



OWASP API 安全 Top 10 2019

十大关键API安全风险

目录

TOC 目录.....	1
FW 前言	2
I 简介	3
RN 发布说明	4
Risk API安全风险	5
T10 OWASP API 安全Top 10 – 2019.....	6
API1:2019 失效的对象级别授权	7
API2:2019 失效的用户身份验证.....	8
API3:2019 过度的数据暴露.....	9
API4:2019 资源缺乏和速率限制.....	10
API5:2019 失效的功能级授权.....	11
API6:2019 批量分配.....	12
API7:2019 安全配置错误.....	13
API8:2019 注入.....	14
API9:2019 资产管理不当.....	15
API10:2019 日志和监控不足.....	16
+D 开发人员下一步做什么?.....	17
+DSO DevSecOps下一步做什么?	18
+DAT 方法和数据.....	19
+ACK 致谢.....	20

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！

在当今应用驱动的世界中，创新的一个基本要素是应用程序编程接口（API）。从银行、零售、运输到物联网、自动驾驶汽车、智能城市，API都是现代移动、SaaS 和 Web应用程序的一个关键组成部分，并且在面向客户、面向合作伙伴和面向内部的应用中都会使用到。

一般来说，API会公开应用程序的逻辑和敏感数据，如：个人识别信息（PII），正因为如此，API越来越成为攻击者的目标。如果没有安全的 API，快速创新将不可能。

虽然更为宽泛的 OWASP Top 10仍然具有重要意义，但由于API的特殊性质，我们需要一份特定的API安全风险清单。API安全聚焦于策略和解决方案，以便理解并且缓解与API相关的独特脆弱点和安全风险。

如果您熟悉 [OWASP Top 10](#)，那么您会发现这两个文件之间的相似之处：它们的目的是为了更容易阅读和被采纳。如果您还不熟悉OWASP Top 10系列，您可能需要先阅读本文 [API 安全风险](#)部分和[方法和数据](#)部分，然后再阅读十大风险列表。

您可以在我们Github上的项目仓库为OWASP API Top 10提出您的问题、意见和想法。

- <https://github.com/OWASP/API-Security/issues>
- <https://github.com/OWASP/API-Security/blob/master/CONTRIBUTING.md>

您可以在以下链接找到《OWASP API 安全 Top 10》：

- https://www.owasp.org/index.php/OWASPAPISecurity_Project
- <https://github.com/OWASP/API-Security>

我们感谢所有这个项目的贡献者，并将他们全部列[致谢](#)部分。谢谢你们！

简介

欢迎参阅2019年版《OWASP API 安全 Top 10》！

欢迎阅读第一版《OWASP API 安全 Top10》。如果您熟悉 OWASP Top 10系列，您会注意到它们之间的相似之处：它们的目的是为了更容易阅读和被采纳。但是如果您不熟悉这个系列，请考虑先访问 [OWASP API 安全项目的维基页面](#)，然后再深入挖掘最关键的API安全风险。

API 在现代应用程序架构中发挥着非常重要的作用。虽然安全意识和创新发展不同步，但专注于常见的 API 安全弱点仍然很重要。

OWASP API 安全 Top 10的主要目的是教育在 API 开发和维护期间的参与者，例如，开发人员、设计师、架构师、管理人员或组织机构。

在[方法和数据](#)部分，您可以阅读有关我们是如何创建本文第一版的更多信息。在未来的版本中，我们希望让安全从业人员参与，并公开征集数据。现在，我们鼓励每个人通过我们的[GitHub 仓库](#)或[邮件列表](#)提出问题、评论和想法。

发布说明

这是第一版《OWASP API 安全 Top 10》，我们计划每三或四年更新一版。

与这个版本不同，在未来的版本中，我们希望公开征集数据，让安全行业的同行参与这项工作。在[方法和数据](#)部分中，您将了解到我们如何构建此版本的详细信息。若想了解更多有关安全风险的信息，请参阅[API 安全风险](#)部分。

重要的是要认识到，在过去的几年里，应用程序架构发生了显著变化。目前，API在微服务、单页应用程序（SPA）、移动应用程序、物联网（IoT）等新体系架构中扮演着非常重要的角色。

OWASP API 安全 Top 10是一项必要的工作，旨在提高人们对现代 API 安全问题的认识。由于志愿者们的不懈努力，这项工作才成为可能。这些志愿者都列在[致谢](#)部分。参与本中文版本项目的成员名单也列在[致谢](#)部分。

谢谢！

API安全风险

本项目采用了[OWASP风险评级方法](#)进行风险分析。

下表汇总了与风险评分相关的术语。

威胁来源	可利用性	弱点普遍性	弱点可检测性	技术影响	业务影响
API详情	易：3	广泛：3	易：3	严重：3	业务详情
	平均：2	常见：2	平均：2	中等：2	
	难：1	少见：1	难：1	小：1	

注意：此方法没有考虑威胁来源的可能性。它也没有考虑与特定应用程序相关的各种技术细节。这些因素都可能影响攻击者发现和利用特定漏洞的总体可能性。这个评估方法也没有考虑到您业务的实际影响。基于企业文化、行业和监管环境，您的组织将决定接受多少应用程序和 API 带来的安全风险。OWASP API 安全 Top 10的目的不是为您做风险分析。

参考资料

OWASP资料

- [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](#)

API1:2019-失效的对象级别授权	API倾向于公开处理对象标识符的端点，从而产生广泛的攻击表层访问控制问题。在使用用户输入访问数据源的每个函数中，都应考虑对象级授权检查。
API2:2019-失效的用户身份验证	身份验证机制的实现往往不正确，使得攻击者能够破坏身份验证令牌或利用漏洞临时或永久地盗用其他用户的身份。破坏了系统识别客户端/用户的能力，损害API的整体安全性。
API3:2019-过度的数据暴露	依赖通用方法，开发人员倾向于公开所有对象属性而不考虑其各自的敏感度，依赖客户端在向用户显示数据前执行数据筛选。
API4:2019-资源缺乏和速率限制	API 通常不会对客户端/用户可以请求的资源的大小或数量施加任何限制。这不仅会影响 API 服务器的性能，导致拒绝服务（DoS），而且还会为诸如暴力破解等身份验证缺陷敞开大门。
API5:2019-失效的功能级授权	具有不同层次结构、组和角色的复杂访问控制策略，以及管理功能和常规功能之间不明确的分离，往往会导致授权漏洞。通过利用这些漏洞，攻击者可以访问其他用户的资源和/或管理功能。
API6:2019-批量分配	将客户端提供的数据（例如 JSON）绑定到数据模型，而无需基于白名单进行适当的属性筛选，通常会导致批量分配。无论是猜测对象属性、探索其他 API 端点、阅读文档或在请求负载中提供其他对象属性，攻击者都可以修改它们不被允许修改的对象属性。
API7:2019-安全配置错误	安全错误配置通常是由于不安全的默认配置、不完整或临时配置、开放云存储、配置错误的HTTP 头、不必要的 HTTP 方法、允许跨域资源共享（CORS）和包含敏感信息的详细错误消息造成的。
API8:2019-注入	当不受信任的数据作为命令或查询的一部分发送给解释器时，就会出现注入缺陷，如 SQL、NoSQL、命令注入等。攻击者的恶意数据可诱使解释器在未经恰当授权的情况下执行非预期的命令或访问数据。
API9:2019-资产管理不当	与传统 Web 应用程序相比，API倾向于公开更多的端点，这使得恰当的文档编制和更新变得非常重要。正确的主机和已部署的 API 版本清单对于缓解弃用的 API 版本和公开的调试终端节点等问题也起着重要的作用。
API10:2019-日志和监视不足	日志记录和监控不足，加上与事件响应的集成缺失或无效，使得攻击者可以进一步攻击系统，保持持久性，转向更多系统以篡改、提取或销毁数据。大多数违规研究表明，检测违规行为的时间超过 200 天，通常由外部方而不是内部程序或监控发现。

失效的对象级授权

API详情	可利用性：3	普遍性：3	可检测性：2	技术：3	业务详情
<p>攻击者可以在发送的请求中改变对象的ID来攻击存在“失效的对象级授权”漏洞的API。这将导致敏感数据的未授权访问。该问题在基于API的应用中非常普遍，因为服务器通常不会完整地跟踪用户的状态，而是依赖用户请求参数中的对象ID来决定访问哪些目标对象。</p>		<p>这已经成为最普遍、且影响广泛的API攻击。授权和访问控制机制在现代应用中已经非常复杂并广泛使用。即使应用已经实现了适当的鉴权设施，开发者在访问敏感对象时仍可能忘记使用这些鉴权设施。通常在静态或动态测试中并不检查访问控制机制。</p>		<p>未授权访问将导致数据向未授权的组织披露、数据丢失或数据篡改。未授权的对象访问也能导致整个账户被控制。</p>	

API脆弱吗？

对象级授权是一种通常在代码层面实现的访问控制机制，用于校验用户仅能访问其有权访问的对象。

所有接收对象ID、并对对象执行任何操作的API，应该实现对象级授权检查。此检查应该校验该登录用户确实有权对其所请求的对象进行所请求的操作。

此校验机制失效通常将会导致数据的未授权信息披露、修改或销毁。

如何防止？

- 基于用户策略和继承关系来实现适当的授权机制。
- 在接收用户输入并访问数据库的所有函数中，都通过一种授权机制来检查该登录用户是否有权在该记录上执行其所请求的操作。
- 建议使用不可预测的随机GUID作为信息记录的ID。
- 通过测试来评估授权机制。不要部署带有脆弱点的变更，这将破坏测试。

攻击案例场景

场景#1：一个在线电子商务平台为他们的入驻商户提供了一个带有利润图表的列表页。通过查看浏览器发出的请求，攻击者可以识别到为此图表提供数据源的API端点及其模式/shops/{shopName}/revenue_data.json。通过另一个API端点，攻击者可以获取入驻平台的商户名列表，用一个简单的脚本即可获取到入驻商户名，通过替换URL中的{shopName}，攻击者获取到数千入驻商户的销售数据。

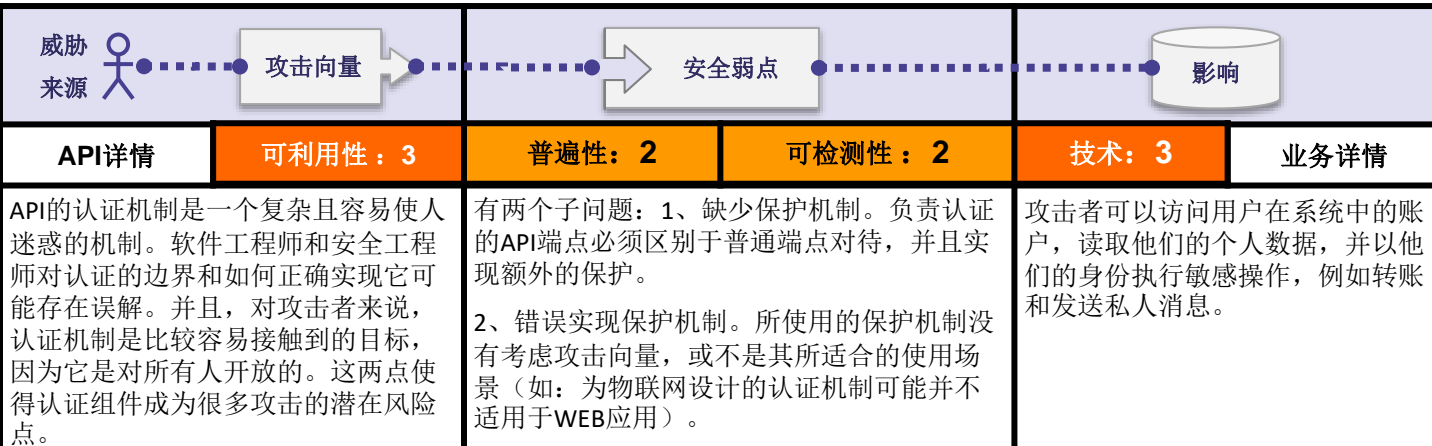
场景#2：通过监控可穿戴设备的网络流量，其HTTP PATCH请求中的自定义请求头部字段 X-User-Id:54796引起了攻击者的注意。通过替换X-User-Id字段的值为54795，攻击者接收到成功的HTTP响应，并可以修改其他用户的账户数据。

参考资料

外部资料

- [CWE-284: Improper Access Control](#)
- [CWE-285: Improper Authorization](#)
- [CWE-639: Authorization Bypass Through User-Controlled Key](#)

失效的用户身份认证



API脆弱吗？

应用端点和工作流是应该被保护的资产。“忘记密码/重置密码”应该和认证机制一样被同等看待。

如果API含有以下项则，则其存在风险：

- 当攻击者拥有有效的用户名和密码列表时，可以进行[凭据填充](#)；
- 缺失验证码或没有账号锁定机制，攻击者可以对同一用户账号进行暴力破解；
- 允许弱密码；
- 发送用于认证的敏感信息，例如在URL中发送令牌和密码；
- 未校验令牌的真实性；
- 接受未签名或弱签名的JWT令牌（“alg”：“none”），或未校验令牌过期时间；
- 使用明文密码、弱加密或者弱哈希密码；
- 使用弱密钥。

如何防止？

- 确保您知晓所有用于API认证的工作流（移动应用、Web应用、执行一键认证的Deep Links等）；
- 询问您的工程师看是否存在您遗漏了的工作流；
- 阅读您所使用的认证机制文档，确保您理解它是如何使用的。OAuth是一种协议，它不是一种认证，也不是API Key；
- 不要在认证、凭据生成、密码存储方面自创算法，使用标准的算法；
- 凭据重置、忘记密码端应被视为认证端点，在暴力破解、请求频率限制和锁定保护上同等对待；
- 使用[OWASP Authentication Cheat Sheet](#)；
- 如有可能，尽量使用多因素认证；
- 在您的认证端上使用反暴力破解机制来缓解凭据填充、字典攻击和暴力破解风险。此机制应该比您一般API具有更严格的请求频率限制；
- 实现[账号锁定](#)、验证码机制来防止对特定用户的暴力破解。
- 实现弱密码检查；
- API Key不应用于用户认证，而用于[客户端应用程序或项目认证](#)。

攻击案例场景

场景#1：[凭据填充](#)（使用[用户名、密码列表](#)）是一种常见的攻击。如果应用没有实现自动的威胁保护或者凭据填充保护，这个应用就能被当作密码验证器来检测凭据是否有效。

场景#2：攻击者通过向/api/system/verification-codes发起一个POST请求并在请求体中提供用户名来启动重置密码工作流。然后一个6位数字的SMS短信验证码发送到受害者的手机。由于该API没有实现调用频率限制，攻击者可以用多线程脚本向/api/system/verification-codes/{smsToken}发送请求来尝试所有可能的验证码组合，从而在几分钟内破解验证码。

参考资料

OWASP资料

- [OWASP Key Management Cheat Sheet](#)
- [OWASP Authentication Cheat Sheet](#)
- [Credential Stuffing](#)

外部资料

- [CWE-798: Use of Hard-coded Credentials](#)

过度的数据暴露

API详情	可利用性：3	普遍性：2	可检测性：2	技术：2	业务详情
利用过度暴露的数据十分容易，通常通过嗅探流量分析API的响应获取不应该返回给用户的多余敏感信息。		API依赖客户端实现数据过滤。当API被用作数据源时，有时开发者会尝试将API用于通用的方法，而不考虑所暴露数据的敏感性。自动化工具通常无法检测到该类漏洞，因为在没有对应用有深入理解的情况下，很难区分API返回的合法数据和不应该返回的敏感数据。			过度的数据暴露通常导致敏感数据的泄露。

API脆弱吗？

API在设计上将敏感数据返回至客户端，这些数据通常在展示给用户前被客户端过滤掉，因此，攻击者可以非常容易地通过嗅探流量获取敏感信息。

如何防止？

- 不要依赖客户端来过滤敏感数据；
- 检查API的响应，确认其中仅包含合法数据；
- 在开放一个新的API端点前，后端工程师应当反复确认“谁才是真正的数据使用者？”；
- 避免使用`to_json()`和`to_string()`之类的通用方法，而是选用您真正想要返回的特定属性；
- 将您的应用存储和工作所使用的敏感信息及个人识别信息（PII）进行分类，并检查所有返回上述信息的API调用，确认这些响应是否构成安全问题；
- 执行schema-based响应验证机制作为额外的安全措施。这种机制定义并强制检查所有方法返回的数据，包括错误信息。

攻击案例场景

场景#1：移动开发团队使用`/api/articles/{articleId}/comments/{commentId}`在article视图中展示comments。通过嗅探移动应用流量，攻击者可以找出被返回的其他comments的敏感信息。端点通过在包含个人识别信息（PII）的User模型上使用通用的`toJSON()`方法，将对象序列化。

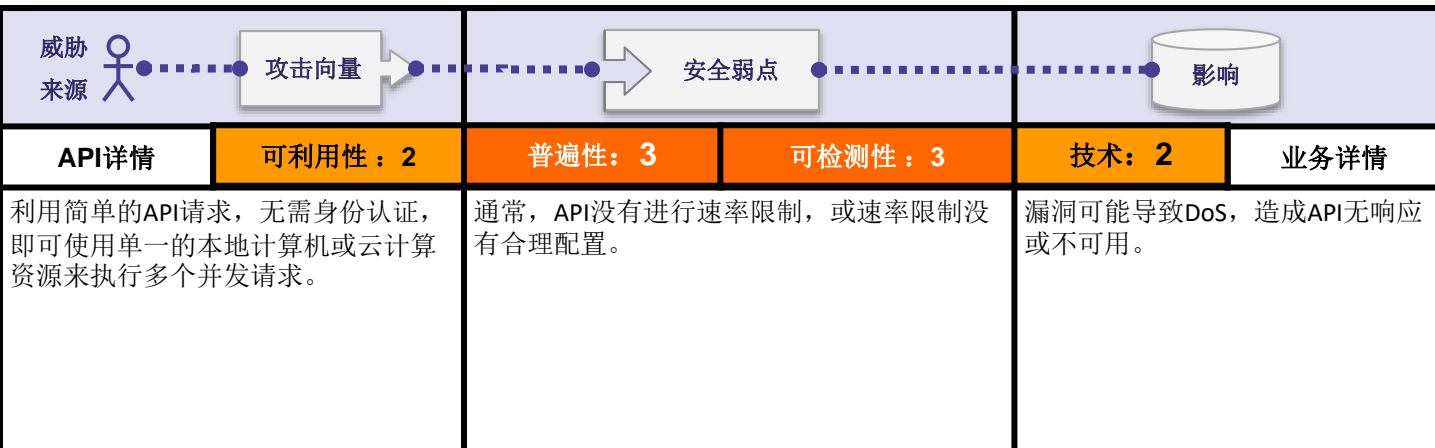
场景#2：一个基于物联网的视频监控系统允许管理员创建具有不同权限的用户。当管理员为一个新的安保人员创建账户时，应当在站点上只分配特定建筑范围的访问权限。当安保人员使用手机应用时，API会触发调用`/api/sites/111/cameras`来接收可用摄像头传输的数据，并显示在显示屏上。这个响应包含了一个基于`{"id":"xxx","live_access_token":"xxxxbbbb","building_id":"yyy"}`格式的摄像头详细列表。因此，当客户端的图形化界面仅显示安保人员应当可以访问的界面时，实际上API返回了站点内所有摄像头的完整列表。

参考资料

外部资料

- [CWE-213: Intentional Information Exposure](#)

资源缺乏和速率限制



API脆弱吗？

API请求会占用网络、CPU、内存和存储等资源，资源的需求取决于用户输入和端点的业务逻辑。同时，考虑到不同API客户端请求的竞争资源情况。当出现下述限制缺失或不恰当配置时（例如，过高或过低），API将存在脆弱点：

- 执行超时；
- 最大化分配内存；
- 文件描述符数；
- 进程数；
- 请求有效负载大小（例如，上传）；
- 单个客户端/资源的请求数；
- 单次请求响应时，每页返回的记录数。

如何防止？

- Docker可以轻松地限制[内存](#)、[CPU](#)、[重启次数](#)、[文件描述符和进程](#)；
- 对用户调用API的频率执行明确的时间窗口限制；
- 在突破限制时通知客户，并提供限制数量及限制重置的时间；
- 在服务器端为字符串查询和主体参数请求提供适当的验证，尤其是那些在响应中控制返回记录数量的验证；
- 定义并强制验证所有传入参数和有效负荷的最大数据量，例如字符串的最大长度和数组中元素的最大数量。

攻击案例场景

场景#1：一攻击者通过向/api/v1/images发送POST请求来上传大尺寸的图像。在上传完成后，API将创建多个不同大小的缩略图。由于上传的图像过大，可用内存存在创建缩略图时被耗尽，API将无法响应。

场景#2：当应用包含一个每页可以显示200个用户的列表界面时，我们可以使用/api/users?page=1&size=100查询从服务器中检索用户列表。攻击者可以将size参数篡改至 200 000，从而导致数据库出现性能问题。同时，API将无法响应该客户端的其他请求，或其他客户端的请求，形成DoS。

该场景也可能造成整数溢出或缓存溢出错误。

参考资料

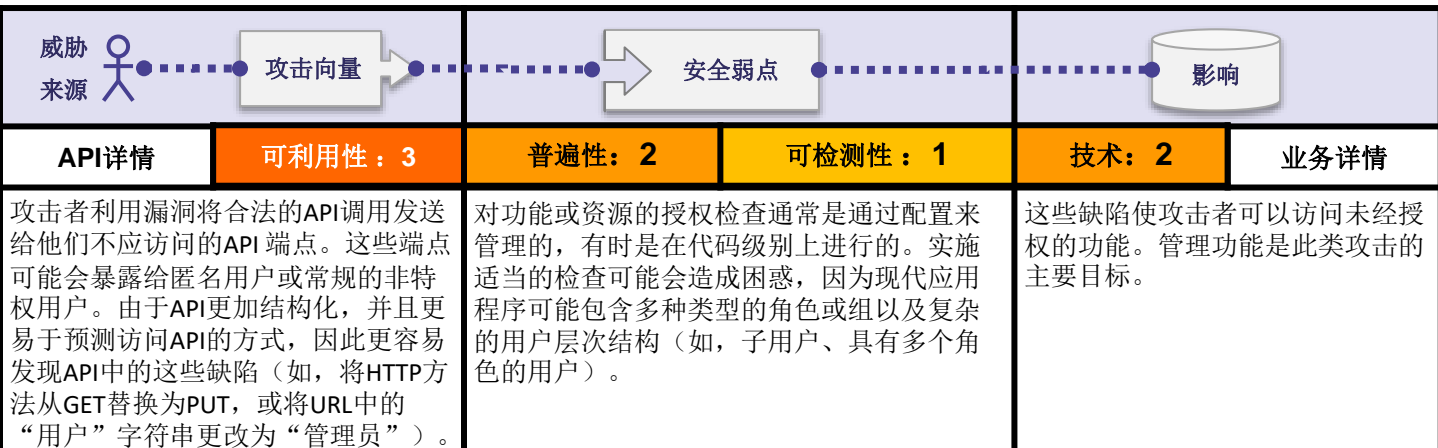
OWASP资料

- [Blocking Brute Force Attacks](#)
- [Docker Cheat Sheet - Limit resources \(memory, CPU, file descriptors, processes, restarts\)](#)
- [REST Assessment Cheat Sheet](#)

外部资料

- [CWE-307: Improper Restriction of Excessive Authentication Attempts](#)
- [CWE-770: Allocation of Resources Without Limits or Throttling](#)
- “Rate Limiting (Throttling)” - [Security Strategies for Microservices-based Application Systems](#), NIST

失效的功能级授权



API脆弱吗？

查找失效的功能级授权问题的最佳方法是对授权机制进行深入分析，同时牢记用户层次结构，应用程序中的不同角色或组，并提出以下问题：

- 普通用户可以访问管理端点吗？
- 用户是否可以通过简单地更改HTTP方法（如，从GET到DELETE）来执行他们不应该授予的敏感动作（如，创建、修改或删除）？
- 通过简单地猜测端点URL和参数（如，`/api/v1/users/export_all`），来自X组的用户是否可以访问本来应该只向Y组用户公开的功能？

不要假设仅基于URL路径的API端点是常规端点或管理端点。

尽管开发人员可能选择将大多数管理端点公开在特定的相对路径下，例如`api/admins`，但在其他相对路径下找到这些管理端点以及常规端点（例如`api/users`）是很常见的。

攻击案例场景

场景#1：在仅允许受邀用户加入的应用程序注册过程中，移动应用程序将触发对GET `/api/invites/{invite_guid}`的API调用。响应包含一个JSON，其中包含有关邀请的详细信息，包括用户的角色和用户的电子邮件。攻击者复制了请求，并操纵HTTP方法和端点进行POST `/api/invites/new`。管理员只能使用管理控制台访问该端点，该控制台未实现功能级别的授权检查。攻击者利用此问题并向自己发送邀请以创建管理员帐户：POST `/api/invites/new`

```
{ "email": "hugo@malicious.com", "role": "admin" }
```

场景#2：API包含仅应向管理员公开的端点GET `/api/admin/v1/users/all`。该端点返回应用程序所有用户的详细信息，并且未实现功能级别的授权检查。研究过API结构的攻击者进行了有根据的猜测，并设法访问了此端点，从而暴露了应用程序用户的敏感细节。

如何防止？

您的应用程序应该具有从所有业务功能中调用的一致且易于分析的授权模块。通常，这种保护是由应用程序代码外部的一个或多个组件提供的。

- 强制执行机制应拒绝所有访问，要求显式授予特定角色才能访问每个功能；
- 在考虑应用程序和组层次结构的业务逻辑的同时，针对功能级授权缺陷检查API端点；
- 确保所有管理控制器都从管理抽象控制器继承，该抽象控制器根据用户的组/角色实施授权检查；
- 确保常规控制器内的管理功能根据用户的组和角色实施授权检查。

参考资料


OWASP资料

- [OWASP Article on Forced Browsing](#)
- [OWASP Top 10 2013-A7-Missing Function Level Access Control](#)
- [OWASP Development Guide: Chapter on Authorization](#)

外部资料

- [CWE-285: Improper Authorization](#)

批量分配

					
API详情	可利用性：2	普遍性：2	可检测性：2	技术：2	业务详情
攻击利用通常需要了解业务逻辑、对象的关系和API结构。在API中更容易利用批量分配，因为它们通过设计公开了应用程序隐含的实现方法以及属性名称。		现代框架鼓励开发人员使用将来自客户端的输入自动绑定到代码变量和内部对象中的功能。攻击者可以使用这种方法来更新或覆盖开发人员从未打算公开的敏感对象属性。			利用该漏洞可能导致特权提升、数据篡改、绕过安全机制等。

API脆弱吗？

现代应用程序中的对象可能包含许多属性。其中一些属性应由客户端直接更新（如，`user.firstname`或`user.address`），而某些属性则不应该更新（如，`user.isvip`标志）。

如果API端点自动将客户端参数转换为内部对象属性，而不考虑这些属性的敏感性和暴露程度，则该端点很容易受到攻击。这可能使攻击者可以更新他们不应该访问的对象属性。

敏感属性示例：

- 与权限相关的属性：`user.isadmin`、`user.isvip`仅应由管理员设置；
- 与流程相关的属性：`user.cash`仅应在付款验证后在内部设置；
- 内部属性：`article.created_time`仅应在应用程序内部设置。

如何防止？

- 如果可能，请避免使用将客户输入自动绑定到代码变量或内部对象中的函数；
- 仅将客户端可更新的属性列入白名单；
- 使用内置功能将客户端不应访问的属性列入黑名单；
- 如果可能，为输入数据有效负载准确、明显的定义和实施schema格式。

攻击案例场景

场景#1：一个乘车共享应用程序为用户提供了编辑个人资料基本信息的选项。在此过程中，将使用以下合法的JSON对象将API调用发送到PUT `/api/v1/users/me`：`{"user_name":"inons","age":24}`。请求GET `/api/v1/users/me`包含一个附加`credit_balance`属性：`{"user_name":"inons","age":24,"credit_balance":10}`。攻击者使用以下负载重播第一个请求：`{"user_name":"attacker","age":60,"credit_balance":99999}`。由于端点易受批量分配缺陷影响，攻击者无需支付即可获得信用。

场景#2：一个视频共享门户允许用户上传并下载不同格式的内容。研究API的攻击者发现，端点GET `/api/v1/videos/{videoid}/metadata`返回具有视频属性的JSON对象。属性之一是`"mp4_conversion_Params": "-v codec h264"`，表示应用程序使用shell命令来转换视频。攻击者还发现端点POST `/api/v1/videos/new`容易受到批量分配缺陷的影响，并允许客户端设置视频对象的任何属性。攻击者将恶意值设置如下：`"mp4_conversion_params":"-v codec h264 && format C:/"`。一旦攻击者将视频下载为MP4，此值将导致注入shell命令。

参考资料

外部资料

- [CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes](#)

安全配置错误

API详情	可利用性：3	普遍性：3	可检测性：3	技术：2	业务详情
攻击者通常会试图查找未修补的缺陷、公共端点或未受保护的文件和目录，以获取对系统未经授权的访问或了解。		从网络层到应用层，在API的任何层级都可能发生安全配置错误。自动化工具可用于检测和利用不必要的服务或遗留选项等错误配置。			安全配置错误不仅会暴露敏感用户数据，还包括系统细节，而这些细节可能导致服务器完全被控制。

API脆弱吗？

在以下情况下，API可能会受到攻击：

- 应用程序堆栈的任何部分都缺少适当的安全加固，或者应用程序堆栈对云服务的权限配置不正确；
- 缺少最新的安全补丁，或者系统已经过期；
- 启用了不必要的功能（如，HTTP Verbs的功能应用）；
- 缺少传输层加密；
- 安全指令未发送给客户端（如，[HTTP的安全返回头](#)）；
- 未配置或错误配置CORS（跨域资源共享）策略；
- 错误提示泄漏了调用栈跟踪信息或其他敏感信息。

如何防止？

API的生命周期应包括：

- 一个可重复的加固流程，可快速、轻松地部署一个适当封闭的环境；
 - 在整个API堆栈中检查和更新配置的任务。审查应包括：文件编排、API组件和云服务（如，S3 bucket权限）；
 - 用于所有API交互访问静态资源（如，图像）的安全通信通道；
 - 在所有环境中持续评估配置和设置有效性的自动化过程。
- 此外：
- 为了防止异常追踪和其他有价值的信息被传回攻击者，如果可以，定义和强制使用统一的API响应格式，包括错误信息；
 - 确定API只能被特定HTTP方法访问，其他的HTTP方法访问都应该被禁止（如，HEAD方法）；
 - 对于主要是浏览器客户端（如，WebApp前端）访问的API应该实现正确的CORS（跨域资源共享）策略。

攻击案例场景

场景#1：攻击者在服务器的根目录下找到.bash_history文件，该文件包含DevOps团队用于访问API的命令：
\$ curl -X GET 'https://api.server/endpoint/' -H 'authorization: Basic Zm9vOmJhcg=='
攻击者还可以通过仅由DevOps团队使用而未记录的新端点。

场景#2：为了锁定特定服务，攻击者使用主流的搜索引擎搜索可从互联网直接访问的计算机。攻击者发现一个主机正在运行主流的数据库管理系统，该系统正在侦听默认端口。主机使用的是默认配置，默认情况下禁用了身份验证，攻击者可以访问包含PII、个人首选项和身份验证数据在内的数百万条记录。

场景#3：检查移动应用上的流量，攻击者发现并非所有的HTTP流量都使用安全协议（如TLS），特别是页面外链，在下载个人头像时，由于用户交互是二进制数的，尽管API的流量运行在安全协议上，但攻击者发现API响应大小的模式，他使用它来跟踪用户对渲染内容的偏好（如，个人头像）。

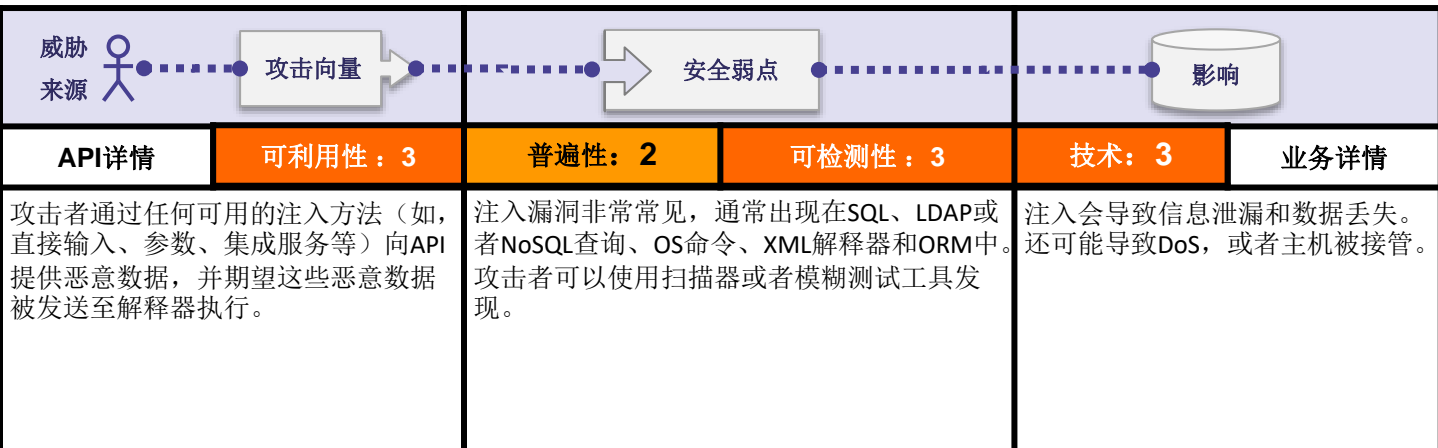
参考资料

OWASP资料

- [OWASP Secure Headers Project](#)
- [OWASP Testing Guide: Configuration Management](#)
- [OWASP Testing Guide: Testing for Error Codes](#)
- [OWASP Testing Guide: Test Cross Origin Resource Sharing](#)

外部资料

- [CWE-2: Environmental Security Flaws](#)
- [CWE-16: Configuration](#)
- [CWE-388: Error Handling](#)
- [Guide to General Server Security](#), NIST
- [Let's Encrypt: a free, automated, and open Certificate Authority](#)



API脆弱吗？

以下情况会导致注入攻击：

- API不对客户端提供的数据进行验证、过滤或净化；
- 客户端提供的数据直接使用或者拼接到SQL/NoSQL/LDAP查询语句、OS命令、XML解释器和ORM（对象关系映射器）/ODM（对象文档映射器）中；
- API不对来自外部系统（如，集成系统）的数据进行验证、过滤或净化。

如何防止？

防止注入需要将数据与命令和查询分开。

- 使用统一、可信并且活跃的库执行数据验证；
- 对客户端提供的数据、或其他来自集成系统的数据进行验证、过滤和清理；
- 应使用目标解释器的特定语法对特殊字符进行转义；
- 首选提供参数化查询的安全API；
- 总是限制返回记录的数量，以防止注入引起大量数据泄漏；
- 使用足够多的过滤器验证输入的数据，保证每一个输入参数只允许有效的值通过；
- 为所有字符串参数定义数据类型和严格模式。

攻击案例场景

场景#1： 家长控制设备的固件提供了一个接口/api/CONFIG/restore，期望将appid作为multipart参数进行传送。通过反编译，攻击者发现appid没有经过任何处理直接传送到系统调用中执行。

```
snprintf(cmd, 128, "%srestore_backup.sh /tmp/postfile.bin %s %d",
"/mnt/shares/usr/bin/scripts/", appid, 66);
system(cmd);
```

攻击者使用以下命令就可关闭任何使用带有该缺陷固件的设备：

```
$ curl -k "https://${deviceIP}:4567/api/CONFIG/restore" -F
'appid=${/etc/pod/power_down.sh}'
```

场景#2： 有一个能实现增删改查功能、用于进行预定操作的应用。攻击者研究发现在删除预定请求中的bookingId字符串查询参数可能存在NoSQL注入。请求类似：DELETE/api/bookings?bookingId=678。

API服务器使用以下函数实现删除请求：

```
router.delete('/bookings', async function (req, res, next) {
  try {
    const deletedBooking = await Bookings.findOneAndRemove({ '_id' : req.query.bookingId });
    res.status(200);
  } catch (err) {
    res.status(400).json({
      error: 'Unexpected error occurred while processing a request'
    });
  }
});
```

攻击者拦截了请求并更改bookId查询字符串参数如下所示。这个例子中，攻击者试图删除其他用户的预定操作。

```
DELETE /api/bookings?bookingId[$ne]=678
```

参考资料

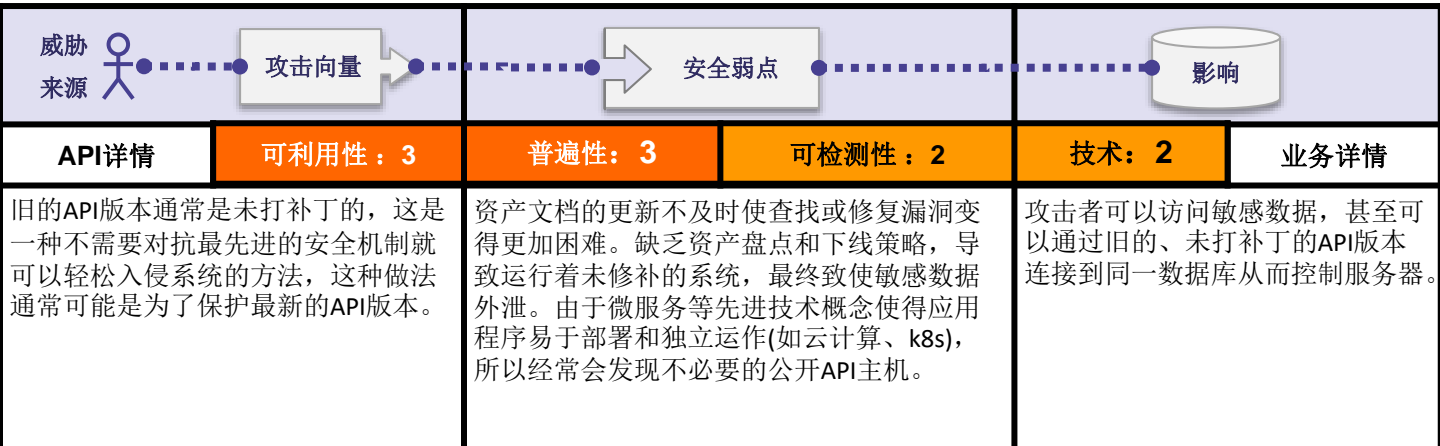
OWASP资料

- [OWASP Injection Flaws](#)
- [SQL Injection](#)
- [NoSQL Injection Fun with Objects and Arrays](#)
- [Command Injection](#)

外部资料

- [CWE-77: Command Injection](#)
- [CWE-89: SQL Injection](#)

资产管理不当



API脆弱吗？

如果API含有以下项，则存在风险：

- API主机的用途不明晰，同时以下问题也没有明确的答案：
 - API在哪个环境中运行（如，生产、准备、测试、开发）？
 - 谁应该对API具有网络访问权限（如，公共、内部、合作伙伴）？
 - 正在运行的是哪个API版本？
 - API收集和处理哪些数据（例如，PII）？
 - 数据流是什么？
- 没有关于API的文档，或者现有文档没有更新；
- 每个API版本都没有下线计划；
- 主机资源清单缺失或过期；
- 集成服务清单中无论是自有还是第三方的信息存在缺失或未更新的现象；
- 旧的API版本在没有打补丁的情况下持续运行。

如何防止？

- 列出所有API主机并记录所有重要信息，重点放在API的环境上(如，生产、准备、测试、开发环境)，谁应该对主机具有网络访问权(如，公共、内部、合作伙伴)和API版本；
- 对集成服务进行清点并记录重要信息，如它们在系统中的角色、交换了什么数据（数据流）及其敏感性；
- 记录API的所有信息，如，身份验证、错误、重定向、速率限制、跨域资源共享(CORS)策略和端点，包括它们的参数、请求和响应；
- 采用开放标准自动生成文档，包括在CI/CD管道中构建的文档；
- 向授权使用API的人提供API文档；
- 为所有公开版本的API使用外部保护措施，如API安全防火墙，不只是针对当前的产品版本；
- 避免在非生产API部署中使用生产数据。如果这是不可避免的，这些端点应该得到与生产端点相同的安全处理；
- 当较新的API版本包含安全改进时，实施风险分析以确定旧版本得到所需的风险缓解措施：如，是否可能在不破坏API兼容性的情况下支持这些改进，或者需要迅速删除旧版本并强制所有客户端迁移到最新版本。

攻击案例场景

场景#1：一个本地搜索服务商在重新设计了他们的应用程序后留下了一个没有保护且运行中的旧API版本（api.someservice.com/v1）并且其能访问用户数据库。而针对最新发布的一版应用程序，攻击者发现了API地址（api.someservice.com/v2）。将URL中的v2替换为v1使攻击者能够访问旧的、不受保护的API，从而暴露超过1亿用户的个人身份信息（PII）。

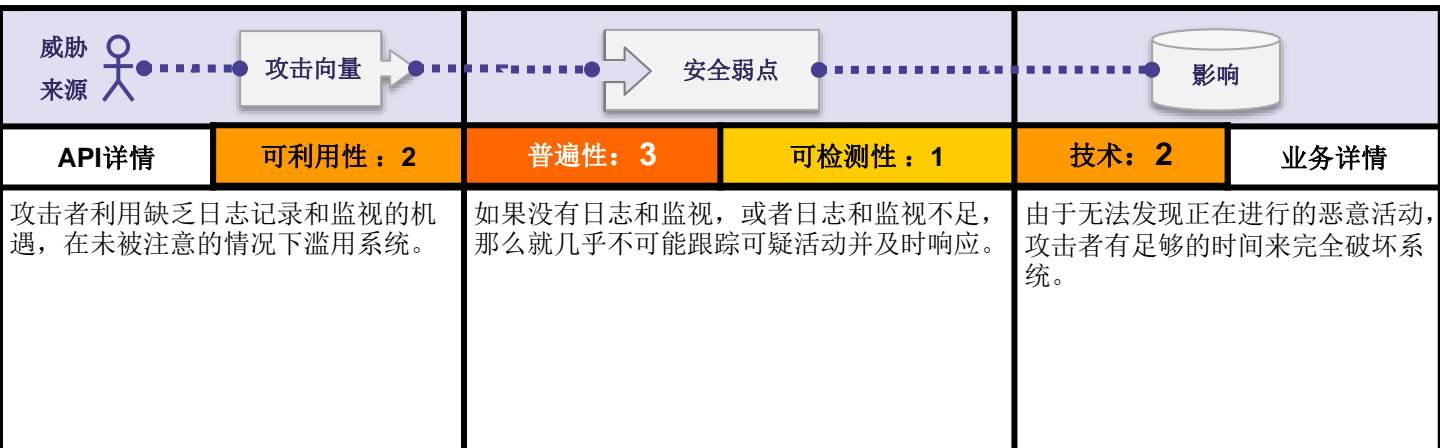
场景#2：一个社交网络实现了一种速率限制机制，阻止攻击者使用暴力猜测重置密码令牌。该机制不是作为API代码本身的一部分实现的，而是作为客户机和官方API之间的一个独立组件实现的（www.socialnetwork.com）。一名研究人员发现了一个运行相同API的beta API主机（www.mbasic.beta.socialnetwork.com），其中包括重置密码机制，但速率限制机制还没有到位。研究人员可以通过一个简单的暴力破解6位令牌来重置任何用户的密码。

参考资料

外部资料

- [CWE-1059: Incomplete Documentation](#)
- [OpenAPI Initiative](#)

日志和监视不足



API脆弱吗？

如果API含有以下项，则存在风险：

- 没有生成任何日志、日志级别没有正确设置、或日志消息缺失足够的细节信息；
- 不能保证日志的完整性（如，[日志注入](#)）；
- 没有对日志进行持续监视；
- API基础设施没有被持续监视。

如何防止？

- 记录所有失败的身份验证尝试、拒绝访问和输入验证错误；
- 日志应该使用适合日志管理解决方案的格式来编写，并且应该包含足够详细的信息来识别恶意的行动者；
- 日志应作为敏感数据处理，其完整性应在空闲和传输时得到保证；
- 配置一个监控系统，以持续监视基础设施、网络和API功能；
- 使用安全信息和事件管理系统（SIEM）来整合和管理来自API和主机的所有组件的日志；
- 配置自定义仪表板和警报，能够更早地检测和响应可疑活动。

攻击案例场景

场景#1：管理API的访问密钥泄漏到公共存储库中。存储库所有者收到了关于潜在泄漏的电子邮件通知，但在事件发生后花费了48个小时采取行动，那么访问密钥暴露就可能导致敏感数据被访问。由于日志记录不足，公司无法评估恶意访问者访问了哪些数据。

场景#2：一个视频共享平台遭到了“大规模”的凭据填充攻击。尽管登录失败，但在攻击期间没有触发警报。作为对用户投诉的回应，我们分析了API日志并检测到了攻击。该公司不得不发布公告，要求用户重置密码，并向监管部门报告此事。

参考资料

OWASP资料

- [OWASP Logging Cheat Sheet](#)
- [OWASP Proactive Controls: Implement Logging and Intrusion Detection](#)
- [OWASP Application Security Verification Standard: V7: Error Handling and](#)

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？

创建和维护安全软件或修复现有软件的任务可能很困难，API 也一样。

我们相信教育和意识是编写安全软件的关键因素。为实现这一目标需要的其他所有事务，都有赖于建立和使用可重复的安全流程和标准安全控制。

从OWASP项目开始以来，OWASP就提供了大量免费和开放的资源来解决安全问题。欢迎访问 [OWASP 项目页面](#) 了解可用项目的综合清单。

教育	您可以根据您的专业和兴趣开始阅读 OWASP教育项目材料 。对于实践学习，我们在 路线图 上添加了“Completely Ridiculous API（crAPI）项目”。同时，您可以使用 OWASP DevSlop Pixi模块 实践WebAppSec，这是一个易受攻击的WebApp和API服务，目的是教用户如何测试现代web应用程序和API的安全问题，以及如何在将来编写更安全的API。您还可以参加 OWASP AppSec 大会 的培训环节，或加入您 当地的OWASP分部 。
安全要求	从一开始，安全就应该是每个项目的一部分。在获取需求时，定义“安全”对该项目意味着什么是很重要的。OWASP建议您使用 OWASP应用程序安全验证标准（ASVS） 作为设定安全要求的指南。如果您是外包，请考虑 OWASP安全软件合同附件 ，该附件应根据当地法律法规进行调整。
安全架构	在项目的所有阶段，安全都应该是一个关注点。 OWASP Prevention Cheat Sheet 是指导如何在架构阶段设计安全性的良好起点。而其他内容，您将找到 REST Security Cheat Sheet 和 REST Assessment Cheat Sheet 。
标准的安全控件	采用标准的安全控件减少了在编写自有逻辑时引入安全弱点的风险。尽管许多现代框架现在都带有内置的标准有效控件，但是 OWASP Proactive Controls 为您提供了一个很好的概述，让您了解应该在项目中包含哪些安全控件。OWASP还提供了一些您可能会发现有价值的库和工具，例如验证控件。
软件安全开发生命周期	在构建API时，可以使用 OWASP软件保障成熟度模型（SAMM） 来改进流程。在不同的API开发阶段可以使用其他OWASP项目（如， OWASP代码审查项目 ），来帮助您。

DevSecOps下一步做什么？

由于API在现代应用程序架构中的重要性，构建安全的API至关重要。安全不能被忽视，它应该是整个开发生命周期的一部分。每年的扫描和渗透测试已经不够了。

DevSecOps应该加入到开发工作中，促进整个系统的持续安全测试。他们的目标是在不影响开发进度的情况下，通过安全自动化来增强开发管道。

如有疑问，请随时了解并经常查阅《[DevSecOps宣言](#)》。

了解威胁模型	测试优先级来自威胁模型。如果您还没有，请考虑参考 OWASP应用程序安全验证标准（ASVS） 和 OWASP测试指南 。让开发团队参与进来可能有助于提高他们的安全意识。
了解SDLC	加入开发团队以更好地了解软件开发生命周期。您贡献的持续安全测试需配置有合适的人员、流程和工具方面。每个人都应该同意这个过程，这样就不会有不必要的摩擦或阻力。
测试策略	由于您的工作不应该影响开发进度，所以您应该明智地选择最佳（简单、最快、最准确）技术来验证安全需求。 OWASP安全知识框架 与 OWASP应用程序安全验证标准 可以是功能性和非功能性安全需求的重要来源，其中重要来源还包括其他类似于 DevSecOps社区 提供的 项目 和 工具 。
实现覆盖范围和准确性	您是开发人员和运维团队之间的桥梁。要实现覆盖，不仅要关注功能，还要关注编排。从开始就和开发团队和运维团队紧密合作，您就可以优化您的时间和效率。您应该追求一个基本安全性不断得到验证的状态。
清晰地沟通调查结果	贡献价值少或没有摩擦。在开发团队正在使用的工具（不是PDF文件）中及时交付结果。加入开发团队解决问题。抓住机会指导他们，清楚地描述漏洞以及如何利用，包括利用漏洞进行的情景。

概述

由于 AppSec 行业并没有特别关注最近的应用程序架构（其中 API 发挥着重要作用），因此，基于公开征集的数据，编制一份包含十大最关键 API 安全风险的列表是一项艰巨的任务。尽管没有征集公开的数据，但由此产生的十大 API 安全风险仍然是基于可用的公开数据、安全专家的贡献、以及与安全社区的公开讨论。

方法和数据

在第一个阶段，由一组安全专家收集、审查和分类有关 API 安全事件的公开数据。这样的数据是在一年的时间内从 bug 赏金平台和漏洞数据库收集的，最终被用于统计。

在下一个阶段，具有渗透测试经验的安全人员被要求编制他们自己的 Top 10 列表。

采用 [OWASP 风险评级方法](#) 进行风险分析。评分是由安全从业人员进行讨论和审查。有关这些问题的考虑，请参阅本文 [API 安全风险](#) 部分。

《OWASP API 安全 Top 10 2019》的初稿是第一阶段统计结果与安全从业人员的列表达成一致的结果。该草案随后提交给另一组在 API 安全领域具有相关经验的安全从业人员，供其参考和审阅。

《OWASP API 安全 Top 10 2019》首次在 OWASP Global AppSec Tel Aviv 活动上发布（2019年5月）。从那时起，它就在 GitHub 上供公众讨论和建议。

本文的贡献者列表请见本文 [致谢](#) 部分。

感谢以下人员的贡献

我们要感谢以下在GitHub上或通过其他方式的贡献者：

- 007divyachawla
- Abid Khan
- Adam Fisher
- anotherik
- bkimminich
- caseysoftware
- Chris Westphal
- dsopas
- DSotnikov
- emilva
- ErezYalon
- flascelles
- Guillaume Benats
- IgorSasovets
- Inonshk
- JonnySchnittger
- jmanico
- jmdx
- Keith Casey
- kozmic
- LauraRosePorter
- Matthieu Estrade
- nathanawmk
- PauloASilva
- pentagramz
- philippederyck
- pleothaud
- r00ter
- Raj kumar
- Sagar Popat
- Stephen Gates
- thomaskonrad
- xycloops123

感谢以下参与本中文版本《OWASP API 安全 Top 10 2019》的OWASP中国成员。

- 组长：肖文棣
- 翻译：陈毓灵、黄鹏华、黄圣超、任博伦、张晓鲁（排名不分先后，按姓氏拼音排列）
- 审查：王颀

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

