



2026 年软件供应链安全 文献报告

AI 采纳与治理问题重塑软件安全格局



目录

关于本报告	2
案例研究简介	4
您的软件供应链中有什么？	7
开发组织中使用的编程语言数量	8
每年每种包类型的新增软件包	11
组织使用的主流软件包技术	13
AI 仓库在组织内的增长	15
热门库	16
新软件包引入组织的速度	19
组织内 AI 库的采用	20
您软件供应链中不断加剧的风险	24
在给定的技术或包类型中发现的漏洞	25
最常见的漏洞类型	27
2025 年高关注度的通用漏洞披露的常见漏洞影响	29
引入您软件供应链中漏洞的严重性	30
开发者工具的攻击面	34
某些恶意软件包比其他软件包危害更大	35
配置错误、失误以及人为错误的影响	36
二进制制品中密钥泄露的现状	37
当今组织如何实施安全措施	40
1. 来源限制	41
2. 扫描，扫描，扫描	43
3. 在整个应用程序流水线中建立可见性和控制	47
4. 新的治理缺口：开发人员工具	51
5. 成本是多少？	53
AI 模型与智能体要素	58
组织如何使用 AI/ML 模型	59
治理模型制品	60
治理 AI 输入和输出	61
对 AI 建议的安全修复的信任	62
检测影子 AI	63
方法论	65
JFrog 平台使用数据	65
JFrog 安全研究团队的分析	65
专项调研结果	66
关于 JFrog	67
关于前瞻性陈述的警示性说明	69



关于本报告

如今的软件供应链与一年前相比已大不相同。AI 已从实验阶段发展为一种结构性力量，正在改变代码的编写方式、依赖项的组装方式，以及软件从开发走向生产的速度。在后 AI 时代，这些挑战只会不断加速，这让大多数技术团队不禁思考：我们如何才能跟上所有这些变化？

JFrog 2026 年软件供应链安全文献报告通过确定风险的范围扩大领域（即依赖项、二进制文件、AI 制品）、控制措施在何处失效，以及攻击者当前正在利用哪些漏洞，有助于让讨论立足于现实。更重要的是，它将噪音转化为可操作的优先事项：



风险优先级确定

根据持久风险和新风险所在位置确定修复优先级



战略投资

基于治理工作在哪些方面跟上了步伐、在哪些方面未能跟上，确定投资应聚焦于何处



董事会理由说明

如何利用更广泛的软件供应链和开发格局背景，向董事会论证投资的合理性

本报告旨在帮助工程和安全专业人员从被动修补转向对软件风险的系统性控制，并以数据为每一项决策提供依据。

作为全球数千家组织以及超过 80% 的财富 100 强企业的记录系统，JFrog 软件供应链平台可深入洞察数十亿软件制品。

本报告是对这些数据的分析结果，并结合了 JFrog 安全研究团队开展的独立漏洞研究，以及来自八个国家/地区 1,508 名安全、开发和运营专业人士委托第三方调查的回复。

希望本报告能为您带来价值，欢迎您提出任何反馈意见，您可通过 data_report@jfrog.com 与我们分享。



案例研究简介

AI 正以前所未有的速度重塑软件供应链，但治理框架却难以跟上步伐。由 AI 生成代码、模型制品和智能体工具推动的新一波风险正在形成，其蔓延速度已快于应对速度。这一趋势正在导致所宣称的防护信心与实际可衡量的风险敞口之间的差距不断扩大，形成一种“掌控错觉”。

二进制文件的冲击已经到来

您的软件供应链不只在增长；而是在成倍扩张。在 AI 辅助开发的推动下，制品创建量已达到创纪录的水平，其构成也正在发生变化，这反映出软件构建方式正在经历根本性转变。

- 截至 2025 年底，JFrog 平台在 JFrog SaaS 客户中托管了 182 亿个制品，较 2024 年增长 136%。
- Hugging Face 于 2025 年发布了 140 万个新软件包，使其成为新软件包的第二大来源，仅次于 Docker Hub。
- npm 超越 Maven，成为流量最高的软件包生态系统。与此同时，PyPI 超越了 YUM，表明 AI/机器学习工作负载正在取代传统基础设施。这不仅仅是流量的转变；智能体工作流和 AI 辅助开发更倾向于脚本语言，而当今承载最多流量的生态系统正反映了这一转变。
- OpenVSX 上的 AI 编码智能体扩展同比增长 262%，新发布的软件包数量从约 1,000 个增至 3,803 个。



风险正在传统和新型攻击面上加速蔓延

威胁态势正在向各个方向不断扩大。尽管 2025 年是 npm 用户有记录以来最危险的一年，但 AI 模型和智能体开发工具这一庞大的新攻击面也同时出现了。

- 2025 年公开披露了超过 48,000 个新的通用漏洞披露，较 2024 年增长了 20%。JFrog 安全研究团队将这部分增长归因于未采用安全编码实践的 AI 生成代码，这导致了 XSS、SQL 注入和其他注入漏洞等已存在数十年的漏洞激增。
- 恶意 npm 软件包激增 451%，达到 171,592 个实例，而起因则是三次重大劫持活动，导致受感染的下载量超过两百万。
- JFrog 安全研究团队在 Hugging Face 上发现了 495 个恶意模型，这些模型携带了包括反向 shell、凭据收集和系统命令执行等可执行的有效负载。
- JFrog 还在 ClawHub 和 Skills.sh 制品库中识别出 969 个恶意 AI 智能体技能，过滤掉了许多误报的恶意“发现”。这些发现表明，在检查新技术的安全问题时，攻击者常常击败防御者。
- 在 OpenVSX 上检测到 56 个恶意扩展，这是本报告首次对这一攻击面进行跟踪。作为一个快速增长的生态系统，OpenVSX 正成为一个越来越有吸引力的目标。

这些模式并未在 2025 年终结。2026 年初，JFrog 由 AI 驱动的 RepoHunter 机器人在攻击者利用之前，于广泛使用的开源项目中识别出 13 个关键的 CI/CD 漏洞，涉及的项目包括 Ansible、CNCf Telepresence 和 JavaScript 标准制品库。

其攻击机制与 Shai-Hulud 攻击活动中使用的机制完全相同：CI/CD 流水线中的错误配置使攻击者能够访问机密、发布令牌和云凭据。Shai-Hulud 的攻击机制至今仍广泛存在于开源生态系统中。

并非报告的所有漏洞都如其表面所示

基于数量的故障分类正在让安全团队不堪重负。需要追踪的通用漏洞披露越多，并不意味着实际风险越大，而是意味着噪音更多。

就连治理机制本身也需要治理

尽管各组织正在迅速采用 AI 开发工具、模型和协议，但针对这些新领域的治理框架仍处于起步阶段，或仅停留在愿景层面。AI 在软件供应链中已不再是一个新兴的考量因素。它就是供应链本身。

- 66% 的已分析通用漏洞披露具有较低的适用性率 (0–20%)，这意味着即使代码中存在漏洞，在实践中也很少满足利用它所需的条件。仅 12% 的漏洞可在企业真实环境中被轻易利用。
- 只有 40% 的受访者采用了恶意软件包检测（与去年持平），而只有 28% 的组织在积极使用机密检测。威胁规模持续上涨，但安全检测覆盖面并未同步扩容。事实上，威胁量增长最快的类别，恰恰是安全工具覆盖最少的类别。
- 41% 的组织正在积极使用 AI 和 ML 库（将应用程序连接到 AI 模型和服务的软件包），高于 2024 年的 34%。当前，组织平均管理的这类软件包数量比去年增加了 47%，因为团队正从依赖单一 AI 服务转向多 AI 共存的现实格局，并同时基于多个服务进行构建。
- 53% 的组织以某种形式自行托管 AI 模型，并从 Hugging Face 和类似的公共注册中心拉取模型，而这些公共注册中心内已检测到 495 个恶意模型。然而，97% 的组织声称模型治理已经过认证，但恶意模型数据令这一数字产生了疑问。
- 59% 的受访者表示具备完整的生产来源可见性。然而，48% 的受访者仍需要一周或更长时间来生成合规审计证明。缺乏问责制的可见性不是真正的治理。
- 23% 的开发人员仅经过快速审查，就会将 AI 建议的安全修复视为近乎确定无疑的方案。18% 的组织对其开发人员环境中的 IDE 扩展和 MCP 服务器没有有效的治理措施，要么根本没有任何策略，要么有策略却未实施。

您的软件供应链中有什么？

企业软件供应链的构成发生了变化，这反映出软件构建方式已出现根本性转变。AI 已不再只是组织正在为之做准备的一个话题。它正切实地影响着每天有哪些代码被编写、哪些组件被集成、哪些内容进入生产交付环节。

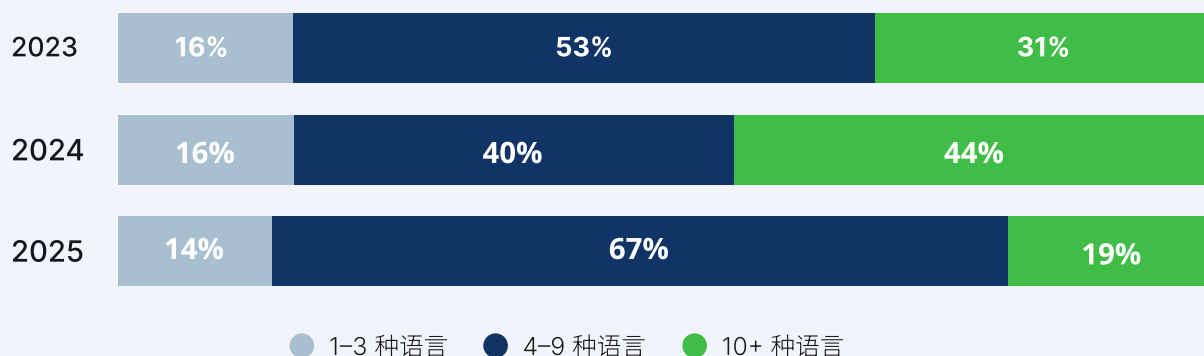


开发组织中使用的编程语言数量

各组织的语言使用分布呈现出明显的整合趋势。14% 的组织表示仅使用 1-3 种语言，67% 使用 4-9 种，19% 使用 10 种或更多。



在您的整个软件开发组织中使用了多少种编程语言？



这表明持续多年的扩张趋势出现了急剧逆转。使用 7 种或更多语言的组织占比已从 2024 年的 64% 下降至 45%。在高端区间，这种收缩更为明显：使用 10 种或更多语言的组织占比从 2024 年的 44% 降至仅 19%。

最合理的解释是有意进行整合：组织正在精简其技术栈，减少多年快速扩张期间积累的蔓延现象，而这一趋势又因 AI 编码助手而加速，因为它们可能默认采用某些语言和框架。

语言越少，可能意味着攻击面更容易治理，但风险也会越集中。当企业开发运行于两个或三个生态系统之上时，基础库中的单个严重漏洞所波及的范围会比以往任何时候都更广。

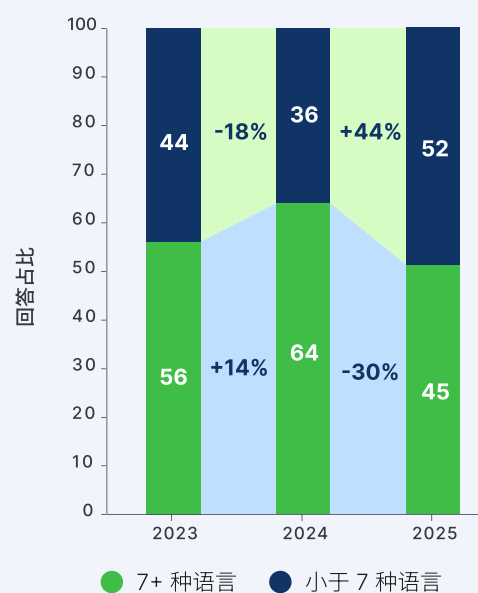
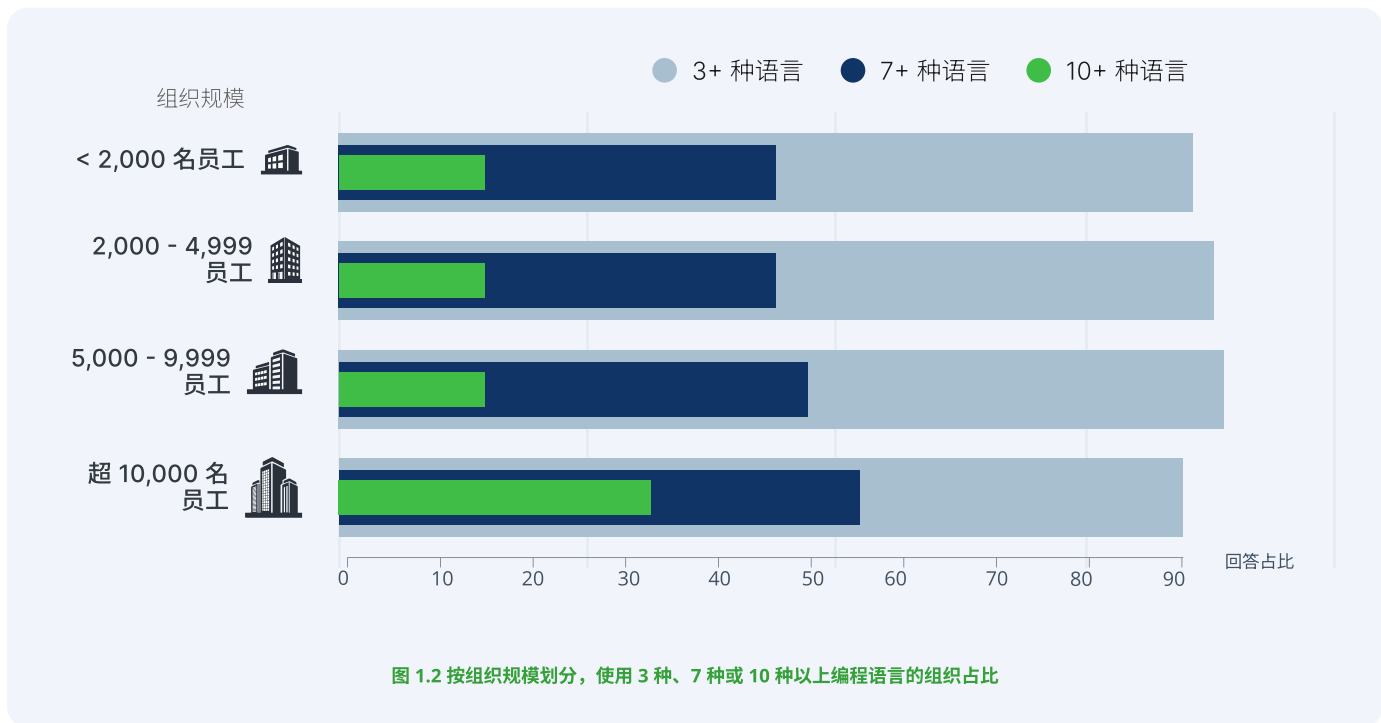


图 1.1 组织所用语言数量的同比变化



按组织规模来看，整合趋势总体上保持一致，但有一个值得注意的例外情况。拥有 1,000–9,999 名员工的组织在使用 7 种以上语言的占比方面紧密集集中在 42%–47% 的范围内，表明该规模的企业在语言使用行为上相对一致。异常情况出现在最大规模层级：拥有 10,000 名或以上员工的组织报告使用 7 种以上语言的占比为 50%，是 2025 年各规模组中最高的，这与去年的模式完全相反，去年最大规模组织这一占比最低。

这一转变在接入 10 种以上编程语言的企业中尤为突出，其中在 10,000+ 规模的组织，该项占比 30%，其余体量企业均值仅 14%。我们在 2024 年观察到的规模倒逼标准化拐点似乎已经发生转变，或者最大型的组织只是在添加新技术栈的速度上超过了其对这些技术栈进行合理化整合的速度。

资历差异带来了显著的认知差距。64%的首席级高管受访者表示其组织内部使用了 7 种以上的语言，而个人贡献者中这一比例仅为 35%，相差 29 个百分点，这与我们在 2024 年观察到的情况一致。高层领导者对各团队间所使用的语言组合可能具有更广泛的可见性，或者他们统计的是自己已批准使用的语言，而非实际部署的语言。无论是哪种情况，这种差异都足够显著，表明组织自行报告的语言复杂性会因您在层级结构中所处的位置不同而呈现出明显差异。



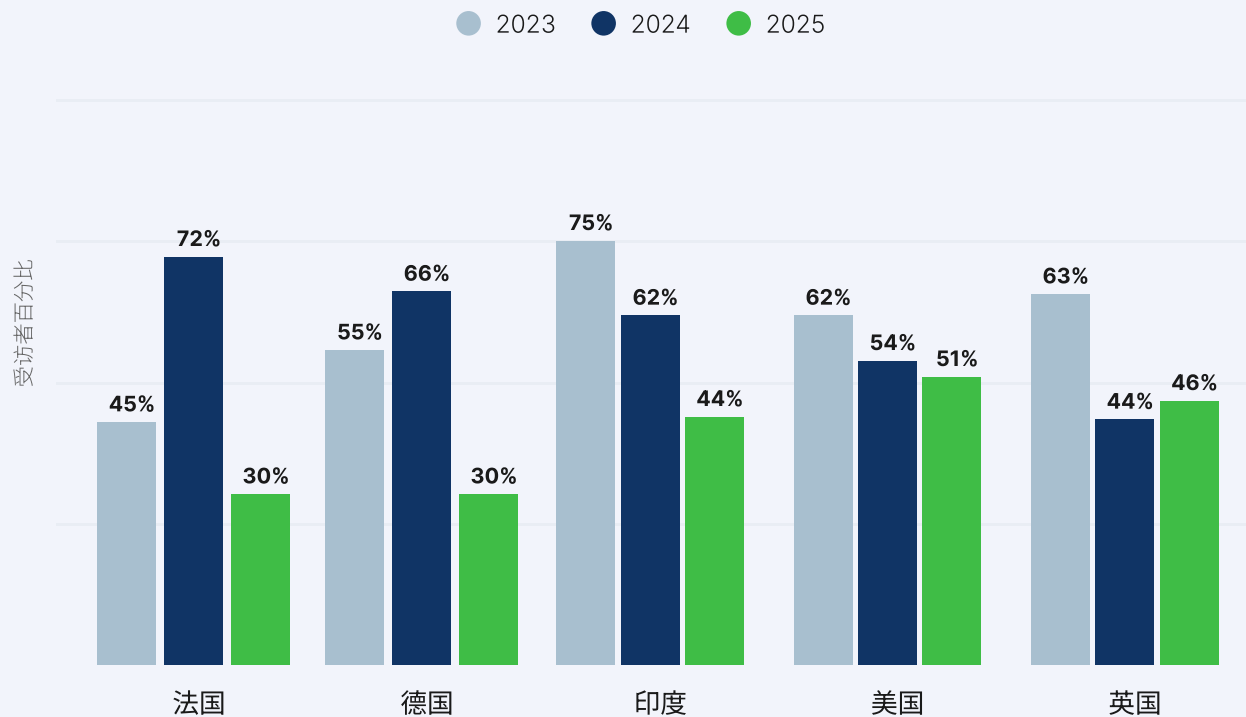


图 1.4 各国家/地区使用 7 种以上语言的占比

国家或地区层面的数据或许最能说明今年调查结果中最引人注目的情况。在受调查国家或地区中，美国目前报告的使用 7 种以上语言的占比最高，达到 51%；这并不是因为美国的组织变得更加复杂，而是因为其他所有国家或地区的这一比例下降得更快。在全部三年调查期间，美国一直呈现逐步且持续的下降趋势 (62% → 54% → 51%)，而其他市场则出现了波动大得多的起伏。

按绝对值计算，印度的降幅最为显著：从 2023 年的 75% 降至 2024 年的 62%，再降至 2025 年的 44%；三年内下降了 31 个百分点。德国和法国则呈现出不同的走势：两国均在 2024 年飙升至 60% 的中段区间，随后大幅下滑，如今两国均为 30%，在所有受访国家或地区中并列垫底。根据记录，法国在所有国家或地区及所有年份的数据集中出现了单次调查的最大波动，从 2024 年 72% 的峰值下降了 42 个百分点。

这两个市场的同步下滑（以及下滑速度）可能反映了欧洲监管压力的影响，因为在《欧盟网络弹性法案》等框架下运营的组织正为了应对新的软件供应链义务而对其技术栈进行合理化调整。

每年每种包类型的新增软件包

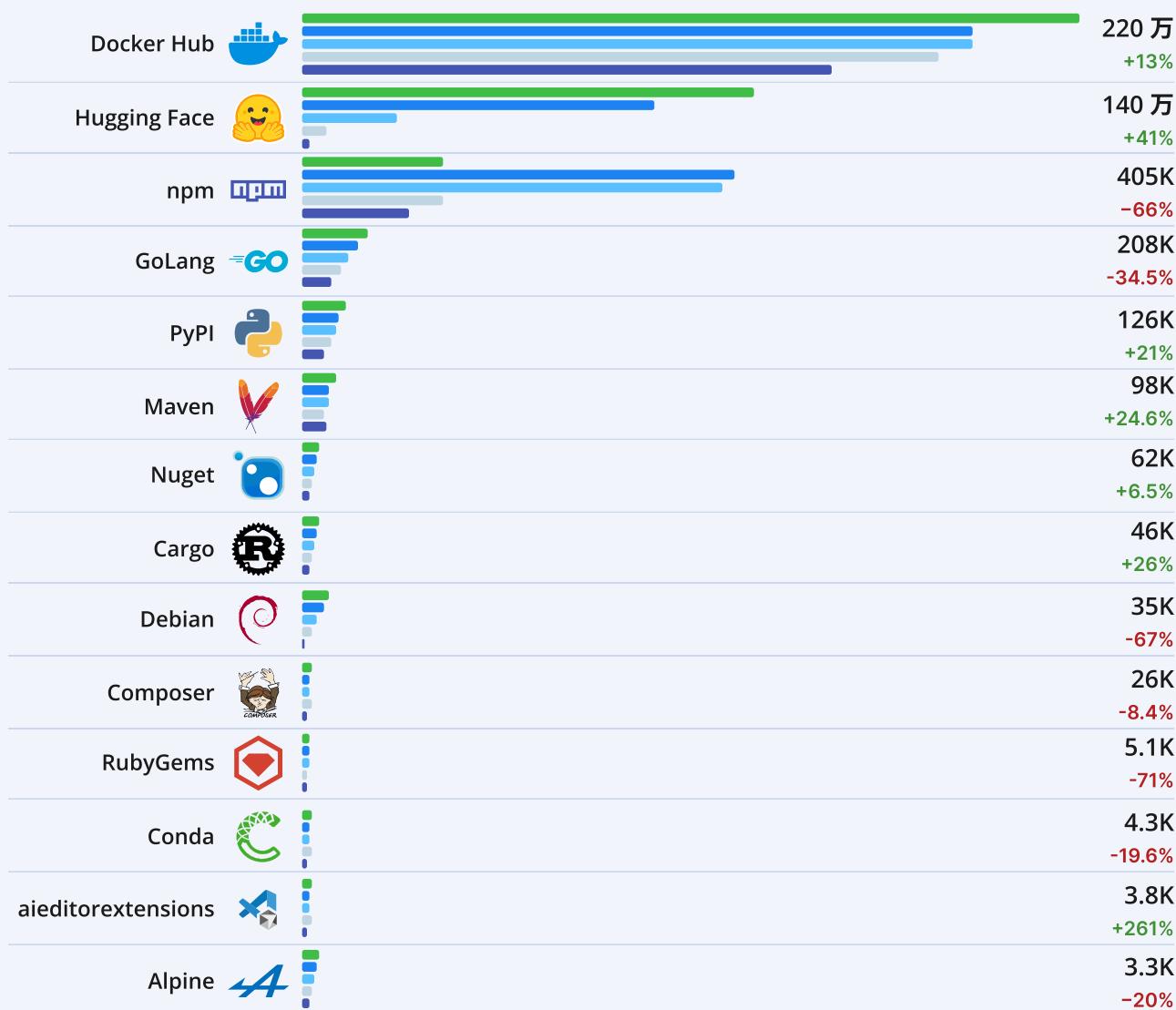


图 2. 按包类型划分的每年新增 OSS 软件包数量，2021-2025 年（JFrog Catalog 数据库，2025 年）

2021 2022 2023 2024 2025

JFrog Catalog 对公共注册中心的调查数据，为软件供应链中已成为最具影响力的一大趋势揭开了新的篇章：AI/机器学习作为独立生态系统的崛起。

按原始数量计算，在我们跟踪的所有注册中心中，Docker Hub 仍然是新软件包的最大来源，2025 年新增 220 万个新软件包，高于 2024 年的 190 万个 (+16%)。这种增长与 Docker Hub 作为事实上的容器制品库这一角色一致：稳定、可预测，而且在很大程度上反映了容器化工作负载的自然扩张，而非任何结构性转变。

更值得关注的态势围绕 Hugging Face 展开。Docker Hub 的增长一直是渐进式的，而 Hugging Face 则从一个快速增长的另类平台演变为一个本质上截然不同的生态系统类别。到 2025 年，其新增模型数量已超过 140 万个，在前一年约 4 倍增长的基础上，同比增长了 41%。如今，Hugging Face 在体量上可与 Docker Hub 媲美，同时它也代表着一种全新的制品类别：AI 模型和数据集，它们在来源、许可和完整性方面面临独特的挑战，而传统的软件包治理并非为应对这些问题而构建。

重要的是，AI 采用在整个生态系统都留下了足迹，而不仅仅是在特定于 AI 的注册中心。扩展 AI 工作负载的组织在 Docker、PyPI 和 npm 方面呈现出相应的增长，这是 AI 流水线运行所依赖的基础设施层。从这个意义上说，传统生态系统的增长本身就是 AI 活动上升的信号，即使所涉及的制品并非特定于 AI。

npm 的情况则更为复杂。在 2024 年新软件包数量激增至 119 万个之后，该注册中心在 2025 年新增了超 40 万个软件包，同比下降 66%。原因尚不完全清楚，但一种可能性是 npm 在发布时检测和屏蔽垃圾及低质量软件包方面变得更加有效。2024 年 npm 软件包数量的激增恰逢自动化软件包活动异常频繁的时期，而注册中心层面管控的收紧或许可以解释其回归较低数量的原因。可以确定的是，新软件包数量的下降并不反映 npm 采用率的下降。JFrog 平台流量数据显示，2025 年，npm 在请求量方面超越 Maven，成为使用量最高的生态系统，这是自我们开始追踪以来其最高的相对排名。

PyPI 持续稳步攀升，在 2025 年新增了 12.6 万个软件包，高于前一年的 10.4 万。这种增长是自然发生的，并且在五年的全部数据中始终保持一致，反映出 Python 作为数据科学、自动化和 AI 工具默认语言的持续扩展。这里没有显著的拐点，而这一点本身就很有意义。Python 生态系统之所以不断发展，是因为开发人员的持续选择，而不是因为某个单一的催化事件。

Maven 多年来首次出现显著加速，2025 年增长了 24%，新增软件包达到 9.8 万，高于此前四年相对持平的 6.3 万-7.9 万区间。这一时机与基于 Java 的 AI 和后端工具的快速扩张相吻合，同时表明，长期以来一直被视为成熟且稳定的 Java 生态系统，正被这股同样重塑所有其他注册中心的 AI 浪潮重新拉回到活跃增长之中。

Rust 软件包注册中心 Cargo 的新增软件包数量已从 2021 年的 2 万个增长至 2025 年的 4.6 万个，四年间增长了 129%，其复合增长率也是此数据集中所有生态系统最稳定的。正如 2025 年报告中所指出，这一发展轨迹反映了政府和行业持续推动采用内存安全语言所带来的压力，而 Cargo 持续不间断的上升趋势表明，这种压力正在转化为实际采用，而非停留在口头意向层面。



组织使用的主流软件包技术

* 请求数 = 上传数和下载数的汇总

包类型	制品库数量	制品	市场份额 (*请求占比)	流量同比增长
npm	143,244	1.62B	33.17%	+35.5%
Maven	315,522	5.93B	30.13%	+20.4%
Docker	305,238	5.90B	14.15%	+31.2%
Generic	287,044	2.65B	7.18%	+126.8%
PyPI	79,232	182.9M	3.31%	+79.4%
YUM	35,298	55.4M	2.49%	+18.0%
Cargo	2,862	140 万	0.28%	+381.2%
OCI	31,708	108.7M	0.20%	+79.2%
HelmOCI	8,350	420 万	0.20%	+565.6%
Hugging Face	2,387	44K	0%	+1190.78%

图 3. 所使用的技术，以及每种技术的操作次数、制品库数量和所存储制品的总大小（JFrog SaaS 平台数据，2024–2025 年）

在所有被跟踪的软件包生态系统中，JFrog SaaS 上的平台总流量同比增长了 33.5%。在原本已创纪录的基数之上，这一增长率反映了现有开发活动的自然扩展以及新型工作负载的涌现，尤其是大规模拉取模型制品的 AI/ML 流水线，以及进一步加快这一速度的智能体工作流。

该报告最重要的发现是史无前例的：npm 取代 Maven 成为使用最多的软件包生态系统，这意味着就请求量而言，JavaScript 已超越 Java，成为占主导地位的企业级软件包生态系统。

npm 目前占平台总流量的 33.2%，而 Maven 占 30.1%。npm 的流量同比增长了 35.5%，超过了 Maven 的增长 (+20.4%) 和整体市场增长 (+33.5%)，在 Maven 份额下降的同时，npm 的份额却在上升。虽然这两个生态系统都没有萎缩，且总体规模实际上还在增长，但 npm 在这个更大的市场中占据了更大的份额。这不仅仅是流量的转移。

智能体工作流和 AI 辅助开发可能更倾向于使用脚本语言；JavaScript 和 Python 的迭代速度更快，更易于非开发人员使用，也是许多 AI 编码助手的默认输出语言。企业级 Java 仍保有市场份额，但依托其开发的新软件层运行在 npm 和 PyPI 上。



同样的态势也解释了为何 PyPI 超过了 YUM。PyPI 增长了 79.4%，超过整体市场增速的两倍，而 YUM 仅增长了 18%，远低于市场水平。从绝对值来看，YUM 仍在增长，但 Python 的采用速度要快得多。

某些生态系统虽然显示出显著的百分比增长，但在整体流量中所占份额仍然很小。Cargo/Rust 的流量增长了 381%，仓库增长了 127%，这与其五年来新软件包增长趋势以及政府持续推动采用内存安全开发语言的要求是一致的。Rust 目前仍仅占平台总流量的 0.3%，但其在一年内几乎增长了四倍，并且没有显示出趋于平稳的迹象。这意味着：如今，在 Python 上运行的人工智能/机器学习工作负载在企业软件活动中的占比，已经超过传统 Linux 软件包管理。这是一个关于企业开发未来走向的重要信号。

OCI 制品库增长了 3,578%，OCI 流量增加了 79%，另外 HelmOCI 制品库增长了 411%，HelmOCI 流量增加了 565%，这反映了 Kubernetes 应用程序打包方式正在发生结构性转变。Kubernetes 社区在平台层面将 OCI 设为 Helm Charts 的默认打包格式；与其说是组织选择了 OCI，不如说是生态系统在设计层面主动向 OCI 迁移。这些数据反映出，这一转变在 2025 年达到了临界规模，而 JFrog 对 Helm Legacy 和 OCI 协议的双重支持使客户的迁移变得简单直接。

同样值得注意的是，Hugging Face 在 JFrog 平台上的增长规模。流量同比增长了 1,190%，存储的制品数量从 12,750 增长到 43,823，增幅达 244%。这些数据共同表明，各组织不只是搭建了模型基础设施，而且正在积极使用它。AI 模型正大规模进入企业软件供应链。



AI 仓库在组织内的增长

2025 年，JFrog 专用的 AI 和机器学习制品库类型实现了大幅增长。随着各组织从“试用 AI”转向“将模型主动集成到开发流水线”，管理这些制品所需的基础设施也在随之扩展，由此带来的规模化追溯来源、许可合规与完整性保障等治理挑战同样与日俱增。请注意，这些数据仅反映了明确标记为“AI”类仓库的增长；而 Docker、PyPI 和 Helm Chart 中 AI 相关制品增长，其实际覆盖范围远超这些数字本身所能呈现的广度。

公共生态系统的增长速度已经超过了组织对其进行治理的能力。根据 JFrog Catalog 的分析，Hugging Face 在 2025 年发布了 140 万个新模型，使其成为我们所跟踪的新软件包的第二大来源，仅次于 Docker Hub。OpenVSX 上的 AI 编辑器扩展从 2023 年的 996 个增长到 2024 年的 1,051 个，并在 2025 年跃升至 3,803 个，单年增长 262%，这反映出开发人员工具层向 AI 原生工作流转变的速度之快。

来自企业环境内部的 JFrog 平台数据也说明了同样的情况。JFrog 平台上的 Hugging Face 流量同比增长了 1,190%。为了更直观地理解这一数据：在 2025 年的全部四个季度中，Hugging Face 流量平均环比增长 182%，相比之下，同期 npm 涨幅为 13%，Docker 为 11%，Maven 为 6%。2025 年全年，三大传统生态系统保持了稳定且可预测的增长。Hugging Face 的增长则完全处于另一个数量级。

季度	Hugging Face	npm	Docker	Maven
第 1 季度	40%	20%	-3%	2%
第 1 季度	543%	10%	12%	7%
第 3 季度	4%	9%	18%	10%
第 4 季度	143%	11%	17%	6%
2025 年均值	183%	13%	11%	6%

图 4. 按生态系统划分的环比增长（JFrog 数据库，2025 年）

另外值得注意的是，JFrog 在 2025 年新增了对两种 AI 专用新软件包类型的支持，即 NIM (NVIDIA Inference Microservices) 和 AI 编辑器扩展，二者都是今年首次出现在平台数据中。

AI 制品正成为企业软件供应链的标准组成部分，而用于管理它们的治理框架仍在不断完善。



热门库

#	 Docker	 Maven	 PyPI	 npm
1	node	commons-io:commons-io	charset-normalizer	debug
2	python	org.slf4j:slf4j-api	certifi	@types/node
3	nginx	org.ow2.asm:asm	urllib3	semver
4	postgres	commons-codec:commons-codec	setuptools	caniuse-lite
5	alpine	org.apache.commons:commons-lang3	requests	get-intrinsic
6	maven	org.apache.commons:commons-text	packaging	electron-to-chromium
7	busybox	org.codehaus.plexus:plexus-utils	typing-extensions	@babel/parser
8	redis	com.fasterxml.jackson.core:jackson-databind	idna	@babel/types
9	openjdk	com.fasterxml.jackson.core:jackson-core	PyYAML	undici-types
10	docker	com.fasterxml.jackson.core:jackson-annotations	click	browserslist
11	ubuntu	com.google.guava:guava	Pygments	@babel/generator
12	golang	org.apache.maven:maven-model	numpy	@babel/traverse
13	mongo	org.codehaus.plexus:plexus-classworlds	cffi	@babel/helper-validator
14	mysql	org.codehaus.plexus:plexus-archiver	MarkupSafe	es-object-atoms
15	centos	org.apache.maven:maven-core	Jinja2	update-browserslist
16	elasticsearch	org.codehaus.plexus:plexus-component	cryptography	qs
17	eclipse-temurin	org.junit.platform:junit-platform-commons	attrs	@babel/compat-data
18	rabbitmq	net.bytebuddy:byte-buddy	pytz	brace-expansion
19	gradle	org.yaml:snakeyaml	pip	@babel/helpers
20	ruby	org.apache.maven:maven-plugin-api	tzdata	resolve

图 5. JFrog SaaS 中 Docker、Maven、PyPI、npm 生态系统中下载量排名前 20 的软件包 (JFrog 数据库, 2025 年)

公共下载次数可告知您某个软件包被获取的频率。但它们无法告知您实际上有多少组织依赖它。单个 CI 流水线每天可能获取同一软件包数千次，使注册中心指标远超实际采用情况数值。我们的研究并非统计下载次数，而是分析在数千个 JFrog SaaS 客户帐户中被主动请求至生产环境的内容，从而更准确地呈现企业实际运行的情况。

Docker

对于 Docker 而言，2025 年官方 Node.js 镜像已超过 Alpine，成为请求量最高的镜像，这与前几年 Alpine 一直作为首选最小基础镜像占据主导地位的情况有所不同。前 20 名仍然反映了使用最广泛的操作系统和运行时环境，这与它们在多阶段构建中作为父镜像的角色一致。

2024 年的三个显著变化：hello-world 跌出榜单；而 hello-world 去年的高排名反映出新开发人员入职引导和概念验证活动的数量相当可观。它跌出榜单可能表明客户群正在走向成熟。Grafana 和 Amazon Corretto 也跌出榜单，取而代之的是 mysql、elasticsearch、rabbitmq 和 gradle，这标志着向数据基础设施和后端服务组件的迁移。这表明组织正在引入更完整的应用程序堆栈，而不仅仅是语言运行时。

Maven

Maven 前 10 名基本逐年保持稳定。更重要的内容在下半部分。JUnit 4 (junit:junit) 已完全跌出榜单，被 org.junit.platform:junit-platform-commons 取代。这清楚地表明，Java 生态系统从 JUnit 4 到 JUnit 5 的长期迁移现已体现在企业生产环境中，而不仅仅是新项目中。

Apache HttpComponents (httpcore、httpClient) 也已跌出，这很可能是因为它已被现代 Java 版本内置的较新 HTTP 客户端实现所取代。org.yaml:snakeyaml 进入前 20 榜单，位列第 19。SnakeYAML 历史上存在大量 CVE，其中最值得注意的是 CVE-2022-1471 (CVSS 9.8 — 构造函数反序列化 RCE)；这一问题直到 2023 年 2 月发布的 2.0 版本中才得到修复。它出现在前 20 榜单中，表明它仍广泛部署于各类企业 Java 环境中。



PyPI 前 20 名是四大生态系统中最稳定的，20 个软件包中有 16 个出现在去年的榜单中。这一新入榜情况颇能说明问题：Jinja2（第 15 名）和 Pygments（第 11 名）是文档工具和 AI 代码生成框架最常引入的模板和语法高亮库。它们的上升趋势与本报告其他部分所记录的基于 Python 的 AI 工具增长情况一致。wheel、importlib-metadata 和 zipp 跌出了榜单；这些较旧的打包基础设施功能如今越来越多地由现代 pip 版本原生提供，而 pip 本身则首次进入前 20 名，位列第 19。



在所有生态系统中，npm 前 20 名的变动幅度最大，共有 9 个新软件包进入榜单。@babel 生态系统的上榜数量从 2024 年的 3 个增至 2025 年的 7 个，目前占据第 7、8、11、12、13、17 和 19 位。Babel 已成为主导性的 JavaScript 转译工具链，可将现代 JS 代码转换为与浏览器兼容的格式。由于它作为传递依赖广泛存在，这意味着大多数 JavaScript 项目无论是否明确直接依赖 Babel，都会引入多个 Babel 软件包。

另外两个新上榜者也值得关注：undici-types 排名第 9，它标志着 Node.js 内置 fetch 实现已开始广泛取代第三方 HTTP 库。get-intrinsic 和 es-object-atoms 反映出，core-js 底层 JavaScript 原语作为传递依赖在整个 npm 依赖关系图中日益普遍。与 2024 年相比，有 9 个软件包跌出榜单，其中包括 chalk、commander、glob 和 minimatch，这些曾经广泛使用的 CLI 实用工具正在被 Node.js 原生的等效项所取代，例如 util.styleText。

组织引入新 OSS 软件包的速率

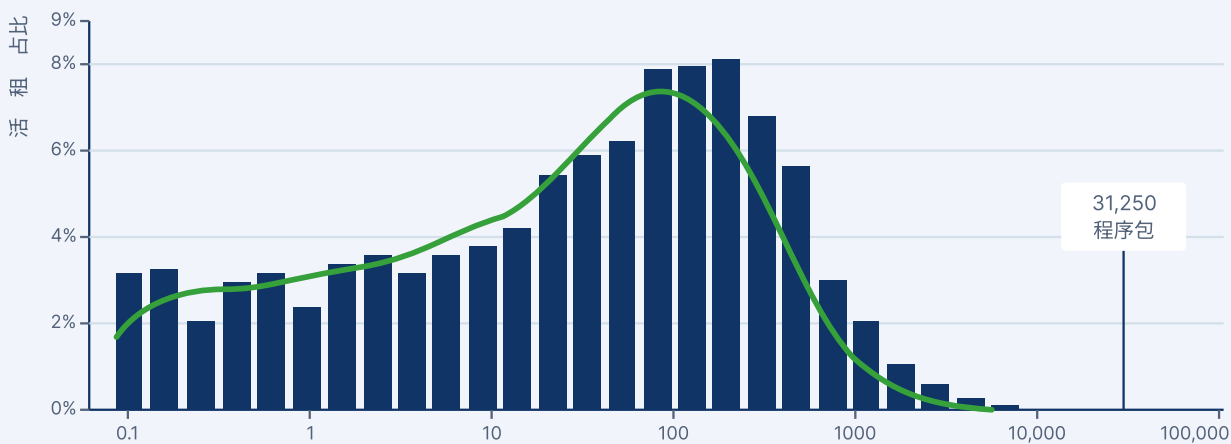


图 6. 2025 年活动租户每月创建的新软件包分布 (JFrog 数据库, 2025 年)

2025 年，使用 JFrog SaaS 的组织共向其软件供应链引入了超 1,170 万个新软件包，高于 2024 年的 700 万个，单年增幅达 67%。对于普通组织而言，全年引入的软件包数量约为 3,500 个。与往年一样，这一数字被少数重度用户拉高。规模最大的单一组织全年共引入了约 375,000 个新软件包。

中位数组织（一个更具代表性的衡量指标）在 2025 年一年引入了 509 个软件包，高于 2024 年的 231 个。这相当于每月约 42 个新软件包，或每天多个新软件包。

这种节奏很重要。每个新软件包都可能成为漏洞、许可证义务或恶意有效负载的潜在入口点。以每天超过一个的速度，问题不在于组织是否暴露于供应链风险之中；而在于它们是否具备相应的工具和流程，能够在每个新的依赖项到达开发人员计算机之前对其进行评估。

组织内 AI 库的采用

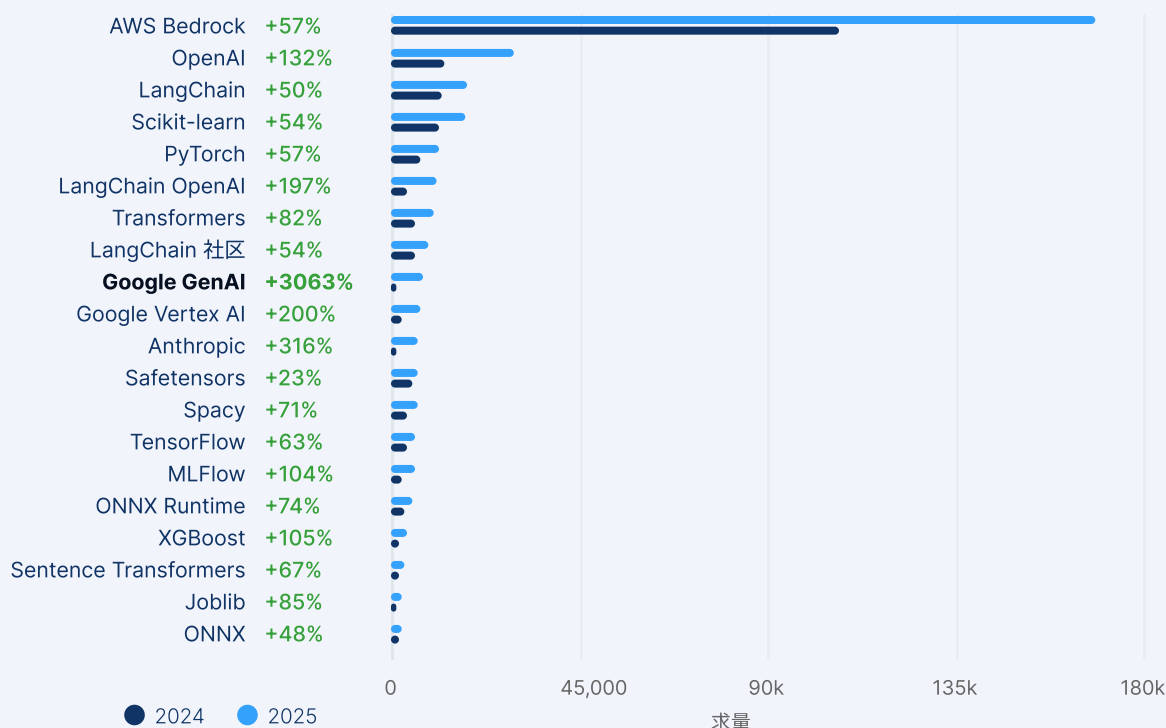


图 7. JFrog 客户环境中，按请求量统计的 AI 和机器学习库使用情况（JFrog 平台数据，2025）

2025 年，JFrog 客户环境中的 AI 和机器学习库总使用量同比增长了 74%，从 194,734 增至 337,573。

在 JFrog 客户中，积极使用 AI 和机器学习库的比例从 2024 年的 34% 增长至 2025 年的 41%。在已使用 AI 软件包的客户中，每位客户的平均库数量从 6.3 增长至 9.3 (+47%)。组织不仅仅是在采用 AI；它们对 AI 的依赖正在加深，随着 AI 工作负载的成熟，会引入更多框架、更多提供商和更多工具。

AWS Bedrock 仍然占据主导地位，但其领先优势正在缩小

Bedrock 以显著优势成为最常被提及的 AI 库，其规模从 107,186 增长到 168,582 (+57%)，占有被追踪 AI 库使用量的近一半 (47%)。其增长保持稳定，但其增长率在成熟的主流库中属于最慢之列，远低于 74% 的市场平均水平。其他所有主要的 API 提供商增长都更快。Bedrock 的主导地位反映了其先发优势和深度企业集成；其低于市场水平的增长率则反映出替代方案的迅速崛起。

API 服务商格局正迅速走向多元化。

Google GenAI 是数据集中增长最显著的产品，从接近于零的基数增长了 3,063%（从 244 增至 7,718），反映出 Gemini 在 2024 年发布后被企业快速采用。Google Vertex AI 增长了 200%（从 2,250 增至 6,742）。Anthropic 增长了 316%（从 1,580 增至 6,565）。Mistral AI 增长了 657%（从 208 增至 1,574）。Groq 增长了 426%（从 254 增至 1,336）。OpenAI 增长了 132%（从 12,651 增至 29,395）。模式已经很清晰：各组织不再依赖单一 AI 提供商来搭建平台。多模型架构正成为默认选择，而库的使用规模也反映了这一趋势。

智能体框架是增长最快的新类别。

CrewAI 增长了 206%（785 增至 2,404），AutoGen 增长了 68%（348 增至 584），二者都支持多智能体 AI 工作流。它们的增长表明，各组织正在从单轮 AI 交互迈向更复杂、经过编排的流水线。

LangChain 正整合为编排层。

LangChain Core 涨幅 +50%，总量达到 18,196。LangChain OpenAI 涨幅 +197%，总量达到 10,893。LangChain Community 涨幅 +54%，总量达到 8,742。旗下三大软件包助力 LangChain 逐步成为模型服务商与应用逻辑之间的衔接载体。

MLflow 的增长反映出模型管理负担正在不断加重。

MLflow 增长了 +104%（从 2,793 增至 5,695）。作为一个模型创建、评估和存储框架，其采用率反映了这样一个事实：组织正在构建和管理比以往任何时候都要多的模型和智能体。一个组织运行的 AI 工作负载越多，就越需要结构化工具来跟踪实验、对模型进行版本管理，并管理由此产生的制品。

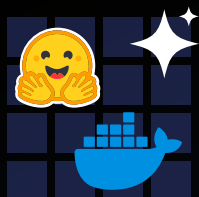
Ollama 的增长证实，自托管并非停留在愿景层面，而是已具备实际运营能力。

Ollama 增长了 +327%（从 404 增至 1,726），这一轨迹反映的是一种真实且不断增长的运营模式，而不仅仅是口头表达的偏好。

本地推理正逐渐被托管 API 取代。

Llama-cpp 是本地运行模型最常见的库之一，它下降了 22%（从 356 降至 276），是该数据集中降幅最大的。随着托管 API 的质量和成本效益不断提高，已经没有充分利用再进行原始本地推理了。各组织正转向 Ollama 或云 API 等托管式服务解决方案。

核心要点

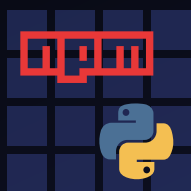


AI 生态系统现已成为新软件包的第二大来源，而且是增长最快的。

Hugging Face 于 2025 年发布了 140 万个新软件包，仅次于 Docker Hub 的 220 万个，同比增长 41%，而上一年的增长幅度约为 4 倍。

供应链出现了一个新的主导输入源，其运作方式与传统软件包生态系统截然不同：增长更快、工具支持成熟度较低，且依赖一个大多数来源控制机制原本无法触达的注册中心。

41% 的组织正在积极使用 AI 库，高于 2024 年的 34%，且这些组织中，平均每个组织使用 9.3 个库。另外 59% 的组织尚未使用任何 AI 软件包。这一差距的缩小速度，很可能比拉开时更快。

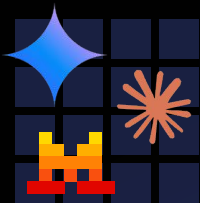


企业如今依靠智能体运行，而非 JAR 包。

在本报告历史上，npm 首次超过 Maven，成为使用量最高的软件包生态系统。PyPI 超过了 YUM。这些并非舍入误差。它们反映了企业软件构建地点及构建者正在发生的结构性转变。

智能体工作流和 AI 辅助开发可能更倾向于使用脚本语言；JavaScript 和 Python 的迭代速度更快，更易于非开发人员使用，也是许多 AI 编码助手的默认输出语言。

大多数大型组织现有的安全工具、审批工作流和治理策略，都是围绕以 Java 为中心的供应链构建的。如今承载最多流量的生态系统已经不同了。而风险就存在于这一差距之中。



多 AI、多模型架构正在重塑您的供应链。

多模型架构已成为新常态，且正在重塑您的供应链。两年前，一个组织的 AI 足迹还只是少数几个 Python 机器学习库。如今，这意味着要使用适用于多家提供商的 API 客户端、编排框架、推理引擎、本地服务工具、从公共注册中心提取的微调模型，以及可直接访问代码库和凭据的 IDE 扩展。

证据就体现在库数据中：Google GenAI 增长了 +3,063%，Anthropic 增长了 +316%，Mistral AI 增长了 +657%，Groq 增长了 +426%，且增速均达到市场增速的数倍。

CrewAI (+206%) 等智能体框架和 Ollama (+327%) 等本地服务工具正在与传统机器学习库一起进入生产流水线。企业软件供应链的构成已发生结构性变化，不仅体现在数量上，更体现在性质上。与传统软件包不同，这些新组件中有许多处于组织花费多年构建的治理框架之外。



语言整合正在加速，但也带来了隐藏的安全影响。

2025 年语言多样性急剧下降，跌破了 2023 年的基线。各组织似乎正在精简其技术栈：使用的语言更少、流水线更易于治理、攻击面也更小。总体而言，这是积极的信号。但整合也会使风险更加集中：当大多数企业开发都运行在两三个生态系统上时，某个基础库中的单个严重漏洞所波及的范围将比以往任何时候都更广。

您软件供应链中不断加剧的风险

AI 驱动的软件供应链威胁格局正在同时朝两个方向演变。现有风险（如通用漏洞披露和恶意软件包）已加速到前所未有的水平。新的威胁类别几乎在毫无预警的情况下出现：CI/CD 流水线被武器化为蠕虫传播渠道，智能体技能也从生产力资源演变为可直接访问开发人员机器的潜在感染媒介。

JFrog 识别出 969 个携带高影响有效负载的恶意 AI 智能体技能，将此前仅停留在理论层面的风险进行了量化。

发生变化的并不是 AI 会带来风险。这一点早已确定。发生变化的是感染点转移到了哪里。攻击者现在已经能够获得与防御者相同的 AI 能力：相同的模型、相同的自动化水平，以及同样大规模高速运作的的能力。这就是 AI 时代的攻防对称性。模型并不知道它在谁的流水线中运行。它不偏袒任何一方。决定结果的，不是哪一方拥有更好的 AI，而是哪一方围绕 AI 实施了更完善的治理措施和政策。

此外，攻击者不再等待开发人员拉取恶意软件包。他们会在软件包发布之前入侵 CI/CD 流水线，并代码尚未编写之前入侵 IDE 扩展。本节记录的攻击活动并非去年威胁的变体。它们代表着供应链可被攻破环节的结构性的转变，也预示着当攻击面遭遇以 AI 速度运作的攻击者时将会发生什么。



在给定的技术或包类型中发现的漏洞

2025 年，共有 48,244 个通用漏洞披露发布到 NVD，比 2024 年的 39,962 个增加了 20.7%。这一增长的部分原因可能归因于 AI 生成代码的激增：vibe 编码的应用程序往往会以超过人工代码审查速度的规模产生常见漏洞类别（例如，SQL 注入、XSS、缺少授权）。



图 8. 由 JFrog 监控的开源生态系统中的通用漏洞披露跟踪情况 (JFrog 数据库, 2025 年)

JFrog 会跟踪其所监控的开源生态系统中的通用漏洞披露，并重点关注面向开发人员且组织面临风险最高的部分通用漏洞披露。分布情况严重偏斜：Linux 发行版生态系统占据了原始总量中的绝大部分，其中仅 Debian 就有 6,825 个通用漏洞披露，SUSE 为 5,549 个，Ubuntu 为 5,110 个，但这一数量主要集中在严重等级为“中危”、且“严重级”占比很低的漏洞上。Linux 发行版的严重性评级将生态系统特定配置纳入考量，这实际上会降低特定漏洞的实际严重性。

从运营角度来看，更相关的情况出现在面向开发人员的软件包生态系统中，这里的通用漏洞披露数量较少，但严重性为严重级别的占比却高得多。RubyGems 的严重级别占比最高，达到 24%，其次是 PyPI 和 NuGet，二者均为 16%。这些是开发人员每天从中拉取依赖项的生态系统，与深藏在 Debian 基础镜像中的严重级 CVE 相比，npm 或 PyPI 中的严重级 CVE 更有可能在生产代码中被触达。

若干开发人员生态系统同比出现了显著变化。npm 大幅增长，在前一年有所下降后，通用漏洞披露数量从 2024 年的 410 增至 2025 年的 539 (+31%)。Maven 增长势头更为强劲，通用漏洞披露数量从 382 增至 566 (+48%)，扭转了 2024 年的大幅下滑趋势；此前的下滑部分归因于 NVD 积压效应。Golang 延续了多年来的上升趋势，通用漏洞披露数量达到 596 (+23%)。PyPI 小幅增长 (+2%)，但其严重级别漏洞密度达 16%，在所有主要生态系统中保持最高水平。Cargo/Rust 的通用漏洞披露数量从 2023 年的 61 大幅增至 2025 年的 135。虽然按规模来看，它仍然是最小的开发人员生态系统，但随着 Rust 采用率加速提升，其发展轨迹值得关注。

对于安全团队而言，最值得密切关注的生态并不是当下绝对 CVE 数量最高的那些，而是增长速度最快的；npm、Maven、Golang 和 Cargo 都呈现出相同的趋势。



最常见的漏洞类型

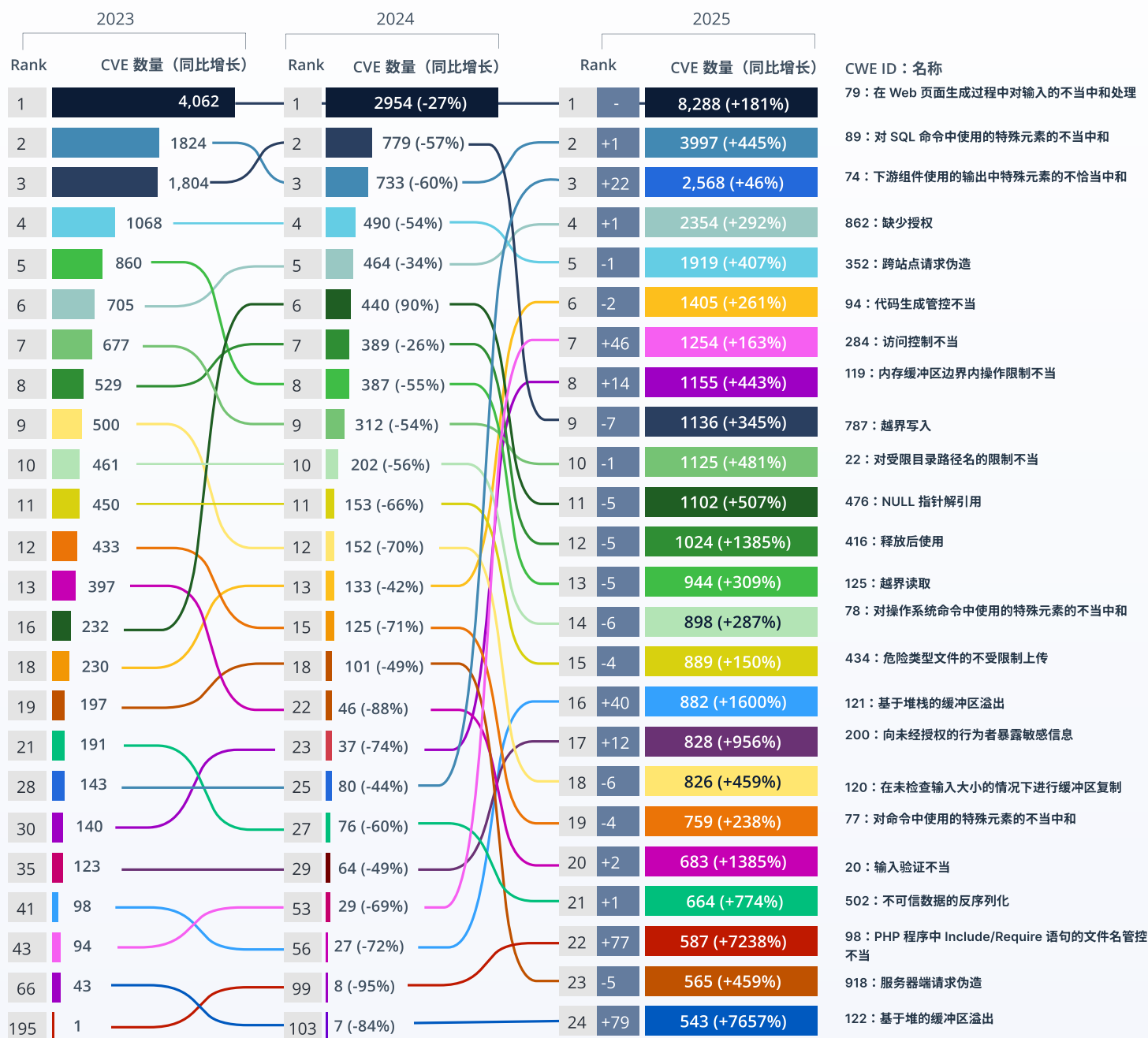


图 9.与 2023 年和 2024 年相比，2025 年披露的最常见漏洞类型



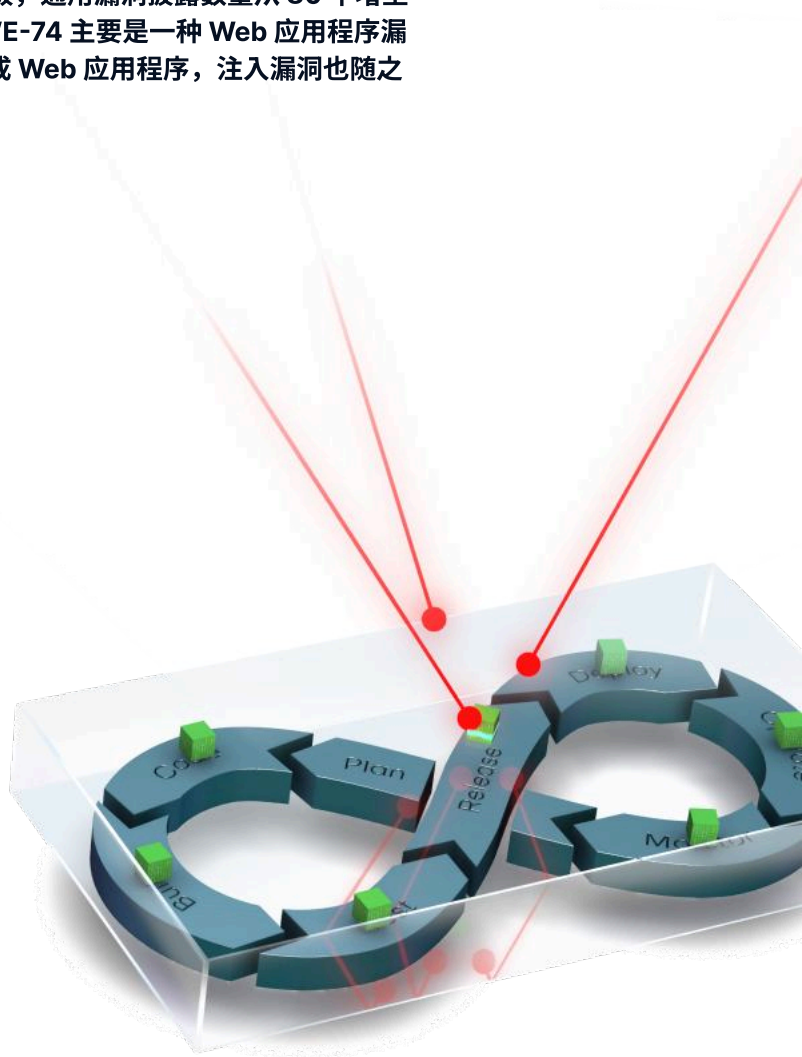
2025 年，共有 562 个唯一的 CWE ID 分配给通用漏洞披露，高于 2024 年的 243 个，增幅达 131%。漏洞类型的范围正在扩大，但最值得关注的是分布顶端的情况，其中一个主题占据主导地位：Web 应用程序漏洞的增加，而这在很大程度上是由未采用安全开发实践的 AI 生成代码所推动的。

2025 年排名前三的漏洞类型传递出一致的趋势：

1. **跨站脚本攻击 (CWE-79)** 保持了其在 2024 年位居第一的位置，但数量几乎增长至原来的三倍，从 2,954 个通用漏洞披露增至 8,288 个 (+181%)。跨站脚本攻击 (XSS) 是一种经典的 Web 应用程序漏洞，而其数量激增与大量“氛围编程”生成的 Web 应用程序在缺乏充分输入验证的情况下发布上线这一趋势高度吻合。
2. **SQL 注入 (CWE-89)** 从 2024 年的第三位跃升至第二位，通用漏洞披露数量从 733 增至 3,997 个 (+445%)。这一增长直接归因于 AI 生成的 Web 应用程序缺乏参数化查询规范，这是所有数据库连接应用程序最基本的安全编码要求之一。
3. **注入 (CWE-74)** 作为最引人注目的新晋类别进入最高层级，通用漏洞披露数量从 80 个增至 2,568 个 (+3,110%)。与 XSS 和 SQL 注入一样，CWE-74 主要是一种 Web 应用程序漏洞，同样的趋势也适用于此：AI 辅助开发正在大规模生成 Web 应用程序，注入漏洞也随之而来。

贯穿这三者的共同点非常显著。XSS、SQL 注入和注入漏洞并不是什么新型或罕见的漏洞类别；几十年来，人们早已了解这些漏洞，并且能够加以预防。它们在 2025 年同时激增，直接表明了当应用程序开发速度快于安全编码实践的落地速度时，会发生什么。

在前三名以外，越界写入 (CWE-787) 展现出不同的发展态势。2024 年，它以 779 个通用漏洞披露位居第 3 位，但 2025 年，尽管通用漏洞披露增至 1,136 个 (+46%)，位次却降至第 10 位。内存不安全代码中的潜在风险并未减弱，只是 Web 应用程序漏洞的激增在其周围形成了更大的威胁面。



2025 年高关注度的通用漏洞披露的常见漏洞影响

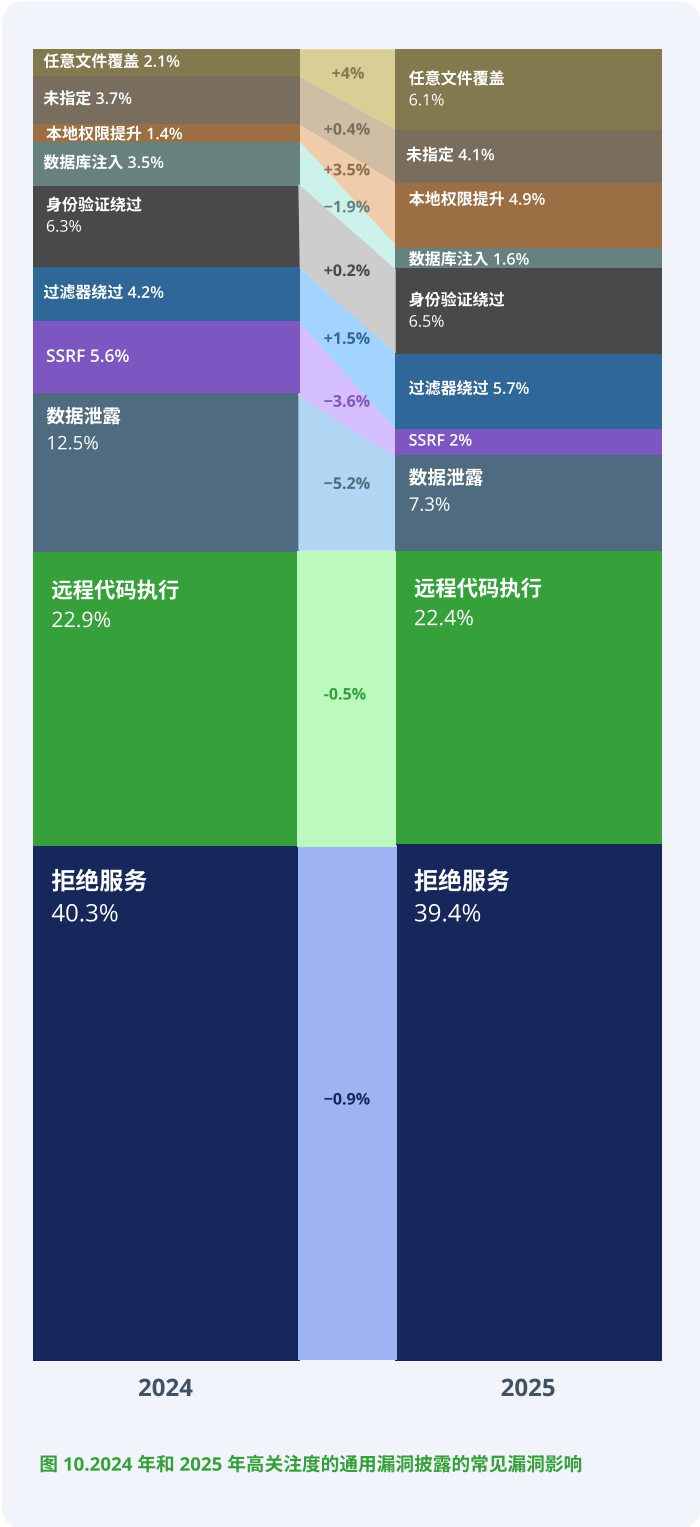


图 10.2024 年和 2025 年高关注度的通用漏洞披露的常见漏洞影响

2025 年，JFrog 安全研究团队分析了开源软件包中的数百个高关注度的通用漏洞披露 (HPCVE)。该团队的通用漏洞披露研究目标是，为广泛使用的开源项目中的通用漏洞披露开展深入分析，研究公开漏洞在现实世界中的严重性，以帮助安全团队了解这些漏洞何时以及如何影响他们的软件项目。

影响类别的分布同比表现得非常稳定。拒绝服务 (DoS) 仍位居第一，占 39.1% (97 个 CVE)；远程代码执行 (RCE) 仍位居第二，占 22% (55 个 CVE)；数据泄露仍位居第三，不过随着整体数据集的增长，其占比从 12.5% 降至 7.3%，而其绝对数量则保持不变。

这种分布历经多年保持稳定本身就很有意义。在企业开源领域中，受到高度关注的漏洞类型并没有变化；拒绝服务和远程代码执行始终占据该列表的主导地位。每年发生变化的是哪些特定软件包存在这类漏洞，以及它们在实践中有多容易被利用。

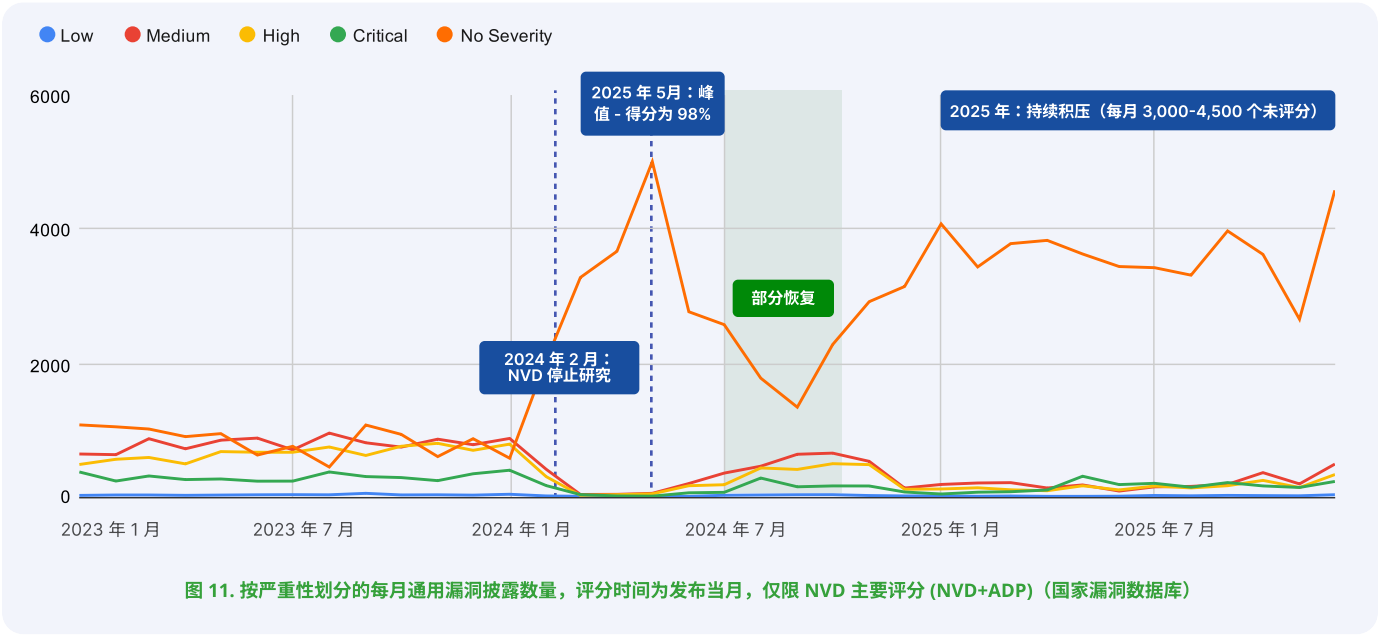
2025 年一个显著的例外是“任意文件覆盖”，其数量从 3 个通用漏洞披露 (2.1%) 跃升至 15 个通用漏洞披露 (6.0%)，在 HPCVE 中的占比几乎增加了两倍，并成为所有影响类别中同比增幅最大的一类。



方法说明

JFrog 安全研究团队在确定通用漏洞披露研究优先级时会考虑多个因素。该团队专注于与 JFrog 客户相关的技术，并优先处理“高危”和“严重”级别的问题 (CVSS ≥ 7.5)；在没有 CVSS 评分时，则辅以基于机器学习的严重性预测。无论公开严重性评级如何，正在被大肆主动利用或媒体关注度较高的通用漏洞披露也会被优先处理。

引入您软件供应链中漏洞的严重性



在去年的报告中，我们重点介绍了 NVD 的通用漏洞披露分析流水线自 2024 年 2 月起遭受的严重中断，彼时，NVD 在重组和预算削减的背景下暂停了其研究工作。当时，这造成了大量未评分通用漏洞披露的积压：到 2024 年 5 月，当月发布的通用漏洞披露中有 98% 完全没有严重性评分，同时促使 NVD 与 CISA 签约，将其作为首个授权数据发布方 (ADP)，来帮助清理积压队列。

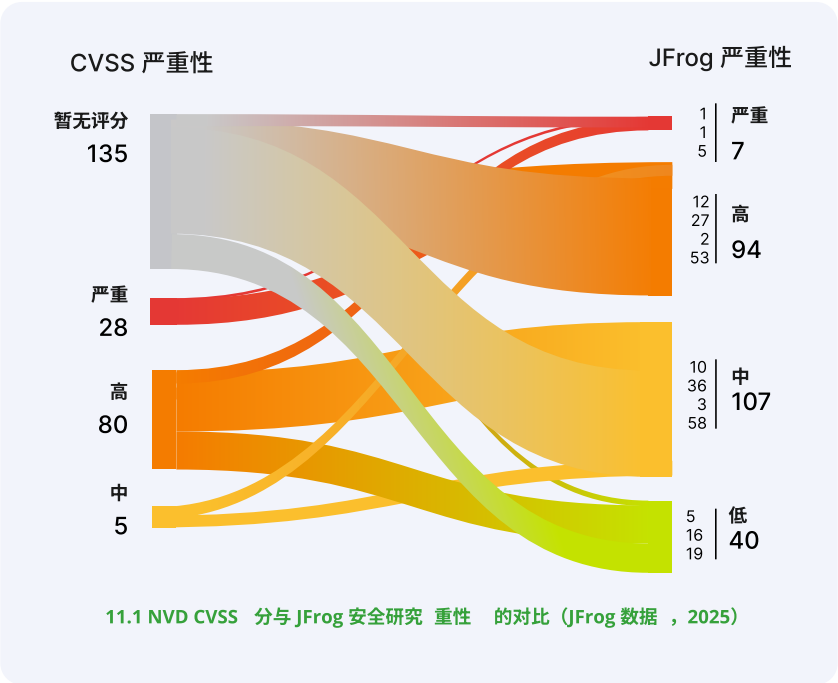
一年后，数据显示，尽管 CISA 的介入在 2024 年年中带来了一定程度的恢复（有评分的通用漏洞披露数量在整个夏季出现回升），但这种改善只是昙花一现。整个 2025 年，积压量始终居高不下，每月有 3,000 至 4,500 个通用漏洞披露在发布时尚未评分，远高于 2023 年 600 至 1,100 个的基线水平。其中一个原因是通用漏洞披露提交量激增，这可能在一定程度上是由于 AI 生成的漏洞报告大量涌入流水线，而这持续超出了 NVD 的分析能力。

在确实获得评分的通用漏洞披露中，严重性分布的总体趋势与往年保持一致：中危和高危通用漏洞披露的数量仍然占主导地位，严重性较低的通用漏洞披露仍然较少，而“严重”级别的通用漏洞披露则介于这两个极端之间。然而，数量庞大且持续存在的“无严重性”类别意味着，真实的严重性状况在任何时候都只是部分可见，这使得组织在确定漏洞修复优先级时，比以往任何时候都更需要将目光投向 NVD 评分之外。



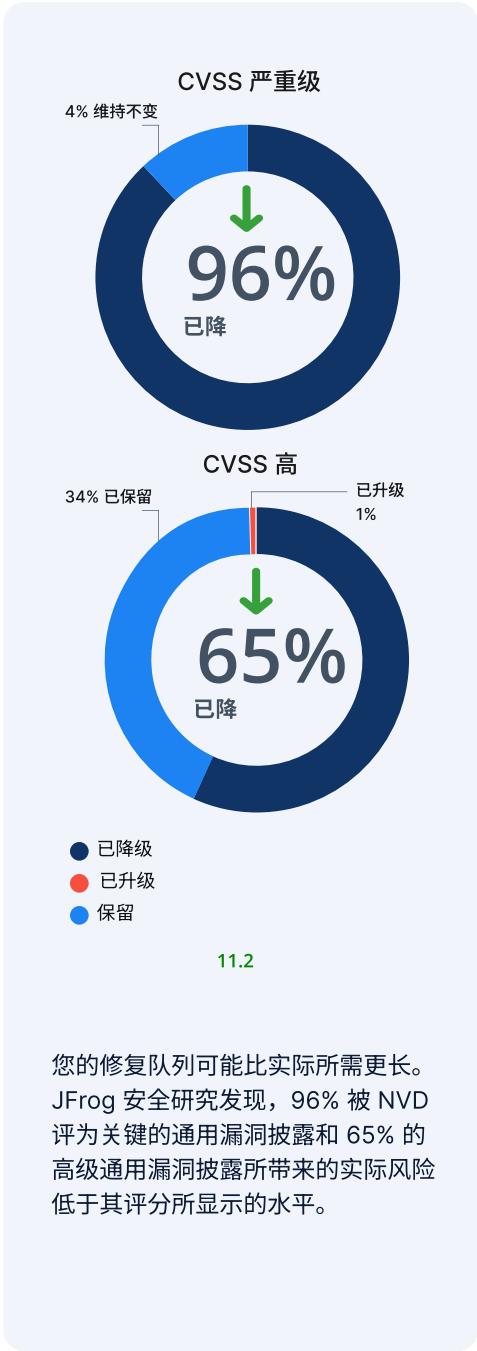
NVD 的 CVSS 评分模型评估的是成功漏洞利用的理论严重性，而非在任何给定环境中被利用的可能性。JFrog 安全研究团队根据真实世界可利用性、上下文因素和实际攻击复杂性，对其分析的每个 HPCVE 重新进行评级。2025 年，NVD 评分与 JFrog 评估之间的差距进一步扩大。

在分析的 248 个 HPCVE 中，113 个具有 NVD 指定的严重性评分，高于 2024 年的 57 个。其余 54% 根本没有 NVD 评分，JFrog 对这些漏洞进行了独立评级。



NVD 与 JFrog 严重性评级之间的分歧继续扩大。在 NVD 评级为“严重”的 28 个 CVE 中，JFrog 仅确认其中 1 个（即 CVE-2025-1550）为“严重”级别，这意味着 96% 被降级，高于 2024 年的 88.2%。NVD 的“高危”评级中也有 65% 被降级，高于 2024 年的 56.8%。

这种模式并不新鲜，但发生频率正在加速。经进一步分析，越来越多被 NVD 评为“严重”或“高危”级别的通用漏洞披露在实践中的可利用性明显降低。仅依赖 NVD 评分来确定修复优先级的安全团队，所依据的风险图景越来越被夸大，他们把时间花在其环境中不太可能被触达的漏洞上，同时却可能降低了那些确实可能被触达的漏洞的优先级。



高关注度的通用漏洞披露的适用性评级

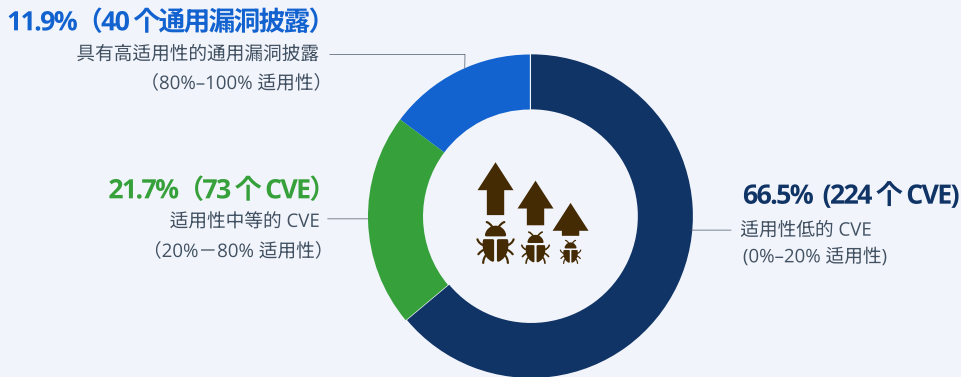


图 11.3.Xray 客户扫描结果中显示的 2025 年通用漏洞披露的适用性评级 (利用 CVE 和 JFrog 数据库开展的 JFrog 安全研究专有数据)

仅为通用漏洞披露分配严重性评分，不足以评估漏洞对特定软件产品的影响。JFrog 安全研究团队不仅会分配 JFrog 严重性评分；该团队还会评估在实际情况下影响这些漏洞可利用性的条件。为此，JFrog 创建了适用性扫描器，以确定其扫描的特定软件产品是否满足可利用性的判定标准。

JFrog 安全研究团队为 2025 年发布的 337 个通用漏洞披露 (CVE-2025-*) 创建了适用性扫描器，重点关注 JFrog 客户中最主流的组件和技术中的高危和严重通用漏洞披露。在 JFrog Xray 扫描的制品中，发现只有 40 个通用漏洞披露 (11.9%) 是高度可利用的，适用性超过 80%。相比之下，发现有 224 个通用漏洞披露 (66%) 的可利用性较低，适用性为 0%–20%。

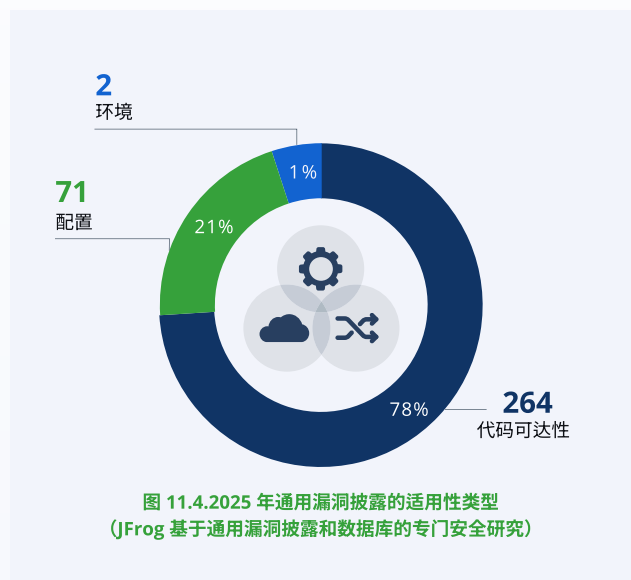
Next.js 中的 CVE-2025-66478 (React2Shell) 是高适用性的最典型示例之一；Xray 标记了该 CVE 的软件制品中，有 96% 被确认为适用，这意味着几乎所有这些制品都满足了可利用条件。该漏洞可通过精心构造的恶意 HTTP 请求触发，当 React 对该请求进行反序列化时，即可在服务器上实现远程代码执行。其高适用率反映出一个直接原因：React 和 Next.js 的默认配置存在漏洞。使用标准 create-next-app 命令创建 Next.js 应用，会导致生成的应用程序存在漏洞。

而在另一个极端，CVE-2025-32988 是适用性最低的通用漏洞披露之一，在扫描的制品中，被判定为适用的不到 0.1%。GnuTLS 中的这个双重释放漏洞，需要攻击者将用户输入提供给特定的 GnuTLS API，并传入一个用户控制的对象标识符 (OID) 参数。在实践中，这种情况发生的可能性极低，因为 OID 通常是固定的，无法从外部进行控制。



高关注度的通用漏洞披露的适用性类型

知道组件中存在漏洞，这仅仅是起点。更难的问题，也是更具运营相关性的问题，是它在给定环境中是否真的能够被利用。JFrog 安全研究团队对这一问题的分析超越了传统的代码可达性分析，后者只会判断易受攻击的代码是否可被调用。现实中的可利用性还取决于应用程序配置以及底层操作系统中存在的条件。遗漏任何一个因素，都会导致两个方向上的误判：标记并不存在的风险，或遗漏确实存在的风险。

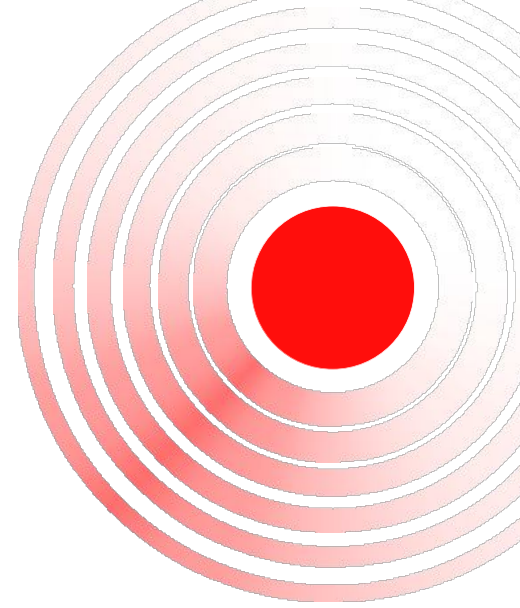


JFrog 将适用性分为三种类型。2025 年，HPCVE 中有大部分 (78%) 属于代码可达性类型，这意味着可通过调用可达性或静态分析对代码本身进行分析，来确定可利用性。另有 21% 属于配置类型，其可利用性取决于特定的应用程序或库配置设置，而不只是代码路径。剩余的 1% 属于环境类型，其可利用性取决于底层操作系统或运行时环境中的条件。

本节前文所述的 GnuTLS 双重释放漏洞 CVE-2025-32988，很好地说明了基于代码可达性的适用性；被认定为适用的已扫描制品不到 0.1%，因为利用该漏洞需要向特定 API 传递用户可控的 OID 参数，而这种情况在实践中几乎不存在。去年针对 sudo 软件包发布的 CVE-2025-32462 很好地说明了基于配置的适用性，其中可利用性完全取决于 sudoers 配置文件中的一个非默认条目，而任何代码可达性分析都无法发现这一点。

开发者工具的攻击面

软件供应链始终围绕着开发人员所引入的内容。2025 年，攻击者向上游转移，渗透到开发人员用于编写代码的工具中。IDE 扩展、AI 编码助手和模型上下文协议 (MCP) 服务器现已成为标准的开发人员基础设施。它们可以访问代码库、凭据和 CI/CD 流水线。而它们是大多数安全框架在设计之初并未考虑应对的攻击面。



IDE 扩展：一个全新的生态系统，一种全新的威胁媒介

JFrog 于 2025 年首次在 OpenVSX 上跟踪 AI 编码助手 IDE 扩展。OpenVSX 注册中心被 Cursor、VSCodium 等 AI 原生 IDE 使用，其扩展数量从 2023 年的约 1,000 个增长到 2025 年的 3,803 个，增幅为 262%。一个不断扩大的生态系统正成为越来越有吸引力的目标。

2025 年，OpenVSX 上检测到 56 个恶意扩展，这是本报告首次追踪此攻击面。其中最受关注的是 GlassWorm，这是首个针对 VS Code 扩展的自传播蠕虫，由 Koi Security 于 2025 年 10 月发现。GlassWorm 通过将恶意代码隐藏在不可见的 Unicode 字符中，入侵了七个扩展，它还收集开发人员凭据以进行自主传播，并在受感染的计算机上部署了远程访问木马，影响了约 35,800 次安装。

MCP 服务器：智能体层现已成为漏洞攻击面

MCP 服务器为 AI 助手提供对工具、API、数据库和本地系统的访问权限，使其成为高价值的攻击面。2025 年，JFrog 在多个 MCP 服务器中发现了 20+ (RCE) 漏洞，其中包括 mcp-remote 中的 CVE-2025-6514，该漏洞于 2025 年 7 月公开发布，CVSS 评分为 9.6，属于远程代码执行漏洞。2026 年初，JFrog 将其扫描范围扩展到 AI 智能体技能注册中心，并识别出 969 个携带高危有效负载的恶意 AI 智能体技能，这进一步证实，智能体攻击面已从传统的软件包注册中心和 IDE 扩展，扩展到了为自主 AI 工作流提供支持的技能和工具。

CVE-2025-11953：严重 RCE 令数百万 React Native 开发人员面临风险

针对开发人员的最重大发现是 CVE-2025-11953，这是一个影响 React Native 开发人员且 CVSS 评分为 9.8 的远程代码执行漏洞。该漏洞由 JFrog 安全研究团队披露，并于 2025 年 11 月发布，它将数百万 React Native 开发人员置于风险之中。这种攻击执行起来极其简单，而且开发人员机器在默认情况下即存在漏洞，甚至本地网络之外的攻击者也可趁虚而入。

某些恶意软件包比其他软件包危害更大

本节讨论的是被故意设计用于造成危害的软件包：不是通用漏洞披露，也不是错误配置，而是带有有效负载的代码。这种区别很重要。意外漏洞可以通过打补丁来修复；而蓄意恶意行为则需要另一种防御方式。

在恶意软件包数量方面，npm 在所有生态系统中位居首位，且领先优势显著。2025 年检测到 171,592 个独特的恶意软件包，较 2024 年的 31,132 个增长了 451%。RubyGems 从 190 个激增至 785 个，增长了 313%；Hugging Face 则首次进入跟踪范围，检测到 495 个恶意模型。在数量方面，npm 超过了其他所有生态系统的总和。

数量并不是衡量危险性的正确标准。
破坏性最大的攻击活动，在软件包数量方面反而是规模最小的。

活动对比

推广活动	程序包	实际影响
<u>Qix</u>	25	超 250 万次受感染下载
<u>s1ngularity</u>	8	83,000 项机密被泄露
<u>Shai-Hulud (1+2)</u>	1,000	凭据收集器、DNS 劫持、破坏性擦除程序
<u>Big Red</u>	80,000	无直接用户有效负载 — 可能为试运行

8 个软件包泄露了 83,000 个机密。25 个软件包造成了 250 万次受感染下载。Big Red 活动发布了约 80,000 个软件包，但两种情况均未发生。要点：软件包数量是衡量供应链风险时最无用的指标。

积极的一面是，NuGet 在这一年中表现出最显著的生态系统改善，恶意软件包数量下降了 97%，从 2024 年的 671 个下降至 2025 年的仅 19 个，部分原因在于 Microsoft 持续努力提升 NuGet 生态系统中的恶意软件包检测和预防能力。



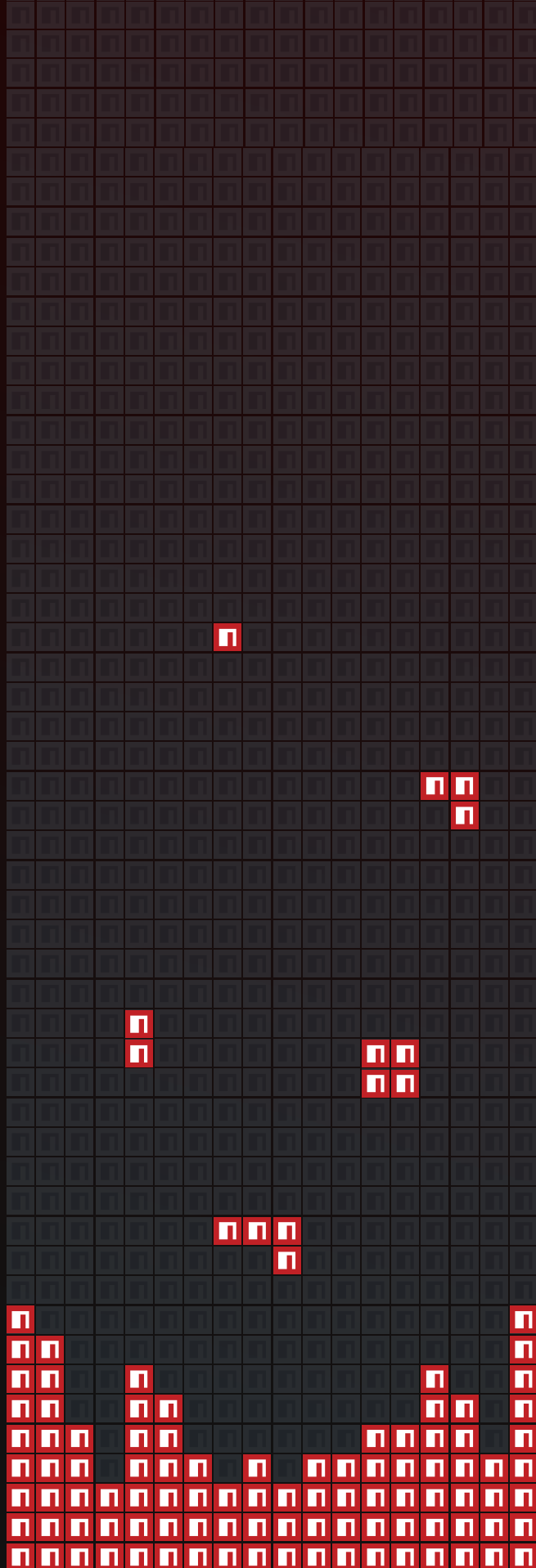
配置错误、失误以及人为错误的影响

s1ngularity 攻击利用了 GitHub Actions 错误配置的一种危险组合。首先是使用了 `pull_request_target` 触发器，这使来自恶意分叉的工作流能够以提升后的制品库权限运行并访问机密。第二个问题是，将未经清理的 PR 标题，具体来说，`{{ github.event.pull_request.title }}`，直接回显到 Shell 脚本中，从而引入了命令注入漏洞。

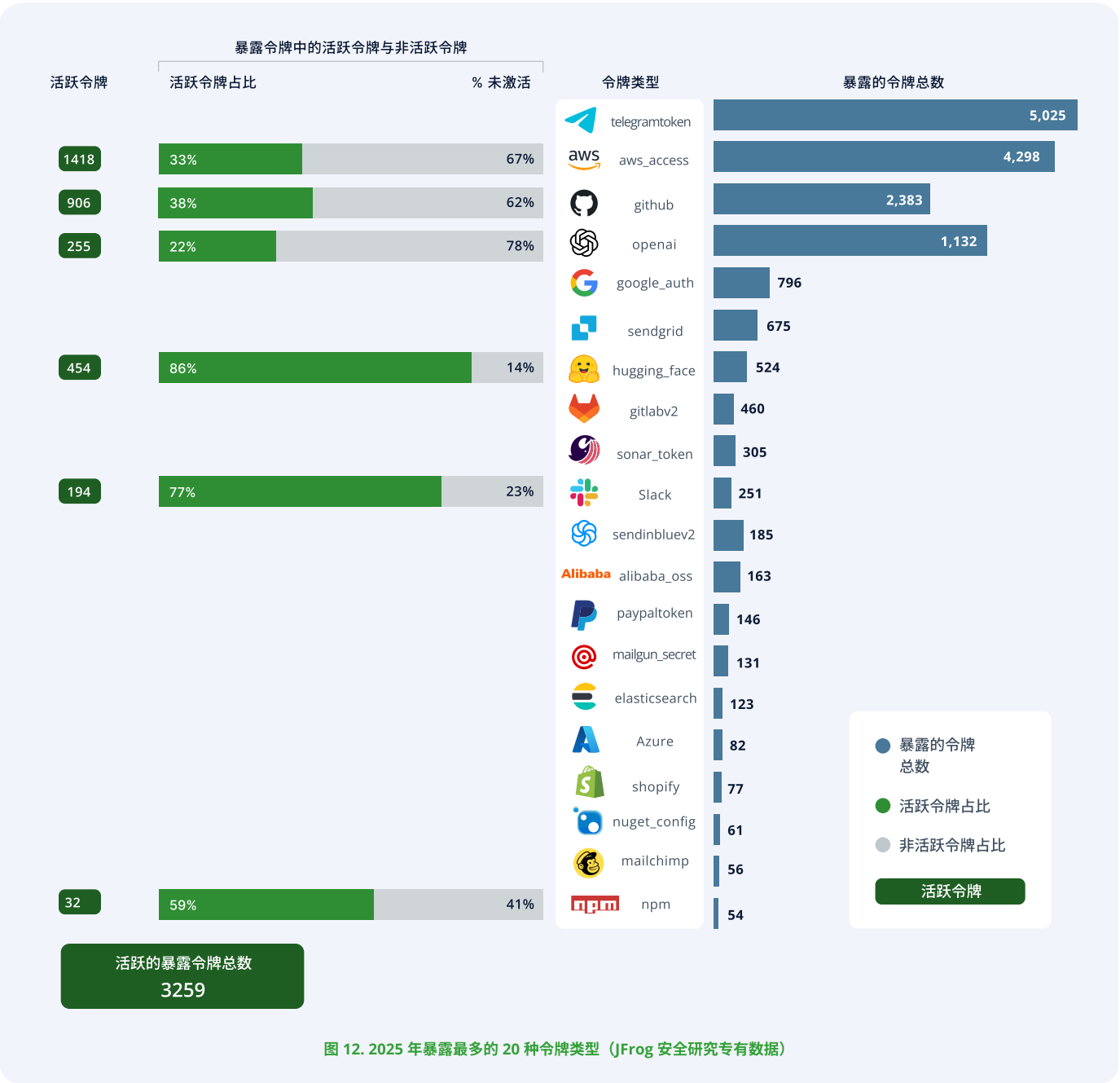
由于 GitHub 令牌被设置为读/写权限，而非推荐的只读权限，攻击者利用注入的命令以编程方式触发了一个单独的 `workflow_dispatch` 事件。该次级工作流可访问制品库的 npm 发布令牌，攻击者会窃取该令牌，从而在该项目范围内发布恶意软件包。

s1ngularity 攻击并不需要零日漏洞或复杂的恶意软件。只需要一个错误配置的工作流，以及一个权限超出所需范围的令牌。这些并不是什么罕见情况。这些都是常见情况。CI/CD 流水线现已成为攻击面的一部分，而其中的错误配置会带来与其所构建代码中的漏洞同样的后果。

仅 8 个软件包和 1 个配置错误的工作流，就足以泄露 83,000 条机密。



二进制制品中密钥泄露的现状



JFrog 安全研究团队扫描了公开的二进制仓库，查找暴露的密钥：遗留在制品、容器镜像和构建输出中的令牌、API 密钥和凭据。2025 年，在跟踪的生态系统中发现了 17,637 个暴露的令牌。其中，经验证有 3,260 个在数据收集时仍处于活跃状态，总体活跃率为 18.5%。

最显眼的数字低估了风险。活跃率因令牌类型不同存在显著差异，而最有可能仍处于活跃状态的令牌，也往往是潜在影响最大的令牌。

AWS 凭据是数量最多的活跃威胁。在发现的 4,298 个 AWS 访问令牌中，已确认有 1,418 个仍处于活跃状态（活跃率为 33%）。攻击者能够利用一个有效的 AWS 凭据来访问云基础设施、存储、计算和数据。

GitHub 令牌既很常见，后果也很严重：在发现的 2,383 个令牌中，906 个已确认处于活跃状态（38%）。一个活跃的 GitHub 令牌可提供对源代码仓库、CI/CD 流水线以及存储在 Actions 中机密的访问权限，使其成为软件供应链环境中最重要的凭据类型之一。

在具有统计意义的发现数量中，Hugging Face 令牌在所有令牌类型中具有在最高的活跃率：共发现 524 个令牌，其中 454 个确认为活跃（87%）。近九成暴露的 Hugging Face 令牌在被发现时仍然处于活跃状态，这与去年约 85% 处于活跃状态的发现结果一致。随着组织越来越多地将专有模型和数据集存储在 **Hugging Face** 上，暴露的令牌会提供对这些资源的访问权限，并且还有可能会允许修改或污染这些资源。

Slack 令牌也显示出类似的高活跃率：在发现的 251 个令牌中，194 个处于活跃状态（77%），可访问内部通信、共享文件和集成工作流。

OpenAI API 密钥代表着一类日益增长的风险。在发现的 1,132 个密钥中，255 个被确认为活跃密钥（23%）。OpenAI 密钥泄露会导致未经授权的 API 使用，费用由受害者承担，但更重要的是，它可能提供对微调模型、存储的对话以及组织 AI 工作流的访问权限。

按数量计算，**Telegram** 是数量最多的单一令牌类型，共发现 5,025 个，但其中 0% 经验证处于活跃状态。这些很可能是已过期或被撤销的机器人令牌：数量很大，但实际风险可忽略不计。



所有令牌类型中的模式都指向同一个根本原因：机密被提交到二进制制品和构建输出中，然后发布到制品库，在凭据创建很久之后仍会保留在那里。

面临最大风险的组织，往往是那些没有采用自动化扫描，无法在暴露的凭据被他人发现之前检测并撤销这些凭据的组织。

核心要点



数量并不总是衡量风险的最佳标准。

2025 年，几乎各个类别的数据都创下了新纪录。但体量和危险性并无关联。Qix 使用 25 个软件包产生了 250 万次受感染的下载。Singularity 使用 8 个软件包泄露了 83,000 个机密。Big Red 发布了 80,000 个软件包，但两个企图均未达成。教训是一致的：本年度最危险的供应链事件并不是规模最大的事件，仅根据数量进行分级的组织，他们总是会优先考虑错误的威胁。



感染点已上移到堆栈的更高层。

攻击者不再等待开发人员从注册中心拉取恶意软件包。他们会入侵 CI/CD 流水线、模型注册中心、IDE 扩展，甚至智能体技能，这进一步证实攻击面正在持续向上游转移，进入了驱动自主工作流的工具层。



在漏洞数据中已可见 AI 生成的代码。

2025 年排名前三的 CWE 类别是 XSS、SQL 注入和注入，这些都是数十年了人们已经理解且可预防的典型 Web 应用漏洞。它们同时再度出现，直接原因在于大规模采用 AI 辅助开发，在未应用基础安全编码实践的情况下生成代码。CVE 的增长率并不仅仅取决于编写了更多软件。还取决于这些软件是如何编写的。



NVD 严重性评分作为故障分类工具的作用正变得越来越弱。

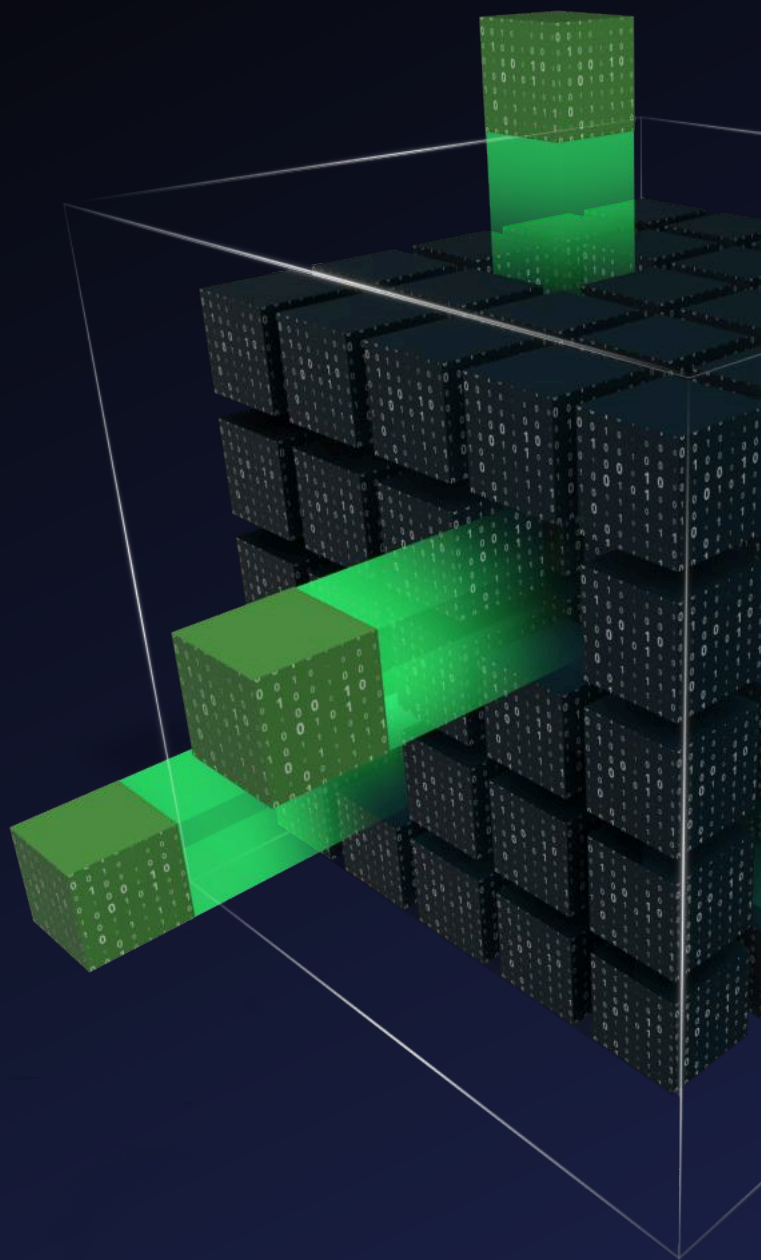
2025 年，NVD 评定为“严重”的 CVE 中有 96% 被 JFrog 降级，高于 2024 年的 88%。只有一个 CVE 被双方同时确认为“严重”级别。以 NVD 评分作为主要优先级排序机制的安全团队正在将修复资源花费在实际威胁较小的漏洞上，同时理论严重性与实际严重性之间的差距持续扩大。

组织如何 立即采取安全措施

今年呈现出的图景是，整个行业被夹在其已构建的基础设施与当下所面临的现实之间，进退维谷。在北美、欧洲和南亚的 1,500 多名安全、DevOps、AppSec 和工程专业人员中，这一模式是一致的：各组织已投入的工具、框架和治理流程，并非为这一新增的层级或复杂度而设计。

AI 生成代码、智能体开发工具与模型制品，正进入那些围绕传统开源依赖项构建的流水线。

防御措施已过时。

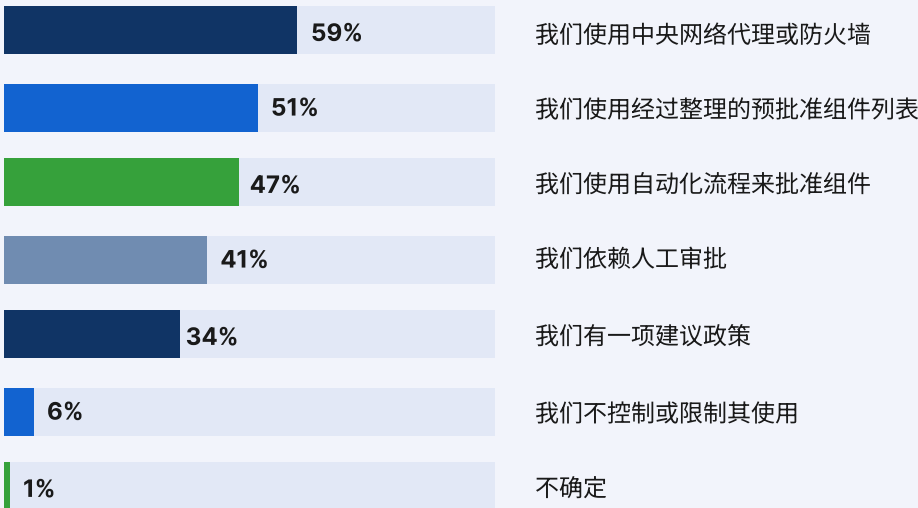


1. 来源限制

来源限制是为软件包生态系统构建的。供应链已经演进到包括模型注册中心和 IDE 扩展；然而，大多数来源控制措施尚未跟上。同一位可能被阻止下载可疑 npm 软件包的开发人员，却仍可自由安装恶意 IDE 扩展、连接到未经治理的 MCP 服务器，或从 Hugging Face 拉取被武器化的模型。边界已扩展。管控措施却没有跟上。

开源组件的来源和治理

Q 您的组织在治理或管控来自公共注册中心的开源软件包和其他软件组件方面采取了哪些方法？

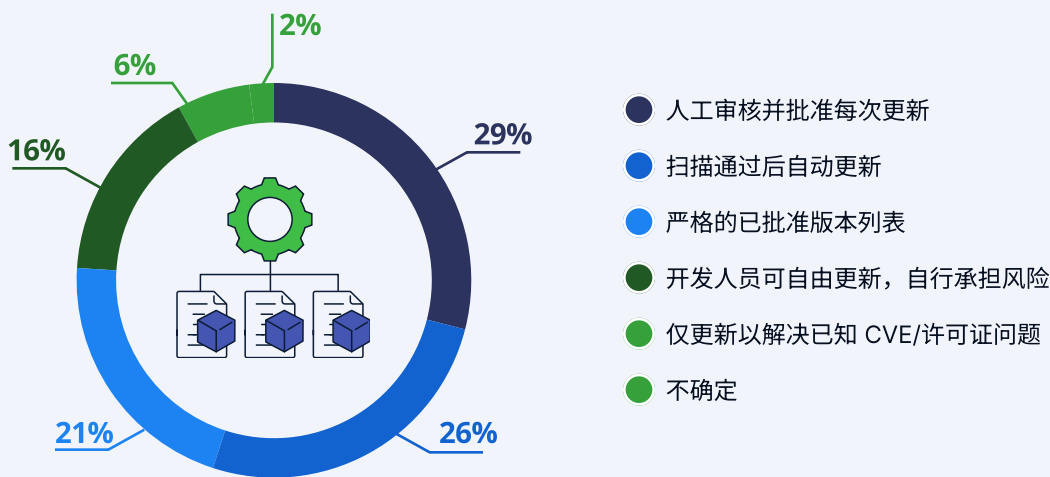


6% 的受访者表示，其组织根本未实施任何管控措施。鉴于本报告仅在 2025 年就记录了 171,592 个恶意 npm 软件包，这种状况令人难以接受。另有 34% 的受访者依赖建议的策略和开发人员自我治理，而不强制实施。未强制实施的策略只是一种假设。在这一年里，就连 Chalk 和 Debug 这样的可信软件包都遭到劫持、其受感染版本的下载量超过 200 万次，因此，假设并不是管控措施。

行业最佳实践依托中央制品管理解决方案代理公共注册中心，在任何内容到达开发人员的机器之前提供单一控制点。59% 的企业通过网络代理或防火墙实现这一点，51% 维护经审核批准的列表，47% 使用基于策略的自动化扫描。从数据层面来看，现状表现良好。但隐患在于，如今供应链的覆盖范围早已突破传统的软件包注册中心。

管理来自公共注册中心的软件包版本更新

Q 您的组织如何管理来自公共注册中心的软件包版本更新？



任何单一方法均不占据主导地位。29% 需要人工审查和批准每次版本更新。26% 会在自动化扫描通过的情况下自动更新。21% 维护严格的经批准版本列表。这种碎片化现象本身值得注意：当同一个组织内的不同团队采用不同方法时，最薄弱的环节决定了实际的管控水平。

另外值得注意的是，三种最常见方法中的两种，即人工审查和静态批准版本列表，并非针对 AI 辅助开发当前的发展速度而设计。

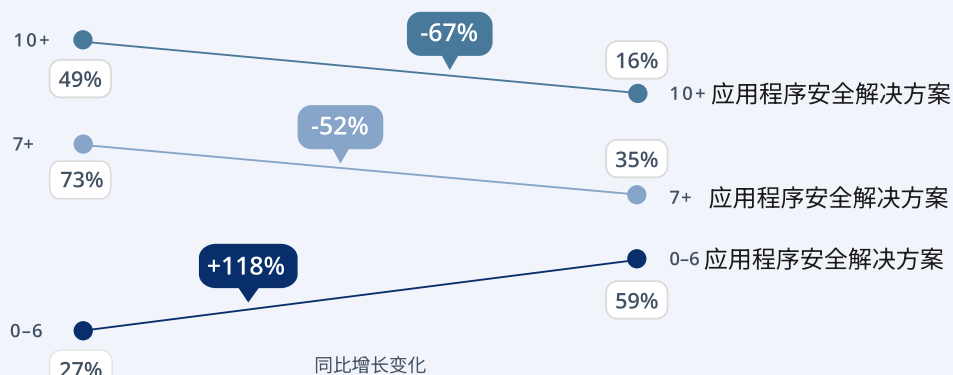
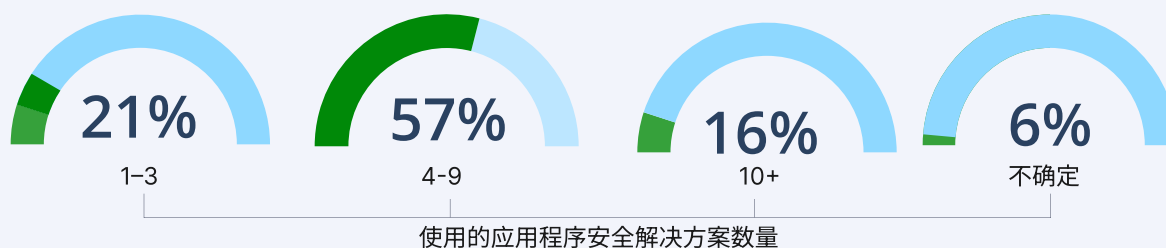
在依赖项频繁更新且智能体工作流无需开发人员直接操作即可引入新软件包的环境中，依赖人工审查周期或手动维护列表的控制措施在设计上就注定会滞后。

2. 扫描，扫描，扫描

各组织报告其所使用的安全工具数量在今年大幅下降。无论这反映的是真正的工具整合，还是向平台原生安全的转变，其揭示出的覆盖缺口都是相同的。

组织正在使用的应用程序安全解决方案数量

Q 贵组织使用了多少应用程序安全解决方案？



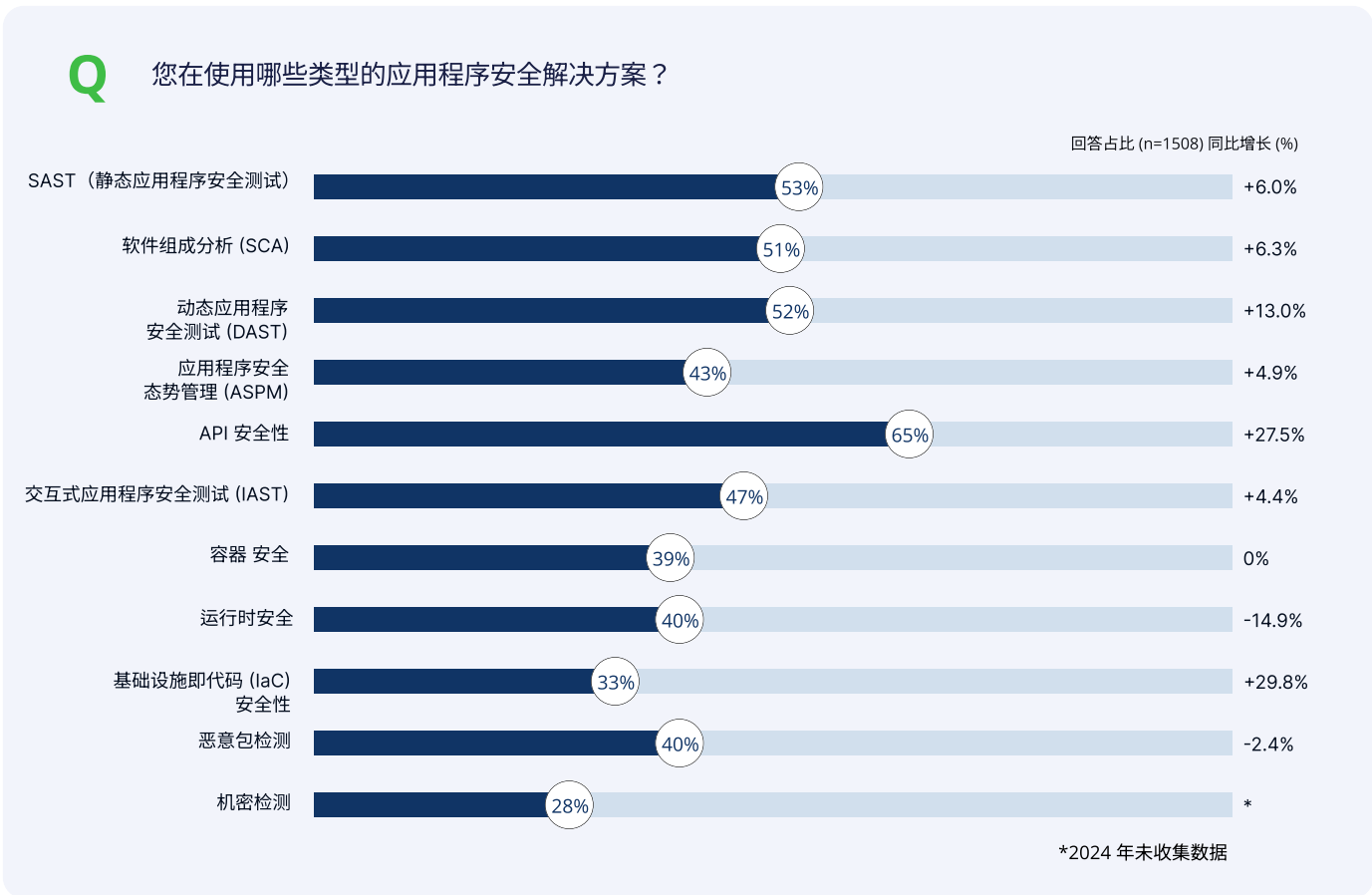
去年，73% 的受访者表示使用了七种或更多应用程序安全解决方案。今年这一比例为35%——降幅过大，无法将其视为随机波动而忽略不计。

两种解释都说得通。第一种是真正的整合：各组织正在对其安全技术栈进行合理化调整，用更少的工具实现更全面的覆盖。第二种是平台原生安全性正在取代零散的单点解决方案。

正如我们将在本节后文中详细分析的那样，38% 的受访者表示，他们现在主要依赖其现有 DevOps 平台的安全功能。如果平台解决方案纳入了先前需要单独工具才能实现的功能，那么受访者可能只是统计了更少的端点型解决方案，但同时仍维持了相当的功能覆盖范围。

尚不明确的是，这种覆盖范围是否确实等同。本节后文所涉及的工具类型数据引发了一些疑问。

组织正在使用的应用程序安全解决方案类型

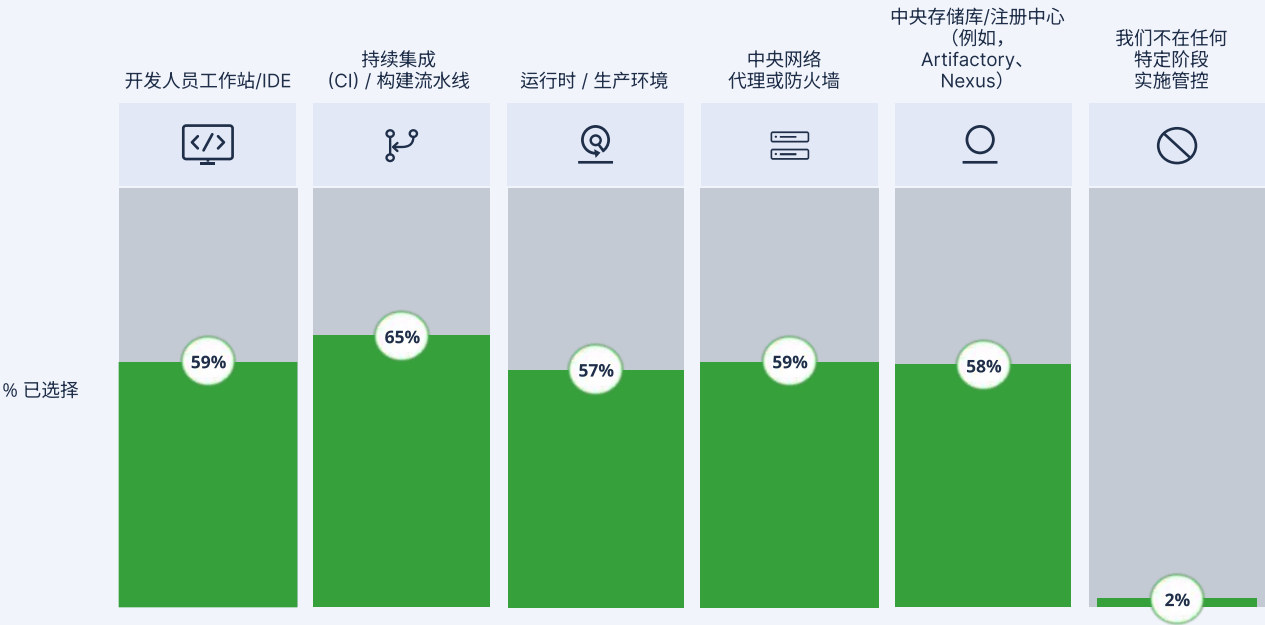


API 安全以 65% 领先，其次是 SAST (53%)、DAST (52%) 和 SCA (51%)。这四项是该行业最接近基本门槛的标准，但没有任何一项的采用率超过 65%。

有两个数字凸显出明显差距。机密检测为 28%，是所有已命名类别中采用率最低的一项。恶意软件包检测为 40%，与去年的 41% 基本持平，尽管 2025 年恶意软件包数量已达到历史新高。威胁大幅增长，但安全覆盖范围却并未提升。增长最快的威胁类别也是覆盖范围最少的类别，而且这一差距并未缩小。

组织实施安全控制的阶段

Q 您的组织在哪个阶段对开源软件包和其他外部组件实施安全控制（例如阻止、扫描、批准）？



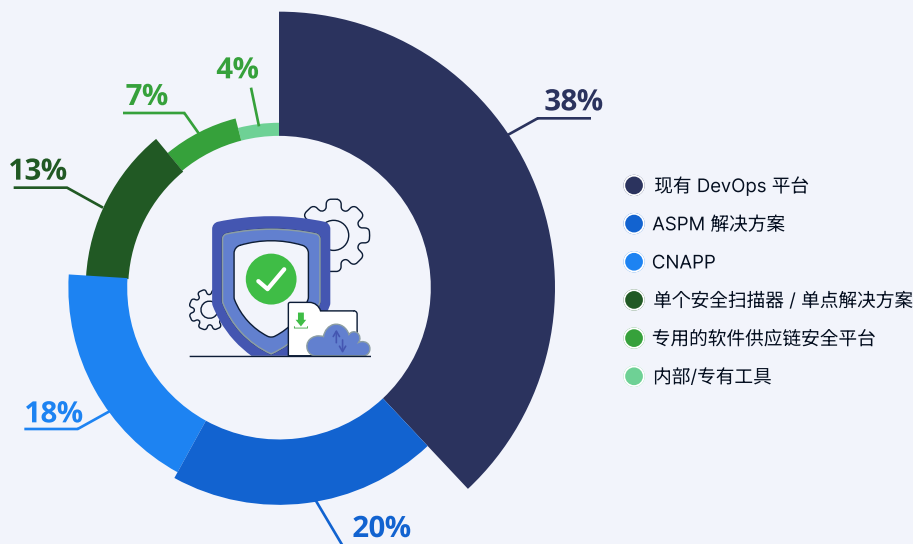
CI/构建流水线仍是主要的执行点，占 65%，反映出行业长期以来对左移投资的持续关注。开发人员工作站/IDE (59%)、中央代理 (59%) 和中央注册中心 (58%) 几乎并列第二。仅有 2% 在任何阶段都不实施控制措施。

问题在于，在 CI/构建阶段实施管控，只能在开发者已引用依赖项之后才能发现风险。无法预防开发人员计算机上的安全暴露。正如 9 月份的 npm 劫持事件所证明的那样，本地环境本身也可能成为被攻击的入口。



首选安全工具偏好

Q 贵组织最依赖哪一类应用程序安全解决方案？



38% 的受访者主要依赖其现有 DevOps 平台作为其主要安全能力。ASPM 解决方案以 20% 位居第二，CNAPP 以 18% 位居第三。仅有 7% 的受访者将专用软件供应链安全平台列为其首要依赖手段。

平台优先的偏好与工具整合的整体情况一致。但并非所有 DevOps 平台都以相同的方式处理安全问题。相关的问题在于，这些安全功能究竟来自何处：它们是真正基于深厚安全积淀而专门打造的，还是仅仅对开源工具进行了浅层封装，且后续持续投入有限？

拥有专用安全研究团队、持续更新的威胁情报，以及贯穿整个软件供应链的原生强制执行能力的平台，与仅仅将安全作为一项功能勾选添加上去的平台相比，体现出显著不同的安全态势。鉴于上文指出的恶意软件包检测和凭据扫描方面的缺口，平台安全能力的来源与深度值得仔细审视：不仅要是否具备安全能力，更要看这些能力是以多么严肃认真的方式构建而成。

3. 建立跨应用程序流水线的可见性和控制

组织中实施的安全框架



以下哪项安全框架级别已在您的组织中实施？

54%

软件包构建元数据已记录

56%

软件包基于专用主机构建

63%

发布前会验证软件包签名

63%

软件包构建数据和元数据已签名

2.37 / 4
平均采用实践数

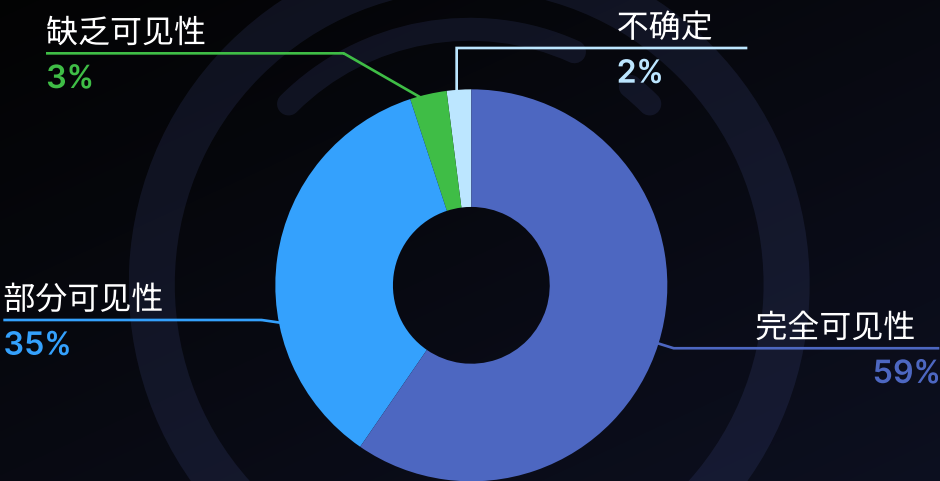
平均而言，组织仅实施了这四项实践中的略多于一半——每项实践单独采用时普及程度较高，但一致的端到端覆盖却较为少见。

能够查看流入流水线的全部内容、跟踪各项内容的创建主体，并按需出具佐证材料的组织，其安全态势与不具备该能力的组织存在本质区别。SLSA 等供应链完整性框架为强化软件构建和签名方式提供了结构化方法，而且各项实践的采用情况相当可观：63% 会对其构建数据和元数据进行签名，63% 会在发布前验证软件包签名，56% 会在专用主机上构建软件包，54% 会记录软件包构建元数据。单看数据，上述占比表现亮眼。

令人担忧的是，那些仅采用了其中部分实践的组织与那些始终如一地全面采用所有这些实践的组织之间存在差距。每项实践本身都有意义。它们共同构成了一条信任链，但存在缺口的信任链，其强度只取决于最薄弱的一环。

追踪生产环境中所运行软件的来源

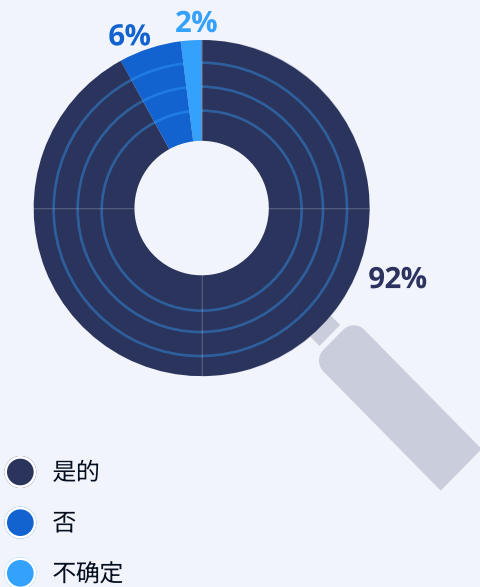
Q 您能否了解在生产环境中所运行软件的来源？



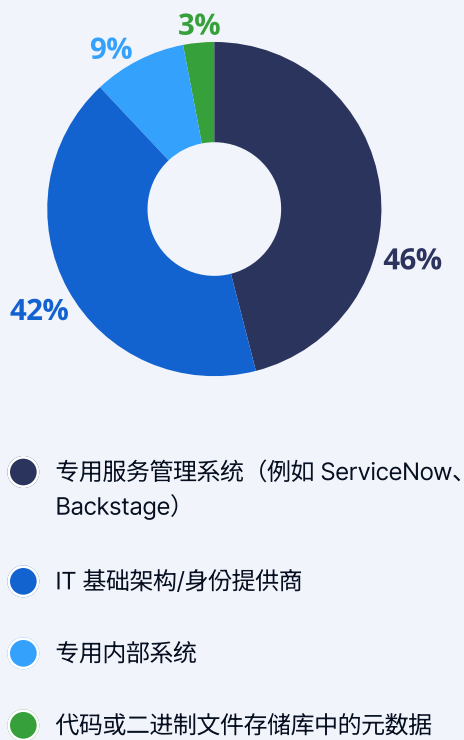
59% 的受访者表示，他们对生产环境中运行的软件来源具有完全可见性：谁提交了代码、经过了哪些测试，以及依赖项来自何处。35% 的受访者具有部分可见性，3% 则完全没有可见性。出于合规目的，部分可见性这一数字最为重要。EU Cyber Resilience Act、US Executive Order 14028 和 SOC 2 等监管框架越来越要求完整的来源可追溯性，而非来源的大概信息。当审计人员提出这些问题时，一个能够回答其中大多数但无法全部回答的组织，其处境与一个完全无法回答这些问题的组织相差无几。

跟踪每个应用程序的所有者

Q 对于您在组织中构建的每个应用程序，您是否会跟踪该应用程序的所有者？



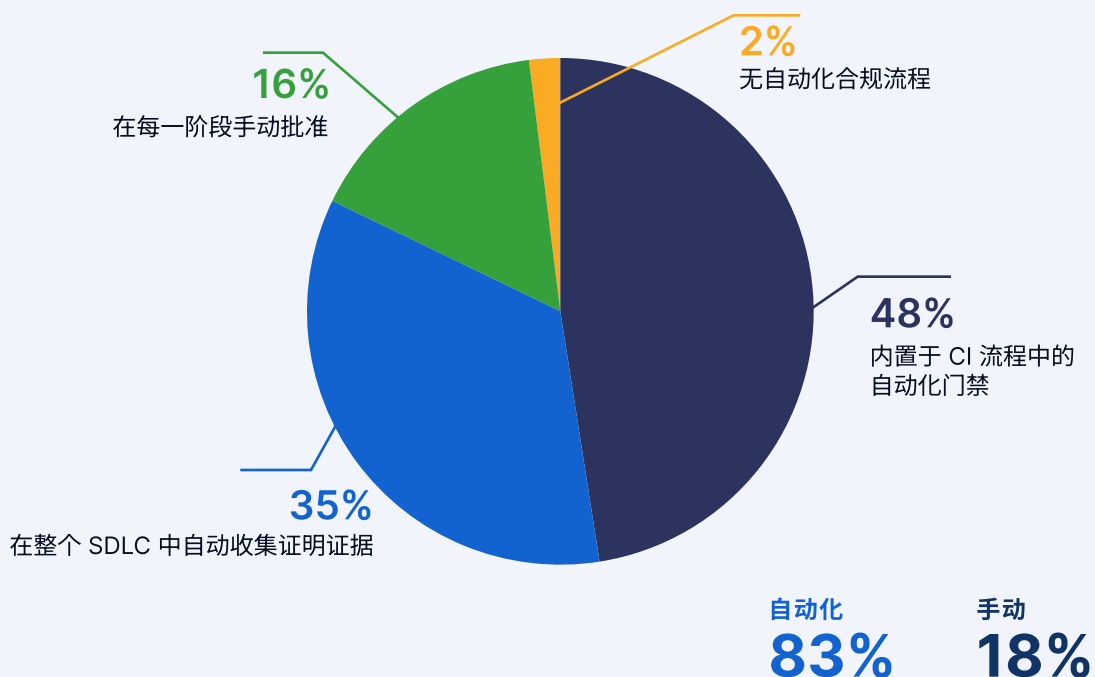
Q 对于您在组织中构建的每个应用程序，您如何跟踪应用程序的所有者？



92% 的受访者表示，他们会跟踪所构建的每个应用程序的所有者，这是一个良好的基准。其中，46% 使用 ServiceNow 或 Backstage 等专用服务管理系统，42% 使用其 IT 基础设施或身份提供商，9% 使用专用内部系统。只有 3% 的受访者直接在代码或二进制仓库中添加元数据。控制方式的多元化特征值得关注。同一行业内的所有权跟踪分散在四种不同的方法中，意味着没有主导标准，这使得跨团队和跨组织的问责制更难以建立。

确保软件质量和合规标准

Q 出于合规性和治理目的，您如何确保在软件创建和发布过程中遵循软件测试/质量标准？

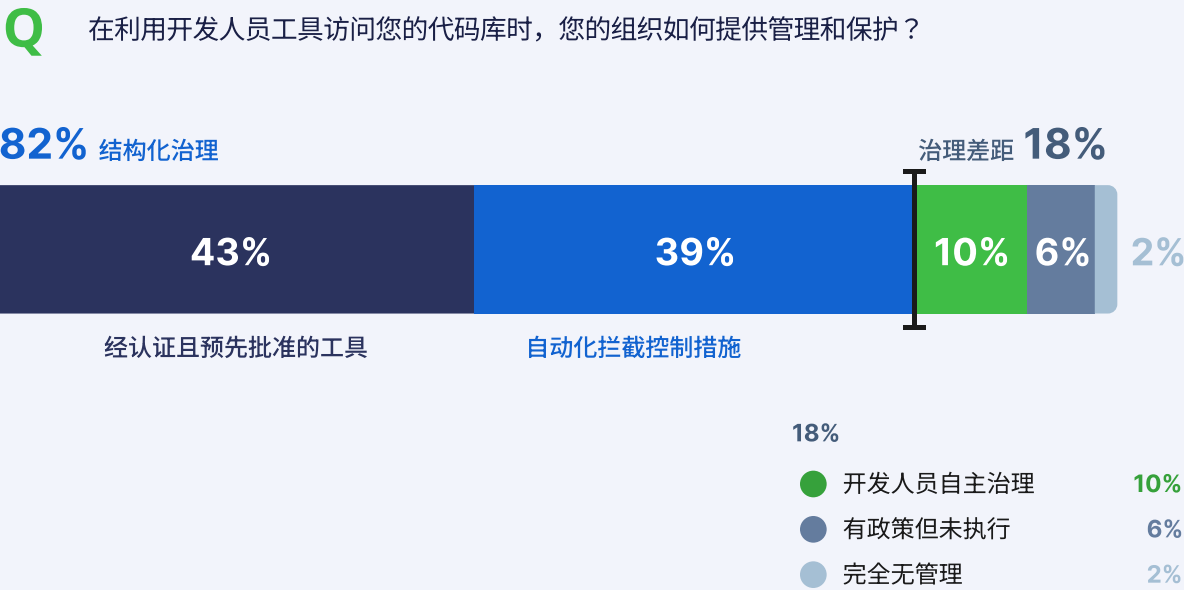


48% 的受访者依赖内置于 CI 流程中的自动化门控，这是最常见的方法。35% 的受访者在整个软件交付生命周期中自动收集证明证据。16% 的受访者仍依赖手动审批将软件从一个阶段推进到下一个阶段。推进流程中的任何手动干预都是潜在的不一致节点、意外错误的机会以及蓄意操纵的途径。在大规模场景下，手动门控会减慢交付速度并引入人为错误，而且在截止日期压力下，它们也是最先被绕过的环节。

4. 新的治理缺口：开发人员工具

59% 的受访者声称已实现完整的软件来源可见性，但 48% 的受访者仍需要一周或更长时间来进行审计准备，由此产生的差距值得审视。完整的可见性应意味着快速完成审计准备。事实并非如此，这表明对于许多组织而言，"完整的可见性"意味着数据存在于某个地方，而不是数据已结构化、可访问，并且可按需用于审计。

管理和保护访问代码库的工具



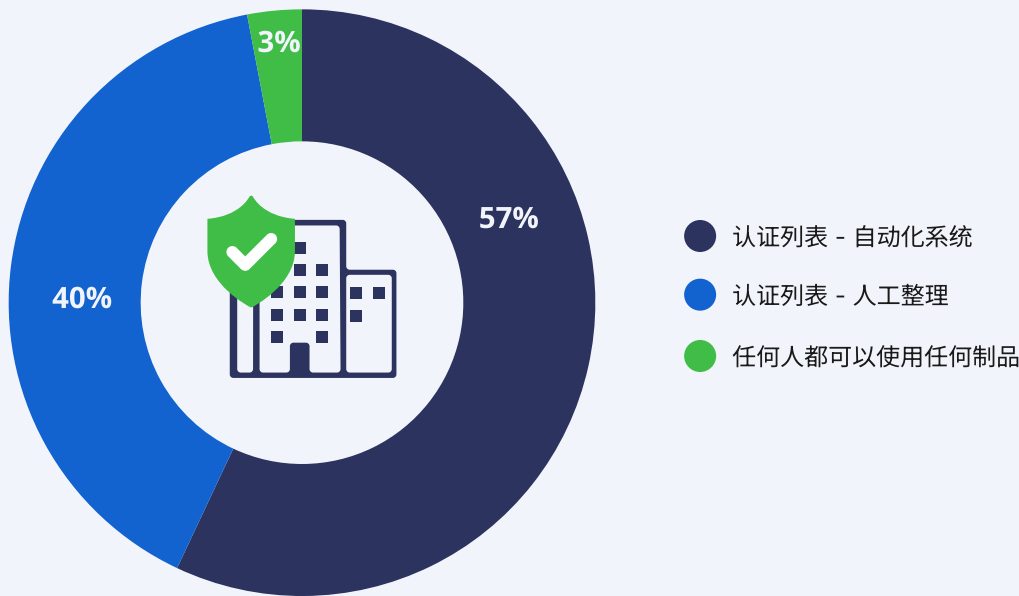
本小节在去年的报告中没有对应内容。2024 年，IDE 扩展、AI 编码助手和 MCP 服务器尚未重要到需要纳入调查的程度。2025 年，它们已成为攻击面，JFrog 在此记录了 56 个恶意扩展、一个 CVSS 9.6 的 RCE 漏洞，以及在 MCP 服务器中发现的数百个高风险问题。如今，治理数据已经存在，可以提出这样的问题：各组织是如何管理这些工具的？

43% 的组织维护一份经过认证的预先批准工具列表，并按需执行相关策略。39% 的组织使用自动化控制措施来监控并阻止未经批准的工具。10% 的组织完全依赖开发人员自行管控其对这些工具的使用。6% 的组织制定了一些策略但未积极执行，2% 的组织则完全没有任何管理或限制措施。这意味着，18% 的受访者对于开发人员 IDE 中可直接访问其代码库、凭据和 CI/CD 流水线的工具，要么完全没有治理，要么只是名义上进行治理。鉴于本报告所记录的有关恶意 IDE 扩展和 MCP 服务器漏洞的内容，这是今年调查中最明显的治理漏洞。

依赖开发人员自我治理的 10% 尤其值得关注。自我治理假设开发人员在截止日期压力下、采用新的 AI 辅助工作流程时，以及在攻击面正被积极利用的情况下，始终能够做出良好的安全工具决策。这是一个非常大胆的假设，这或许有理由将“无治理”组织的比例拉近至四分之一左右。

治理模型上下文协议 (MCP) 使用情况

Q 您的组织如何管控模型上下文协议 (MCP) 的使用？

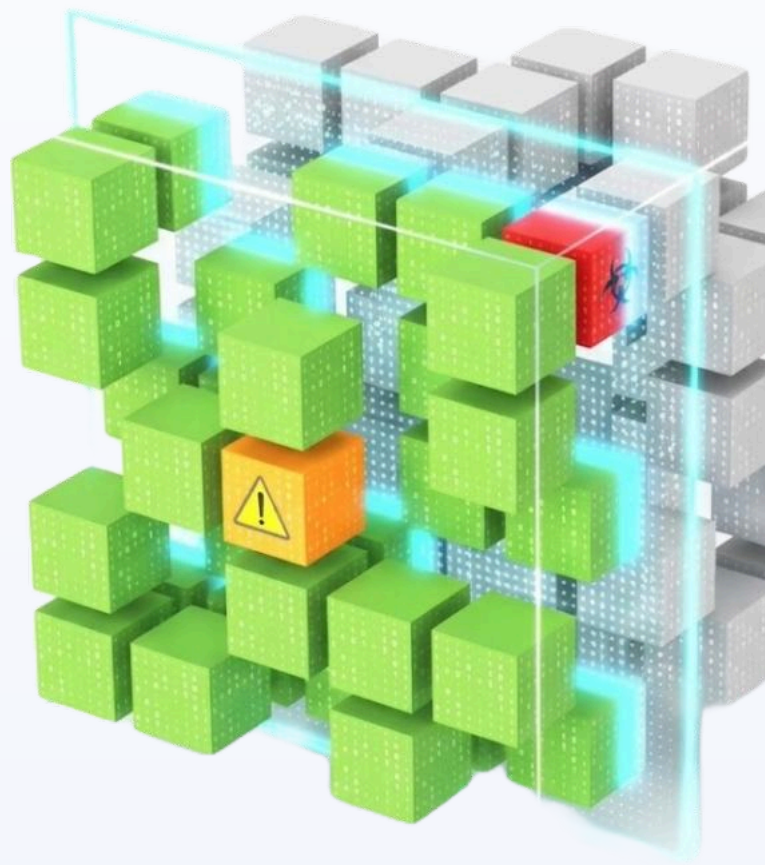


与开发者工具治理图景相比，MCP 治理乍一看显得成熟得多。97% 的受访者表示，他们采用某种形式的认证清单，或通过人工监管 (40%)，或通过自动化系统 (57%) 进行管理。只有 3% 的受访者表示，任何人都可以随意使用他们想要的任何内容。

JFrog 对数千个 MCP 服务器和实用工具进行扫描后，发现了数百项高风险问题，其中包括 CVE-2025-6514，这是一个影响 mcp-remote 实用工具 0.0.5 至 0.1.15 版本的 RCE 漏洞，CVSS 评分为 9.6，于 2025 年 7 月发布。如果 97% 的组织声称已通过 MCP 治理认证，那么值得提出的是，该认证流程实际检查的是什么。如果只是按名称跟踪已批准服务器的列表，而不持续扫描这些服务器是否存在新披露的恶意活动和漏洞，就会始终落后于威胁。不进行扫描的治理只是一个列表，而非管控措施。

5. 成本是多少？

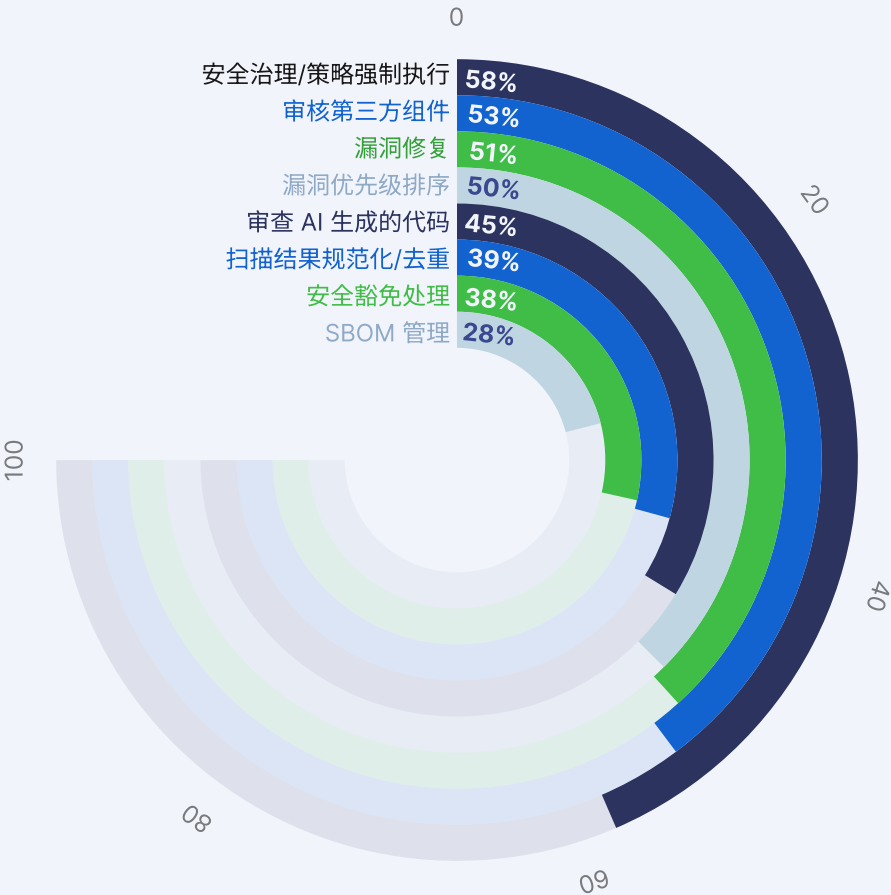
本节所记录的安全工作一直都需要大量人力投入，但其必须跟上的开发速度却已大幅加快。AI 辅助开发压缩了时间周期，智能体工作流在无需开发人员直接操作的情况下引入了依赖项，而通过流水线流转的制品数量增长速度已超越任何人工流程所能企及的范围。依赖人工审查周期、手动审批或手工整理证据的安全与合规职能，不仅效率低下，在结构上与其旨在保护的环境也不匹配。



DevSecOps 供应链中主要的耗时问题



哪些软件供应链安全职责占用了您的 DevSecOps 利益相关者最多的时间？



整个软件开发生命周期中，安全治理与策略执行是最常被提及的环节，占比为 58%。鉴于本报告其他部分所记录的合规负担，这并不令人意外。审查第三方组件以 53% 位居第二，漏洞修复以 51% 位居第三。

今年新出现的情况是：审查和加固 AI 生成的代码占比达 45%，其耗时负担几乎与漏洞修复一样常见。这一类别在去年的调查中并不存在，且目前尚不存在专为大规模审查 AI 生成的代码而设计的成熟的自动化工具。组织大多仍在手动执行这项工作，这意味着成本高昂、缺乏一致性，而且不太可能跟上本报告前文所述的 AI 辅助开发规模。



开发人员等待新 OSS 软件包审批的时间

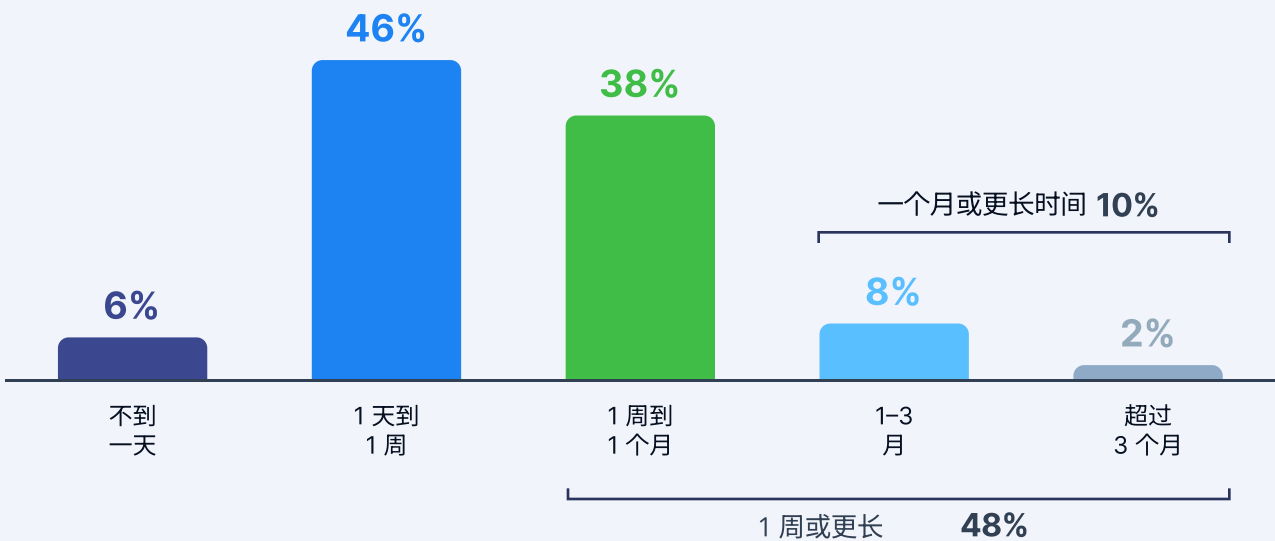
Q 贵组织中的开发人员通常需要等待多长时间，新 OSS 软件包才能获批？



今年的情况比去年要好。50% 的受访者需要等待一周或更长时间才能获批使用一个新软件包，低于 2024 年的 68%。等待两周或更长时间的受访者比例从 41% 降至 23%。这是一个意义重大的转变。这表明，来源层面的自动化（例如自动化扫描、整理的经批准列表、基于策略的审批）正开始缓解人工审批的瓶颈问题。摩擦并未消失，但正在缩小。目标应该是让这一趋势持续下去：审批时间每缩短一周，开发人员绕过该流程的动机就会减少一分。不过，正如我们所展示的那样，至少几天的等待期实际上可能是一件好事，有助于避免软件包劫持攻击。

生成合规审计证明所需时间

Q 贵组织的开发团队平均需要多长时间才能为每个应用程序生成合规审计证明（例如：SOC 2、HIPAA）？



59% 声明完整溯源可见性

但可见性 ≠ 审计就绪

48% 花费一周或更长时间审计准备

48% 的受访者需要一周或更长时间才能生成每个应用程序的合规审计证明，10% 需要一个月或更长时间，而只有 6% 能在不到一天的时间内完成。这一数字与本节前文中提到的 59% 受访者声称具有完整来源可见性的结果存在矛盾。这一差距表明，对于许多组织而言，数据虽然存在，但其结构或可访问性尚不足以支持按需实现审计就绪，而且仍然需要人工汇总证据。这正是大多数组织尚未打通的治理自动化的最后一公里。

核心要点



边界已扩展。但配套安全控制措施并未同步扩容。

来源控制机制最初专为软件包注册中心而构建，多数组织的控制边界仍止步于此。模型注册中心、MCP 服务器和 IDE 扩展在很大程度上游离于相同的治理框架之外，尽管本报告记录的 2025 年活跃攻击正是发生在这些层面上。将来源控制扩展至这些新层面已成为一项硬性要求。



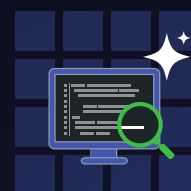
工具减少了，但缺口依然如故。

使用七种或更多安全工具的组织占比从 73% 降至 35%，这可能反映出围绕平台原生安全的真正整合。但最关键的覆盖缺口并未缩小：恶意软件包检测率停滞于 40%，而此次调查中首次纳入的机密检测率仅为 28%。只有在剩下的部分覆盖了实际风险内容的情况下，整合才算有了真正的进展。



未被执行的治理不是真正的治理。

97% 的受访者声称已实现经认证的 MCP 治理，但实际上只有 57% 会对接入的 MCP 服务器进行筛选整理。18% 的组织对于访问其代码库的开发人员工具没有治理措施，或虽有政策但未得到执行。59% 的受访者表示已具备完整的来源可见性，但近一半仍需要一周或更长时间才能生成审计证明材料。在采购来源、工具和合规性各方面，已报告的治理情况与实际运营现实之间的差距，是本节最关键的发现。



在缺乏成熟工具支持的情况下，审查 AI 生成的代码如今已成为一项重大的安全负担。

45% 的 DevSecOps 利益相关者表示，审查和加固 AI 生成的代码是主要的耗时项。尽管这一类别未出现在去年的调查中，但如今它已几乎成为与漏洞修复一样常见的负担。然而，与漏洞修复不同的是，在本次调查进行时，用于 AI 生成代码审查的专用自动化工具仍处于起步阶段。随着 AI 辅助开发加速推进以及专用工具不断成熟，这种负担与应对工具之间的“赛跑”，将成为 2026 年影响最重大的 DevSecOps 挑战之一。

AI 模型与智能体因素

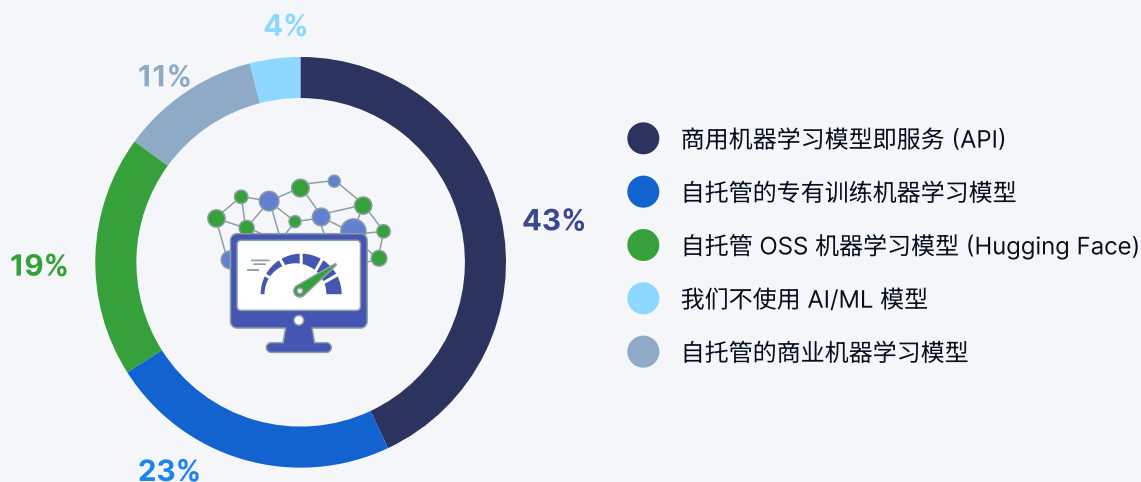
前三节记录了一条供应链在已知威胁冲击下所承受的压力，这些威胁的规模和速度都在不断增长。最后一节记录了一些新情况：一种新兴风险类别，其中治理框架正在实时构建中，而威胁面的扩展速度超过了组织对其进行测绘的速度。

AI 不再只是软件供应链中的一个前瞻性考量，它本身就是供应链。模型就是依赖项，AI 生成的代码就是生产代码，而能够访问代码库和凭据的智能体工具如今已成为开发人员的标准基础设施。问题不再是是否要对供应链中的 AI 进行治理，而是治理是否跟得上。



组织如何使用 AI/ML 模型

Q 作为您正在开发的应用程序的一部分，您使用 AI/ML 模型的主要方式是什么？



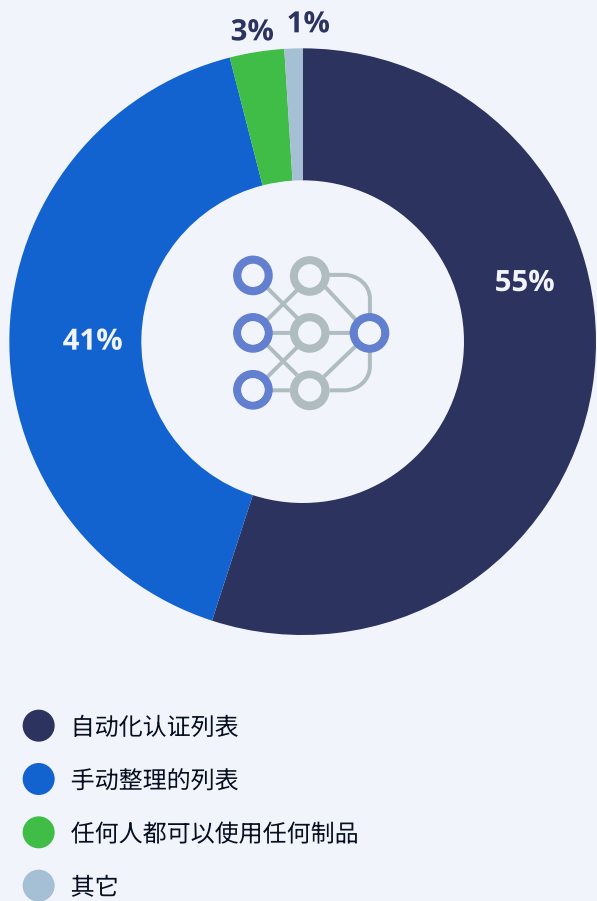
43% 的受访者主要通过商业 API 服务形式使用 AI/ML 模型：OpenAI、Claude、Gemini 及同类服务。基于 API 的使用方式通常是最易于治理的模式：模型不会进入组织的基础架构，访问可通过凭据进行控制，并且调用可记录到日志中并接受审计。话虽如此，这并非毫无风险，通过 API 调用实施的提示注入和敏感数据泄露都是已经记录在案的攻击媒介，本节稍后将对此进行介绍。

更复杂的治理挑战在于剩余的 53% 受访者，他们以某种形式自托管模型：23% 自托管专有训练模型，19% 自托管从 Hugging Face 等注册中心下载的 OSS 模型，11% 自托管商业模型。与基于 API 的使用方式不同，自托管模型会引入一类风险，即模型制品本身会进入组织的基础设施。与任何软件依赖项一样，它可能携带恶意有效负载、易受攻击的组件和许可证义务，并且需要像供应链中的任何其他软件包一样遵循相同的治理规范。

本报告记录了 2025 年 Hugging Face 上的 495 个恶意模型。19% 主要采用自托管 OSS 模型的组织恰恰是从同样的注册中心拉取模型。仅有 4% 的受访者表示完全不使用 AI/机器学习模型，这一数字可能在下一个报告周期内趋近于零。

治理模型制品

Q 贵组织如何管控 AI/ML 模型制品的使用？

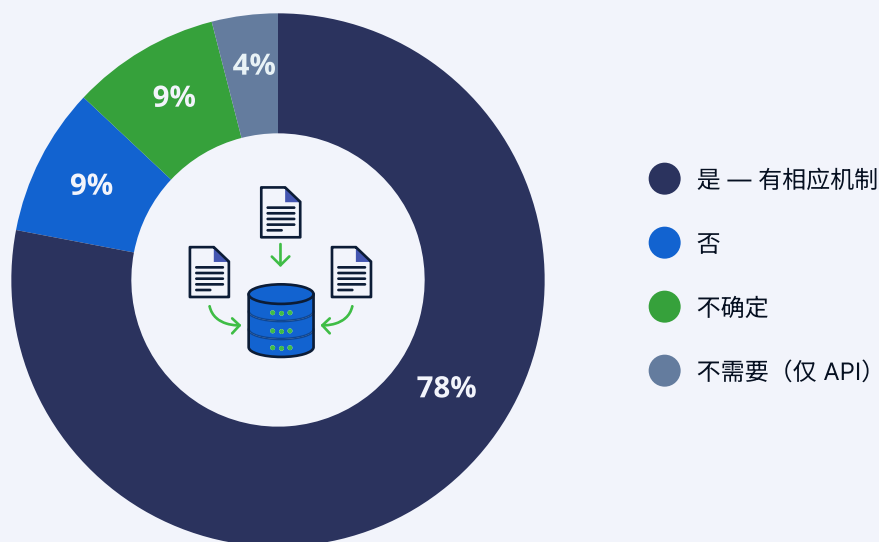


55% 使用模型及其版本的自动化认证批准名单，41% 使用人工整理的认证名单，而 3% 则允许任何人随意使用任何内容。总体来看，96% 采用了某种形式的认证治理，这与去年的 94% 基本持平，也表明模型制品治理正趋于成熟。

如上一节所指出，问题在于“认证”在实践中究竟意味着什么。未经恶意有效负载或漏洞扫描的认证列表，只是一份名称清单，并非安全控制措施。如果 Hugging Face 上那 495 个恶意模型在其有效负载被发现之前就已出现在认证列表中，那么该治理框架就未提供任何保护。

治理 AI 输入和输出

Q 您的组织是否已建立相应机制来检查 AI/ML 服务的输入和输出？

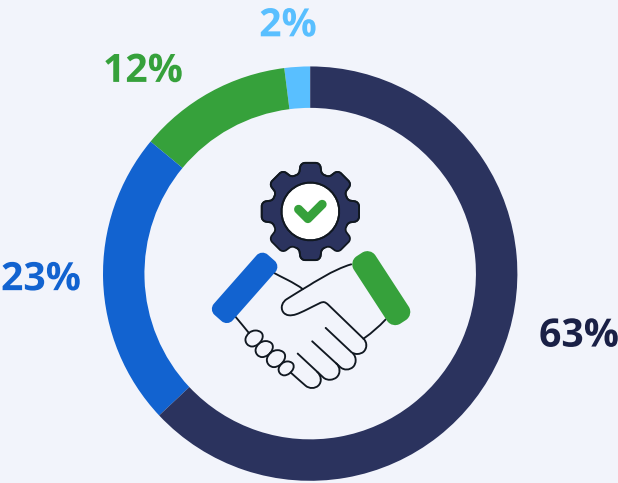


78% 的受访者表示，是的，他们已建立相关机制来检查提示注入、敏感数据泄露及类似的 AI 特有风险。22% 的受访者没有相关机制、对此不确定，或认为由于他们只使用基于 API 的模型，因此不需要此类机制。

22% 缺乏输入/输出检查，这代表着一个值得重视的缺口。提示注入如今已是一个记录在案的攻击媒介，而非理论上的攻击媒介，并且通过 AI 服务发生的敏感数据泄露已经引发了真实事件。那些仅依赖 API 提供商控制措施的组织，是将这项责任外包了出去，而不是消除了风险。

对 AI 建议的安全修复的信任

Q 以下哪项陈述最贴切地描述了您对 AI 工具建议的安全修复的信任程度？



- 坚实的起点，细致审查
- 确定性解决方案，仅需快速评估
- 仅提供大致方向 — 由我自行撰写
- 不会考虑 AI 建议的修复方案

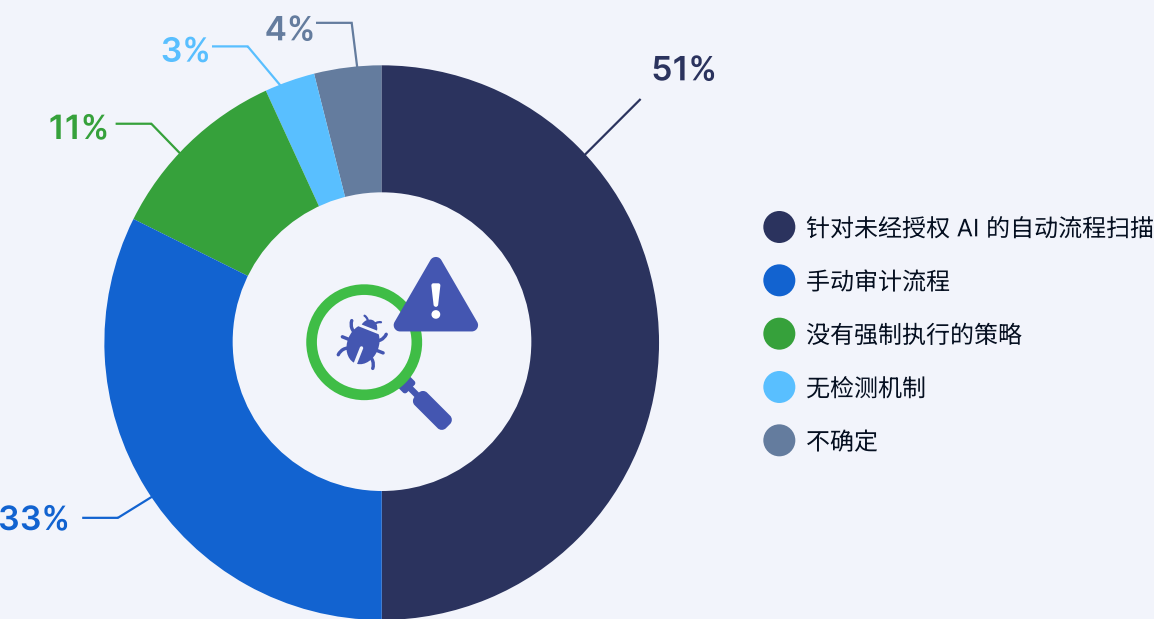
这个问题触及了 2025 年最重大的治理问题之一的核心：当 AI 工具提供代码修复、安全补丁和修复步骤相关建议时，开发人员在评估建议的过程中保持了多大的批判性？

23% 的人会将 AI 的建议用作最终解决方案，只需在实施前进行快速审查即可。63% 的人会将其作为一个强有力的起点，在实施前仔细审查其逻辑。12% 的人只会用它来了解大致方向，然后自行从头编写修复方案。2% 的人根本不会使用 AI 建议的安全修复方案。

将 AI 建议视为近乎最终方案的占比达到 23%，这一数字值得关注。众所周知，AI 编码工具会生成看似合理却不正确的安全修复方案，这些方案只是解决了表面症状，并未消除潜在漏洞，或者还会引入新的漏洞。快速审查有时发现不了这些问题。这与上一部分的调查结果直接相关：审查并加固 AI 生成的代码如今已成为 DevSecOps 的首要负担，而这一发现表明该负担是合理的。

检测影子 AI

Q 您是否有方法检测开发环境或应用程序中的影子 AI（未授权使用 AI 驱动的工具或服务）？



51% 的受访者拥有自动化流程，可扫描并标记未经授权的 AI 使用。33% 的受访者使用手动审计流程。总体而言，83% 的受访者具备某种形式的影子 AI 检测。

剩余 18% 的受访者属于以下情况之一：有策略但未检测 (11%)、完全未检测 (3%)，或不确定 (4%)。针对未经授权 AI 的策略如果未主动实施，其情况与本报告前文记录的来源限制策略如出一辙，只能作为一种假设，而不是一种管控措施。

此外，本报告显示，各地区在采用用于检测影子 AI 的工具和流程方面差异很大，这意味着跨国公司在自动化方面存在显著差距。

影子 AI 是组织看不见的供应链风险。使用未经授权的 AI 编码助手的开发人员可能会通过外部基础设施路由代码、将敏感上下文存储在第三方服务中，或生成带有依赖项的代码，从而绕过所有经批准的来源控制。18% 未进行主动检测的组织对这一切毫无可见性。



API 优先/无头式 AI 使用掩盖了自托管风险。

43% 的组织主要通过商业 API 来使用 AI，这是最易于管控的模式。但 53% 的组织采用某种形式的自托管，从同一注册中心拉取模型，而 2025 年正是在这些注册中心检测出 495 个恶意模型。自托管模型属于供应链依赖项，需要按此进行管控。



经认证 ≠ 安全：AI 治理的假象。

97% 的组织声称拥有经过认证的模型治理，78% 会检查 AI 输入和输出。这些数字看起来不错。但是，未经有效负载扫描的认证列表，以及未覆盖完整攻击面的输入/输出检查器，提供的更多是治理的表象，而非现实情况。真正重要的是认证流程实际能发现什么，而本报告中的恶意模型数据表明，当前的标准需要提高。



23% 的人对 AI 安全修复方案抱有过高信心。

近四分之一的受访者仅经过快速审查，就会将 AI 建议的安全修复视为近乎确定无疑的方案。AI 工具生成看似可信但实际上错误的安全修复的频率之高，足以使这种做法带来实际风险。63% 受访者将 AI 建议视为有力起点但认为仍需仔细审查，他们的判断是正确的。



影子 AI 是不受治理的边缘地带。

18% 的组织尚未启用任何有效的影子 AI 检测机制。每一位使用未授权 AI 工具的开发人员，都可能成为一个潜在的数据泄露点、一个绕过源码控制的缺口，以及一个治理盲点。检测不是可选项；这是了解组织内 AI 使用实际范围的唯一方式。不同地区的组织在检测影子 AI 的方式上存在显著差异，这为任何跨境运营的企业带来不平衡的管控面。

方法论

本报告综合了源自 JFrog 使用数据的洞察、JFrog 安全研究团队的 CVE 分析结果以及委托第三方调查数据。以下是对每个来源的详细介绍：

JFrog Platform 使用数据

本报告重点介绍的技术使用趋势来自对匿名化 JFrog Platform 数据的全年分析，并借助了多年的历史数据，这些数据涵盖数千家客户、数十万个制品库，以及 PB 级制品。

软件包受欢迎程度通过三个维度的同比增长来体现：制品库数量、制品数量和请求量（上传和下载）。请求数量很好地反映了不同包类型在软件开发中被主动使用的频率。

少数几家企业可能会影响这些排名结果。但是，由于我们还会考察制品操作，因此我们可以有把握地得出结论，判断哪种包类型正在被积极用作开发流程的一部分。

JFrog 安全研究团队的分析

作为指定的 CNA，JFrog 安全研究团队会定期监控和调查新漏洞，以了解其真实严重性，并发布此类信息，让整个社区和所有 JFrog 客户获益。

本报告包含通过 JFrog Catalog 服务从公开来源获取的数据、从国家漏洞数据库获取的 CVE 信息，以及 JFrog 安全研究团队针对这些数据源执行的专有分析。



专项调研结果

JFrog 委托 Atomik Research 开展了一项国际在线调查，调查对象为来自美国 (n=508)、英国 (n=125)、印度 (n=167)、德国 (n=120)、法国 (n=125)、澳大利亚 (n=165)、新加坡 (n=174) 和西班牙 (n=124) 特定行业¹ 的 1,508 名 IT 专业人士。

该样本由在其所在组织的信息技术、信息系统或技术部门中担任特定职位² 的全职员工组成。此外，所有受访者均表示，其所在组织的员工总数为 1,000 人或以上，并确认其组织内有一个至少由 50 名成员组成的软件开发团队。所有参与者均可选择查看该在线问卷的英文、法文、德文、印地文和西班牙文译本。

整体样本的误差范围为 ± 3 个百分点，置信水平为 95%。实地调研于 2026 年 1 月 28 日至 2 月 13 日期间进行。Atomik Research 是一家富有创意的市场调研机构。

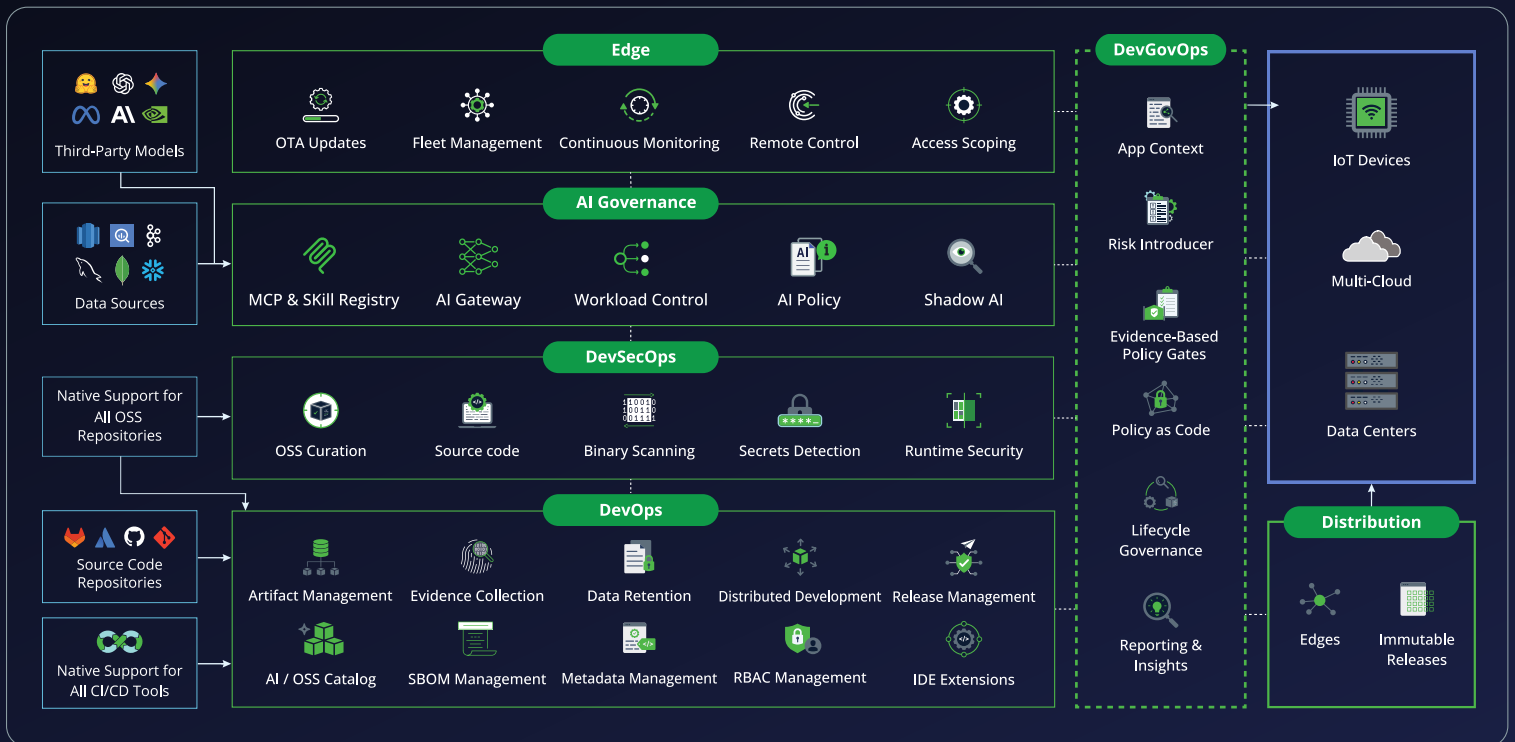
¹ 要获得参与资格，所有受访者均须表明其就职于服务以下行业的组织：(a.) 航空航天 (b.) 建筑与工程 (c.) 汽车 (d.) 银行、金融服务、保险与金融科技 (e.) 能源、石油、天然气 (f.) 政府或公共部门 (g.) 医疗保健与生命科学 (h.) 酒店服务业 (i.) 制造业 (j.) 零售业 (k.) 科技 (l.) 交通运输与物流 (m.) 公用事业、电信与电力

² 要获得参与资格，所有受访者均须表明其工作职能属于或类似于以下岗位：(a.) AI 专家或 AI 工程师 (b.) 应用程序安全工程师 (c.) 网络安全工程师 (d.) 数据科学家 (e.) 开发人员 (f.) DevOps 架构师 (g.) DevOps 工程师 (h.) 工程经理 (i.) 机器学习专家或机器学习工程师 (j.) 平台工程师 (k.) 安全架构师 (l.) 安全研究员 (m.) 站点可靠性工程师 (n.) 软件架构师 (o.) 软件开发人员 (p.) 软件工程师 (q.) 解决方案架构师，并且还须表明其受雇于所在组织的信息技术、信息系统、技术部门或 IT 产品开发部门。



关于 JFrog

JFrog Ltd. (纳斯达克：FROG) 作为统一 DevOps、DevSecOps、DevGovOps 和 MLOps 平台的创建者，致力于打造一个从开发到生产均能无摩擦交付软件的世界。在“流动软件”愿景的推动下，JFrog 平台作为软件供应链记录系统，旨在助力组织快速、大规模地构建、管理和分发安全软件。全面的安全功能有助于识别、防护和修复威胁及漏洞。通用、混合与多云的 JFrog 平台可通过主要云服务提供商以 SaaS 服务形式提供，也可进行自托管部署。全球数百万用户和约 6,600 家组织（包括大多数财富 100 强企业）都依赖 JFrog 解决方案，在 AI 时代安全地实现数字化转型。要了解更多信息，请访问 <https://www.jfrogchina.com/>



JFrog ARTIFACTORY

管理软件制品、容器和机器学习模型生命周期的黄金标准，原生支持 60 多种不同的软件包和制品技术。

JFrog RUNTIME

获取软件包级别运行时漏洞的实时可见性，对潜在威胁进行优先级排序，并快速识别其来源和开发人员，从而实现快速修复。

JFrog CURATION

软件供应链的守护者，可在恶意或有风险的软件包、模型和 IDE 扩展到达开发人员的机器之前就挡在门外。

JFrog XRAY

贯穿整个生命周期的安全层，持续扫描二进制文件、容器和 AI 制品，检测从构建到发布整个过程中的漏洞和许可证合规性问题。

JFrog ADVANCED SECURITY

通过软件供应链安全暴露扫描、代码扫描和上下文感知的漏洞分析，打造新一代应用程序安全性，从而削减告警噪音，让修复工作真正产生实效。

JFrog APPTRUST

用于跟踪所有权、生命周期阶段和合规性证据的治理层，以便在应用程序层面管理风险。

JFrog CONNECT

将企业 DevOps 和安全实践引入 IoT 开发，以实现大规模 IoT 机群和软件更新管理。

JFrog ML

借助这一站式解决方案将创意转化为生产成果，构建、部署、管理并监控您所有的 AI 工作流，无论是生成式 AI、大语言模型，还是经典机器学习。

JFrog DISTRIBUTION

将您的信任圈扩展到软件交付的最后一英里，并将软件带入理想位置，以实现最佳使用。

JFrog AI CATALOG

您的 AI 生态系统的可信数据源，包含用于模型、MCP 服务器、智能体 skill 和其他 AI 制品的注册表，让团队能够快速采用 AI，而无需在治理、安全或合规性方面妥协。

本报告美国联邦证券法定义的“前瞻性”陈述，包括但不限于关于 JFrog 使用数据的陈述，以及 AI 对软件供应链增长和安全影响的风险。

这些前瞻性陈述基于我们当前的假设、期望和信念，并受到重大风险、不确定性、假设和情况变化的影响，因此可能导致 JFrog 的实际结果、业绩或成就与任何前瞻性陈述中明示或暗示的内容存在重大差异。

有许多因素可能导致实际结果、业绩或成就与本报告中的陈述存在重大差异，包括但不限于我们向美国证券交易委员会提交的文件中详述的风险，这些文件包括我们截至 2025 年 12 月 31 日的 Form 10-K 年度报告、Form 10-Q 季度报告，以及我们可能不时向美国证券交易委员会提交的其他文件和报告。前瞻性陈述仅代表我们截至本新闻稿发布之日的信念和假设。除法律要求外，我们不承担更新前瞻性陈述的任何义务。

