



TOP10

2025

中文版



欢迎阅读 OWASP Top 10: 2025版本发布内容。

OWASP Top 10是面向开发人员与Web应用安全的标准安全意识文档，代表了业界对 Web 应用最关键安全风险的广泛共识。

项目组成员

项目组成员：张坤 肖文棣 王斌

审核组成员：王颀

由于成员知识与翻译水平有限，存在的错误敬请指正。邮箱：wangj@owasp.org.cn

关于本次发布

这是 OWASP Top 10的2025 版本。本版本基于最新数据与安全趋势进行了更新。

项目主页

项目主页包含旧版本信息以及本项目的元数据。

入门

建议从“引言”开始，了解2025版本有哪些新增与变化。

- [引言](#)
- [关于 OWASP](#)
- [什么是应用安全风险?](#)
- [建立现代应用安全计划](#)

2025 年版 Top 10 排行榜

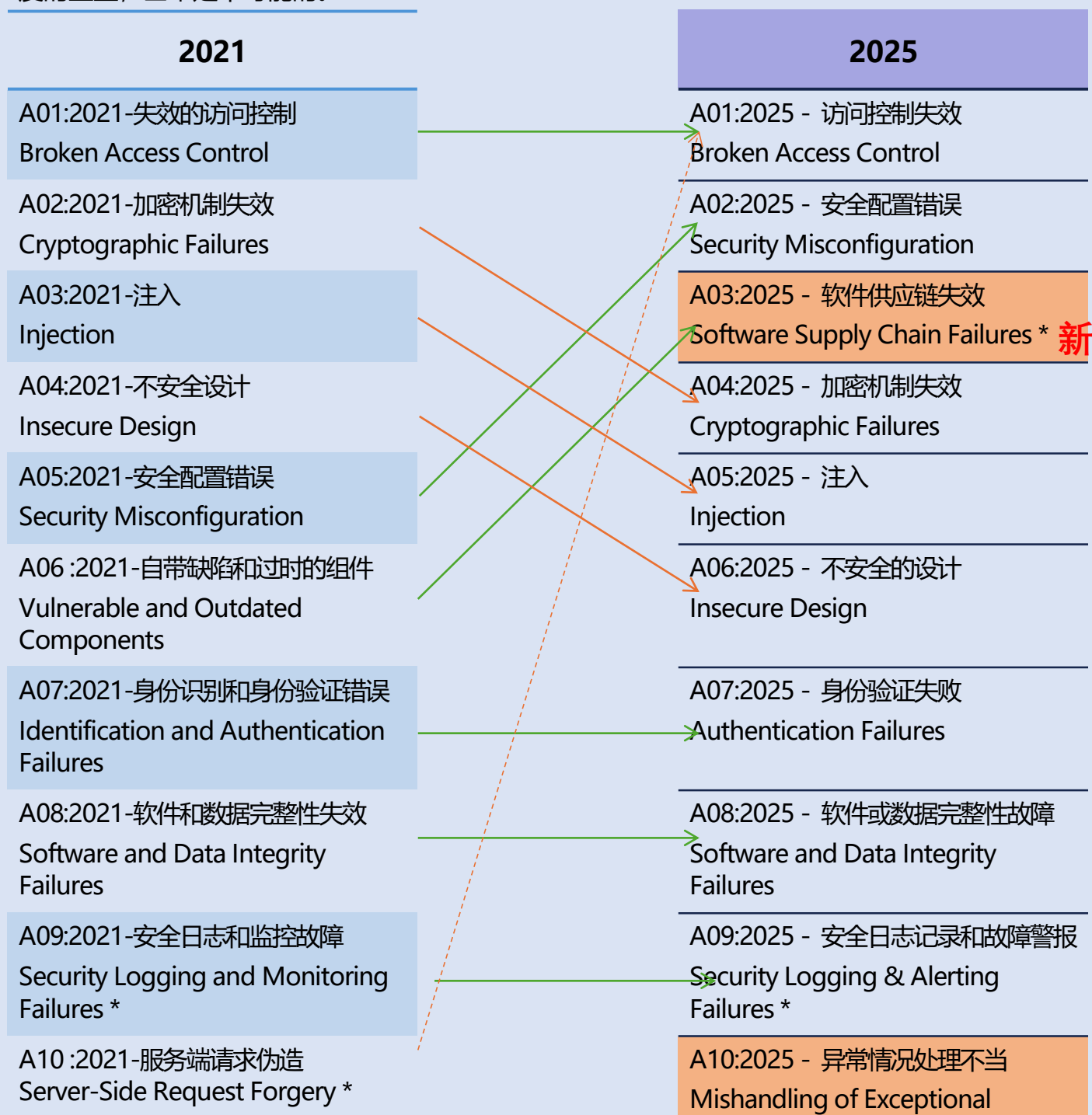
1. A01:2025 - 访问控制失效
2. A02:2025 - 安全配置错误
3. A03:2025 - 软件供应链失效
4. A04:2025 - 加密机制失效
5. A05:2025 - 注入
6. A06:2025 - 不安全的设计
7. A07:2025 - 身份认证失败
8. A08:2025 - 软件或数据完整性失效
9. A09:2025 - 安全日志记录和告警失效
10. A10:2025 - 异常情况处理不当

欢迎阅读 OWASP Top 10 的第八版！

衷心感谢所有在调研中贡献数据与观点的参与者。没有你们，本版将无法完成。谢谢大家！

2025 年十大排行榜发生了哪些变化？

2025 版 Top 10 中新增两个类别，并对一个类别进行了合并。我们尽可能保持对“根因”而非“症状”的关注。由于软件工程与软件安全本身的复杂性，要在仅 10 个类别中完全避免一定程度的重叠，基本是不可能的。



Conditions

- **A01:2025 - 访问控制失效** 继续位列第 1，仍是最严重的应用安全风险；贡献数据表明，平均而言，受测应用中有 3.73% 至少存在该类别 40 个通用弱点枚举（CWE）中的一个或多个。正如上图中的虚线所示，服务器端请求伪造（SSRF）已被并入此类别。
- **A02:2025 - 安全配置错误** 从 2021 年的第 5 名上升到 2025 年的第 2 名。本轮数据中配置错误更为常见：受测应用中有 3.00% 至少存在该类别 16 个 CWE 中的一个或多个。这并不意外，因为软件工程正在持续增加应用行为中由配置驱动的比例。
- **A03:2025 - 软件供应链失效** 是对 **A06:2021 - 易受攻击和过时的组件** 的扩展，将范围扩大到软件依赖生态、构建系统以及分发基础设施内部或跨域发生的更广泛妥协事件。在社区调研中，该类别被压倒性投票认为是最重要的关注点之一。该类别包含 5 个 CWE，在收集到的数据中出现频率有限，但我们认为这与测试难度相关，并希望后续测试能力在该领域跟上。该类别在数据中的出现次数最少，但其相关 CVE 的平均可利用性评分与影响评分最高。
- **A04:2025 - 加密机制失效** 从第 2 名下降到第 4 名。贡献数据表明，平均而言，3.80% 的应用至少存在该类别 32 个 CWE 中的一个或多个。该类别常常导致敏感数据泄露或系统被攻破。
- **A05:2025 - 注入** 从第 3 名下降到第 5 名，仍保持其相对于“加密机制失效”和“不安全的设计”的位置关系。注入是测试最充分的类别之一，与该类别 38 个 CWE 相关联的 CVE 数量最多。注入涵盖的问题范围很广，从跨站脚本（XSS，发生频率高/影响较低）到 SQL 注入（发生频率低/影响较高）漏洞。

- **A06:2025 - 不安全设计** 从第 4 名下降到第 6 名，被“安全配置错误”和“软件供应链失效”超越。该类别于 2021 年引入；我们已经看到行业在威胁建模方面有明显改进，并更加强调安全设计。
- **A07:2025 - 身份验证失败** 继续位列第 7，并对名称做了轻微调整（之前为“身份识别和身份验证失效”），以更准确反映该类别的 36 个 CWE。该类别依然重要，但身份认证标准化框架的使用增加，似乎对降低身份认证失效的发生有积极影响。
- **A08:2025 - 软件或数据完整性失效** 继续位列第 8。该类别聚焦于未能维持信任边界，以及未能在较低层面验证软件、代码与数据工件的完整性；其关注点比“软件供应链失效”更偏底层。
- **A09:2025 - 安全日志记录和告警失效** 继续位列第 9。该类别名称做了轻微调整（原名为“安全日志记录和监控故障”），以强调告警功能的重要性——告警能够促使针对相关日志事件采取适当行动。只有高质量日志但没有告警，在识别安全事件方面价值有限。该类别在数据中往往会被低估，本次也再次通过社区调研参与者投票进入榜单并获得当前排名。
- **A10:2025 - 异常情况处理不当** 是 2025 年新增类别。该类别包含 24 个 CWE，重点关注错误处理不当、逻辑错误、失败时默认放行（Fail Open）等，以及系统在异常条件下可能遇到的其他相关场景。

方法论

本期十大安全风险榜单依然以数据为依据，但并非盲目由数据驱动。我们基于贡献数据对 12 个类别进行排名，并允许其中两个类别由社区调研的反馈进行“提升”或“重点突出”。我们这样做有一个根本原因：审视贡献数据本质上是在回顾过去。应用安全研究人员会投入时间识别新漏洞并开发新的测试方法，而这些测试从提出到融入工具与流程，往往需要数周到数年不等。等到我们

能够在大规模范围内可靠地测试某一弱点时，可能已经过去了好几年。此外，还存在一些重要风险，我们可能永远无法可靠地进行规模化测试，因此也不会出现在数据中。为平衡这一视角，我们通过社区调研向一线的应用安全与开发实践者征询：他们认为哪些关键风险在测试数据中可能被低估。

类别是如何构建的

与之前的 OWASP 十大安全漏洞报告相比，本次有少量类别发生变化。以下为类别变化的高层概述。

在本次迭代中，我们征集数据时不再像 2021 版那样对 CWE 施加限制。我们要求提供：某一年度（从 2021 年开始）受测应用的数量，以及在测试中至少发现一次某个 CWE 的应用数量。该数据格式使我们能够追踪每个 CWE 在应用总体中的普遍程度。我们在本项目中不使用“出现次数（frequency）”，因为尽管在其他场景可能需要它，但它只会掩盖该弱点在应用总体中的真实普遍性。一个应用里某个 CWE 出现 4 次还是 4000 次，并不纳入 Top Ten 的计算。尤其是，人工测试人员往往无论漏洞在应用中重复多少次都只记录一次，而自动化测试框架则会把每一个漏洞实例都作为独立记录。我们从 2017 年约 30 个 CWE，增长到 2021 年接近 400 个 CWE，再到本版用于数据集分析的 589 个 CWE。未来我们计划补充开展更多数据分析。CWE 数量的大幅增长，要求我们调整类别的构成方式。

我们用了数月时间对 CWE 进行分组与归类，而且若继续还可以再做数月。我们必须在某个点上停下来。CWE 既包含“根因型”也包含“症状型”：根因型例如“加密机制失效”“配置错误”，而症状型例如“敏感数据泄露”“拒绝服务”。我们决定在可能的情况下优先聚焦根因，因为这更符合提供识别与修复指导的逻辑。强调“根因优先于症状”并非新概念；Top Ten 历来就是症状与根因的混合。CWE 本身也是症状与根因的混合；我们只是更有意识地将这一点明确出来。本版中每个类别平均包含 25 个 CWE：下限为 5 个 CWE（A03：2025-软件供应链失效 与 A09：2025-安全日志记录与告警失效），上限为 40 个 CWE（A01：2025-访问控制失效）。我们决

定将单个类别中的 CWE 数量上限设置为 40。更新后的类别结构也带来额外的培训收益——企业可以聚焦于对其所用编程语言/框架更有意义的 CWE。

有人问我们：为什么不把 Top 10 直接改为“10 个 CWE 列表”，类似 MITRE 的“最危险的 25 个软件弱点 (Top 25)”？我们在类别中使用多个 CWE 主要有两个原因。第一，并非所有 CWE 都存在于所有编程语言或框架中，这会给工具与培训/意识项目带来问题，因为 Top Ten 的某些项可能不适用。第二，常见漏洞往往对应多个 CWE。例如：通用“注入”、命令注入、跨站脚本 (XSS)、硬编码密码、缺乏校验、缓冲区溢出、敏感信息明文存储等，都可能对应多个 CWE。不同组织或测试人员可能会选用不同的 CWE。通过使用包含多个 CWE 的类别，我们可以提高基线认知，帮助理解同一类别名称下可能出现的不同弱点类型。在 2025 版 Top Ten 的 10 个类别中，共包含 248 个 CWE。在本次发布时，[MITRE 可下载词典](#)中共有 968 个 CWE。

数据如何用于选择类别

与 2021 版类似，我们利用 CVE 数据来衡量“可利用性 (Exploitability)”与“(技术) 影响 ((Technical) Impact)”。我们下载了 OWASP Dependency Check 并提取其中的 CVSS 可利用性与影响分数，然后按 CVE 所列出的相关 CWE 进行分组。这个过程需要相当多的研究与工作量，因为所有 CVE 都有 CVSSv2 评分，但 CVSSv2 存在缺陷，而 CVSSv3 旨在解决这些问题。在某个时间点之后，所有 CVE 也会同时分配 CVSSv3 评分。此外，CVSSv2 与 CVSSv3 的评分范围与公式也进行了更新。

在 CVSSv2 中，可利用性与 (技术) 影响的原始分值都可达到 10.0，但公式会将可利用性按 60%、影响按 40% 进行折算。在 CVSSv3 中，理论最大值被限制为：可利用性 6.0、影响 4.0。考虑权重后，CVSSv3 的影响评分整体上移，平均提高接近 1.5 分；可利用性则平均下降接近 0.5 分。

从 OWASP Dependency Check 中提取的数据中，NVD（国家漏洞数据库）里大约有 17.5 万条 CVE→CWE 映射记录（2021 年约为 12.5 万）。同时，映射到 CVE 的唯一 CWE 数量为 643 个（2021 年为 241 个）。在提取出的近 22 万个 CVE 中，16 万有 CVSS v2 分数，15.6 万有 CVSS v3 分数，6000 有 CVSS v4 分数。许多 CVE 同时拥有多个版本的评分，因此这些数量相加会超过 22 万。

对于 2025 版十大漏洞，我们按如下方式计算平均可利用性与影响分数：将所有带 CVSS 分数的 CVE 按 CWE 分组，并按采用 CVSSv3 的占比及剩余采用 CVSSv2 的占比对可利用性与影响分数进行加权，得到总体平均值。然后将这些平均值映射到数据集中的 CWE，用作风险方程另一半的“可利用性”与“（技术）影响”评分。

你可能会问：为什么不使用 CVSS v4.0？原因在于其评分算法发生了根本性变化，不再像 CVSS v2 与 CVSSv3 那样容易直接得到“可利用性”或“影响”分数。我们将尝试在未来版本的 Top Ten 中找到使用 CVSS v4.0 评分的方法，但在 2025 版中未能找到可在合理时间内完成的实现路径。

为什么使用社区调查

数据结果很大程度上受限于行业能够以自动化方式测试的内容。与资深应用安全（AppSec）从业者交流，他们会提到许多尚未进入数据的发现与趋势。针对某些漏洞类型，测试方法学的形成需要时间；随后将这些测试自动化并在大量应用上运行，也需要更多时间。我们所看到的一切都在回望过去，可能遗漏过去一年出现的新趋势——这些趋势尚未体现在数据中。

因此，我们只从数据中选取十个类别中的八个，因为数据并不完整。剩下两个类别来自 Top 10 社区调研。该调研允许一线实践者投票选出他们认为风险最高、但可能未体现在数据中的类别（甚至可能永远无法通过数据充分表达）。

数据贡献者

感谢以下组织（以及若干匿名捐赠者）慷慨提供了覆盖超过 280 万个应用的数据，使本次数据集成为规模最大、覆盖最全面的应用安全数据集。没有你们的支持，这一切将无法实现。

- Accenture (Prague)
- Anonymous (multiple)
- Bugcrowd
- Contrast Security
- CryptoNet Labs
- Intuitor SoftTech Services
- Orca Security
- Probely
- Semgrep
- Sonar
- usd AG
- Veracode
- Wallarm

核心作者

- Andrew van der Stock - X: [@vanderaj](#)
- Brian Glas - X: [@infosecdad](#)
- Neil Smithline - X: [@appsecneil](#)
- Tanya Janca - X: [@shehackspurple](#)

-
- Torsten Gigler - Mastodon: [@torsten_gigler@infosec.exchange](https://mastodon.social/@torsten_gigler)

开放式 Web 应用安全项目（Open Worldwide Application Security Project, OWASP）是一个开放社区，致力于帮助组织开发、采购并维护值得信赖的应用与 API。

在 OWASP，你可以免费获取并开放使用的资源包括：

- 应用安全工具和标准
- 前沿研究成果
- 标准安全控制措施和库
- 关于应用程序安全测试、安全编码开发和安全代码审查的完整书籍
- 演示文稿和[视频](#)
- 许多常见主题的[速查表](#)
- [在全球及线上举办的分会（Chapter）会议与活动。](#)
- [活动、培训和会议。](#)
- [Google 群组](#)

了解更多信息，请访问：<https://owasp.org>。

OWASP 的所有工具、文档、视频、演示材料以及各地分会资源，均对任何希望提升应用安全的人免费开放。

我们倡导将应用安全视为“人员、流程与技术”的综合问题，因为最有效的应用安全方法需要在这三个方面同时改进。

OWASP 是一种不同类型的组织。我们不受商业压力影响，因此能够提供客观、公正、实用且具成本效益的应用安全信息。

OWASP 不隶属于任何科技公司，但我们支持在充分理解的前提下使用商业安全技术。

OWASP 以协作、透明与开放的方式产出多种类型的资料。

OWASP 基金会 (OWASP Foundation) 是确保项目长期成功的非营利实体。几乎所有参与 OWASP 的人员都是志愿者，包括 OWASP 董事会、分会负责人、项目负责人和项目成员。我们通过资助与基础设施支持创新性的安全研究。

加入我们！

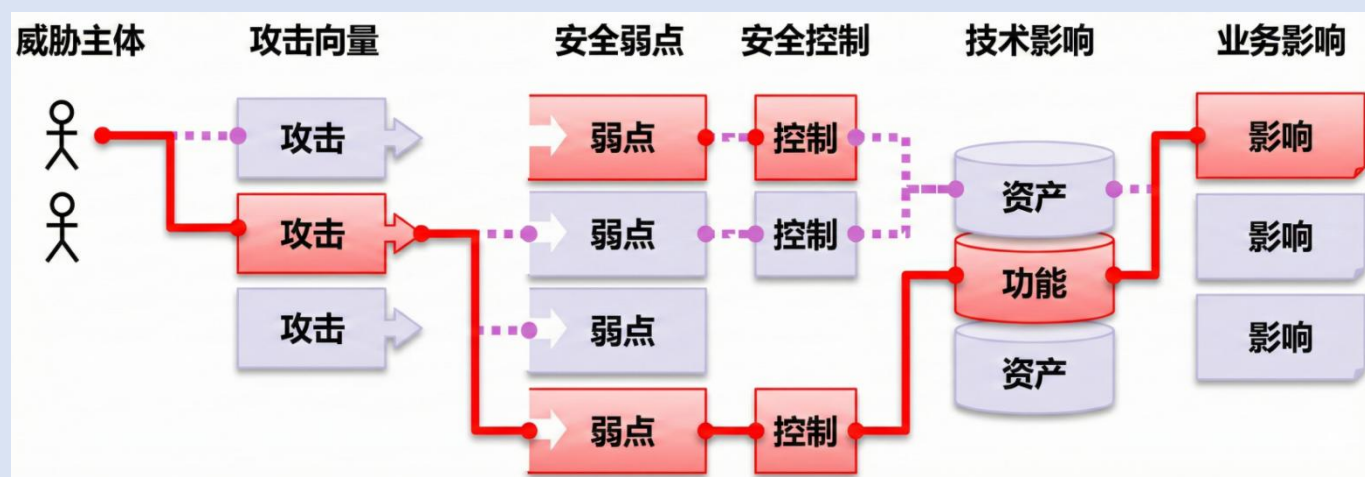
版权和许可



版权所有 © 2003-2025 OWASP® Foundation, Inc. 本文档依据 “知识共享 署名-相同方式共享 4.0 (Creative Commons Attribution Share-Alike 4.0) ” 许可协议发布。任何再利用或再分发时，你必须向他人明确说明本作品的许可条款。

什么是应用程序安全风险？

攻击者可能通过你应用中的多种不同路径对你的企业或组织造成损害。每一种方式都构成潜在风险，需要进行调查与评估。



威胁主体	攻击向量	可利用性	安全控制缺失的可能性	技术的影响	商业影响
按环境维度、随场景动态变化	按应用暴露程度 (按环境)	平均加权可利用性	安全控制缺失：按平均发生率并按覆盖度加权	平均加权影响	按业务纬度

在我们的风险评级中，我们综合考虑了通用参数：可利用性、某项弱点安全控制缺失的平均可能性，以及其技术影响。

每个组织都是独特的，该组织面临的威胁主体、其目标以及任何安全事件造成的影响也各不相同。举例来说，某公益组织使用内容管理系统(CMS)发布公开信息，而某医疗系统使用同一套 CMS 存储敏感健康记录；即使软件完全相同，威胁主体与业务影响也可能截然不同。因此，基于应用的暴露面、在特定场景下适用的威胁主体画像（按业务与地点划分的定向攻击与非定向攻击），以及组织自身的业务影响来理解本组织面临的风险至关重要。

数据如何用于选择类别和进行排名

在 2017 年，我们先依据发生率 (incidence rate) 选择类别以确定“可能性”，再基于团队数十年的经验讨论，对可利用性、可检测性（也属于可能性）和技术影响进行排序。2021 年，我

什么是应用程序安全风险？



们使用国家漏洞数据库 (NVD) 中 CVE 的 CVSSv2 与 CVSSv3 评分数据来衡量可利用性与 (技术) 影响。2025 年，我们延续了 2021 年建立的方法论。

我们下载了 OWASP Dependency Check，并提取按相关 CWE 分组后的 CVSS 可利用性与影响分数。这个过程需要相当多的研究与工作量，因为所有 CVE 都有 CVSSv2 分数，但 CVSSv2 存在缺陷，而这些缺陷应由 CVSSv3 予以改进。在某个时间点之后，所有 CVE 也都会同时分配 CVSSv3 分数。此外，CVSSv2 与 CVSSv3 的评分范围与公式也进行了更新。

在 CVSSv2 中，可利用性与 (技术) 影响的原始分值都可达到 10.0，但公式会将可利用性按 60%、影响按 40% 进行折算。在 CVSSv3 中，理论最大值被限制为：可利用性 6.0、影响 4.0。考虑权重后，在我们对 2021 版 Top Ten 进行分析时发现，CVSSv3 的影响评分整体上移，平均提高接近 1.5 分；可利用性则平均下降接近 0.5 分。

从 OWASP Dependency Check 中提取的数据中，NVD 里大约有 17.5 万条 CVE→CWE 映射记录 (2021 年约为 12.5 万)。同时，映射到 CVE 的唯一 CWE 数量为 643 个 (2021 年为 241 个)。在提取出的近 22 万个 CVE 中，16 万有 CVSS v2 分数，15.6 万有 CVSS v3 分数，6000 有 CVSS v4 分数。许多 CVE 同时拥有多个版本的评分，因此这些数量相加会超过 22 万。

对于 2025 年十大漏洞，我们按如下方式计算平均可利用性与影响分数：将所有带 CVSS 分数的 CVE 按 CWE 分组，并按采用 CVSSv3 的占比及剩余采用 CVSSv2 的占比对可利用性与影响分数进行加权，得到总体平均值。然后将这些平均值映射到数据集中的 CWE，用作风险方程另一半的“可利用性”与“(技术) 影响”评分。

你可能会问，为什么不使用 CVSS v4.0？这是因为评分算法发生了根本性的变化，它不再像 CVSSv2 和 CVSSv3 那样能够轻松地提供漏洞利用或影响评分。我们会尝试在未来版本的 Top Ten 中采用 CVSS v4.0 评分，但在 2025 版中未能找到可在合理时间内完成的实现路径。

什么是应用程序安全风险？

对于发生率 (incidence rate) , 我们计算的是: 在某一时间段内, 某组织测试过的应用总体中, 每个 CWE 所对应的 “存在漏洞的应用占比” 。再次提醒, 我们不使用 “频次 (frequency, 即某问题在单个应用中出现多少次) ” , 我们关注的是: 在应用总体中, 有多少比例的应用被发现存在该 CWE。

对于覆盖度 (coverage) , 我们关注的是: 所有组织针对某一 CWE 进行测试的应用占比。覆盖度越高, 说明样本规模越能代表总体, 从而对发生率的准确性提供更强的保证。

本次迭代使用的公式与 2021 年类似, 但对权重做了一些调整: $(\text{最大发生率}\% \times 1000) + (\text{最大覆盖度}\% \times 100) + (\text{平均可利用性} \times 10) + (\text{平均影响} \times 20) + (\text{出现次数合计} / 10000)$
= 风险评分 (Risk Score)

计算得到的评分范围为: 访问控制失效 (Broken Access Control) 类别 621.60, 到内存管理错误 (Memory Management Errors) 类别 271.08.

这并不是一个完美的体系, 但对于风险类别的排序仍然具有价值。

另一个日益突出的挑战是对 “应用 (application) ” 的定义。随着行业转向由微服务等组成的新架构, 这些实现粒度小于传统意义上的应用, 计算会更困难。例如, 当组织测试的是代码仓库时, 它如何界定 “一个应用” ? 类似于 CVSSv4 的发展, 下一版 Top Ten 可能需要调整分析与评分方法, 以适应不断变化的行业形态。

数据因素

前十大类别均列出了相应的数据因素, 以下是这些因素的含义:

- 映射的 CWE (CWE Mapped) : Top Ten 团队映射到该类别的 CWE 数量。
- 发生率 (Incidence Rate) : 在该年度, 某组织测试的应用总体中, 存在该 CWE 漏洞的应用占比。

什么是应用程序安全风险？

- 加权可利用性 (Weighted Exploit) : 将映射至该 CWE 的 CVE 对应的 CVSSv2、CVSSv3 可利用性子分数归一化至 10 分制。
- 加权影响 (Weighted Impact) : 映射到 CWE 的 CVE 所对应的 CVSSv2 与 CVSSv3 “影响 (Impact)” 子分数, 经归一化后换算到 10 分制。
- (测试) 覆盖度 ((Testing) Coverage) : 所有组织针对某一 CWE 进行测试的应用占比。
- 总出现次数 (Total Occurrences) : 检测出存在该类别映射 CWE 弱点的应用总数。
- CVE 总数: NVD 数据库中映射到该类别 CWE 的 CVE 总数。
- 公式 (Formula) : $(\text{最大发生率} \% \times 1000) + (\text{最大覆盖度} \% \times 100) + (\text{平均可利用性} \times 10) + (\text{平均影响} \times 20) + (\text{出现次数合计} / 10000) = \text{风险评分 (Risk Score)}$

建立现代应用程序安全计划



OWASP Top 10 系列清单是安全意识文档，旨在提升对其所覆盖主题中最关键风险的认知。它们并非完整清单，而只是一个起点。在该清单的以往版本中，我们一直建议：启动应用安全计划是避免这些风险及更多风险的最佳方式。本节将介绍如何启动并构建一个现代化的应用安全计划。

如果您已经拥有应用程序安全计划，不妨考虑使用 [OWASP SAMM](#)（软件保障成熟度模型）或 [DSOMM](#)（DevSecOps 成熟度模型）对其进行成熟度评估。这些成熟度模型全面且覆盖范围广，可帮助你识别在扩展与成熟化计划时最应聚焦的方向。请注意：并不需要把 OWASP SAMM 或 DSOMM 的所有内容都做完了才算做得好——它们旨在为你提供指引并给出多种选择；并非要提出难以企及的标准或描述成本高昂、难以负担的计划。它们之所以覆盖面广，是为了提供更多思路与可选项。

如果你要从零开始建立计划，或者你觉得 OWASP SAMM 或 DSOMM 对你的团队来说目前“过于庞大”，请参考以下建议。

1. 建立基于风险的投资组合方法：

- 从业务角度确定应用程序组合的保护需求。这部分应由隐私法和与受保护数据资产相关的其他法规驱动。
- 建立一套[通用的风险评级模型](#)，该模型包含一组一致的可能性和影响因素，能够反映贵组织对风险的承受能力。
- 据此对所有应用和 API 进行评估和优先级排序，并将结果添加到[配置管理数据库 \(CMDB\)](#)中。
- 建立保障指南，明确覆盖范围与所需严谨程度的要求。

2. 打好坚实的基础：

- 制定一套重点明确的政策和标准，为所有开发团队提供应用程序安全基线。
- 定义一套可复用的通用安全控制措施，用以补充上述政策与标准，并为其使用提供设计与开发层面的指导。
- 建立一套针对不同开发角色和主题的、必要的应用程序安全培训课程。

3. 将安全措施融入现有流程：

- 将安全实施和验证活动纳入现有开发和运营流程中。
- 活动包括威胁建模、安全设计和设计审查、安全编码和代码审查、渗透测试和整改措施。
- 为研发与项目团队提供主题专家与支持服务，帮助团队成功落地。
- 审查当前系统开发生命周期以及所有软件安全活动、工具、政策与流程，并形成文档。
- 对新软件而言，在系统开发生命周期（SDLC）的每个阶段加入一个或多个安全活动。下文将提供许多可选建议。确保在每一个新项目或软件举措中都执行这些新增活动，这样你就能确保每一项新交付的软件都具备组织可接受的安全态势。
- 选择合适的活动，确保最终产品符合贵组织可接受的风险水平。
- 对现有软件（有时称为遗留系统/存量系统）而言，你需要制定正式的维护计划；关于如何在“运营与变更管理”部分维护安全应用，请参考下文建议。

4. 应用安全教育：

- 不妨考虑启动一个安全倡导者计划，或者面向开发人员的通用安全教育计划（有时也称为安全倡导或安全意识计划），让他们掌握您希望他们了解的一切。这将帮助他们及时了解最新知识，掌握安全的工作方法，并营造更积极的工作安全文化。此外，它通常还能增进团队间的信任，建立更和谐的工作关系。OWASP 通过不断完善的《OWASP 安全倡导者指南》为您提供支持。
- OWASP 教育项目提供培训资料，帮助开发人员了解 Web 应用程序安全。如需进行漏洞方面的实践学习，请尝试 OWASP Juice Shop 项目或 OWASP WebGoat。为了保持知识更新，您可以参加 OWASP AppSec 大会、OWASP 大会培训或当地 OWASP 分会会议。

5. 提供管理层可视性：

- 通过度量指标进行管理。基于已采集的指标与分析数据来推动改进，并据此做出资金投入与资源配置决策。指标包括：对安全实践与活动的遵循情况、引入的漏洞数量、已缓解/修复的漏洞数量、应用覆盖度、按类型统计的缺陷密度及实例计数等。
- 分析实施和验证活动的的数据，找出根本原因和漏洞模式，从而推动企业范围内的战略性和系统性改进。从错误中吸取教训，并提供积极的激励措施来促进改进。

建立并使用可重复的安全流程和标准安全控制措施

需求和资源管理阶段：

- 与业务部门收集并协商应用程序的业务需求，包括所有数据资产的保密性、真实性、完整性和可用性方面的保护要求，以及预期的业务逻辑。

建立现代应用程序安全计划



- 编制技术需求，包括功能性和非功能性安全需求。OWASP 建议您使用 [OWASP 应用程序安全验证标准 (ASVS) (<https://owasp.org/www-project-application-security-verification-standard/>)] 作为指导，来设定应用程序的安全需求。
- 制定并协商涵盖设计、建造、测试和运营（包括安全活动）各个方面的预算。
- 在项目进度计划中加入安全保障活动。
- 在项目启动会上自我介绍，表明你是安保代表，以便他们知道该和谁交谈。

征求建议书 (RFP) 和合同签订：

- 与内部或外部开发人员协商需求，包括有关安全计划的指导方针和安全要求，例如软件开发生命周期 (SDLC) 和最佳实践。
- 评估所有技术要求的完成情况，包括规划和设计阶段。
- 协商所有技术要求，包括设计、安全和服务级别协议 (SLA) 。
- 采用模板和清单，例如 [OWASP 安全软件合同附件](#)。

注意：该附件适用于美国合同法，因此在使用示例附件前，请咨询合格的法律顾问。

规划设计阶段：

- 与开发商和内部股东（例如安全专家）协商规划和设计。
- 根据防护需求和预期威胁级别，制定相应的安全架构、控制措施、应对措施和设计评审方案。这项工作应由安全专家提供支持。

建立现代应用程序安全计划

- 与其事后在应用程序和 API 中添加安全措施，不如从一开始就将安全性融入设计，这样更具成本效益。OWASP 推荐使用 [OWASP 速查表](#)和 [OWASP 主动控制措施](#)作为指导，了解如何从一开始就将安全性融入设计。
- 执行威胁建模，请参阅 [OWASP 速查表：威胁建模](#)。
- 向软件架构师讲授安全设计概念与模式，并在可能的情况下要求他们将其纳入设计中。
- 与开发人员一起梳理并检查数据流。
- 将安全用户故事添加到所有其他用户故事中。

安全开发生命周期：

- 为了改进贵组织在构建应用程序和 API 时所遵循的流程，OWASP 推荐使用 [OWASP 软件保障成熟度模型 \(SAMM\)](#)。该模型可帮助组织制定并实施针对其面临的特定风险的软件安全策略。
- 为软件开发人员提供安全编码培训，并根据需要提供其他培训，帮助他们开发更健壮、更安全的应用。
- 代码审查，请参阅 [OWASP 速查表：安全代码审查](#)。
- 为开发人员提供安全工具并培训其使用方法，尤其包括静态分析、软件成分分析、密钥 (secret) 扫描，以及[基础设施即代码 \(IaC\)](#)扫描工具。
- 尽可能为开发人员建立“护栏” (guardrails)，即通过技术性防护措施引导他们做出更安全的选择。

- 构建强大且易用的安全控制并非易事。应尽可能提供安全的默认设置，并尽可能创建“铺好的路”（即让最简单的方法也成为最安全的方法，成为显而易见的首选方法）。OWASP [安全指南](#)是开发人员的良好起点，许多现代框架现在都自带标准且有效的安全控制，用于授权、验证、CSRF 防护等。
- 为您的开发人员提供与安全相关的 IDE 插件，并鼓励他们使用这些插件。
- 向他们提供秘密管理工具、许可证以及使用说明文档。
- 为他们提供一个私有的 AI 系统，理想情况下，该系统应配备一个 RAG 服务器，其中包含丰富的实用安全文档、团队编写的用于提升性能的提示信息，以及一个 MCP 服务器，用于调用贵组织选择的安全工具。教会他们如何安全地使用 AI，因为不管你喜不喜欢，他们都会这么做。

建立持续的应用安全测试：

- 测试技术功能及其与 IT 架构的集成，并协调业务测试。
- 从技术和业务角度创建“使用”和“滥用”测试用例。
- 根据内部流程、保护需求以及应用程序假定的威胁级别来管理安全测试。
- 提供安全测试工具（模糊测试器、分布式安全测试工具等）、安全的测试环境以及使用培训，或者代为进行测试，或者聘请测试人员。
- 如果您需要高度的保证，请考虑进行正式的渗透测试，以及压力测试和性能测试。
- 与开发人员合作，帮助他们根据错误报告确定需要修复的内容，并确保他们的经理给他们时间来完成这项工作。

建立现代应用程序安全计划

推出：

- 将应用程序投入运行，并根据需要从以前使用的应用程序进行迁移。
- 完成所有文档，包括变更管理数据库（CMDB）和安全架构。

运营与变革管理：

- 操作必须包含应用程序安全管理指南（例如补丁管理）。
- 提高用户的安全意识，并解决可用性与安全性之间的冲突。
- 计划和管理变更，例如迁移到应用程序或其他组件（如操作系统、中间件和库）的新版本。
- 确保所有应用都已列入清单，并记录所有重要细节。更新所有文档，包括配置管理数据库（CMDB）和安全架构、控制措施和应对措施，以及任何运行手册或项目文档。
- 对所有应用程序执行日志记录、监控和警报功能。如果缺少此功能，请添加。
- 建立高效的更新和补丁流程。
- 制定定期扫描计划（最好包括动态扫描、静态扫描、密钥扫描、IaC 扫描和软件成分分析）。
- 修复安全漏洞的服务级别协议（SLA）。
- 为员工（最好也包括客户）提供报告漏洞的途径。
- 建立一支训练有素的事件响应团队，了解软件攻击的特征和可观测性工具。
- 运行拦截或防护工具来阻止自动化攻击。
- 每年（或更频繁地）对设施进行加固。
- 至少每年进行一次渗透测试（具体取决于您的应用程序所需的安全级别）。

建立现代应用程序安全计划

- 建立流程和工具，以加强和保护您的软件供应链。
- 建立并更新业务连续性和灾难恢复计划，其中包括您最重要的应用程序以及您用于维护这些应用程序的工具。

退役系统：

- 任何必要的数​​据都应存档。所有其他数据都应安全擦除。
- 安全地停用该应用程序，包括删除未使用的帐户、角色和权限。
- 在 CMDB 中将应用程序状态设置为“已停用”。

以 OWASP Top 10 为标准

OWASP Top 10 主要是一个安全意识文档。然而，自 2003 年发布以来，这并没有阻止许多组织将其作为事实上的行业应用安全标准。如果您想将 OWASP Top 10 用作编码或测试标准，请注意它只是最低要求，仅供参考。

使用 OWASP Top 10 作为标准的一大难点在于，我们记录的是应用安全风险，而非易于测试的问题。例如，[A06:2025 不安全设计](#)超出了大多数测试形式的范围。另一个例子是：验证是否实现了“就地、在用且有效”的日志记录与监控——这通常只能通过访谈，以及抽样查看有效的事件响应案例来完成。静态代码分析工具可以检查是否缺少日志，但它可能无法判断：业务逻辑或访问控制是否对关键安全破坏事件进行了日志记录。渗透测试人员也可能只能确认他们在测试环境中触发了事件响应，而测试环境往往并不会像生产环境那样被同等方式监控。

以下是我们建议何时适合使用 OWASP Top 10：

建立现代应用程序安全计划



用例	OWASP2025 年十大漏洞	OWASP 应用安全验证标准
意识	是的	
训练	入门级	综合的
设计与建筑	偶尔	是的
编码标准	最低限度	是的
安全代码审查	最低限度	是的
同行评审清单	最低限度	是的
单元测试	偶尔	是的
集成测试	偶尔	是的
渗透测试	最低限度	是的
工具支持	最低限度	是的
安全供应链	偶尔	是的

我们鼓励任何想要采用应用程序安全标准的人使用 [OWASP 应用程序安全验证标准\(ASVS\)](#)，因为它旨在可验证和可测试，并且可以在安全开发生命周期的所有部分中使用。

对于工具供应商而言，ASVS 是唯一可接受的选择。由于 OWASP Top 10 风险中某些风险的性质（参见 [A06:2025-不安全设计](#)），工具无法全面检测、测试或防御 OWASP Top 10 风险。OWASP 不鼓励任何声称能够完全覆盖 OWASP Top 10 风险的说法，因为这根本不可能。

A01:2025 访问控制失效

背景

该漏洞在十大漏洞中稳居榜首，所有受测应用程序均被发现存在某种形式的访问控制缺陷。值得注意的 CWE 漏洞包括 CWE-200：向未经授权的攻击者泄露敏感信息、CWE-201：通过发送的数据泄露敏感信息、CWE-918：服务器端请求伪造 (SSRF) 和 CWE-352：跨站请求伪造 (CSRF)。该漏洞类别在提交的数据中出现次数最多，相关 CVE 数量也位居第二。



记分表

映射的常见弱点枚举数量 (CWE)	最大发生率	平均发生率	最大覆盖率	平均覆盖率	平均加权漏洞利用	平均加权影响	总发生次数	CVE 总数
40	20.15%	3.74%	100.00%	42.93%	7.04	3.84	1,839,701	32,654

描述

访问控制通过强制执行策略，防止用户超出其权限范围进行操作。访问控制失效通常会导致未经授权的信息泄露、数据被修改或销毁，或者用户执行超出其权限范围的业务功能。常见的访问控制漏洞包括：

违反了最小权限原则（通常称为默认拒绝），即只应授予特定能力、角色或用户访问权限，但实际上任何人都可以访问。

A01:2025 访问控制失效

通过修改 URL (篡改参数或强制浏览)、内部应用程序状态或 HTML 页面, 或者使用修改 API 请求的攻击工具来绕过访问控制检查。

允许通过提供唯一标识符来查看或编辑他人的帐户 (不安全的直接对象引用)

一个易于使用的 API, 但缺少 POST、PUT 和 DELETE 的访问控制。

权限提升。指在未登录的情况下以用户身份行事, 或获取超出已登录用户应有的权限 (例如管理员权限) 。

元数据操纵, 例如重放或篡改 JSON Web Token (JWT) 访问控制令牌、操纵 cookie 或隐藏字段以提升权限, 或滥用 JWT 失效。

CORS 配置错误会导致 API 可以从未经授权或不受信任的来源访问。

强制未认证用户浏览 (猜测 URL) 到已认证页面, 或强制标准用户浏览 (猜测 URL) 到特权页面。

如何预防

访问控制只有在受信任的服务器端代码或无服务器 API 中实现时才有效, 攻击者无法修改访问控制检查或元数据。

- 除公共资源外, 默认拒绝。
- 只需实施一次访问控制机制, 即可在整个应用程序中重复使用, 包括最大限度地减少跨域资源共享 (CORS) 的使用。
- 模型访问控制应该强制执行记录所有权, 而不是允许用户创建、读取、更新或删除任何记录。
- 独特的应用程序业务限制要求应通过领域模型来强制执行。

A01:2025 访问控制失效

- 禁用 Web 服务器目录列表，并确保 Web 根目录中不存在文件元数据（例如 .git）和备份文件。
- 记录访问控制失败情况，并在适当时（例如，重复失败）向管理员发出警报。
- 对 API 和控制器访问实施速率限制，以最大限度地减少自动化攻击工具造成的损害。
- 用户注销后，服务器应使有状态会话标识符失效。无状态 JWT 令牌的有效期应尽可能短，以最大程度地减少攻击者可利用的机会。对于有效期较长的 JWT，请考虑使用刷新令牌并遵循 OAuth 标准来撤销访问权限。
- 使用成熟的工具包或模式，提供简单、声明式的访问控制。

开发人员和质量保证人员应在单元测试和集成测试中加入功能访问控制。

攻击场景示例

场景 1：应用程序在访问帐户信息的 SQL 调用中使用未经验证的数据：

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

攻击者只需修改浏览器的“acct”参数，即可发送任何所需的账号。如果验证不当，攻击者就能访问任何用户的账号。

```
https://example.com/app/accountInfo?acct=notmyacct
```

场景 2：攻击者只需强制浏览器访问特定 URL 即可。访问管理页面需要管理员权限。

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

A01:2025 访问控制失效

如果未经身份验证的用户可以访问任一页面，那就是一个漏洞。如果非管理员用户可以访问管理员页面，这也是一个漏洞。

场景 3：应用程序将所有访问控制都放在前端。虽然攻击者

```
https://example.com/app/admin_getappInfo
```

由于浏览器中运行的 JavaScript 代码而无法直接访问前端，但他们可以简单地执行以下操作：

```
$ curl https://example.com/app/admin_getappInfo
```

从命令行。

参考文献

- [OWASP 主动控制：C1：实施访问控制](#)
- [OWASP 应用安全验证标准：V8 授权](#)
- [OWASP 测试指南：授权测试](#)
- [OWASP 速查表：授权](#)
- [PortSwigger：利用 CORS 配置错误](#)
- [OAuth：撤销访问权限](#)

CWE 映射列表

- [CWE-22 路径名对受限目录的限制不当（“路径遍历”）](#)
- [CWE-23 相对路径遍历](#)
- [CWE-36 绝对路径遍历](#)

A01:2025 访问控制失效

- CWE-59 文件访问前链接解析不正确（“链接跟踪”）
- CWE-61 UNIX 符号链接（Symlink）如下
- CWE-65 Windows 硬链接
- CWE-200 向未经授权的人员泄露敏感信息
- CWE-201 通过发送数据泄露敏感信息
- CWE-219 在 Web 根目录下存储敏感数据文件
- CWE-276 默认权限错误
- CWE-281 许可保存不当
- CWE-282 所有权管理不当
- CWE-283 未经核实的所有权
- CWE-284 访问控制不当
- CWE-285 授权不当
- CWE-352 跨站请求伪造（CSRF）
- CWE-359 向未经授权的人员泄露私人个人信息
- CWE-377 不安全的临时文件
- CWE-379 在权限不安全的目录中创建临时文件
- CWE-402 私人资源向新领域的传输（“资源泄漏”）
- CWE-424 备用路径保护不当
- CWE-425 直接请求（“强制浏览”）

A01:2025 访问控制失效

- CWE-441 非预期代理人或中间人（“受骗代理人”）
- CWE-497 敏感系统信息暴露于未经授权的控制范围
- CWE-538 将敏感信息插入外部可访问的文件或目录
- CWE-540 源代码中包含敏感信息
- CWE-548 通过目录列表泄露信息
- CWE-552 外部方可访问的文件或目录
- CWE-566 通过用户控制的 SQL 主键绕过授权
- CWE-601 URL 重定向到不受信任的站点（“开放重定向”）
- CWE-615 源代码注释中包含敏感信息
- CWE-639 通过用户控制密钥绕过授权
- CWE-668 资源暴露于错误领域
- CWE-732 关键资源权限分配错误
- CWE-749 暴露危险方法或功能
- CWE-862 缺少授权
- CWE-863 授权错误
- CWE-918 服务器端请求伪造 (SSRF)
- CWE-922 敏感信息存储不安全
- CWE-1275 敏感 Cookie 的 SameSite 属性不正确

A02:2025 安全配置错误

背景

较上一版排名第五有所上升，本次测试中所有应用程序均发现存在某种形式的配置错误，平均发生率为 3.00%，且该风险类别中常见缺陷枚举 (CWE) 的出现次数超过 71.9 万次。随着高度可配置软件的普及，该类别排名上升并不令人意外。值得注意的 CWE 包括 CWE-16 配置错误和 CWE-611 XML 外部实体引用 (XXE) 限制不当。



记分表

映射的常见弱点枚举数量 (CWE)	最大发生率	平均发生率	最大覆盖率	平均覆盖率	平均加权漏洞利用	平均加权影响	总发生次数	CVE 总数
16	27.70%	3.00%	100.00%	52.35%	7.96	3.97	719,084	1,375

描述

安全配置错误是指系统、应用程序或云服务从安全角度来看设置不正确，从而造成漏洞。

如果出现以下情况，该应用程序可能存在漏洞：

- 它缺少应用程序堆栈任何部分的适当安全加固，或者云服务上的权限配置不当。
- 启用或安装了不必要的功能（例如，不必要的端口、服务、页面、帐户、测试框架或权限）。
- 默认账户及其密码仍然启用且未更改。

- 缺乏用于拦截过多错误消息的集中配置。错误处理会向用户显示堆栈跟踪或其他过于详细的错误消息。
- 对于升级后的系统，最新的安全功能被禁用或未进行安全配置。
- 过度重视向后兼容性导致配置不安全。
- 应用程序服务器、应用程序框架（例如 Struts、Spring、ASP.NET）、库、数据库等中的安全设置未设置为安全值。
- 服务器未发送安全标头或指令，或者未将其设置为安全值。

如果没有协调一致、可重复的应用安全配置加固流程，系统将面临更高的风险。

如何预防

应实施安全的安装流程，包括：

- 可重复的加固流程能够快速便捷地部署另一个经过适当安全防护的环境。开发、测试和生产环境的配置应完全相同，但每个环境使用不同的凭证。此流程应实现自动化，以最大限度地减少设置新安全环境所需的工作量。
- 一个极简平台，不包含任何不必要的功能、组件、文档或示例。移除或不要安装未使用的功能和框架。
- 一项任务是审查并更新所有安全说明、更新和补丁的相应配置，作为补丁管理流程的一部分（参见 [A03 软件供应链失效](#)）。审查云存储权限（例如，S3 存储桶权限）。

- 分段式应用程序架构通过分段、容器化或云安全组（ACL）在组件或租户之间提供有效且安全的隔离。
- 向客户端发送安全指令，例如安全标头。
- 用于验证所有环境中配置和设置的有效性的自动化流程。
- 主动添加中央配置以拦截过多的错误消息作为备份。
- 如果这些验证不是自动化的，至少应该每年进行一次人工验证。
- 使用底层平台提供的身份联合、短期凭证或基于角色的访问机制，而不是在代码、配置文件或管道中嵌入静态密钥或秘密。

攻击场景示例

场景 1：应用服务器预装了一些未从生产服务器移除的示例应用程序。这些示例应用程序存在已知的安全漏洞，攻击者可以利用这些漏洞入侵服务器。假设其中一个应用程序是管理控制台，并且默认账户未被更改。在这种情况下，攻击者可以使用默认密码登录并控制服务器。

场景 2：服务器上的目录列表功能未禁用。攻击者发现他们可以轻松列出目录。攻击者找到并下载了已编译的 Java 类，然后对其进行反编译和逆向工程以查看代码。攻击者随后发现应用程序中存在严重的访问控制漏洞。

场景 3：应用服务器的配置允许向用户返回详细的错误信息，例如堆栈跟踪。这可能会暴露敏感信息或潜在缺陷，例如已知存在漏洞的组件版本。

场景 4：云服务提供商（CSP）默认将共享权限开放给互联网。这使得存储在云存储中的敏感数据可以被访问。

参考

- [OWASP 测试指南：配置管理](#)
- [OWASP 测试指南：错误代码测试](#)
- [应用程序安全验证标准 V13 配置](#)
- [NIST 通用服务器加固指南](#)
- [CIS 安全配置指南/基准](#)
- [Amazon S3 存储桶发现和枚举](#)
- [ScienceDirect：安全配置错误](#)

已映射 CWE 列表

- [CWE-5 J2EE 配置错误：数据传输未加密](#)
- [CWE-11 ASP.NET 配置错误：创建调试二进制文件](#)
- [CWE-13 ASP.NET 配置错误：配置文件中的密码](#)
- [CWE-15 系统或配置设置的外部控制](#)
- [CWE-16 配置](#)
- [CWE-260 配置文件中的密码](#)
- [CWE-315 以明文形式在 Cookie 中存储敏感信息](#)
- [CWE-489 活动调试代码](#)
- [CWE-526 通过环境因素暴露敏感信息](#)

A02:2025 安全配置错误

- [CWE-547 使用硬编码的安全相关常量](#)
- [CWE-611 XML 外部实体引用限制不当](#)
- [CWE-614 HTTPS 会话中缺少“安全”属性的敏感 Cookie](#)
- [CWE-776 DTD 中递归实体引用的限制不当（“XML 实体扩展”）](#)
- [CWE-942 允许跨域策略，但不受信任域的限制](#)
- [CWE-1004 缺少“HttpOnly”标志的敏感 Cookie](#)
- [CWE-1174 ASP.NET 配置错误：模型验证不当](#)

A03:2025 软件供应链失效

背景

在十大社区调查中，该风险位居榜首，50% 的受访者将其评为第一。自 2013 年首次以“A9 – 使用已知漏洞的组件”之名出现在十大风险榜单以来，该风险的范围已扩大到所有供应链故障，而不仅仅是涉及已知漏洞的故障。尽管范围扩大，但供应链故障的识别仍然面临挑战，目前只有 11 个常见漏洞披露 (CVE) 拥有相关的 CWE。然而，在已提交的数据中，经测试和报告，该类别的平均发生率最高，达到 5.19%。相关的 CWE 包括：CWE-477：使用过时的功能、CWE-1104：使用未维护的第三方组件、CWE-1329：依赖不可更新的组件以及 CWE-1395：依赖易受攻击的第三方组件。



记分表

映射的常见弱点枚举数量 (CWE)	最大发生率	平均发生率	最大覆盖率	平均覆盖率	平均加权漏洞利用	平均加权影响	总发生次数	CVE 总数
6	9.56%	5.72%	65.42%	27.47%	8.17	5.23	215,248	11

描述

软件供应链失效是指软件构建、分发或更新过程中出现的中断或其他问题。它们通常是由系统所依赖的第三方代码、工具或其他依赖项中的漏洞或恶意更改引起的。

如果您有以下情况，您可能容易受到攻击：

- 您没有仔细跟踪所有使用的组件（包括客户端和服务端组件）的版本。这包括您直接使用的组件以及嵌套的（传递性）依赖项。
- 该软件存在漏洞、不受支持或已过时。这包括操作系统、Web/应用服务器、数据库管理系统（DBMS）、应用程序、API 及其所有组件、运行时环境和库。
- 您没有定期扫描漏洞，也没有订阅与您使用的组件相关的安全公告。
- 你们的供应链中没有变更管理流程或变更跟踪机制，包括跟踪 IDE、IDE 扩展和更新、组织代码库的变更、沙箱、镜像和库库、工件的创建和存储方式等等。供应链的每个环节都应该记录下来，尤其是变更。
- 你们尚未加强供应链的每个环节，尤其没有重点加强访问控制和最小权限原则的应用。
- 你们的供应链系统缺乏职责分离机制。任何人都不应该在没有其他人监督的情况下编写代码并将其部署到生产环境。
- 在技术栈的任何部分中使用来自不受信任来源的组件，都可能对生产环境造成影响。
- 您未能以基于风险且及时的方式修复或升级底层平台、框架和依赖项。这种情况通常发生在补丁程序作为变更控制下的月度或季度任务的环境中，导致组织在修复漏洞之前面临数天甚至数月的不必要风险暴露。
- 软件开发人员不会测试更新、升级或补丁修复的库的兼容性。
- 您没有确保系统每个部分的配置安全（参见 [A02:2025-安全配置错误](#)）。
- 您的 CI/CD 流水线的安全性比它构建和部署的系统要弱，尤其是在它很复杂的情况下。

如何预防

应该建立一套补丁管理流程，以：

- 集中生成和管理整个软件的软件物料清单 (SBOM)。
- 不仅要跟踪你的直接依赖项，还要跟踪它们的（传递）依赖项，依此类推。
- 通过移除未使用的依赖项、不必要的功能、组件、文件和文档来减少攻击面。
- 使用 OWASP Dependency Track、OWASP Dependency Check、retire.js 等工具，持续盘点客户端和服务端组件（例如框架、库）及其依赖项的版本。
- 持续监控诸如通用漏洞披露 (CVE)、国家漏洞数据库 (NVD) 和[开源漏洞 \(OSV\)](#)等来源，以发现您使用的组件中的漏洞。使用软件成分分析、软件供应链或专注于安全的软件物料清单 (SBOM) 工具来实现流程自动化。订阅与您使用的组件相关的安全漏洞警报。
- 仅通过安全链接从官方（可信）来源获取组件。优先使用签名软件包，以降低包含已修改、恶意组件的风险（参见 [A08:2025-软件和数据完整性失效](#)）。
- 谨慎选择要使用的依赖项版本，并且仅在需要时才进行升级。
- 监控那些不再维护或未针对旧版本发布安全补丁的库和组件。如果无法补丁修复，请考虑迁移到替代方案。如果迁移也不可行，请考虑部署虚拟补丁来监控、检测或防御已发现的问题。
- 定期更新您的 CI/CD、IDE 和任何其他开发人员工具。
- 避免同时向所有系统部署更新。使用分阶段发布或金丝雀部署来限制风险，以防受信任的供应商遭到入侵。

A03:2025 软件供应链失效

应该建立变更管理流程或跟踪系统，以跟踪以下方面的变更：

- CI/CD 设置（所有构建工具和流水线）
- 代码库
- 沙盒区域
- 开发者 IDE
- SBOM 工具和创建的工件
- 日志系统和日志
- 第三方集成，例如 SaaS
- 制品仓库
- 容器注册表

加强以下系统，包括启用多因素身份验证 (MFA) 和锁定身份与访问管理 (IAM)：

- 你的代码仓库（包括不提交机密信息、保护分支、备份）
- 开发人员工作站（定期补丁、多因素身份验证、监控等）
- 您的构建服务器和 CI/CD（职责分离、访问控制、签名构建、环境范围密钥、防篡改日志等）
- 您的工件（通过溯源、签名和时间戳确保完整性，推广工件而不是为每个环境重新构建，确保构建是不可变的）
- 基础设施即代码（像所有代码一样进行管理，包括使用 PR 和版本控制）

每个组织都必须确保制定持续的计划，以便在应用程序或产品组合的整个生命周期内进行监控、分类和应用更新或配置更改。

攻击场景示例

场景 1：受信任的供应商被恶意软件入侵，导致您升级时计算机系统受到感染。最著名的例子可能是：

- 2019 年 SolarWinds 系统遭到入侵，导致约 18,000 家机构的数据泄露。
<https://www.npr.org/2021/04/16/985439655/a-worst-nightmare-cyberattack-the-untold-story-of-the-solarwinds-hack>

场景 2：受信任的供应商遭到入侵，以至于它只在特定条件下才会表现出恶意行为。

- 2025 年 Bybit 钱包被盗 15 亿美元，原因是钱包软件中的供应链攻击，该攻击仅在目标钱包被使用时执行。

场景 3：2025 年的 Shai-Hulud 供应链攻击是首个成功实现自我传播的 npm 蠕虫病毒。攻击者植入了常用软件包的恶意版本，这些恶意版本利用安装后脚本收集敏感数据并将其泄露到公共 GitHub 代码库。该恶意软件还会检测受害者环境中的 npm 令牌，并自动使用这些令牌推送任何可访问软件包的恶意版本。在被 npm 系统拦截之前，该蠕虫病毒已感染了超过 500 个软件包版本。此次供应链攻击技术先进、传播迅速且破坏性极强，并且通过攻击开发者机器，表明开发者本身也成为了供应链攻击的主要目标。

场景 4：组件通常以与应用程序本身相同的权限运行，因此任何组件中的缺陷都可能造成严重后果。此类缺陷可能是意外的（例如，编码错误），也可能是故意的（例如，组件中的后门）。已发现的一些可利用的组件漏洞示例包括：

- CVE-2017-5638 是一个 Struts 2 远程代码执行漏洞，允许在服务器上执行任意代码，该漏洞被认为是造成重大安全漏洞的原因。
- CVE-2021-44228 (“Log4Shell”) 是 Apache Log4j 的一个远程代码执行零日漏洞，已被指责为勒索软件、加密货币挖矿和其他攻击活动的罪魁祸首。

参考文献

- [OWASP 应用安全验证标准：V15 安全编码和架构](#)
- [OWASP 速查表系列：依赖关系图 SBOM](#)
- [OWASP 速查表系列：漏洞依赖管理](#)
- [OWASP 依赖项跟踪](#)
- [OWASP CycloneDX](#)
- [OWASP 应用安全验证标准：V1 架构、设计和威胁建模](#)
- [OWASP 依赖项检查（适用于 Java 和 .NET 库）](#)
- [OWASP 测试指南 - 地图应用程序架构 \(OTG-INFO-010\)](#)
- [OWASP 虚拟补丁最佳实践](#)
- [图书馆安全隐患的不幸现实](#)
- [MITRE 通用漏洞披露 \(CVE\) 搜索](#)

A03:2025 软件供应链失效

- [国家漏洞数据库 \(NVD\)](#)
- [Retire.js](#) 用于检测已知的易受攻击的 JavaScript 库
- [GitHub 咨询数据库](#)
- [Ruby 库安全咨询数据库和工具](#)
- [SAFECode 软件完整性控制 \(PDF\)](#)
- [玻璃虫供应链攻击](#)
- [幻影渡鸦供应链攻击活动](#)

CWE 映射列表

- CWE-447 使用过时功能
- CWE-1035 2017 Top 10 A9: 使用已知漏洞的组件
- CWE-1104 使用未经维护的第三方组件
- CWE-1329 依赖于不可更新的组件
- CWE-1357 依赖于不可信组件
- CWE-1395 依赖于易受攻击的第三方组件

A04:2025 加密机制失效

背景

排名下降两位至第 4 位，这一弱点主要集中在与缺乏加密、加密强度不足、加密密钥泄露及相关错误相关的故障上。该风险中最常见的三个常见弱点枚举（CWE）都涉及使用弱伪随机数生成器：CWE-327 使用有缺陷或风险的加密算法、CWE-331 熵不足、CWE-1241 在随机数生成器中使用可预测算法以及 CWE-338 使用加密强度较弱的伪随机数生成器（PRNG）。



记分表

映射的常见弱点枚举数量 (CWE)	最大发生率	平均发生率	最大覆盖率	平均覆盖率	平均加权漏洞利用	平均加权影响	总发生次数	CVE 总数
32	13.77%	3.80%	100.00%	47.74%	7.23	3.90	1,665,348	2,185

描述

一般来说，所有传输中的数据都应在传输层（OSI 第四层）加密。以往的瓶颈，例如 CPU 性能和私钥/证书管理，现在已通过 CPU 指令集（例如 AES 支持）得到解决，这些指令集旨在加速加密；而 Let's Encrypt.org 等服务则简化了私钥和证书管理；此外，主流云厂商还为其特定平台提供更加紧密集成的证书管理服务。

除了确保传输层安全之外，确定哪些数据需要静态加密，以及哪些数据需要在传输过程中（应用层，OSI 第 7 层）进行额外加密也至关重要。例如，密码、信用卡号、健康记录、个人信息和

A04:2025 加密机制失效

商业机密都需要额外保护，尤其是在这些数据受隐私法（例如欧盟的《通用数据保护条例》(GDPR)）或支付卡行业数据安全标准（PCI DSS）等法规约束的情况下。对于所有此类数据：

- 是否存在默认使用或旧代码中使用的过时或弱加密算法或协议？
- 是否正在使用默认加密密钥？是否生成了弱加密密钥？密钥是否被重复使用？或者是否缺少适当的密钥管理和轮换？
- 加密密钥是否已提交到源代码库中？
- 是否未强制执行加密，例如，是否缺少任何 HTTP 标头（浏览器）安全指令或标头？
- 收到的服务器证书和信任链是否已正确验证？
- 初始化向量是否被忽略、重用，或者生成的向量是否足够安全以适应加密操作模式？是否使用了不安全的操作模式（例如 ECB）？是否在更适合使用认证加密的情况下使用了加密？
- 在没有基于密码的密钥派生函数的情况下，密码是否被用作加密密钥？
- 是否使用了不符合加密要求的随机数？即使选择了正确的函数，开发者是否需要为其设置种子？如果不需要，开发者是否用缺乏足够熵/不可预测性的种子覆盖了函数内置的强种子设置功能？
- 是否使用了已弃用的哈希函数（例如 MD5 或 SHA1），或者在需要加密哈希函数时是否使用了非加密哈希函数？
- 加密错误信息或侧信道信息是否可被利用，例如以填充预言机攻击的形式？
- 加密算法可以被降级或绕过吗？

参见 ASVS 参考资料：密码学（V11）、安全通信（V12）和数据保护（V14）。

如何预防

至少要完成以下步骤，并参考相关资料：

- 对应用程序处理、存储或传输的数据进行分类和标记。根据隐私法、监管要求或业务需求，识别哪些数据属于敏感数据。
- 将您最敏感的密钥存储在硬件或云端 HSM 中。
- 尽可能使用可信度高的加密算法实现。
- 不要不必要地存储敏感数据。应尽快丢弃，或使用符合 PCI DSS 标准的令牌化技术，甚至进行数据截断。不保留的数据不会被窃取。
- 务必对所有静态敏感数据进行加密。
- 确保采用最新且强大的标准算法、协议和密钥；使用适当的密钥管理。
- 仅使用 TLS 1.2 或更高版本协议加密所有传输中的数据，并采用前向保密 (FS) 密码套件，放弃对密码块链接 (CBC) 密码套件的支持，支持量子密钥变更算法。对于 HTTPS，强制使用 HTTP 严格传输安全 (HSTS) 进行加密。使用工具检查所有设置。
- 禁用包含敏感数据的响应的缓存。这包括 CDN、Web 服务器以及任何应用程序缓存（例如：Redis）中的缓存。
- 根据数据分类应用必要的安全控制措施。
- 请勿使用未加密的协议，例如 FTP 和 STARTTLS。避免使用 SMTP 传输机密数据。

- 使用具有工作因子（延迟因子）的强自适应加盐哈希函数（例如 Argon2、yescrypt、scrypt 或 PBKDF2-HMAC-SHA-512）来存储密码。对于使用 bcrypt 的旧系统，请参阅 [OWASP 速查表：密码存储](#)，获取更多建议。
- 初始化向量的选择必须与操作模式相匹配。这可能意味着需要使用密码学安全的伪随机数生成器（CSPRNG）。对于需要随机数（nonce）的模式，初始化向量（IV）则不需要 CSPRNG。在任何情况下，对于固定的密钥，初始化向量都绝不能重复使用。
- 始终使用经过身份验证的加密，而不仅仅是加密。
- 密钥应通过加密方式随机生成，并以字节数组的形式存储在内存中。如果使用密码，则必须通过适当的基于密码的密钥派生函数将其转换为密钥。
- 确保在适当情况下使用加密随机数，并且其种子不是以可预测的方式或低熵值生成的。大多数现代 API 不需要开发者为 CSPRNG 设置种子即可保证安全。
- 避免使用已弃用的加密函数、块构建方法和填充方案，例如 MD5、SHA1、密码块链接模式（CBC）、PKCS 1 号 v1.5。
- 请安全专家使用专门设计的工具或两者结合的方式审查设置和配置，以确保其符合安全要求。
- 您现在需要为后量子密码学（PQC）做好准备，请参阅参考资料（ENISA），以便高风险系统最迟在 2030 年底前安全无虞。

攻击场景示例

场景 1：网站未对所有页面启用或强制使用 TLS，或仅支持弱加密。攻击者监控网络流量（例如，在不安全的无线网络中），将连接从 HTTPS 降级为 HTTP，拦截请求并窃取用户的会话

A04:2025 加密机制失效

cookie。攻击者随后重放此 cookie 并劫持用户的（已认证的）会话，从而访问或修改用户的私人数据。此外，攻击者还可以篡改所有传输的数据，例如，更改汇款的收款人信息。

场景 2：密码数据库使用未加盐或简单的哈希算法存储所有用户的密码。文件上传漏洞使得攻击者能够获取密码数据库。所有未加盐的哈希值都可以通过预先计算的哈希值的彩虹表暴露出来。即使加盐，由简单或快速哈希函数生成的哈希值也可能被 GPU 破解。

参考文献

- [OWASP 主动控制：C2：使用加密技术保护数据](#)
- [OWASP 应用程序安全验证标准 \(ASVS\)：V11、12、14](#)
- [OWASP 速查表：传输层保护](#)
- [OWASP 速查表：用户隐私保护](#)
- [OWASP 速查表：密码存储](#)
- [OWASP 速查表：加密存储](#)
- [OWASP 速查表：HSTS](#)
- [OWASP 测试指南：弱加密测试](#)
- [ENISA：向后量子密码学过渡的协调实施路线图](#)
- [美国国家标准与技术研究院发布首批三项最终版后量子加密标准](#)

CWE 映射列表

- [CWE-261 密码编码强度不足](#)

A04:2025 加密机制失效

- [CWE-296 证书信任链遵循不当](#)
- [CWE-319 明文传输敏感信息](#)
- [CWE-320 密钥管理错误（禁止）](#)
- [CWE-321 使用硬编码加密密钥](#)
- [CWE-322 密钥交换无需实体认证](#)
- [CWE-323 加密中重用随机数和密钥对](#)
- [CWE-324 使用过期密钥](#)
- [CWE-325 缺少必要的加密步骤](#)
- [CWE-326 加密强度不足](#)
- [CWE-327 使用有缺陷或存在风险的加密算法](#)
- [CWE-328 可逆单向哈希](#)
- [CWE-329 未为 CBC 模式使用随机初始化向量](#)
- [CWE-330 使用随机性不足的值](#)
- [CWE-331 熵不足](#)
- [CWE-332 伪随机数生成器中的熵不足](#)
- [CWE-334 小空间随机值](#)
- [CWE-335 伪随机数生成器 \(PRNG\) 中种子使用错误](#)
- [CWE-336 伪随机数生成器 \(PRNG\) 中的相同种子](#)

A04:2025 加密机制失效

- [CWE-337](#) 伪随机数生成器 (PRNG) 中的可预测种子
- [CWE-338](#) 使用密码学上薄弱的伪随机数生成器 (PRNG)
- [CWE-340](#) 生成可预测的数字或标识符
- [CWE-342](#) 根据先前值预测精确值
- [CWE-347](#) 密码签名验证不当
- [CWE-523](#) 凭证传输未受保护
- [CWE-757](#) 协商过程中选择安全性较低的算法（“算法降级”）
- [CWE-759](#) 使用无盐单向哈希
- [CWE-760](#) 使用可预测盐值的单向哈希
- [CWE-780](#) 不使用 OAEP 的 RSA 算法
- [CWE-916](#) 使用计算复杂度不足的密码哈希算法
- [CWE-1240](#) 使用存在风险的实现方式的密码学原语
- [CWE-1241](#) 随机数生成器中可预测算法的使用

背景



注入攻击的排名从第 3 位下降到第 5 位，但其相对于 A04:2025-加密机制失效和 A06:2025-不安全设计的位置保持不变。注入攻击是测试最多的类别之一，所有应用程序都接受了某种形式的注入攻击测试。它是所有类别中 CVE 数量最多的，共有 37 个 CWE。注入攻击包括跨站脚本攻击（高频/低影响），拥有超过 3 万个 CVE，以及 SQL 注入攻击（低频/高影响），拥有超过 1.4 万个 CVE。CWE-79 “网页生成期间输入处理不当”（即跨站脚本攻击）的大量 CVE 报告拉低了该类别的平均加权影响。

记分表

映射的常见弱点枚举数量 (CWE)	最大发生率	平均发生率	最大覆盖率	平均覆盖率	平均加权漏洞利用	平均加权影响	总发生次数	CVE 总数
37	13.77%	3.08%	100.00%	42.93%	7.15	4.32	1,404,249	62,445

描述

注入漏洞是一种应用程序缺陷，它允许将不受信任的用户输入发送到解释器（例如浏览器、数据库、命令行），并导致解释器将该输入的一部分作为命令执行。

应用程序在以下情况下容易受到攻击：

- 用户提供的数据未经应用程序验证、过滤或净化。

- 动态查询或不带上下文感知转义的非参数化调用直接在解释器中使用。
- 在对象关系映射 (ORM) 搜索参数中使用未经清理的数据来提取其他敏感记录。
- 潜在的恶意数据被直接使用或拼接在一起。SQL 或命令中包含动态查询、命令或存储过程中的结构和恶意数据。

一些常见的注入攻击包括 SQL 注入、NoSQL 注入、操作系统命令注入、对象关系映射 (ORM) 注入、LDAP 注入以及表达式语言 (EL) 或对象图导航库 (OGNL) 注入。所有解释器中的注入原理都相同。最佳的检测方法是结合源代码审查和自动化测试 (包括模糊测试)，对所有参数、标头、URL、Cookie、JSON、SOAP 和 XML 数据输入进行测试。在 CI/CD 流水线中添加静态应用安全测试 (SAST)、动态应用安全测试 (DAST) 和交互式应用安全测试 (IAST) 工具也有助于在生产部署前识别注入漏洞。

另一类相关的注入漏洞在 LLM 中变得很常见。OWASP [LLM Top 10](#) 中单独讨论了这些漏洞，特别是 [LLM01:2025 提示注入](#)。

如何预防

防止注入攻击的最佳方法是将数据与命令和查询分开：

- 首选方案是使用安全的 API，完全避免使用解释器，提供参数化接口，或迁移到对象关系映射工具 (ORM)。注意：即使使用了参数化，如果 PL/SQL 或 T-SQL 将查询和数据连接起来，或者使用 EXECUTE IMMEDIATE 或 exec() 执行恶意数据，存储过程仍然可能引入 SQL 注入。

当无法将数据与命令分离时，可以使用以下技术来降低威胁。

- 使用服务器端输入验证。但这并非万无一失，因为许多应用程序需要特殊字符，例如文本区域或移动应用程序的 API。
- 对于任何残留的动态查询，请使用该解释器的特定转义语法转义特殊字符。注意：SQL 结构（例如表名、列名等）无法转义，因此用户提供的结构名称存在风险。这是报表编写软件中常见的问题。

警告：这些技术涉及解析和转义复杂的字符串，因此容易出错，并且在底层系统发生微小变化时不够稳健。

攻击场景示例

场景 1：应用程序在构建以下存在漏洞的 SQL 调用时使用了不受信任的数据：

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

攻击者修改浏览器中的 “id” 参数值，使其发送：

```
' OR '1'='1
```

例如：

```
http://example.com/app/accountView?id=' OR '1'='1
```

这会改变查询的含义，使其返回账户表中的所有记录。更危险的攻击可能会修改或删除数据，甚至调用存储过程。

场景 2：应用程序对框架的盲目信任可能导致查询仍然存在漏洞。例如，Hibernate 查询语言 (HQL)：

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

攻击者提供：

```
' OR custID IS NOT NULL OR custID='
```

这会绕过过滤器并返回所有帐户。虽然 HQL 比原始 SQL 包含的危险函数较少，但当用户输入被拼接成查询语句时，仍然允许未经授权的数据访问。

场景 3：应用程序直接将用户输入传递给操作系统命令：

```
String cmd = "nslookup " + request.getParameter("domain");  
Runtime.getRuntime().exec(cmd);
```

攻击者准备

```
example.com; cat /etc/passwd
```

在服务器上执行任意命令。

参考文献

- [OWASP 主动控制：安全数据库访问](#)
- [OWASP ASVS：V5 输入验证和编码](#)
- [OWASP 测试指南：SQL 注入、命令注入和 ORM 注入](#)
- [OWASP 速查表：注入预防](#)

- [OWASP 速查表: SQL 注入防护](#)
- [OWASP 速查表: Java 中的注入防御](#)
- [OWASP 速查表: 查询参数化](#)
- [OWASP Web 应用程序自动化威胁 – OAT-014](#)
- [PortSwigger: 服务器端模板注入](#)
- [超棒的模糊测试: 模糊测试资源列表](#)

CWE 映射列表

- [CWE-20 输入验证不当](#)
- [CWE-74 下游组件使用的输出中特殊元素的中和不当 \(“注入” \)](#)
- [CWE-76 等效特殊元素的中和不当](#)
- [CWE-77 指令 \(“指令注入” \) 中使用的特殊元素的不当中和](#)
- [CWE-78 操作系统命令中使用的特殊元素的未正确中和 \(“操作系统命令注入” \)](#)
- [CWE-79 网页生成过程中输入未正确中和 \(“跨站脚本攻击” \)](#)
- [CWE-80 网页中脚本相关 HTML 标签的异常中和 \(基本 XSS \)](#)
- [CWE-83 网页属性中脚本的不正确中和](#)
- [CWE-86 网页标识符中无效字符处理不当](#)
- [CWE-88 命令中参数分隔符的不正确中和 \(“参数注入” \)](#)
- [CWE-89 SQL 命令中使用的特殊元素未正确中和 \(“SQL 注入” \)](#)

- [CWE-90 LDAP 查询中使用的特殊元素未正确中和 \(“LDAP 注入” \)](#)
- [CWE-91 XML 注入 \(又称盲注 XPath 注入 \)](#)
- [CWE-93 CRLF 序列中和不当 \(“CRLF 注入” \)](#)
- [CWE-94 代码生成控制不当 \(“代码注入” \)](#)
- [CWE-95 动态求值代码中指令的不当中和 \(“求值注入” \)](#)
- [CWE-96 静态保存代码中指令的不当中和 \(“静态代码注入” \)](#)
- [CWE-97 网页中服务器端包含 \(SSI\) 的中和不当](#)
- [CWE-98 PHP 程序中 include/require 语句的文件名控制不当 \(“PHP 远程文件包含” \)](#)
- [CWE-99 资源标识符控制不当 \(“资源注入” \)](#)
- [CWE-103 Struts: validate\(\) 方法定义不完整](#)
- [CWE-104 Struts: 表单 Bean 未扩展验证类](#)
- [CWE-112 缺少 XML 验证](#)
- [CWE-113 HTTP 标头中 CRLF 序列处理不当 \(“HTTP 响应拆分” \)](#)
- [CWE-114 过程控制](#)
- [CWE-115 输出结果误解](#)
- [CWE-116 输出编码或转义不当](#)
- [CWE-129 数组索引验证不当](#)
- [CWE-159 特殊元素无效使用处理不当](#)

A05:2025 注入

- CWE-470 使用外部控制输入选择类或代码（“不安全反射”）
- CWE-493 缺少最终修饰符的关键公共变量
- CWE-500 公共静态场未标记为最终版
- CWE-564 SQL 注入：Hibernate
- CWE-610 外部控制引用另一领域的资源
- CWE-643 XPath 表达式中数据中和不当（“XPath 注入”）
- CWE-644 脚本语法中和 HTTP 标头不正确
- CWE-917 表达式语言语句中使用的特殊元素的不正确中和（“表达式语言注入”）

A06:2025 不安全设计

背景

由于 [A02:2025-安全配置错误](#) 和 [A03:2025-软件供应链失效](#) 跃升至其之前，“不安全设计”在排名中从第 4 位下滑两位至第 6 位。该类别于 2021 年引入，安全专家看到行业在威胁建模方面有了显著改进，并且对安全设计更加重视。该类别聚焦于与设计 and 架构缺陷相关的风险，同时强调应采用威胁建模、安全设计模式和参考架构进行开发。这包括应用程序业务逻辑中的缺陷，例如未定义应用程序内部不需要或意外的状态变化。作为一个社区，安全专家需要超越编码领域的“左移”，转向编码前的活动，如需求编写和应用程序设计，这些对于“设计即安全”原则至关重要（例如，参见[建立现代应用安全计划：规划和设计阶段](#)）。值得注意的通用弱点枚举 (CWE) 包括 CWE-256: 凭证的未受保护存储、CWE-269: 不当权限管理、CWE-434: 危险类型文件的无限制上传、CWE-501: 信任边界违规以及 CWE-522: 凭证保护不足。



评分表

映射的通用弱点枚举 (CWE)	最大发生率	平均发生率	最大覆盖率	平均覆盖率	平均加权漏洞利用	平均加权影响	总发生次数	CVE 总数
39	22.18%	1.86%	88.76%	35.18%	6.96	4.05	729,882	7,647

描述

不安全设计是一个广泛的类别，代表着不同的弱点，表现为“缺失或无效的控制设计”。不安全设计并非其他所有十大风险类别的源头。需要注意的是，不安全设计和不安全实施之间存在差异。安全专家对设计缺陷和实施缺陷进行区分是有原因的，两者有着不同的根本原因，发生在开发过程的不同阶段，并且有着不同的补救措施。一个安全的设计仍然可能存在实施缺陷，从而导致可能被利用的漏洞。不安全的设计无法通过完美的实施来修复，因为系统缺乏必要的安全控制措施以抵御特定的攻击。导致不安全设计的其中一个因素是，正在开发的软件或系统中缺乏固有的业务风险分析，因此无法确定所需的安全设计级别。

安全设计的三个关键部分是：

- 需求收集与资源管理
- 创建安全设计
- 拥有安全开发生命周期

需求与资源管理

与业务方收集并协商应用程序的业务需求，包括所有数据资产在保密性、完整性、可用性和真实性方面的保护要求，以及预期的业务逻辑。考虑应用程序的暴露程度，以及是否需要租户隔离（超出访问控制所需的隔离）。编制技术需求，包括功能性需求和非功能性的安全需求。规划并协商涵盖所有设计、构建、测试和运营（包括安全活动）的预算。

安全设计

安全设计是一种文化和方法，安全专家使用安全设计的方法不断评估威胁，并确保代码经过稳健设计和测试，以防范已知的攻击手段。威胁建模应融入项目会议（或类似活动）中；要关注数据

流、访问控制或其他安全控制措施的变化。在用户故事开发过程中，确定正确的流程和失败状态，确保负责方和受影响方都能充分理解并达成共识。分析预期流程和失败流程的假设与条件，确保这些假设与条件始终准确且符合预期。确定如何验证这些假设，并执行正常行为所需的条件。确保结果记录在用户故事中。团队需要从错误中学习，并提供积极激励以促进改进。安全设计既不是附加组件，也不是可以添加到软件中的工具。

安全开发生命周期

安全的软件需要安全的开发生命周期、安全的设计模式、规范的开发方法、安全的组件库、适当的工具、威胁建模以及用于改进流程的事件事后分析。在软件项目开始时、项目进行过程中以及持续的软件维护阶段，都要与安全专家沟通。可以考虑利用 [OWASP 软件保障成熟度模型 \(SAMM\)](#) 来帮助构建安全的软件开发工作。

开发者的自我责任感往往未得到足够重视。组织应培养一种具备意识、责任感和主动风险缓解能力的文化，并定期就安全问题进行交流（例如在威胁建模会议期间），可以培养在所有重要设计决策中纳入安全考量的思维模式。

如何预防

- 与应用安全专业人员一起建立并使用安全开发生命周期，以帮助评估和设计与安全及隐私相关的控制措施
- 建立并使用安全设计模式库或标准化组件库
- 对应用程序的关键部分（如身份验证、访问控制、业务逻辑和关键流程）使用威胁建模
- 将威胁建模作为一种教育工具，以培养安全思维

- 将安全语言和控制措施整合到用户故事中
- 在应用程序的每个层级（从前端到后端）集成合理性检查
- 编写单元测试和集成测试，以验证所有关键流程都能抵御威胁模型提出的威胁。为应用程序的每个层级编写用例和误用案例。
- 根据暴露程度和保护需求，在系统层和网络层对层级进行隔离
- 在所有层级中，通过设计对租户进行强有力的隔离

示例攻击场景

场景 1：凭证恢复流程可能包含“问答”环节，而这是 NIST 800-63b、OWASP ASVS 以及 OWASP Top 10 所禁止的。问答不能作为身份的可靠证明，因为可能有多个人知道答案。此类功能应移除，并替换为更安全的设计。

场景 2：一家连锁影院提供团体订票折扣，且在要求支付押金前最多可容纳 15 名观众。攻击者可以对这一流程进行威胁建模，并测试能否在应用程序的业务逻辑中找到攻击向量，例如通过几次请求一次性预订 600 个座位以及所有影院的座位，从而造成巨大的收入损失。

场景 3：一家零售连锁企业的电子商务网站没有针对黄牛运行的机器人程序的防护措施，这些机器人会抢购高端显卡，然后在拍卖网站上转售。这给显卡制造商和零售连锁企业带来了极坏的名誉，并与那些无论出多少钱都无法买到这些显卡的发烧友结下了长久的梁子。精心设计的反机器人程序和领域逻辑规则（例如，在显卡上架后几秒内完成的购买）可能会识别出虚假购买并拒绝此类交易。

参考文献

- [OWASP Cheat Sheet: Secure Design Principles](#)OWASP cheat sheet: 安全设计原则
- [OWASP SAMM: Design | Secure Architecture](#)OWASP SAMM: 设计 | 安全架构
- [OWASP SAMM: Design | Threat Assessment](#)OWASP SAMM: 设计 | 威胁评估
- [NIST – Guidelines on Minimum Standards for Developer Verification of Software](#)NIST——软件开发者验证最低标准指南
- [The Threat Modeling Manifesto](#) 威胁建模宣言
- [Awesome Threat Modeling](#) 出色的威胁建模

CWE 映射列表

- [CWE-73](#) 文件名或路径的外部控制
- [CWE-183](#) 允许输入的宽松列表
- [CWE-256](#) 凭证的未受保护存储
- [CWE-266](#) 权限分配不当
- [CWE-269](#) 权限管理不当
- [CWE-286](#) 不正确的用户管理
- [CWE-311](#) 敏感数据未加密
- [CWE-312](#) 敏感信息的明文存储
- [CWE-313](#) 在文件或磁盘中明文存储
- [CWE-316](#) 内存中敏感信息的明文存储
- [CWE-362](#) 利用共享资源的并发执行且同步不当（“竞争条件”）
- [CWE-382](#) J2EE 不良实践：使用 System.exit()

- [CWE-419 未受保护的主通道](#)
- [CWE-434 危险类型文件的无限制上传](#)
- [CWE-436 解释冲突](#)
- [CWE-444 HTTP 请求的不一致解析（“HTTP 请求走私”）](#)
- [CWE-451 用户界面（UI）对关键信息的错误呈现](#)
- [CWE-454 可信变量或数据存储的外部初始化](#)
- [CWE-472 对假定不可变的 Web 参数的外部控制](#)
- [CWE-501 信任边界违规](#)
- [CWE-522 凭证保护不足](#)
- [CWE-525 使用包含敏感信息的 Web 浏览器缓存](#)
- [CWE-539 使用包含敏感信息的持久性 Cookie](#)
- [CWE-598 使用带有敏感查询字符串的 GET 请求方法](#)
- [CWE-602 服务器端安全的客户端实施](#)
- [CWE-628 带有错误指定参数的函数调用](#)
- [CWE-642 关键状态数据的外部控制](#)
- [CWE-646 依赖外部提供文件的文件名或扩展名](#)
- [CWE-653 隔离不充分](#)
- [CWE-656 依赖通过隐匿实现的安全性](#)
- [CWE-657 违反安全设计原则](#)
- [CWE-676 使用潜在危险函数](#)

- [CWE-693](#) 保护机制失效
- [CWE-799](#) 交互频率控制不当
- [CWE-807](#) 安全决策中对不可信输入的依赖
- [CWE-841](#) 行为 workflow 执行不当
- [CWE-1021](#) 渲染的用户界面层或框架的不当限制
- [CWE-1022](#) 带有 window.opener 访问权限的指向不可信目标的 Web 链接的使用
- [CWE-1125](#) 过大的攻击面

A07:2025 身份认证失效

背景

“身份验证失败”类别保持在第 7 位，其名称略有更改，以更准确地反映该类别中的 36 个 CWE。尽管标准化框架带来了益处，但该类别自 2021 年以来一直保持第 7 位。其中值得注意的 CWE 包括 CWE-259：使用硬编码密码、CWE-297：证书与主机不匹配的不当验证、CWE-287：身份验证不当、CWE-384：会话固定和 CWE-798：使用硬编码凭据。



评分表

映射的常见弱点枚举数量 (CWE)	最大发生率	平均发生率	最大覆盖率	平均覆盖率	平均加权漏洞利用	平均加权影响	总发生次数	CVE 总数
36	15.80%	2.92%	100.00%	37.14%	7.69	4.44	1,120,673	7,147

描述

当攻击者能够诱骗系统将无效或错误的用户识别为合法用户时，就存在这种漏洞。如果应用程序存在以下情况，则可能存在身份验证漏洞：

- 允许进行自动攻击，例如凭据填充攻击，即攻击者拥有已泄露的有效用户名和密码列表。
最近，这类攻击已扩展到包括混合密码攻击性质的凭据填充攻击（也称为密码喷洒攻击），攻击者会使用泄露凭据的变体或递增形式来获取访问权限，例如尝试 Password1!、Password2!、Password3! 等等。
- 允许暴力攻击或其他未能迅速阻止的自动化、脚本化攻击。

- 允许使用默认密码、弱密码或知名密码，例如 “Password1” 或 “admin” 用户名搭配 “admin” 密码。
- 允许用户使用已知已泄露的凭据创建新账户。
- 允许使用薄弱或无效的凭证恢复和密码找回流程，例如 “基于知识的答案”，这类流程无法保证安全。
- 使用纯文本、加密或弱哈希密码数据存储（参见 [A04:2025-加密失败](#)）。
- 存在缺失或无效的多因素认证。
- 如果多因素认证不可用，则允许使用薄弱或无效的备用方案。
- 在 URL、隐藏字段或客户端可访问的其他不安全位置中暴露会话标识符。
- 成功登录后重复使用相同的会话标识符。
- 在登出或一段不活动期间，无法正确使用户会话或身份验证令牌（主要是单点登录 (SSO) 令牌）失效。
- 未能正确声明所提供凭证的范围和目标受众。

如何预防

- 在可能的情况下，实施并强制使用多因素认证，以防止自动化凭证填充、暴力破解和失窃的凭证的重复使用攻击。
- 在可能的情况下，鼓励并支持使用密码管理器，以帮助用户做出更好的选择。
- 不要使用任何默认凭据进行发布或部署，尤其是管理员用户的凭据。
- 实施弱密码检查，例如将新密码或更改后的密码与排名前 10,000 的最差密码列表进行比对。

- 在创建新账户和更改密码时，要对照已知的泄露凭据列表进行验证（例如：使用 haveibeenpwned.com）。
- 使密码长度、复杂性和轮换策略符合《国家标准与技术研究院（NIST）800-63b 指南》中第 5.1.1 节关于记忆密码的规定，或其他现代、基于证据的密码策略。
- 除非怀疑发生数据泄露，否则不要强迫用户定期更换密码。如果怀疑发生数据泄露，应立即强制用户重置密码。
- 确保注册、凭证恢复和 API 路径通过对所有结果使用相同的消息（“用户名或密码无效。”）来增强抵御账户枚举攻击的能力。
- 限制或逐步延迟失败的登录尝试，但要注意不要造成拒绝服务的情况。记录所有失败的登录尝试，并在检测到或怀疑存在凭证填充、暴力破解或其他攻击时向管理员发出警报。
- 使用服务器端、安全的内置会话管理器，该管理器在登录后生成具有高熵值的新随机会话 ID。会话标识符不应出现在 URL 中，而应安全地存储在安全的 Cookie 中，并在注销、闲置和绝对超时后失效。
- 理想情况下，应使用预制的、广受信任的系统来处理身份验证、身份识别和会话管理。只要有可能，就通过购买和使用经过强化且充分测试的系统来转移这种风险。
- 验证所提供凭据的预期用途，例如对于 JWT，验证 aud、iss 声明和范围。

示例攻击场景

场景 1：凭证填充，即使用已知的用户名和密码组合列表，如今是一种非常常见的攻击方式。最近发现，攻击者会根据人类的常见行为对密码进行“递增”或其他调整。例如，将“Winter2025”改为“Winter2026”，或者将“I Love My Dog6”改为“I Love My Dog7”或“I Love My Dog5”。这种对密码尝试的调整被称为混合凭证填充攻击或密码喷洒攻击，其效果甚至可能比传统版本更好。

如果应用程序没有实施针对自动化威胁（暴力破解、脚本或机器人程序）或凭证填充的防御措施，那么该应用程序可能会被用作密码验证机，从而让攻击者确定凭证是否有效并获取未授权访问权限。

场景 2：大多数成功的身份验证攻击都源于继续将密码作为唯一的身份验证因素。曾经被视为最佳实践的密码轮换和复杂性要求，反而促使用户重复使用密码并使用弱密码。根据 NIST 800-63 标准，建议各组织停止这些做法，并在所有重要系统上强制使用多因素身份验证。

场景 3：应用程序会话超时未正确实现。用户使用公共计算机访问应用程序时，没有选择“注销”，而只是关闭浏览器标签页后离开。另一个例子是，如果单点登录（SSO）会话无法通过单点登出（SLO）关闭。也就是说，一次登录会让你进入例如邮件阅读器、文档系统和聊天系统。但登出只对当前系统生效。如果攻击者在受害者认为自己已成功登出后使用同一浏览器，而用户仍在某些应用程序中保持认证状态，那么攻击者就可以访问受害者的账户。在办公室和企业中也可能出现同样的问题：当敏感应用程序未被正确退出时，同事（暂时）可以访问未锁定的计算机。

参考文献

- [OWASP Authentication Cheat Sheet OWASP 身份验证 cheat sheet](#)
- [OWASP Secure Coding Practices OWASP 安全编码实践](#)

CWE 映射列表

- [CWE-258 配置文件中的空密码](#)
- [CWE-259 硬编码密码的使用](#)
- [CWE-287 身份验证不当](#)
- [CWE-288 使用替代路径或通道绕过身份验证](#)

A07:2025 身份认证失效

- [CWE-289 通过替代名称绕过身份验证](#)
- [CWE-290 通过欺骗绕过身份验证](#)
- [CWE-291 依赖 IP 地址进行身份验证](#)
- [CWE-293 使用 Referer 字段进行身份验证](#)
- [CWE-294 通过捕获重放实现的身份验证绕过](#)
- [CWE-295 证书验证不当](#)
- [CWE-297 证书与主机不匹配时的不当验证](#)
- [CWE-298 证书与主机不匹配的不当验证](#)
- [CWE-299 证书与主机不匹配的不当验证](#)
- [CWE-300 非端点可访问的信道](#)
- [CWE-302 因假设数据不可变导致的身份验证绕过](#)
- [CWE-303 身份验证算法实现不正确](#)
- [CWE-304 身份验证中缺失关键步骤](#)
- [CWE-305 因主要缺陷导致的身份验证绕过](#)
- [CWE-306 关键功能缺少身份验证](#)
- [CWE-307 对过多身份验证尝试的不当限制](#)
- [CWE-308 使用单因素认证](#)
- [CWE-309 使用密码系统进行主要身份验证](#)
- [CWE-346 来源验证错误](#)
- [CWE-350 安全关键操作依赖反向 DNS 解析](#)
- [CWE-384 会话固定](#)

- [CWE-521 弱密码要求](#)
- [CWE-613 会话过期不充分](#)
- [CWE-620 未验证的密码更改](#)
- [CWE-640 遗忘密码的弱密码恢复机制](#)
- [CWE-798 硬编码凭证的使用](#)
- [CWE-940 通信渠道来源验证不当](#)
- [CWE-941 通信信道中错误指定的目的地](#)
- [CWE-1390 弱认证](#)
- [CWE-1391 使用弱凭据](#)
- [CWE-1392 使用默认凭据](#)
- [CWE-1393 使用默认密码](#)

A08:2025 软件或数据完整性失效

背景

软件或数据完整性故障继续列为第 8 类, 名称略有更改从“软件和数据完整性失效”更清晰地表达了其含义。该类别聚焦于未能维护信任边界, 且未在底层技术层面验证软件、代码和数据工件的完整性; 其关注维度相较于“软件供应链失效”更偏向底层实现。该类别关注的是未验证完整性就对软件更新和关键数据做出假设的情况。值得注意的常见弱点枚举 (CWE) 包括 CWE-829: 包含来自不受信任控制领域的功能、CWE-915: 对动态确定的对象属性进行不当控制的修改以及 CWE-502: 反序列化不受信任的数据。



评分表

映射的常见弱点枚举数量 (CWE)	最大发生率	平均发生率	最大覆盖率	平均覆盖率	平均加权漏洞利用	平均加权影响	总发生次数	CVE 总数
14	8.98%	2.75%	78.52%	45.49%	7.11	4.79	501,327	3,331

描述

- 软件和数据完整性失效与代码和基础设施有关, 这些代码和基础设施未能防止将无效或不受信任的代码或数据视为可信有效。例如, 应用程序依赖于来自不受信任的来源、代码库和内容分发网络 (CDN) 的插件、库或模块。不安全的 CI/CD 管道如果未使用和

提供软件完整性检查，则可能导致未经授权的访问、不安全或恶意代码，甚至系统遭到入侵。另一个例子是 CI/CD 从不受信任的位置拉取代码或工件，并且/或者在使用前未对其进行验证（例如，检查签名或类似机制）。此外，许多应用程序现在都包含自动更新功能，这些更新在未进行充分完整性验证的情况下下载，并应用于之前受信任的应用程序。攻击者可能上传自己的更新，并将其分发到所有安装位置运行。另一个例子是，如果对象或数据被编码或序列化成为攻击者可以查看和修改的结构，则容易受到不安全的反序列化攻击。

如何预防

- 使用数字签名或类似机制来验证软件或数据是否来自预期来源且未被篡改。
- 确保库和依赖项（例如 npm 或 Maven）仅使用可信的存储库。如果您的风险等级较高，请考虑托管一个经过验证的内部可靠存储库。
- 确保对代码和配置变更进行审查，以最大程度地降低恶意代码或配置被引入软件管道的可能性。
- 确保您的 CI/CD 流水线具有适当的隔离、配置和访问控制，以保证流经构建和部署过程的代码的完整性。
- 确保不会从不受信任的客户端接收未签名或未加密的序列化数据，并且随后在没有某种形式的完整性检查或数字签名来检测序列化数据的篡改或重放的情况下使用这些数据。

示例攻击场景

场景 1：引入来自不受信任来源的 Web 功能：一家公司使用外部服务提供商提供支持功能。

为了方便起见，该公司将域名映射到外部服务提供商 myCompany.SupportProvider.com。

support.myCompany.com 这意味着在该 myCompany.com 域名上设置的所有 Cookie（包括身份验证 Cookie）现在都会被发送到支持服务提供商。任何有权访问支持服务提供商基础架构的人都可以窃取所有访问过该公司的用户的 Cookie support.myCompany.com，并执行会话劫持攻击。

场景 2：未经签名的更新：许多家用路由器、机顶盒、设备固件等并不通过签名固件来验证更新。未签名固件正日益成为攻击者的目标，而且这种情况预计只会更加恶化。这是一个重大隐患，因为很多时候除了在未来的版本中修复并等待旧版本过时之外，没有其他补救措施。

场景 3：使用来自不可信来源的软件包：一位开发者找不到所需软件包的更新版本，于是他没有从常用的可信软件包管理器下载，而是从某个在线网站下载。该软件包未经签名，因此无法确保其完整性。此外，该软件包还包含恶意代码。

场景四：不安全的反序列化：一个 React 应用调用了一组 Spring Boot 微服务。由于开发者是函数式程序员，他们试图确保代码的不可变性。他们想出的解决方案是将用户状态序列化，并在每次请求中来回传递。攻击者注意到 Java 对象签名（base64 编码）为“rO0”，并利用 Java 反序列化扫描器在应用服务器上获取远程代码执行权限。

参考文献

- [OWASP 速查表：软件供应链安全](#)
- [OWASP 速查表：基础设施即代码](#)
- [OWASP 速查表：反序列化](#)
- [SAFECode 软件完整性控制](#)
- [一场“噩梦般的”网络攻击：SolarWinds 黑客事件不为人知的故事](#)

- [CodeCov Bash 上传器漏洞](#)
- [Julien Vehent 著《保障 DevOps》](#)
- [Tenendo 的不安全反序列化](#)

CWE 映射列表

- [CWE-345 数据真实性验证不足](#)
- [CWE-353 缺少对完整性检查的支持](#)
- [CWE-426 不受信任的搜索路径](#)
- [CWE-427 非受控搜索路径元素](#)
- [CWE-494 下载代码未进行完整性检查](#)
- [CWE-502 不受信任数据的反序列化](#)
- [CWE-506 嵌入式恶意代码](#)
- [CWE-509 复制恶意代码（病毒或蠕虫）](#)
- [CWE-565 依赖 Cookie 而未进行验证和完整性检查](#)
- [CWE-784 安全决策中依赖未经验证和完整性检查的 Cookie](#)
- [CWE-829 包含来自不受信任控制域的功能](#)
- [CWE-830 包含来自不受信任来源的 Web 功能](#)
- [CWE-915 动态确定对象属性的不当控制修改](#)
- [CWE-926 Android 应用程序组件导出不当](#)

背景

安全日志记录与警报故障仍位居第 9 位。该类别名称略有更改，以强调针对相关日志事件采取行动所需的警报功能。这类问题在数据中总是体现不足，这是第三次通过社区调查参与者的投票进入该榜单。此类问题极难测试，在 CVE/CVSS 数据中的占比极低（仅 723 个 CVE），但对可见性、事件警报和取证可能具有极大影响。该类别包括以下问题：对日志文件的输出编码处理不当（CWE-117）、将敏感数据插入日志文件（CWE-532）以及日志记录不充分（CWE-778）。



评分表

映射的常见弱点 枚举数量 (CWE)	最大发 生率	平均发 生率	最大覆 盖率	平均覆 盖率	平均加 权漏洞 利用	平均加 权影响	总发生 次数	CVE 总数
5	11.33%	3.91%	85.96%	46.48%	7.19	2.65	260,288	723

描述

如果没有日志记录和监控，就无法检测到攻击和漏洞；而如果没有警报，在安全事件发生时就很难快速有效地做出响应。以下任何时候都会出现日志记录、持续监控、检测和警报不足，从而无法启动主动响应的情况：

- 可审计事件（如登录、登录失败和高价值交易）未被记录，或记录不一致（例如，仅记录成功的登录，而不记录失败的尝试）。

- 警告和错误不会生成日志消息，或者生成的日志消息不充分、不清晰。
- 日志的完整性没有得到适当保护，无法防止被篡改。
- 未对应用程序和 API 的日志进行可疑活动监控。
- 日志仅存储在本地，且未进行适当的备份。
- 适当的警报阈值和响应升级流程未到位或无效。运营人员未能在合理时间内接收到警报或查看警报。
- 渗透测试以及通过动态应用安全测试（DAST）工具（如 Burp 或 ZAP）进行的扫描不会触发警报。
- 该应用程序无法实时或近实时地检测、上报或警示正在进行的攻击。
- 通过让用户或攻击者能够看到日志记录和告警事件（参见 [A01:2025-访问控制失效](#)），或者通过日志中记录的敏感信息（这些不应该记录）（如个人身份信息或受保护的健康信息），系统可能会面临敏感信息泄露的风险。
- 如果日志数据没有正确编码，那么系统容易受到针对日志记录或监控系统的注入攻击。
- 该应用程序缺失或错误处理错误及其他异常情况，导致系统未察觉错误的发生，因此无法记录出现的问题。
- 用于发出警报以识别特殊情况的足够“用例”缺失或过时。
- 过多的误报使得无法区分重要警报和不重要警报，导致运营人员识别这些警报太晚，或者根本没有识别到这些警报（安全运营中心团队的人力负担过重）。
- 由于用例的剧本不完整、过时或缺失，检测到的警报无法正确处理。

如何预防

开发人员应根据应用程序的风险，实施以下部分或全部控制措施：

- 确保所有登录、访问控制和服务器端输入验证失败都能在记录时包含足够的用户上下文，以便识别可疑或恶意账户，并保留足够长的时间以进行延迟的取证分析。
- 确保应用程序中包含安全控制的每个部分都被记录下来，无论其成功与否。
- 确保日志的生成格式便于日志管理解决方案轻松处理。
- 确保日志数据编码正确，以防止对日志记录或监控系统的注入攻击。
- 确保所有交易都有带有完整性控制的审计跟踪，以防止篡改或删除，例如仅追加数据库表或类似机制。
- 确保所有抛出错误的交易都应该回滚并重新开始交易。交易始终以关闭状态失败。
- 如果应用程序或其用户的行为可疑，请发出警报。为你的开发人员制定关于此主题的指南，以便开发人员能够针对此进行编码或购买相关系统。
- DevSecOps 和安全团队应建立有效的监控和警报用例，包括操作手册，以便安全运营中心（SOC）团队能够快速检测和响应可疑活动。
- 在应用程序中添加“蜜罐令牌”作为针对攻击者的陷阱，例如在数据库、数据中，以真实和/或技术用户身份的形式存在。由于系统在不会在正常业务中使用“蜜罐令牌”，任何对“蜜罐令牌”的访问都会生成日志数据，这些数据可以触发警报，且几乎不会产生误报。
- 行为分析和人工智能支持可以作为一种可选的附加技术，以降低警报的误报率。
- 制定或采用事件响应与恢复计划，例如美国国家标准与技术研究院（NIST）800-61r2 或其后续版本。向软件开发人员传授应用程序攻击和事件的特征，以便他们能够进行报告。

有一些商业和开源的应用程序保护产品，例如 OWASP ModSecurity 核心规则集，以及开源的日志关联软件，如 Elasticsearch、Logstash、Kibana (ELK) 堆栈，这些软件具有自定义仪表板和警报功能，可能有助于运营人员解决上述问题。还有一些商业可观测性工具可以帮助运营人员近乎实时地响应或阻止攻击。

示例攻击场景

场景 1：一家儿童健康计划提供商的网站运营者因缺乏监控和日志记录而未能发现数据泄露。一名外部人士告知该健康计划提供商，有攻击者访问并修改了超过 350 万名儿童的数千份敏感健康记录。事后审查发现，网站开发人员并未处理重大漏洞。由于没有对系统进行日志记录或监控，此次数据泄露可能从 2013 年就已开始，持续了七年多。

场景 2：一家印度大型航空公司发生了数据泄露事件，涉及数百万乘客超过十年的个人数据，包括护照和信用卡信息。此次数据泄露发生在一家第三方云托管提供商处，该提供商过了一段时间后才将此事通知该航空公司。

场景 3：一家欧洲主要航空公司发生了一起需按《通用数据保护条例》（GDPR）上报的违规事件。据报道，该违规事件是由攻击者利用支付应用程序的安全漏洞引发的，攻击者窃取了超过 40 万条客户支付记录。因此，这家航空公司被隐私监管机构处以 2000 万英镑的罚款。

参考文献

- [OWASP Proactive Controls: C9: Implement Logging and Monitoring](#) OWASP 主动控制措施：C9：实施日志记录和监控

- OWASP Application Security Verification Standard: V16 Security Logging and Error HandlingOWASP 应用程序安全验证标准: V16 安全日志记录与错误处理
- OWASP Cheat Sheet: Application Logging VocabularyOWASP cheat sheet: 应用程序日志词汇
- OWASP Cheat Sheet: Logging OWASP cheat sheet: 日志记录
- Data Integrity: Recovering from Ransomware and Other Destructive Events 数据完整性: 从勒索软件及其他破坏性事件中恢复
- Data Integrity: Identifying and Protecting Assets Against Ransomware and Other Destructive Events 数据完整性: 识别资产并保护其免受勒索软件及其他破坏性事件的影响
- Data Integrity: Detecting and Responding to Ransomware and Other Destructive Events 数据完整性: 检测并响应勒索软件及其他破坏性事件
- Real world example of such failures in Snowflake BreachSnowflake 数据泄露中此类失败的真实世界案例

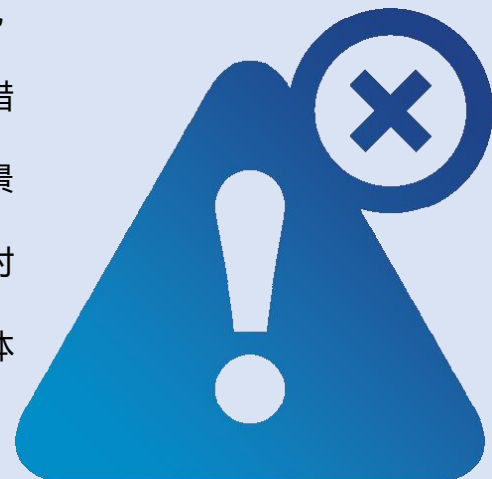
CWE 映射列表

- CWE-117 日志的不当输出中和
- CWE-221 信息遗漏丢失
- CWE-223 安全相关信息的遗漏
- CWE-532 敏感信息插入日志文件中
- CWE-778 日志记录不足

A10:2025 异常情况处理不当

背景

“异常情况处理不当”是 2025 年新增的一个类别。该类别包含 24 个通用弱点枚举（CWE），重点关注错误处理不当、逻辑错误、故障开放以及其他由系统可能遇到的异常情况引发的相关场景。这个类别中的一些常见弱点枚举以前与糟糕的代码质量相关联。对安全专家而言，这一表述过于笼统；在安全专家看来，这个更具体的类别能提供更好的指导。



本类别中包含的重要 CWE 有：CWE-209 生成包含敏感信息的错误消息、CWE-234 未能处理缺失参数、CWE-274 对权限不足的处理不当、CWE-476 空指针解引用以及 CWE-636 未能安全失败（“开放失败”）。

评分表

映射的常见弱点枚举数量 (CWE)	最大发生率	平均发生率	最大覆盖率	平均覆盖率	平均加权漏洞利用	平均加权影响	总发生次数	CVE 总数
24	20.67%	2.95%	100.00%	37.95%	7.11	3.81	769,581	3,416

描述

软件的异常情况处理不当，指程序未能对异常及不可预测的场景进行预防、检测和响应，易导致系统崩溃、异常运行，甚至引发安全漏洞。这可能涉及以下三种失误中的一种或多种：应用程序未能阻止异常情况的发生，未能在情况发生时识别出该情况，以及在事后对该情况应对不力或完全没有应对。

A10:2025 异常情况处理不当

异常情况可能由以下原因导致：输入验证缺失、不完善或不完整；在错误发生的函数处未进行处理，而是采用滞后的高层级错误处理；出现意外的环境状态，如内存、权限或网络问题；异常处理不一致；或者完全未处理异常，导致系统陷入未知且不可预测的状态。每当应用程序不确定其下一步指令时，就说明异常情况处理不当。难以发现的错误和异常可能会长期威胁整个应用程序的安全性。

当开发人员错误处理异常情况时，可能会出现许多不同的安全漏洞。

例如逻辑漏洞、溢出、竞争条件、欺诈交易，或者与内存、状态、资源、时间、身份验证和授权相关的问题。这类漏洞可能会对系统或其数据的机密性、可用性和完整性产生负面影响。攻击者会利用应用程序有缺陷的错误处理来利用此漏洞。

如何预防

为了妥善处理异常情况，系统必须为此类情况做好规划（做最坏的打算）。系统必须在每个可能的系统错误发生的地方直接“捕获”它们，然后进行处理（这意味着采取一些有意义的措施来解决问题，并确保系统能从该问题中恢复）。作为处理的一部分，系统应该包括抛出错误（以用户能理解的方式告知用户）、记录事件，以及在系统认为有必要时发出警报。系统还应该设置一个全局异常处理器，以防出现遗漏的情况。理想情况下，系统还应该配备监控和/或可观测性工具或功能，用于监测重复出现的错误或表明正在遭受攻击的模式，这些工具或功能能够发出某种响应、进行防御或实施拦截。这有助于系统拦截并应对那些利用系统错误处理弱点的脚本和机器人程序。

捕获和处理异常情况可以确保程序的底层基础设施不会陷入处理不可预测情况的境地。如果系统正在进行任何类型的事务处理，那么回滚该事务的所有部分并重新开始（也称为故障关闭）是极为重要的。在事务处理过程中尝试进行部分恢复往往会导致无法挽回的错误。

只要有可能，就应添加速率限制、资源配额、节流和其他限制，从一开始就防止异常情况的发生。信息技术中没有任何东西应该是无限制的，因为这会导致应用程序弹性失效、服务中断、暴力攻击成功以及高昂的云服务费用。考虑一下，当相同的重复错误超过一定频率时，是否应该只以统计数据的形式输出，显示它们发生的频率和时间范围。此信息应附加到原始消息中，以免干扰自动日志记录和监控，参见 [A09:2025 安全日志记录和警报故障](#)。

除此之外，系统还希望包含严格的输入验证（对必须接受的潜在危险字符进行净化或转义），以及集中式错误处理、日志记录、监控和告警，还有一个全局异常处理器。一个应用程序不应有多个处理异常情况的函数，而应在一个地方以相同的方式进行处理。开发人员还应为本节中的所有建议制定项目安全要求，在项目的设计阶段进行威胁建模和/或安全设计审查活动，进行代码审查或静态分析，并对最终系统执行压力测试、性能测试和渗透测试。

如果可能的话，整个组织应该以相同的方式处理异常情况，因为这样可以更轻松地审查和审计这一重要安全控制措施中的代码错误。

示例攻击场景

场景 1：如果应用程序在文件上传时捕获到异常，但之后没有正确释放资源，可能会因异常情况处理不当导致资源耗尽（拒绝服务）。每次新的异常都会使资源处于锁定状态或无法使用，直至所有资源被耗尽。

场景 2：由于处理不当或数据库错误导致敏感数据泄露，向用户展示了完整的系统错误。攻击者持续制造错误，以便利用敏感的系统信息发起更有效的 SQL 注入攻击。用户错误消息中的敏感数据属于侦察信息。

场景 3：攻击者可能通过网络中断干扰多步骤交易，从而导致金融交易中的状态损坏。试想交易顺序为：从用户账户扣款、向目标账户贷记、记录交易。如果系统在交易过程中出现错误时没有正确回滚整个交易（故障关闭），攻击者可能会耗尽用户的账户，或者可能出现竞态条件，使攻击者能够多次向目标账户转账。

参考文献

- [OWASP MASVS- RESILIENCE OWASP 移动应用安全验证标准-韧性](#)
- [OWASP Cheat Sheet: Logging OWASP cheat sheet: 日志记录](#)
- [OWASP Cheat Sheet: Error Handling OWASP cheat sheet: 错误处理](#)
- [OWASP Application Security Verification Standard \(ASVS\): V16.5 Error Handling](#)
[OWASP 应用程序安全验证标准 \(ASVS\) : V16.5 错误处理](#)
- [OWASP Testing Guide: 4.8.1 Testing for Error Handling](#)
[OWASP 测试指南: 4.8.1 错误处理测试](#)
- [Best practices for exceptions \(Microsoft, .Net\)](#)
[异常的最佳实践 \(微软, .Net\)](#)
- [Clean Code and the Art of Exception Handling \(Toptal\)](#)
[《整洁代码与异常处理的艺术》 \(Toptal\)](#)
- [General error handling rules \(Google for Developers\)](#)
[通用错误处理规则 \(谷歌开发者\)](#)

- Example of real-world mishandling of an exceptional condition 现实世界中异常情况处理不当的示例

CWE 映射列表

- CWE-209 生成包含敏感信息的错误消息
- CWE-215 敏感信息插入调试代码中
- CWE-234 未能处理缺失的参数
- CWE-235 对额外参数的不当处理
- CWE-248 未捕获的异常
- CWE-252 未检查的返回值
- CWE-274 权限不足的不当处理
- CWE-280 权限不足或特权处理不当
- CWE-369 除以零
- CWE-390 检测到错误状态却未采取行动
- CWE-391 未检查的错误条件
- CWE-394 意外的状态码或返回值
- CWE-396 泛型异常的捕获声明
- CWE-397 泛型异常的抛出声明
- CWE-460 抛出异常时的不当净化
- CWE-476 空指针解引用
- CWE-478 多条件表达式中缺少默认情况

A10:2025 异常情况处理不当

- [CWE-484 switch 语句中省略 break 语句](#)
- [CWE-550 服务器生成的包含敏感信息的错误消息](#)
- [CWE-636 不安全的失败（“开放失败”）](#)
- [CWE-703 异常条件的不当检查或处理](#)
- [CWE-754 对异常或特殊情况检查不当](#)
- [CWE-755 异常条件处理不当](#)
- [CWE-756 缺少自定义错误页面](#)

从设计上讲，OWASP Top 10 本质上仅限于十种最重大的风险。但每一份 OWASP Top 10 报告中都有一些“处于边缘”的风险，这些风险经过了长时间的慎重评估，但最终未能入选。其他入选的风险更为普遍且影响更大。

对于致力于打造成熟应用安全计划的组织、安全咨询公司或希望扩大其产品覆盖范围的工具供应商而言，以下两个问题非常值得花功夫去识别和补救。

X01:2025 应用程序弹性不足

背景

这是对 2021 年“拒绝服务”类别的重命名。之所以重命名，是因为原来的名称描述的是一种症状而非根本原因。此类别聚焦于描述与弹性问题相关的弱点的常见弱点枚举（CWE）。该类别的评分与 A10:2025-异常条件处理不当非常接近。相关的常见弱点枚举包括：CWE-400 不受控制的资源消耗、CWE-409 对高压缩数据的不当处理（数据放大）、CWE-674 不受控制的递归以及 CWE-835 存在不可达退出条件的循环（“无限循环”）。

评分表

映射的常见弱点枚举数量 (CWE)	最大发生率	平均发生率	最大覆盖率	平均覆盖率	平均加权漏洞利用	平均加权影响	总发生次数	CVE 总数
16	20.05%	4.55%	86.01%	41.47%	7.92	3.49	865,066	4,423

描述

该类别漏洞反映了应用程序在应对压力、故障和边界情况时存在的系统性缺陷，导致其无法从故障中恢复。当应用程序无法妥善处理、承受意外情况、资源限制和其他不利事件，或从中恢复时，很容易导致可用性问题（最常见），还可能造成数据损坏、敏感数据泄露、级联故障和/或安全控制被绕过。

此外，[X02:2025 内存管理故障](#)也可能导致应用程序甚至整个系统发生故障。

如何预防

为了防止此类漏洞，开发人员必须在设计时考虑系统的故障与恢复。

- 添加限制、配额和故障转移功能，尤其要关注资源消耗最大的操作。
- 识别资源密集型页面并提前规划：减少攻击面，尤其不要向未知或不可信用户暴露不需要的、需要大量资源（如 CPU、内存）的“工具”和功能。
- 使用允许列表和大小限制执行严格的输入验证，然后进行彻底测试。
- 限制响应大小，并且绝不要将原始响应发送回客户端（在服务器端处理）。
- 默认设置为安全/关闭（永不开启），默认拒绝，若出现错误则回滚。
- 避免在请求线程中阻塞同步调用（使用异步/非阻塞方式，设置超时，设置并发限制等）。
- 仔细测试错误处理功能。
- 实现弹性模式，例如断路器模式、舱壁模式、重试逻辑和优雅降级。
- 进行性能和负载测试；如果有风险承受能力，可以加入混沌工程。
- 在合理且经济可行的情况下，实现并构建冗余架构。
- 实施监控、可观测性和告警功能。
- 根据 RFC 2267 过滤无效的发件人地址。
- 通过指纹、IP 地址或动态地通过行为来拦截已知的僵尸网络。
- 工作量证明（PoW）：在攻击者一侧启动资源消耗操作，这对普通用户影响不大，但会影响试图发送大量请求的机器人。如果系统的总体负载增加，应提高工作量证明的难度，特别是对于那些可信度较低或看似是机器人的系统。
- 基于不活动状态和最终超时限制服务器端会话时间。
- 限制与会话绑定的信息存储。

X01:2025 应用程序弹性不足

场景 1：攻击者故意消耗应用程序资源，以触发系统内部故障，从而导致拒绝服务。这可能是内存耗尽、磁盘空间填满、CPU 饱和或打开无限连接。

场景 2：输入模糊测试导致精心构造的响应破坏了应用程序的业务逻辑。

场景 3：攻击者将重点放在应用程序的依赖项上，瘫痪 API 或其他外部服务，导致应用程序无法继续运行。

参考文献

- [OWASP Cheat Sheet: Denial of Service](#)OWASP cheat sheet: 拒绝服务
- [OWASP MASVS- RESILIENCE](#)OWASP 移动应用安全验证标准-韧性
- [ASP.NET Core Best Practices \(Microsoft\)](#)ASP.NET Core 最佳实践 (微软)
- [Resilience in Microservices: Bulkhead vs Circuit Breaker \(Parser\)](#) 微服务中的弹性：舱壁与断路器 (解析器)
- [Bulkhead Pattern \(Geeks for Geeks\)](#) 舱壁模式 (极客论坛)
- [NIST Cybersecurity Framework \(CSF\)](#) NIST 网络安全框架 (CSF)
- [Avoid Blocking Calls: Go Async in Java \(Devlane\)](#)避免阻塞调用：在 Java 中实现异步 (Devlane)

CWE 映射列表

- [CWE-73](#) 文件名或路径的外部控制
- [CWE-183](#) 允许输入的宽松列表

- [CWE-256](#) 密码的明文存储
- [CWE-266](#) 权限分配不当
- [CWE-269](#) 权限管理不当
- [CWE-286](#) 不正确的用户管理
- [CWE-311](#) 敏感数据未加密
- [CWE-312](#) 敏感信息的明文存储
- [CWE-313](#) 在文件或磁盘中明文存储
- [CWE-316](#) 内存中敏感信息的明文存储
- [CWE-362](#) 利用共享资源的并发执行且同步不当（“竞争条件”）
- [CWE-382](#) J2EE 不良实践：使用 `System.exit()`
- [CWE-419](#) 未受保护的主通道
- [CWE-434](#) 危险类型文件的无限制上传
- [CWE-436](#) 解释冲突
- [CWE-444](#) HTTP 请求的不一致解析（“HTTP 请求/响应走私”）
- [CWE-451](#) 用户界面（UI）对关键信息的错误呈现
- [CWE-454](#) 可信变量或数据存储的外部初始化
- [CWE-472](#) 对假定不可变的 Web 参数的外部控制
- [CWE-501](#) 信任边界违规
- [CWE-522](#) 凭证保护不足
- [CWE-525](#) 使用包含敏感信息的 Web 浏览器缓存

- CWE-539 使用包含敏感信息的持久性 Cookie
- CWE-598 使用带有敏感查询字符串的 GET 请求方法
- CWE-602 服务器端安全的客户端实施
- CWE-628 带有错误指定参数的函数调用
- CWE-642 关键状态数据的外部控制
- CWE-646 依赖外部提供文件的文件名或扩展名
- CWE-653 隔离或分区不当
- CWE-656 依赖通过隐匿实现的安全性
- CWE-657 违反安全设计原则
- CWE-676 使用潜在危险函数
- CWE-693 保护机制失效
- CWE-799 交互频率控制不当
- CWE-807 安全决策中对不可信输入的依赖
- CWE-841 行为工作流执行不当
- CWE-1021 渲染的用户界面层或框架的不当限制
- CWE-1022 带有 window.opener 访问权限的指向不可信目标的 Web 链接的使用
- CWE-1125 过大的攻击面

X02:2025 内存管理故障

背景

像 Java、C#、JavaScript/TypeScript (node.js)、Go 以及“安全的” Rust 这类语言是内存安全的。内存管理问题往往出现在 C 和 C++ 等非内存安全语言中。尽管这一类别涉及的常见漏洞披露 (CVE) 数量排在第三位，但在社区调查中的得分最低，在数据中的表现也较差。我们认为，这是因为 Web 应用程序比更传统的桌面应用程序更为普及。内存管理漏洞通常具有最高的通用漏洞评分系统 (CVSS) 分数。

评分表

映射的常见弱点 枚举数量 (CWE)	最大发 生率	平均发 生率	最大覆 盖率	平均覆 盖率	平均加 权漏洞 利用	平均加 权影响	总发生 次数	CVE 总 数
24	2.96%	1.13%	55.62%	28.45%	6.75	4.82	220,414	30,978

描述

当应用程序被迫自行管理内存时，很容易出错。内存安全语言的使用越来越普遍，但全球范围内仍有许多遗留系统在运行，新的底层系统需要使用非内存安全语言，还有一些 Web 应用程序会与大型机、物联网设备、固件以及其他可能被迫自行管理内存的系统进行交互。典型的 CWE 包括 CWE-120 未检查输入大小的缓冲区复制（“经典缓冲区溢出”）和 CWE-121 基于栈的缓冲区溢出。

内存管理失败可能发生在以下情况：

- 开发人员没有为变量分配足够的内存

- 开发人员没有验证输入，导致堆、栈、缓冲区溢出
- 开发人员存储的数据值大于变量类型所能容纳的范围
- 开发人员尝试使用未分配的内存或地址空间
- 开发人员会产生差一错误（从 1 开始计数而不是从 0 开始）
- 开发人员尝试在对象被释放后访问它
- 开发人员使用了未初始化的变量
- 系统出现内存泄漏，或者因其他错误耗尽所有可用内存，直到应用程序崩溃

内存管理故障可能导致应用程序甚至整个系统崩溃，另见 [X01:2025 应用程序弹性失效](#)

如何预防

防止内存管理失败的最佳方法是使用内存安全的语言。例如 Rust、Java、Go、C#、Python、Swift、Kotlin、JavaScript 等。在创建新应用程序时，要努力说服你的组织，值得付出学习成本来切换到内存安全的语言。如果要进行全面重构，在可能且可行的情况下，争取用内存安全的语言重写。

如果无法使用内存安全的语言，请执行以下操作：

- 启用以下服务器功能，这些功能可降低内存管理错误被利用的可能性：地址空间布局随机化（ASLR）、数据执行保护（DEP）和结构化异常处理覆盖保护（SEHOP）。
- 监控应用程序是否存在内存泄漏。
- 非常仔细地验证进入你系统的所有输入，并拒绝所有不符合预期的输入。

- 研究正在使用的语言，列出不安全函数和更安全的函数，并将该列表与整个团队共享。
如果可能，将其添加到安全编码指南或标准中。例如，在 C 语言中，相较于 `strcpy()`，应优先使用 `strncpy()`；相较于 `strcat()`，应优先使用 `strncat()`。
- 如果语言或框架提供内存安全库，请使用这些库。例如：Safestringlib 或 SafeStr。
- 尽可能使用托管缓冲区和字符串，而不是原始数组和指针。
- 参加侧重于内存问题和/或所选编程语言的安全编码培训。告知培训师开发人员关注内存管理故障问题。
- 进行代码审查和/或静态分析。
- 使用有助于内存管理的编译器工具，如 StackShield、StackGuard 和 Libsafe。
- 对系统的每个输入进行模糊测试。
- 如果安全专家进行了渗透测试，请告知安全专家，开发人员担心内存管理故障，并希望安全专家在测试时特别关注这一点。
- 修复所有编译器错误和警告。不要因为程序能够编译就忽略警告。
- 确保底层基础设施定期进行补丁更新、扫描和加固。
- 专门监控底层基础设施，以排查潜在的内存漏洞和其他故障。
- 考虑使用[金丝雀](#)来保护地址栈免受溢出攻击。

示例攻击场景

场景 1: 缓冲区溢出是最著名的内存漏洞，指攻击者向某个字段提交的信息超出其可接受范围，导致为底层变量创建的缓冲区发生溢出的情况。在成功的攻击中，溢出的字符会覆盖栈指针，使攻击者能够向你的程序中插入恶意指令。

场景 2：释放后使用（UAF）的发生频率很高，以至于该漏洞是一种较为常见的浏览器漏洞赏金提交类型。想象一下，一个网页浏览器正在处理操纵 DOM 元素的 JavaScript。攻击者精心设计一个 JavaScript 有效载荷，创建一个对象（例如一个 DOM 元素）并获取对它的引用。通过精心操纵，攻击者促使浏览器释放该对象的内存，同时保留一个指向它的悬空指针。在浏览器意识到内存已被释放之前，攻击者分配一个新对象，该对象占据相同的内存空间。当浏览器尝试使用原始指针时，该指针现在指向攻击者控制的数据。如果这个指针是用于虚函数表的，攻击者就可以将代码执行重定向到他们的有效载荷。

场景 3：一种网络服务接收用户输入，但未对其进行适当验证或净化，然后直接将其传递给日志记录函数。用户的输入通过 `syslog(user_input)` 而非 `syslog("%s", user_input)` 传递给日志记录函数，后者没有指定格式。攻击者发送包含 `%x` 等格式说明符的恶意负载，以读取堆栈内存（敏感数据泄露），或使用 `%n` 写入内存地址。通过将多个格式说明符链接在一起，他们可以绘制出堆栈，找到重要地址，然后覆盖这些地址。这将构成一种格式字符串漏洞（不受控制的字符串格式）。

注意：现代浏览器采用多种防护措施来抵御此类攻击，包括[浏览器沙箱](#)、地址空间布局随机化（ASLR）、数据执行保护/不可执行（DEP/NX）、重定位只读（RELRO）和位置无关可执行文件（PIE）。对浏览器实施内存管理故障攻击并非易事。

参考文献

- [OWASP community pages: Memory leak, Doubly freeing memory, & Buffer Overflow](#)
OWASP 社区页面：内存泄漏，双重释放内存，以及缓冲区溢出

- [Awesome Fuzzing: a list of fuzzing resources](#) 出色的模糊测试：模糊测试资源列表
- [Project Zero Blog](#) 零号项目博客
- [Microsoft MSRC Blog](#) 微软 MSRC 博客

CWE 映射列表

- [CWE-14](#) 编译器移除用于清除缓冲区的代码
- [CWE-119](#) 对内存缓冲区边界内操作的不恰当限制
- [CWE-120](#) 未检查输入大小的缓冲区复制（“经典缓冲区溢出”）
- [CWE-121](#) 基于栈的缓冲区溢出
- [CWE-122](#) 基于堆的缓冲区溢出
- [CWE-124](#) 缓冲区下溢（“缓冲区下写”）
- [CWE-125](#) 越界读取
- [CWE-126](#) 缓冲区过度读取
- [CWE-190](#) 整数溢出或回绕
- [CWE-191](#) 整数下溢（回绕或环绕）
- [CWE-196](#) 无符号到有符号转换错误
- [CWE-367](#) 检查时间与使用时间（TOCTOU）竞争条件
- [CWE-415](#) 双重释放
- [CWE-416](#) 释放后使用
- [CWE-457](#) 使用未初始化的变量

X02:2025 内存管理故障

- [CWE-459 净化不彻底](#)
- [CWE-467 对指针类型使用 sizeof\(\)](#)
- [CWE-787 越界写入](#)
- [CWE-788 缓冲区结束后的内存位置访问](#)
- [CWE-824 未初始化指针的访问](#)

的不当信任（“氛围编程”）

背景

目前，全世界都在谈论和使用人工智能，软件开发人员也不例外。尽管目前还没有与人工智能生成的代码相关的通用漏洞披露（CVE）或通用弱点枚举（CWE），但众所周知且有记录显示，人工智能生成的代码往往比人类编写的代码包含更多漏洞。

描述

安全专家发现软件开发实践正在发生变化，其中不仅包括在人工智能辅助下编写的代码，还包括几乎完全在没有人类监督的情况下编写和提交的代码（通常被称为“氛围编程”）。就像从不加思索地从博客或网站上复制代码片段从来都不是一个好主意一样，这种情况下的问题更为严重。优质、安全的代码片段过去和现在都很罕见，而且由于系统限制，人工智能在统计上可能会忽略它们。

如何预防

安全专家敦促所有编写代码的人在使用人工智能时考虑以下几点：

- 开发人员应当能够阅读并完全理解你提交的所有代码，即使这些代码是由人工智能编写的或从在线论坛复制的。开发人员要对自己提交的所有代码负责。
- 开发人员应该彻底检查所有 AI 辅助编写的代码是否存在漏洞，理想情况下，既要亲自检查，也要使用为此目的设计的安全工具（如静态分析工具）。可以考虑采用《OWASP cheat Sheet 系列：安全代码审查》中描述的经典代码审查技术。

的不当信任（“氛围编程”）

- 理想情况下，开发人员可以自己编写代码，让人工智能提出改进建议，检查人工智能的代码，然后让人工智能进行修正，直到开发人员对结果满意为止。
- 考虑使用检索增强生成（RAG）服务器，结合开发人员自己收集和审核的安全代码样本及文档，例如组织的安全编码指南、标准或政策，并让 RAG 服务器执行任何政策或标准。
- 考虑购买能为开发人员所选的人工智能实施隐私和安全防护措施的工具。
- 考虑购买一个私有的人工智能系统，理想情况下要签订合同协议（包括隐私协议），规定该人工智能不得使用您组织的数据、查询内容、代码或任何其他敏感信息进行训练。
- 可以考虑在开发人员的集成开发环境（IDE）和人工智能之间部署一个模型上下文协议（MCP）服务器，然后对其进行设置，以强制使用开发人员所选的安全工具。
- 在软件开发生命周期中实施相关政策和流程，以告知开发人员（及所有员工）在组织内部应如何以及不应如何使用人工智能。
- 创建一份优质且有效的提示词列表，其中需考虑 IT 安全最佳实践。理想情况下，这些提示词还应符合贵公司内部的安全编码指南。开发人员可将这些提示词作为其程序开发的起点。
- 人工智能很可能会成为系统开发生命周期每个阶段的一部分，包括如何有效且安全地使用它。请明智地使用它。
- 实际上，不建议将氛围编码用于复杂函数、业务关键程序或长期使用的程序。
- 实施针对影子人工智能使用的技术检查和保障措施。

的不当信任（“氛围编程”）

- 对开发人员进行有关公司政策、安全使用人工智能以及在软件开发中运用人工智能的最佳实践的培训。

参考文献

- [OWASP Cheat Sheet: Secure Code Review](#)OWASP cheat sheet: 安全代码审查

CWE 映射列表

- 无