的各种十大列表。那些列表只指出可能或者不可能在你的网络中运行的第三方软件中的漏洞。使用这个列表中的信息则会让你自己开发的软件和其他的列表无关！”

■ ***约翰 维依迦 Secure Software Inc 首席科学家，“Building Secure Software”作者之一***

目录

介绍

开放 Web 应用安全项目(OWASP)目的在于帮助理解和提高 Web 应用和服务的安全性。本列表帮助企业和政府机构关注最严重的漏洞。随着各个公司都将内容和服务放在网上，Web 应用安全已经成为一个热门话题。同事，黑客也将他们的注意力转移到应用开发人员常犯的错误中。

当一个组织建立一个 web 应用程序，他们邀请整个世界给他们发送 HTTP 请求。攻击被嵌在这些请求中，穿过防火墙，过滤器，被加强了的平台，和入侵检测系统，而不被察觉，因为它们是在合法的 HTTP 请求中。甚至使用 SSL 的“安全”网站也接受这些通过加密通道传送的请求，而不仔细检查。**这就意味着你的 web 应用代码就是你的安全参数之一。**随着你的 web 应用的数目和复杂度的增加，对外的暴露也会增加。

这些安全问题并不新。事实上，有些问题已经存在，并被了解有很多年了。然而，由于种种原因，主要的软件开发项目还是存在这样的问题。这不仅将他们的客户置于险地，也将整个因特网的置于险地。治疗这些问题没有万能药。现在的评估和保护的技术在提高，但是只能处理以上问题的有限的一部分。为了解决在本文谈到的问题，公司需要改变他们的开发文化，训练开发人员，改变他们的软件开发流程，并且使用合适的技术。

OWASP 十大列表是一个需要立刻处理的漏洞列表。应该立刻检查现存的代码里面的漏洞。软件开发项目应该在需求文档里涉及到这些漏洞，设计，实现和测试他们的应用程序以保证没有这样的漏洞。项目经理应该将应用安全活动如开发人员培训，应用安全策略开发，安全机制设计和开发，渗透测试，和安全代码检查所需的时间和金钱考虑一下。

我们鼓励大家加入到**使用 OWASP 十大作为最低标准**的队伍里来，承诺开发不包含这些漏洞的 web 应用程序。

我们选择了类似于非常成功的 SANS/FBI 二十大列表的形式，来确保它被很好的理解。SANS 列表侧重于网络和基本框架的产品的漏洞上。由于每个网站都是与众不同的，OWASP 十大列表围绕着漏洞的特定的形式或者类别来组织的。

本列表体现了 OWASP 专家的集体智慧。他们有很多年的政府机构，金融机构，制药，生产公司的应用安全工作经验。本文档的目的是介绍最严重的 web 应用漏洞。有很多书籍和指导性的文档更仔细地描述了这些漏洞，提供了仔细的消除它们的办法。OWASP 的指南 http://www.owasp.org 就是其中一个。

OWASP 十大是一个活着的文档。它包括解决这些类型安全漏洞的办法，和指向更多信息的指针。当更严重的威胁和更加流行的方法被了解时，我们就会更新这个列表。我们欢迎你的建议。这是一个描述共识的文档－你和攻击者的斗争，和消除漏洞的经验能够帮助其他人。请将你的建议寄给 topten@owasp.org，标题为”OWASP Top Ten Comments”。

OWASP 感谢 Aspect Security 在本文档的研究和准备中作出的贡献。

http://www.aspectsecurity.com

背景

定义最严重的 web 应用漏洞基本上是一个不可能的任务。甚至连“web 应用安全”应该包含什么都没有共识。有些人认为我们应该只关注影响开发人员写 web 应用代码的安全问题。另一些人则认为该概念应该更广，涉及整个应用层，包括库，服务器配置，应用层协议。针对最严重风险，我们决定使用广义上的 web 应用安全的定义，同时不涉及网络和基础建设的安全问题。

另一个挑战是每一个漏洞对于一个特定的组织的网站是独一无二的。指出具体的漏洞没有太大的意义，尤其很多人知道其存在之后，会被很快修复。因而，我们侧重于 web 应用漏洞的类别，类型或者种类。

在本文档的第一个版本中，我们决定将很广泛的 web 应用问题分成有意义的类别。我们研究了很多漏洞分类体系，提出了我们的类别。决定一个好的漏洞类别的因素包括这些漏洞是否紧密相关，能够使用类似的反制措施，和经常出现在 web 应用体系结构中。在这个版本里，我们引入一个更仔细的体系。我们在 OASIS WAS 技术委员会里有一个正在进行的项目，描述了很多问题，包含安全研究人员能够在 XML 格式里描述签名。

从很多备选项中选出十大本身就有很多困难。没有有关 web 应用安全问题的可靠的统计数据。在将来，我们想收集一些有关漏洞出现频率的统计数据，来使用该数据给十大列表排序。然而，由于种种原因，这样的测量在不久的将来是不太可能发生的。

我们认识到，对于某个漏洞类别是否应该出现在十大列表里没有“正确的”答案。每一个组织应该考虑他们可能由于有这样的漏洞而产生的风险，以及其独特的后果。当前情况下，我们将本列表当作一系列的问题，体现了很多组织所面对的很大的风险。十大本身没有有限次序，因为我们基本不可能决定哪一些体现了最大的风险。

OWASP 十大计划是一个正在进行中的尝试，用来将有关关键的 web 应用安全问题的信息提供给大众。我们期盼根据 OWASP 邮件列表和发给 topten@owasp.org 的反馈每年更新本文档一次。

更新

自从 2003 年一月发布十大文档以来，OWASP 走过了很多的路。本更新包括过去 12 月以来在 OWASP 社区内发生的所有的讨论，最新的看法，观点和辩论。总的来说，十大的各部分都有了的小改动。少量的大改动如下：

- **WAS-XML Alignment** － 2003 年在 OASIS 的 Web 应用安全技术委员会（WAS TC）发起一个项目。WAS TC 的目标是产生一个安全漏洞的分类策略，一个提供威胁，影响，和风险评估的模型，和一个可以被评估和保护工具使用的来描述 web 安全情况的 XML 结构。OWASP 十大计划被 WAS TC 用作 web 应用安全漏洞分类标准化的参考。WAS 词典定义了一个标准语言用于讨论 web 应用安全，我们在这里使用里面的词汇。
- **加入了“拒绝访问”**。唯一对定级的分类的改变是将 A9 拒绝访问加入到表中。我们的研究表明，很多组织容易受到这种类型的攻击。基于拒绝访问发生的可能性，以及其后果，我们将去年的 A9“远程管理漏洞”放到了 A2“不正确的访问控制”里，因为它是其特例。由于 A2 里面的漏洞和 A9 里面的漏洞通常相同，都需要相同的弥补措施，我们相信这是合理的。

下表强调了新的十大，去年的十大，和 WAS TC 词典之间的关系：

新十大（2004）	十大（2003）	新的 WAS 辞典
A1 未验证的输入	A1 未验证的参数	输入验证
A2 错误的访问控制	A2 错误的访问控制 (A9 远程管理漏洞)	访问控制
A3 错误的验证和会话管理	A3 错误的验证和会话管理	验证和会话管理
A4 跨站脚本	A4 跨站脚本	输入验证->跨站教本
A5 缓冲溢出	A5 缓冲溢出	缓冲溢出
A6 注入式漏洞	A6 命令注入式漏洞	输入验证->注入
A7 不正确的错误处理	A7 错误处理的问题	错误处理
A8 不安全的存储	A8 密码学的不安全的使用	数据保护
A9 拒绝访问	无	可使用性
A10 不安全的配置管理	A10 Web 和应用服务器的错误配置	应用配置管理 基础设施配置管理

十大列表

以下就是一个最严重的 web 应用安全漏洞的简短摘要。它们的每一个都会在接下来的章节里仔细描述。

Web 应用十大漏洞		
A1	**未验证的输入**	Web 请求在被 web 应用使用前没有被验证。攻击者可以使用该漏洞通过 web 应用程序来攻击后端的组件。
A2	**错误的访问控制**	验证过的用户的权限限制未被执行。攻击者可以利用这些漏洞来访问其他用户的帐号，浏览敏感文件，或者使用未被授权的功能。
A3	**错误的验证和会话管理**	帐号信息和会话 token 没有被正确的保护。攻击者通过获取密码，密钥，会话 cookie，或者其他 token 来破坏认证限制，从而获得其他用户的身份。
A4	**跨站脚本**	Web 应用可以被用来对用户的浏览器攻击。一个成功的攻击可以暴露用户的会话 token，攻击本地机器，或者用假消息来欺骗用户。
A5	**缓冲溢出**	有些不验证输入的 Web 应用组件可能会崩溃。在某些情况下，来用于获得一个进程的控制权。这些组件包括 CGI，库，驱动程序，和 web 应用服务器组件。
A6	**注入式漏洞**	Web 应用在访问外部系统时会传递参数。如果一个攻击者可以将恶意代码嵌入到参数中，外部系统可能以 web 应用的身份会执行这些命令。
A7	**不合适的错误处理**	在正常运作中出现的错误没有合理地处理。如果攻击者可以产成系统不能处理的错误，他们就可以获得详细的系统信息，进一步可以拒绝服务，使安全机制无效，或者使系统崩溃。
A8	**不安全的存储**	Web 应用常常使用密码函数来保护信息。这些函数以及使用他们的代码被证明很难来正确的使用，常常使保护变弱。
A9	**拒绝服务**	攻击者可以消耗 web 应用资源，以至于合法的用户不能访问或者使用应用程序。攻击者也可以使用户帐号被锁不能使用，或者导致整个应用程序不能运行。
A10	**不安全的配置管理**	严格的服务器配置标准对于安全 web 应用程序很重要。服务器通常有很多有关安全的配置选项，而这些选项缺省是不安全的。

A1 未被验证的输入

A1.1 描述

Web 应用使用 HTTP 请求（有时是文件）的输入来决定如何反应。攻击者可以篡改 HTTP 请求的任意部分，包括 URL，请求字符串，头，cookie，表的内容项，以及隐藏的项，来试图绕过网站的安全机制。常见的输入篡改攻击有：强制浏览，命令行插入，跨站教本，缓存溢出，格式化串攻击，SQL 注入，毒化 cookie，和隐藏项篡改。每种攻击类型在本文档中会仔细描述。

- A4－跨站教本讨论输入中包含在其他用户浏览器上执行的教本
- A5－缓存溢出讨论用于覆盖程序执行空间的输入
- A6－注入式漏洞讨论包含可执行命令的输入

有些网站通过过滤出恶意输入来保护自己。问题是，有非常多种不同的信息编码方法。这些编码格式不像加密，因为它们容易破解。然而，开发人员常常忘记在使用参数前进行解码。参数在被验证前，必须被转化成最简单的形式，否则，恶意输入可以绕过这些过滤器。这个使编码最简单化的过程被称为“canonicalization”。由于几乎所有的 HTTP 都可以多种形式表现出来，该技术可以用来使本文中描述的代码模糊攻击。这使得过滤变得很困难。

大量的 web 应用只使用了客户端的机制来验证输入。客户端的验证机制很容易被绕过，使得 web 应用对恶意参数缺乏保护。攻击者可以使用简单的工具比如 telnet，来产生他们自己的 HTTP 请求。他们不必理会开发人员放在客户端的限制。注意，客户端验证的性能和实用性很好，可是它没有任何的安全性。我们必须使用服务端的检查来对付参数篡改攻击。一旦这些检查设置好了，客户端的检查也可以用来增强用户的体验，以及减少对服务器的无效流量。

随着支持参数“模糊化（fuzzing）”，数据损坏，蛮力攻击（brute-force）的工具越来越多，这样的攻击会不断增多。使用未被验证的输入所产生的影响不应该被低估。如果开发人员只是在使用前做一个简单的验证，很多的攻击都会变得困难，或者不可能。除非一个 web 应用有一个强大的，集中的机制用于验证来自 HTTP 和其他来源的请求，基于恶意输入的漏洞非常可能存在。

A1.2 影响的环境

所有的 web 服务器，应用服务器，和 web 应用环境都会受到参数篡改的影响。

A1.3 例子和参考

- OWASP 建立安全 web 应用和 Web 服务指南，第八章：数据验证 http://www.owasp.org/documentation/guide/
- Modsecurity 计划（Apache 的 HTTP 验证模块）http://www.modsecurity.org
- 如何建立一个 HTTP 请求验证引擎（用 Stinger 做 J2EE 验证）
  http://www.owasp.org/columns/jeffwilliams/jeffwilliams2
- Have Your Cake and Eat it Too (.Net 验证) http://www.owasp.org/columns/jpoteet/jpoteet2

A1.4 如何确定你是否容易被攻击

被 Web 应用使用而未被仔细验证的任何 HTTP 请求被称为“被污染的（tainted）”参数。最简单的寻找被污染的参数的方法是做一个自己的代码检查，找所有的将 HTTP 请求中将具体信息取出的调用。例如，在 J2EE 应用里，在 HttpServletRequest 类里的方法。然后，你就可以看代码中什么变量被使用了。如果变量在使用前没有被检查，很可能就有问题。在 Perl 里，你应该考虑使用”tain” (-T)的选项。

你也可以使用工具，比如 OWASP 的 WebScarab 来寻找被污染的参数。通过在 HTTP 请求中发送“意外的”值，观察 web 应用的反应，你可以找到被污染的参数被使用的地方。

A1.5 如何保护你自己

防止参数篡改最好的办法是确保所有的参数在使用前被验证过。一个统一的组件或者库可能是最有效的办法，因为做检查的代码应该在同一个地方。每一个参数都应该限制输入具体是什么样子的形式。使用过滤某些坏的输入的“反向的（negative）”方法，或者依赖于签名的方法不太可能奏效，并且可能难以维护。

参数应该针对“正面的”规范来进行验证，它定义了：

- 数据类型（串，整型，实数，等）
- 允许的字符集
- 最小和最大的长度
- 是否允许空输入
- 参数是否是必须的
- 重复是否允许
- 数值范围
- 特定的值（枚举型）
- 特定的模式（正规表达式）

有一类新的被成为 Web 应用防火墙的安全设备能够提供一些参数验证服务。然而，为了让他们有效，该设备必须严格的配置每一个参数。这包括，正确的保护所有类型的来自 HTTP 的输入，包括 URL，表，cookies，查询串，隐藏项，和参数。

OWASP 过滤器项目正在产生多种语言下可复用的组件，用来帮助保护很多种形式的参数篡改。Stinger HTTP 请求验证引擎(stinger.sourceforge.net) 也是 OWASP 为 J2EE 环境下开发的。

A2 错误的访问控制

A2.1 描述

访问控制 (Access Control)，有时也被称为授权(authorization)，是指 web 应用程序如何给不同用户授予对内容和功能的不同访问权限。访问控制的检查在身份认证之后执行，用来决定被授权的用户能干什么。访问控制看起来是个简单的问题但是要正确实现却很难。一个 web 应用程序的访问控制模型是和这个网站提供的内容和功能是紧密相连的。而且，用户会被分为具有不同能力或权限的用户组或者角色（role）。

开发人员常常低估实现一个可靠的访问控制机制的难度。很多访问控制方案事先都没有经过仔细设计而只是随着网站的发展而改进。在这种情况下，访问控制的规则被加入到源代码中不同的位置。当这样的网站接近部署阶段时，这些没有考虑全局的规则组合在一起变得非常复杂，几乎无法理解。

很多这样有问题的访问控制方案不难被发现和利用。通常，攻击者只需要编写一个对无权访问的功能和内容的请求即可达到这一目的。一旦一个访问控制的漏洞被发现，具有破坏性其后果可能是非常具有破坏性的。除了可以查看未经授权的内容，一个攻击者还可以篡改和删除这些内容，执行未经授权的功能， 甚至夺取网站的管理权限。

一个特定的访问控制问题是让网站管理员通过 Internet 管理网站的管理员界面。这样的功能通常被用来方便网站管理员管理网站的用户，数据和内容。在很多情况下，网站通过支持不同的管理员角色来实现细化（finer granularity）的网站管理。应为功能强大，这些管理员界面往往成为内部和外部攻击的目标。

A2.2 影响的环境

所有已知的 web 服务器，应用程序服务器和 web 应用程序环境至少都会被某些这样的问题影响。即使一个网站是完全静态的，如果它没有被正确的设置，黑客还是可以访问到保密文件，破坏网站页面，或者进行其它的破坏活动。

A2.3 例子和参考

- “OWASP 指南：如何建立安全的 Web 应用程序和 Web 服务”， 第八章，访问控制
  http://www.owasp.org/guide/
- J2EE 应用程序中的访问控制（授权）
  http://www.owasp.org/columns/jeffwilliams/jeffwilliams3
- http://www.infosecuritymag.com/2002/jun/insecurity.shtml

A2.4 如何确定你是否容易被攻击

基本上所有的网站都有一定的访问控制需求。因此，访问控制的策略应当用文档清晰的描述。而且，设计文档应该描述实施这一因此，一个网站的访问控制的策略应当用文档清晰的描述。而且，设计文档应该描述实施这一策略的方法。如果缺乏这些设计文档，那么这个网站很有可能就存在安全问题。

实现访问控制策略的代码应该被检查。这些代码应该是结构合理的，模块化的，而且大多数情况下是集中化的。一个仔细的代码检查应该被执行以验证访问控制机制实现的正确性。而且，渗透性测试（penetration testing）对于发现访问控制中的问题非常有帮助。

了解你的网站是如何被管理的。你应当了解网络页面是如何被更改的，在哪里被测试的，而且是如何被传输到生产系统的。如果系统管理员可以远程进行修改，你应当了解这些通讯渠道是如何被保护的。仔细检查各个界面，确保只有被授权的管理员才可以访问。同时，如果不同的类型或分组的数据可以通过同一个界面被访问，确保只有被授权的数据可以被访问。如果这样的界面使用外部命令，检查这些命令的使用，确保它们不具有本文中所描述的命令注入漏洞。

A2.5 如何保护你自己

最重要的一步是全面考虑整个应用程序的访问控制需求，然后将其在 web 应用程序的安全策略文档中描述下来。我们极力推荐使用一个访问控制矩阵来定义访问控制的规则。如果缺乏用文档描述的安全策略，就没法定义这个网站如何算安全。这个访问策略应该定义何种类型的用户可以访问系统，每个类型的用户可以访问哪些功能和内容。访问控制机制应该被全面的测试以保证没法被绕过。这样测试需要不同的帐号并要求对未授权的内容和功能进行广泛的访问尝试。

一些特定的访问控制问题包括：

- 不安全的 Id – 大部分的 web 网站都使用一定形式的 id, 关键字，或者索引来代表用户，角色，内容，对象，或功能。如果一个黑客可以猜测这些 id, 并且没有一定的检测机制来确保提供的值是授权给当前用户的，攻击者可以自由使用安全访问机制来看他们到底可以访问什么。Web 应用程序不应该依靠任何 id 的保密性来保证其安全。
- 强迫访问绕过访问控制检查 － 很多网站都要求用户通过一定的检查才赋予网站深层 URL 的访问权利。当用户绕过带有安全检查的页面直接访问网站深层的 URL 时，这些检查不能被绕过。
- 路径遍历 － 这一攻击方法是将相对路径（例如，“../../target_dir/target_file”）作为请求的一部分提交。这样的攻击试图访问那些正常情况下任何用户都不能直接访问，或者直接访问会被拒绝的文件。这样的攻击不仅可以通过 URL 提交，而且可以通过别的最终能访问一个文件的输入提交（例如，系统调用和 shell 命令）。
- 文件属性 －很多 web 和应用程序服务器依靠底层平台文件系统提供的访问控制列表。即使所有的数据都存储到后台的服务器，在 web 和应用程序服务器上总是存在一些不应该被共享的本地文件，特别是配置文件，缺省文件，以及安装在大部分 web 和应用程序服务器上的脚本文件。只有那些特别指定是被 web 用户访问的文件才应使用操作系统的文件属性机制标为可读文件。大部分目录不应该是可读的，只有少数的文件被标志为可以执行
- 客户端缓存－很多用户通过在图书馆，学校，机场和其他公共数据访问点共享的计算机访问网站。浏览器缓存的 web 页面可以被黑客用来访问网站未授权的部分。开发人员应该使用多重机制，包括 HTTP header 和 meta tag，来保证包含有敏感信息得页面不会被缓存在用户的浏览器中。

有一些应用程序层的安全组件可以用来帮助正确实现访问控制方案的某些方面。要实现有效的参数验证，安全组件的配置必须严格的定义怎么样的访问请求对于你的网站是合法的。当使用一个组件的时候，你必须了解清楚，对于你的网站的安全策略，这个组件到底能够提供怎样的访问控制帮助，而你的访问控制策略的哪一部分这个组件不能处理，从而确保这一部分在你自己编写的代码中被正确的处理。

对于管理员功能，主要建议是尽量禁止管理员通过网站的前门访问。考虑到这些界面的强大功能，大部分的组织都不应该将这些界面暴露给外部攻击，这样风险很大。如果远程管理员访问是绝对需要的，可以不用敞开网站的前门而做到这一点。使用 VPN 技术可以给处于外部的管理员提供一个访问公司（或网站）内部网络的安全后台连接。

A3 错误的认证和会话管理

A3.1 描述

认证和会话管理包括了处理用户认证和管理活跃的会话。认证是该过程关键的一部分，但是就算是强大的认证机制也会被有问题的认证信息管理功能所破坏，比如改变密码，忘记我的密码，记住我的密码，帐号更新，和其他相关功能。由于“过路人(walk by)”攻击对于很多 web 应用可能发生，所有的帐号管理功能应该要求重新验证，即使当该用户有一个有效的会话 ID。

Web 的用户认证通常使用用户 id（userid）和密码。商业级的更强的认证使用基于密码学的 token 和生物学技术的软件和硬件，但是，这些方法对于大多数 web 应用成本太高。很多的帐号和会话管理的漏洞可以危及用户，甚至系统管理员帐号的安全。设计一个认证和会话管理系统来保护好网站上各种各样的认证信息，是很复杂的。而开发团队往往低估了其复杂度。

Web 应用必须使用会话来保存每个用户的请求流。HTTP 本身不提供这样的功能，所以，web 应用必须自己来创建。通常 web 应用环境提供了会话能力，但是很多开发人员更喜欢创建他们自己的会话 token。不管哪一种情况，如果会话 token 没有被好好保护，攻击着就可以劫持一个活跃会话，来冒充用户的身份。建立强的会话 token，并在它们的生命周期里保护好他们，被证明对于很多开发者而言是难以掌握的。

除非所有的验证信息和会话标识符被 SSL 保护，并且不暴露给其他漏洞（比如跨站脚本），攻击者可以劫持用户的会话，冒充他们的身份。

A3.2 受影响的环境

所有已知的 web 服务器，应用服务器，和 web 应用环境都可能存在错误的认证和会话管理的问题。

A3.3 例子和参考

- OWASP 指南：建立安全 Web 应用和 Web 服务，第 6 章：认证，第 7 章：会话管理 http://www.owasp.org/guide/
- 基于 Web 的应用的会话固定漏洞白皮书 http://www.acros.si/papers/session_fixation.pdf
- 基于 Web 的应用的密码恢复白皮书 http://fishbowl.pastiche.org/archives/docs/PasswordRecovery.pdf

A3.4 如何确定你容易受攻击

代码检查和渗透性测试都可以用来检查认证和会话管理的问题。仔细检查你的认证机制，确保用户认证信息在存储时（磁盘上）和传输时（登陆时）都是被保护的。检查你的会话管理机制，以确保所有的会话标识符总是被保护的，而且偶然或者恶意暴露的可能性最小化。

A3.5 如何保护你自己

仔细合理的使用自定义的或者通用的认证和会话管理机制，会极度的降低出现问题的可能性。一个很好的开始包含定义，记录下你网站的有关安全管理用户认证信息的策略。确保你的实现和策略的一致，这是安全的强壮的认证和会话管理机制的关键。一些关键的领域如下：

- 密码强度－密码应该有限制，来保障最小长度和复杂度。复杂度通常要求在用户的密码里使用最少数目的字母，数字，和/或符号的结合，例如，至少每种一个。用户应该被要求周期性的更改密码。用户应该不被允许重复使用以前的密码。
- 密码使用－用户有失败登陆次数的限制，所有的重复的失败登陆应该被记到日志中。失败的登陆的密码不应该被记录下来，因为这样会将用户的密码暴露给访问该日志的人。如果一个登陆失败了，系统不应该指出到底是用户名还是密码错误。用户应该被告知他们上次登陆的日期和时间，和从该时间开始，失败的登陆发生的次数。

- 密码修改控制－不论在什么地方允许用户休息密码，都应该使用一个简单的密码修改机制。当修改密码时，用户应该总是被要求提供老的和新的密码（像所有的帐号信息）。如果忘记了的密码被用 Email 寄回给用户，当用户修改他们的 email 地址时，系统应该要求用户来重新认证，否则，暂时拥有会话访问权（比如过路人的方式）的攻击者可以修改他们的 email 地址，然后要求将“忘记了”的密码寄给他们。
- 密码存储－不论存在什么地方，所有的密码必须以杂凑函数（Hash）或者加密的方式存储以防止暴露。由于不可逆，Hash 更好些。加密应该在明文密码使用时使用，例如当使用密码登陆到另一个系统。密码不能被直接写在任何源程序中。解密密钥必须被严格的保护，以确保他们不能被获得来解密密码文件。
- 保护传输中的认证信息－唯一有效的技巧是加密整个登陆过程，比如用 SSL。简单的密码的转换，比如传输前的在客户端的 Hash 不能提供有效的保护，因为被 Hash 的版本可能被拦截和重传输，即使实际的明文密码也许不被发现。
- 会话 ID 保护－理想情况下，用户的所有会话应该使用 SSL 保护。如果是的话，会话 ID（会话 Cookie）就不会被被从网络上取到，杜绝了最大的会话 ID 暴露的风险。如果 SSL 由于性能或其他因素而不能使用，就应该使用其他办法来保护会话 ID。首先，他们不应该被包含到一个 URL 里，因为他们可以被浏览器放入缓存，在 Referrer header 里发送，或者偶然被转发给一个“朋友”。会话 ID 应该是很长的复杂的很难猜的随机数。当切换到 SSL 认证，或者其他主要的过程时，会话 ID 必须改变。绝对不要接受用户自己选择的会话 ID。
- 帐号列表－系统不应该允许用户列出网站所有帐号。如果必须需要一个用户列表，我们推荐使用某种形式的假名（屏幕名）来指向实际的帐号。这样的话，假名不能被使用来登陆或者攻击用户帐号。
- 浏览器缓存－认证和会话数据不应该作为 GET 的一部分来发送，应该使用 POST。认证总是应该使用“不缓存”的标记，防止被人使用浏览器的“后退(back)”按钮，到达登陆页面，重新发送以前输入的认证信息。很多浏览器现在支持“autocomplete=false”的标记来防止存储“自动完成”的认证信息。
- 信任关系－在可能的情况下，你的网站的体系结构应该防止组件之间隐形的信任。每个组件应该对和它有界面的组件认证自己，除非有充足的理由不这样做（比如性能，或者缺乏适用的机制）。如果需要信任的关系，应该有严格的过程和体系结构机制，来确保随着体系结构的发展，这样的信任不会被滥用。

A4 跨站脚本（XSS）

A4.1 描述

攻击者使用 web 应用来发出恶意代码（通常以脚本的形式）给另一个末端用户，这叫做跨站教本（Cross-site scripting - XSS）。这些漏洞非常普遍，在任意一个 web 应用都可能发生－在输出中不经验证而使用一个用户的输入时。

攻击者能够使用跨站教本来发送恶意脚本给没有发觉的用户。末端用户的浏览器没有办法知道应该不信任该脚本，而导致脚本被执行。由于浏览器认为脚本来自一个可信的来源，恶意脚本可以访问任意的 cookie，会话 token，或者其他敏感信息。这些脚本甚至可以重写 HTML 页面内容。

XSS 攻击通常分为两类：存储型(stored)和反射型(reflected)。存储型攻击是指注入代码（injected code）永久性存于目标服务器上，比如数据库，论坛，访问日志等。受害者从服务器上取到恶意脚本。反射性攻击指的是注入代码是从 web 服务器上反射出的，比如错误信息，搜索结果，或者是其他反应。反射性攻击通过另一条途径到达受害者，比如电子邮件，或者其他的 web 服务器。当用户点击一个恶意连接或者发送一个特别定制的表的时候，注入的代码就会到达容易受到攻击的服务器，而后者将攻击“发射”回用户的浏览器。浏览器便执行代码，由于它来自“可信”的服务器。

不论攻击是否是存储的，还是反射的，它们的后果都是相同的。他们的不同之处在于代码如何到达服务器。不要认为个“只读”或者“手册型（brochureware）”的网站不怕反射型 XSS 攻击。XSS 可以给末端用户造成一系列严重的问题，从讨厌，到危及帐号安全。最严重的 XSS 攻击暴露用户的会话 cookie，导致攻击者劫持用户会话，从而获得帐号。其他破坏性的攻击包含暴露用户文件，安装木马，将用户重定向到其他页面或者网站，修改内容。一个允许攻击者修改网站的新闻稿的 XSS 漏洞可以影响一个公司的股票价格，或者降低消费者信息。一个制药厂网站上的 XSS 可以导致攻击者修改剂量信息，从而导致病人用药过量。

攻击者常常使用各种方法来对恶意部分进行编码，比如 Unicode，因而请求看起来不太可疑。现存有几百种攻击的变种，包括甚至不使用<>符号的。鉴于此，将代码“过滤”不太可能成功。取而代之的，我们推荐使用一个严格的规范来对输入进行验证。XSS 通常以嵌入式的 JavaScript 的形式出现。然而，任何嵌入式的内容都是有可能存在危险的，包括 ActiveX（OLE），VBScript，Shockwave，Flash 等等。

XSS 的问题也在下层的 web 和应用服务器上存在。大多数的 web 和应用服务器会产生简单的错误页面，比如 404‘网页未找到’，和 500‘内部服务器错误’。如果这些页面反射出用户请求信息，比如他们试图访问的 URL，这有可能受到 XSS 攻击。

一个网站包含跨站脚本漏洞的可能性非常大。存在一系列的办法使 web 应用传播恶意代码。开发人员试图过滤掉请求种的恶意部分时，很可能忽略掉可能的攻击或者编码。攻击者找到这些漏洞不是极度困难，因为他们所需的只是浏览器和一些时间。存在很多的免费工具来帮助黑客找到这些漏洞，定制和注入 XSS 攻击到一个目标网站。

A4.2 受影响的环境

所有的 web 服务器，应用服务器，和 web 应用环境都受跨站脚本影响。

A4.3 例子和参考

- 跨站脚本 FAQ： The Cross Site Scripting FAQ: http://www.cgisecurity.com/articles/xss-faq.shtml
- CERT 恶意 HTML 标记报告 CERT Advisory on Malicious HTML Tags: http://www.cert.org/advisories/CA-2000-02.html
- CERT “Understanding Malicious Content Mitigation” http://www.cert.org/tech_tips/malicious_code_mitigation.html
- 跨站脚本安全曝光摘要 Cross-Site Scripting Security Exposure Executive Summary: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/topics/ExSumCS.asp
- 理解 CSS 的原因和影响 Understanding the cause and effect of CSS Vulnerabilities: http://www.technicalinfo.net/papers/CSS.html

- OWASP 建立安全 Web 应用和 Web 服务指南，第八章：数据验证 OWASP Guide to Building Secure Web Applications and Web Services, Chapter 8: Data Validation http://www.owasp.org/documentation/guide/
- 如何建立 HTTP 请求验证引擎（使用 Stinger 的 J2EE 验证）How to Build an HTTP Request Validation Engine (J2EE validation with Stinger) http://www.owasp.org/columns/jeffwilliams/jeffwilliams2
- Have Your Cake and Eat it Too (.NET validation) http://www.owasp.org/columns/jpoteet/jpoteet2

A4.4 如何判断你是否容易被攻击

XSS 漏洞很难确认，和从 web 应用中除去。最好的办法时做安全检查，寻找所有的 HTTP 请求可能到达 HTML 输出的地方。注意，很多 HTML 标记都可以用来传输恶意的 JavaScript。Nessus，Nikto 和其他工具能够帮助扫描网站来发现这些漏洞，但是程度不够深入。如果网站的某部分存在漏洞，那其他问题很可能也存在。

A4.5 如何保护你自己

最好的保护 web 应用免于 XSS 攻击的办法，是确保针对一个严格的规范来验证所有的头，cookie，查询串，表项，和隐藏项（即所有的参数）。验证的过程不应该试图找到相关的内容，删掉，过滤，或者除去有害部分。现存有太多类型的有效的内容，有太多编码的方式来绕过过滤器。我们强烈推荐一个“正面”的安全策略，该策略定义了什么是允许的。“负向”的，或者基于攻击签名的策略非常难于维护，而且可能不完整。

通过防止被插入的脚本以可执行的形式传输到用户，对用户提供的输出进行编码也对跨站脚本有效。通过将以下生成的字符转化成合适的 HTML 编码，应用程序可以获得针对 JavaScript 攻击的极大的保护。

从	到
<	&lt;
>	&gt;
(	&#40;
)	&#41;
#	&#35;
&	&#38;

OWASP 过滤器计划正在产生一些多种语言下可复用的组件，以帮助防范很多种形式的参数篡改，包括 XSS 注入。OWASP 也发布了一个应用层的防火墙－CodeSeeker。不仅如此，OWASP 的 WebGoat 培训程序在跨站脚本和数据编码有专门的课程。

A5 缓冲溢出

A5.1 描述

攻击者使用缓冲溢出来破坏 web 应用程序的栈。通过发送特别编写的代码到 web 应用程序中，攻击者可以让 web 应用程序来执行任意的代码，基本上可以控制机器。缓冲溢出不容易被发现，就算发现了，总的来说利用起来非常困难。尽管如此，攻击者还是成功地在很多的产品和组件中找到了缓冲溢出的问题。另一类相似的问题是格式化串攻击。

缓冲溢出可能在 web 服务器，应用服务器，或者应用程序本身中出现。在大规模使用的服务器产品中发现的缓冲溢出对用户会产生非常大的风险。当 web 应用程序使用库，比如图形库来产生图像时，他们将自己暴露在可能的缓冲溢出攻击中。

缓冲溢出也出现在用户自己的 web 应用程序代码中，在不谨慎的情况下尤其如此。在用户自己的 web 应用程序中的缓冲溢出一般来说比较不容易被发觉，因为非常少的黑客会找寻和攻击这样的漏洞。如果在用户应用程序里发现了这样的问题，由于黑客通常看不到用户源码和具体的出错信息，使用漏洞（而不是让程序崩溃）的可能性会大大降低。

A5.2 影响的环境

几乎所有已知的 web 服务器，应用服务器，和 web 应用程序环境都可能出现缓冲溢出。唯一的例外是 Java 和 J2EE，这样的攻击对它们无效（除了在 Java 虚拟机本身的溢出）。

A5.3 例子和参考

- OWASP 指导：建立安全的 web 应用程序和 web 服务器，第八章：数据验证 http://www.owasp.org/documentation/guide/
- Aleph One, 自娱有利搞垮栈 http://www.phrack.com/show.php?p=49&a=14
- 马克 • 堂纳森，缓冲溢出攻击详解：机制，方法，及预防 http://rr.sans.org/code/inside_buffer.php

A5.4 如何确定你容易受到攻击

对于服务器产品和库，留意最新的 bug 报告。对于用户自己的应用程序软件，所有的通过 HTTP 接受的用户输入都必须被验证以确保它可以处理任意大的输入。

A5.5 如何保护你自己

留意最新的有关你的 web 服务器和应用服务器，以及在你的网络中其他产品的 bug 报告。确保打了最近的补丁。定期地用网络扫描器来扫描你的网站，寻找你的服务器和你自己 web 应用程序中的缓冲溢出。

对于你自己的应用代码，你需要审查所有的通过 HTTP 接受请求的部分代码，确保他们能检查所有输入的长度。就算是对于不产生这样情况的环境也要做检查，因为过于庞大的输入会产生拒绝服务，或者其他运行的问题。

A6 注入式漏洞

A6.1 描述

注入式漏洞允许攻击者通过一个 web 应用中转恶意代码到另一个系统。这些攻击包括对操作系统的调用，使用 shell 命令来调用外部程序，以及通过 SQL 注入来调用后端数据库。使用 perl，python，和其他语言写的教本可以被注入到没有很好设计的 web 应用程序中，被执行。每当一个 web 应用程序使用任何类型的解释器的时候，就会存在被注入式攻击的危险。

很多 web 应用程序使用操作系统和外部程序来完成它们的功能。Sendmail 可能是被调用最多的外部程序，但是其他的很多程序也被使用着。当一个 web 应用程序通过完不请求来传递 HTTP 请求信，它必须被仔细检查。否则，攻击者可能注入特别的（亚）字符，恶意代码，或者命令修饰符到信息中，web 应用程序会把这些信息原封不动地传递到外部程序。

SQL 注入是一种很流行并且危险的注入的形式。为了使用一个 SQL 注入漏洞，攻击者必须找到一个 web 应用程序会传递到数据库的参数。通过将恶意的 SQL 命令仔细嵌入到参数中，攻击者可以让 web 应用程序将恶意的数据库查询送给数据库。这样的攻击实现并不困难，而且有很多的工具可以寻找到这样的漏洞。如果攻击者得到，或者毁坏数据库内容，后果非常具有破坏性。

注入式攻击很容易被发现和使用，但是它们也可能极度的不明显。后果的严重性变化很大，从没有破坏性，到整个系统的被入侵，甚至毁灭。在任何情况下，使用外部调用是很常见的，因而，一个 web 应用程序有注入式漏洞的可能性比较高。

A6.2 受影响的环境

每一个 web 应用程序的环境都允许执行外部命令，比如系统调用，shell 命令，以及 SQL 请求。一个外部命令会受到注入攻击的可能性取决于调用是如何完成的，以及被调用的组件是什么，但是，如果 web 应用程序没有写好的话，几乎所有的外部调用都有可能被攻击。

A6.3 例子和参考

- 例子：一个恶意的参数可以修改一个检索当前用户文件的系统调用的行为，到访问其他用户的文件（比如，通过加入"../"到文件名的一部分中）。可以在当前的参数后加入多余的命令给 shell 脚本（比如，"; rm –r *"）。SQL 查询可以被一个“约束”来修饰（比如，"OR 1=1"）来访问或者修改未授权的数据。
- OWASP 建立安全 Web 应用程序和 Web 服务指南，第八章，数据验证。
  http://www.owasp.org/documentation/guide/
- 如何建立一个 HTTP 请求验证引擎（J2EE 下用 Stinger 的验证）
  http://www.owasp.org/columns/jeffwilliams/jeffwilliams2
- 吃你的蛋糕，要吃。(.NET 验证) http://www.owasp.org/columns/jpoteet/jpoteet2
- SQL 注入白皮书 http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf

A6.4 如何确定你容易受到攻击

最好的方法是在你的源码中找所有对发布资源的调用（既：system, exec, fork, Runtime.exec,SQL 查询，或者任何在你的环境中对外部请求的语法）。注意，很多语言都有很多种不同的方法来调用外部命令。开发人员应该检查他们的代码中有关 HTTP 请求的部分。你应该自己检查这些调用，实行以下的保护措施。

A6.5 如何保护你自己

最简单的办法就是能够不访问外部程序就不要访问。对很多 shell 命令和有些系统调用提供的功能而言，每个编程语言都有很多内部的库来实现相同的功能。使用这个内部库不会涉及到操作系统 shell 解释器，因而避免了 shell 命令的很多问题。

对于必须使用的外部调用，比如对后端数据库的调用，你必须仔细的验证数据输入，保证它不包含任何恶意的内容。你也可以将所有的参数都当作数据，而不是可能被执行的内容。使用存储过程或者预准备的语句会提供很大的保护，保证输入被当作数据处理。这些方法会减少，但不会完全消除这些外部调用的风险。你仍然需要验证输入，以确定它满足应用程序的要求。

另一个有效的保护是保证 web 应用程序只在它需要的特权下运行。所以，你不应该以 root 运行 web 服务器，或者以管理员帐号访问数据库，否则，攻击者可以使用这些管理员特权。有些 J2EE 环境中允许使用 Java 沙盘，用以防止执行系统命令。

如果必须使用外部命令，任何用户信息输入都必须被严格的检查。应该存在处理在调用时可能出现错误，超时的机制。

所有的输出，返回代码，和错误代码必须被检查以保证期望的数据处理确实发生。至少，这会让你发现错误。否则，攻击可能发生，却不被察觉到。

OWASP 过滤器项目正在产生几种语言下的可复用的组件，用于帮助防范多种形式的注入。OWASP 也发布了 CodeSeeker，一个应用层的防火墙。

A7 不恰当的异常处理

A7.1 描述

不恰当的异常处理可能导致一个网站各种各样的安全问题。最常见的问题出现在详细的内部错误信息比如堆栈追溯，数据库清空以及错误的代码呈现给黑客的时候。这些信息显示了不应该显示的执行细节所以能够提供给黑客重要的网站潜在漏洞。

当网站的应用程序在正常操作的时候会频繁地产生错误情况：比如内存不够，空指针，系统调用失败，数据库不存在，网络不通等等很多常见的错误。这些错误异常应该很好地被处理。给用户提供详细有意义的错误信息，给网站维护者提供诊断信息但是不能够给黑客提供任何信息。

即使这些错误信息不能够提供详情，信息之间的不一致仍然能够提供线索这个网站如何工作，哪些信息是存在的。比如，当用户想要存取一个并不存在的文件，错误信息会显示：“文件未找到”。当用户想要存取一个没有授予权限的文件，信息会显示：“权限被拒绝”。用户并不应该知道文件是否存在。但是这样的不一致性就会提示不可存取的文件存在与否以及站点的目录结构。

一个常见的安全问题由不恰当的异常处理引起的是打开失败的安全检查。所有的安全机制应该拒绝存取直到被特别地给予了权限，只有在拒绝的时候才给于权限。这就是为什么打开失败的错误会出现。其他的错误会导致系统崩溃或者消耗大量的资源，拒绝或者减少给合法用户提供服务。

好的错误异常处理机制应该能够处理任何几组可行的输入同时执行恰当的安全策略。简单的错误信息应该被生成并且记录，这样他们的成因，无论是网站本身的错误或者是黑客攻击引起的，可以被重审。错误异常处理不应该仅仅致力于用户提供的输入还要包括内部模块像系统调用，数据库查询，内部函数等等引起的错误。

A7.2 受影响的环境

所有的 web 服务器，web 应用服务器和 web 应用环境都面临异常处理问题。

A7.3 例子和参考

- OWASP 关于错误代码的讨论是在 http://www.owasp.org/documentation/guide

A7.4 如何决定你是否容易被攻击的

一般来说，简单的测试可以确定你的网站怎样来响应各种不同的输入错误。更加彻底的测试通常需要程序出现内部错误来看你的网站如何响应。

另外一种有价值的方法是检查一遍代码看看它的异常处理逻辑。异常处理在整个站点应该是一致的。每一部分都是整个好设计的一部分。代码检查可以显示系统是怎样处理各种不同的错误的。如果你发现整个异常处理机制没有组织或者出现几种不一致的异常处理机制。很有可能就是一个问题所在。

A7.5 如何保护自己

一种特别的如何处理错误的策略应该写入文档，包括错误处理的类型，什么样的信息回报给用户，什么样的信息要被记入日志。所有的开发者应该理解这种策略并且在代码中执行它。

在执行代码中保证网站能够处理各种类型的错误。当错误出现的时候，网站可以响应用户提供有用的信息同时又不揭示不必要的内部祥情。特殊类型的错误要记录下来帮助查找网站代码中的漏洞以及黑客可能攻击的地方。

很少有网站在应用程序中具有入侵测试功能，但是确实是可以设计每一个应用程序能够跟踪重复失效的黑客的企图并生成警告。大量的网络应用程序攻击是不能被察觉因为很少有站点可以检测到它们。因此很可能大量的网络应用的安全攻击被大大低估了。

OWASP 过滤项目正在用几种语言生产重复可用的模块阻止错误的代码流入用户页面，在这些页面被 web 应用程序动态生成的时候过滤他们。

A8 不安全的储存

A8.1 描述

大多数 web 应用程序都需要存储敏感信息在数据库或者文件系统中。这些信息可能是密码，信用卡号码，帐号记录，或者是专利信息。通常加密技术用于保护敏感信息。尽管加密变得相对容易来实现和使用，开发人员还是经常将其集成到 web 应用程序时犯错误。开发人员通常会过度依靠使用加密获得的保护，而忽视其他的方面。一些常见的错误包括：

- 没有加密关键数据
- 密钥，证书和密码的不安全存储
- 秘密不恰当的存在内存中
- 随机性来源不好
- 算法选择不好
- 试图发明新的加密算法
- 没有包括对加密密钥交换的支持，及其他必要的维护过程

以上错误对于网站的安全有毁灭性的影响。加密通常被用于保护网站的最敏感的信息，而信息很可能被以上的错误所危及。

A8.2 受影响的环境

绝大多数 web 应用程序的环境包含了一些密码学支持。在这样的支持不存在时，有很多第三方的产品可以选择。只有使用加密来保护存储的或者传输的信息的网站会受到这样攻击。注意，本部分不包含 SSL 的使用，我们在第 10 章中会谈及不安全的配置管理。这一部分只涉及应用层数据的加密。

A8.3 例子和参考

- OWASP 建立安全 web 应用和 web 服务指南 http://www.owasp.org/documentation/guide/
- Bruce Schneier,应用密码学，第二版，John Wiley & Sons 1995

A8.4 如何确定你容易受到攻击

在没有源代码的情况下发现密码学的漏洞是极度花费时间的。然而，有可能检查 token，会话 ID，cookie 和其他信息，看他们是否随机。所有的传统的密码分析方法都可以用于发现一个网站如何使用密码函数。

迄今为止最容易的方法就是检查代码，看密码学函数是如何实现的。应该有一个对于密码模块的结构，质量和实现仔细的检查。检查者应该在应用密码学和常见漏洞上有很强的背景。这个检查应该也包含对密钥，密码，和其他秘密的存储，保护，调入，处理，和从内存中清除的过程。

A8.5 如何保护你自己

最容易的办法就是将加密的使用最小化，只存储绝对必要的信息。比如，不应该存储信用卡号码，而是让用户重新输入。而且，不应存储加密后的密码，而是使用单向函数，如 SHA-1 来哈希密码。

如果必须使用密码学的话，选择一个经过公众检验过的库，确信它没有漏洞。将密码函数封装，仔细检查代码。确信秘密，比如密钥，证书和密码，被安全的存储。为了让攻击变得困难，主秘密应该被分开在两个地点，在运行时组装起来。这样的地点包括配置文件，一个外部的服务器，或者在代码里面。

A9 拒绝服务

A9.1 描述

Web 应用非常受拒绝服务式(Denial of Service)攻击的印象。需要注意的是，网络拒绝服务攻击，比如 SYN Floods，是本文档之外的一个问题。

一个 web 应用很难分辨出攻击和普通的访问。有很多因素决定了这一点，但是，最重要的是，IP 地址不能作为判断的依据。因为没有可靠的办法判断出一个 HTTP 请求从哪里来，所以滤出恶意的访问非常困难。对于分布式的攻击，一个应用程序如何分辨得出真正的攻击，多个用户同时刷新网页（当网站出现暂时的问题时会发生），或者是被 slashdotted（出现在 slashdot 首页上，链接被数以万计的 slashdot 用户访问）了呢？

绝大多数的 web 服务器在正常情况下能够同时处理几百个用户。一个攻击者可以从一个主机产生足够多的流量来耗尽很多应用程序。负载均衡(load balancing)可以使这样的攻击变得困难，但是不可能完全消除攻击，尤其当会话是和服务器绑定的。这就是为什么应用程序的会话数据要越小越好，这样使得开始新的会话变得困难。

当攻击者能够消耗所有的资源时，他们就可以阻止合法的用户使用系统。有限的资源包括带宽，数据库链接，磁盘存贮，CPU，内存，线程，或者应用程序特定的资源。所有的这些资源可以被攻击所消耗。例如，一个网站允许未被授权的用户来请求消息版的访问，它可能对于每一个 HTTP 请求发出很多数据库的查询。一个攻击者可以很容易地发送很多这样的请求，数据库链接池很快就会被耗光，合法的用户将不能使用服务。

这些攻击的变种瞄准一个特定的用户。例如，一个攻击者可能通过不停地发送无效的有关一个用户的信息（如密码），导致系统拒绝该合法用户的访问。或者，攻击者可以为用户请求新的密码，强迫他们访问 email 来重新获得访问。另一种方法就是，如果系统给一个用户锁定了一些资源，攻击者有可能可以不停的使用他们以致其他用户不能使用资源。

有些 web 应用程序甚至会被立刻攻击下线。没有处理好错误的应用程序甚至会将 web 应用程序 container 弄垮。这些攻击尤其具有毁灭性，因为他们立刻防止其他用户使用应用程序。

有很多中这样的攻击，他们中的大多数可以很容易地通过一台简陋电脑上地几行 PERL 代码来发起。尽管没有完全防范攻击的办法，使攻击变得困难还是可能的。

A9.2 受影响的环境

所有的 web 服务器，应用服务器，和 web 应用程序都会受到影响。

A9.3 例子和参考

- OWASP 建立安全 web 应用程序和 web 服务 http://www.owasp.org/documentation/guide/

A9.4 如何确定你容易受到攻击

如何确定你容易受攻击非常困难。负载测试工具比如 JMeter 能够产生 web 流量，你可以测试你的网站在重负载下的性能。当然一个重要的测试就是看你的应用程序每秒能够处理多少请求。从单个 IP 测试很有用，它可以告诉你一个攻击者需要发出多少请求以到达破坏你的网站的目的。

为了确定是否有资源被用来导致拒绝服务，你应该分析每一个资源，看是否有耗尽它的办法。你尤其应该关注未授权用户能够干什么，但是除非你信任所有的用户，你也应该检查授权用户能够做什么。

A9.5 如何保护你自己

防范拒绝服务是困难的，因为没有完美的保护的办法。

一个通用的规则是，你应该将最少的资源分配给用户。对于授权用户，可以使用配额来限制用户在系统上可以使用的资源。具体的说，你可以考虑通过将用户的对话同步的办法来保证在一个时间内只处理一个请求。如果你正在处理一个用户的请求，你也可以考虑忽略掉来自该用户的请求。

对于未授权的用户，你应该避免任何任何不必要的对数据库或者其他费时的资源的访问。试着去设计你网站的数据流，使未授权的用户不能激发费时的操作。你也可以将未授权的用户的访问内容放在缓存中，而不是直接访问数据库。

你也应该检查你的错误处理机制，以确保一个错误不能影响整个应用程序的操作。

A10 不安全的配置管理

A10.1 描述

Web 服务器和应用成熟配置在 web 应用安全中有关键的作用。这些服务器负责提供内容，调用产生内容的应用程序。另外，很多应用服务器也提供很多服务给应用程序，包括数据存储，目录服务，邮件，消息，等等。对这些配置的管理如果有错误，会导致一系列的安全问题。

通常，web 的开发团队和运作团队是分开的。实际上，写应用程序的人和运作它的人存在很大的沟通鸿沟。Web 应用安全的考虑常常需要两方面来正确的保证一个网站的安全。

存在很多影响网站安全的服务器配置问题。它们包括：

- 服务器软件未做安全补丁
- 服务器软件漏洞和错误配置允许列出目录，和目录遍历攻击
- 不必要的缺省，备份，或者例子文件，包括教本，应用程序，配置文件，和网页
- 不正确的文件和目录权限
- 不必要的服务被运行，包括内容管理和远程系统管理
- 有缺省密码的缺省的帐号
- 被激活的、能够被反问的管理和调试功能
- 提供太多信息的错误信息（在错误处理中会有更多细节）
- 不正确的 SSL 证书和加密设定
- 使用自己签名的证书达到验证和中间人保护
- 使用缺省证书
- 不正确的通过外部系统的验证

有些问题可以被扫描工具很容易的检测到。一旦被检测到，这些问题很容易被利用，导致网站被攻陷。成功的攻击也可以导致后端系统的攻陷，包括数据库和企业网络。拥有安全软件和安全的配置是安全网站必要的条件。

A10.2 被影响的环境

所有的 web 服务器，应用服务器，和 web 应用程序环境都会受到错误配置的影响。

A10.3 例子和参考

- OWASP 建立安全 Web 应用程序和 Web 服务 http://www.owasp.org/documentation/guide/
- Web 服务器安全实践 http://www.pcmag.com/article2/0,4149,11525,00.asp
- 让公众 Web 服务器变得安全（来自 CERT）http://www.cert.org/security-improvement/modules/m11.html

A10.4 如何确定你容易被攻击

如果你没有花时间加固你的 web 和应用服务器的话，你很可能容易被攻击。很少，如果有的话，服务器产品出来就是安全的。一个安全的配置应该被记录下来，不断更新。你应该经常检查检查这个配置，保证它没有过时，并且是一致的。我们也推荐做和已经使用了的系统做比较。

除此之外，有很多扫描工具（如 Nessus 和 Nikto）来从外部扫描 web 服务器或者应用程序，找寻已知漏洞。你应该至少每个月运行这些工具一次。这些攻击应该在网站的内部和外部都运行。外部的扫描应该从从网站所在网络之外主机上运行。内部扫描应该从服务器所在的网络进行。

A10.5 如何保护你自己

第一步是创建一个指导性的文档用于加强你的 web 服务器和应用程序服务器配置。这个配置应该用在所有的运行该应用程序的主机和开发环境下的主机上。我们推荐最开始的时候使用已存在的文档，例如从你的产品生产商或者其他的来自 OWASP，CERT 和 SANS 等安全组织得到，然后根据你的独特要求而修改该文档。该文档应该包括以下范围：

- 配置所有的安全机制
- 关掉所有不使用的服务
- 设置角色，权限，帐号，包括关掉所有的缺省帐号，或者改他们的密码
- 使用日志和警报

一旦你的指导文档建立起来，用它来配置和维护你的服务器。如果你有大量的服务器需要配置，可以考虑使用半自动，或者完全自动的办法来配置。使用现存的配置工具或者开发你自己的配置工具。已经存在很多这样的工具。你也可以使用磁盘复制工具，比如 ghost 来建立一个服务器的映像，然后将它复制到新的服务器上去。这样的过程可能对于你独特的环境有效，或者无效。

保证服务器配置的安全需要很谨慎。你应该将服务器配置及时更新的任务分配到具体的人或者团队。这个维护的过程包括：

- 监视最新发布的安全漏洞
- 给系统使用最新的安全补丁
- 更新安全配置指导文档
- 周期性从内部和外部网络做漏洞扫描
- 周期性给上层管理层报告整体的安全情况

结论

OWASP 完成本列表，用于提高对 Web 应用安全的认识。OWASP 的专家们得出结论：这些漏洞体现了公司和机构将他们的商业逻辑暴露在 Internet 里所面对的严重的风险。Web 应用安全问题和网络安全一样严重，尽管他们传统上没有受到受到太大注意。攻击者开始将焦点转移到 web 应用安全问题，并且积极地开发工具和技巧，用于检测和使用漏洞。

本十大列表仅仅是个开始。我们相信这些漏洞体现了 web 应用安全面对的最严重的风险，但是还有很多其他的在部署 web 应用程序时需要考虑的领域：

- 无必要的和恶意代码
- 线程安全和同步编程中的问题
- 未被授权的信息收集
- 责任制度的问题和弱日志系统
- 数据错误
- 缓存，池和复用的问题

我们欢迎您对本列表的意见。请参加 OWASP 邮件列表，一起来提高 web 应用安全。请访问 http://www.owasp.org 及 OWASP 中文邮件列表 owasp-chinese@lists.sourceforge.net 获得更多信息。