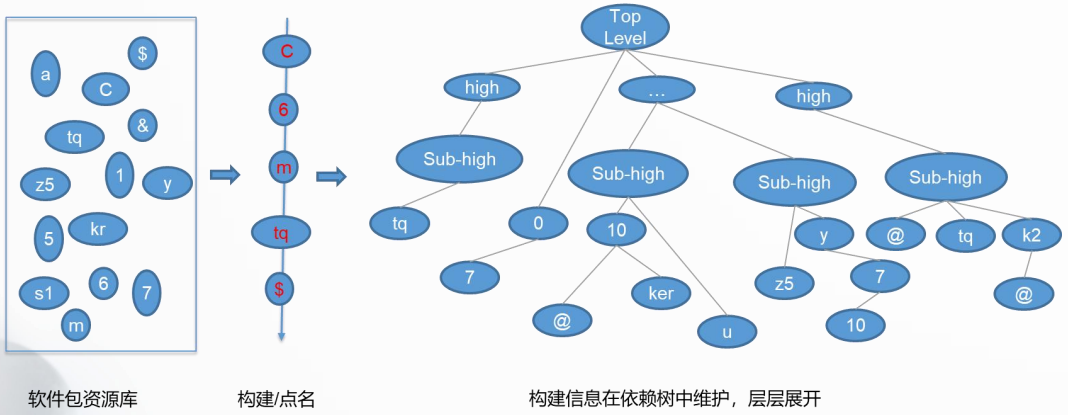

GOTC 2023
全球开源技术峰会
THE GLOBAL OPENSOURCE TECHNOLOGY CONFERENCE
OPEN SOURCE, INTO THE FUTURE
「分论坛标题」专场
开源软件供应链安全的挑战与实践
王宇 I 思探明中国区（Scantist China）产品专家
2023年5月29日

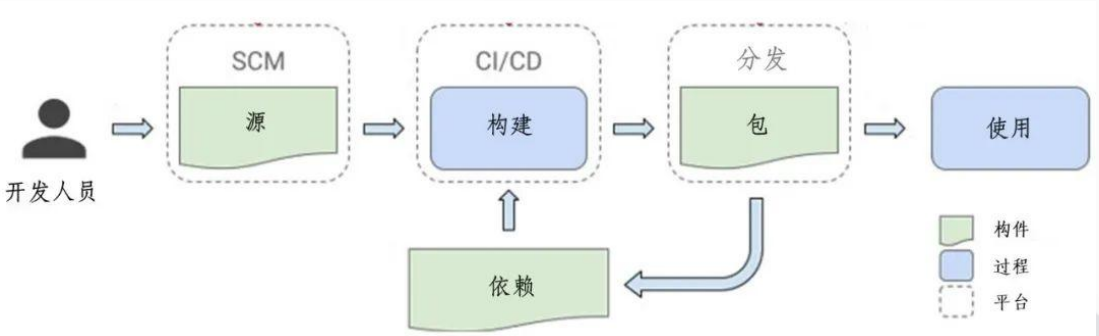
软件供应链定义

解释1： 软件系统在开发和运行中按照调用关系组织起来的软件包集合。

解释2：一个构件的供应链是构件所依赖关系的供应链加上本身的源和构建过程的组合。



软件包资源库　　构建/点名　　构建信息在依赖树中维护，层层展开



软件供应链是软件开发生产过程中的所有过程及要素的集合，包括源代码、第三依赖、硬件和基础设施、操作系统、编译器和编辑器、驱动程序和依赖关系、开源脚本和打包软件、存储库引擎、测试套件和CI/CD工具、云服务和数据中心以及人员。

开源软件已经成为软件供应链的最重要的基础设施

60 – 90%

APPLICATION CODE IS OPEN SOURCE

100K+

OPEN-SOURCE RELEASES PER DAY

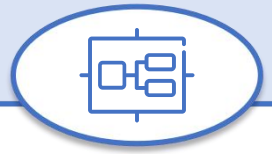
300M

OPEN-SOURCE LIBRARIES BY 2026

1 : 8

ONE OPEN-SOURCE COMPONENT CALLS 8 OTHERS ON AVERAGE

软件供应链面临风险



供应风险

- 断供/出口管制
- 技术垄断
- 停服/无人维护
- 漏洞不公开
- 难以获得源码

安全风险

- 恶意代码植入
- 供应链投毒
- 主动/被动引入漏洞
- 信息泄露
- 漏洞攻击/利用
- 漏洞传递

知识产权风险

- 侵权
- 专利
- 商标
- 许可证兼容

技术演进/复杂性风险

- 软件成分分析复杂（闭源、混源、开源）
- 开发过程逐步迭代（瀑布、敏捷、Devops）
- 应用架构升级（单体、微服务）
- 基础设施升级（物理机、虚拟化、容器化）

开源软件　软件更新　开发人员

软件供应链风险根因分析

供应链条长

行业应用软件供应链环节多，闭源、开源混杂，层层打包封装

不透明传递

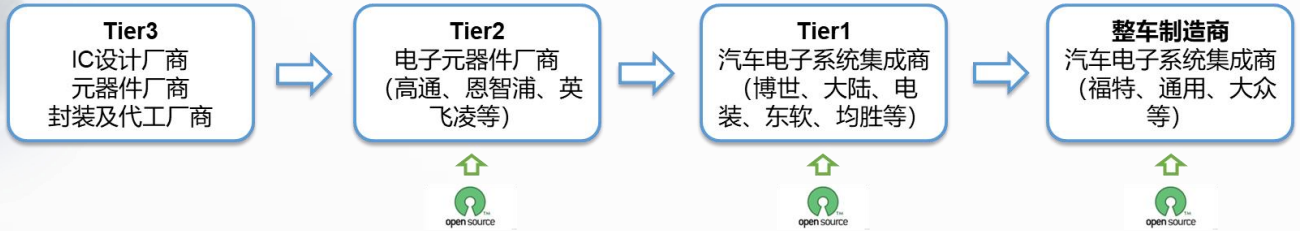
供应链上下游之间采用二进制制品方式进行传递，并删除各种元数据

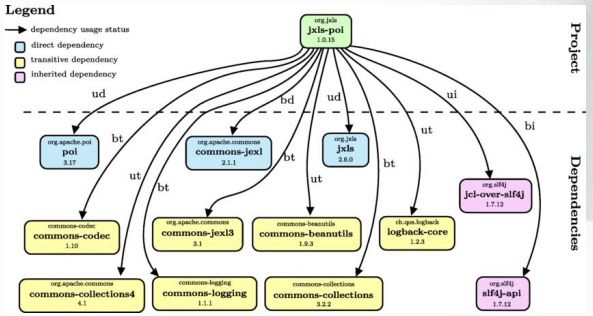
依赖关系复杂

包管理器依赖管理规则差别大
克隆方式的依赖缺乏基础数据

缺乏统一标准

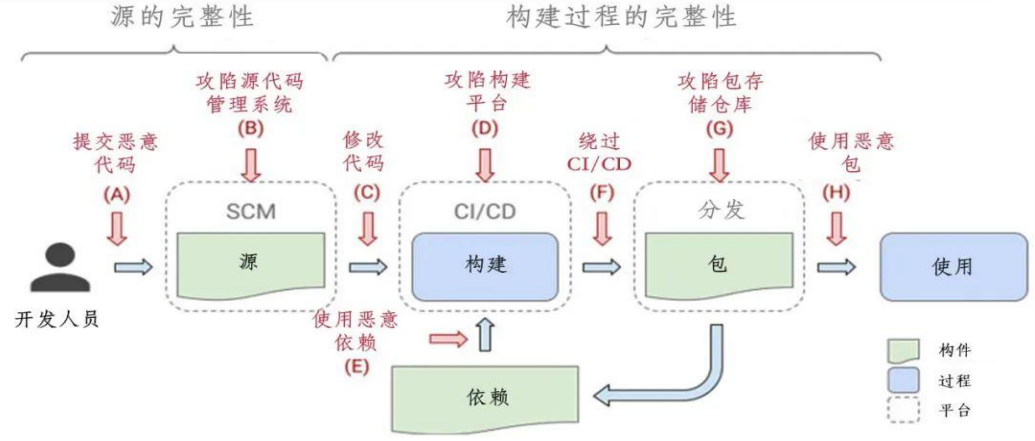
开源社区、软件企业、行业等多层面缺乏统一的软件供应链基础物料标准





软件供应链安全框架实践 – Google SLSA

谷歌SLSA（Supply Chain Level for Security Artifact）框架围绕来源、构建、包、依赖、使用5个环节识别8类威胁，定义4个SLSA安全级别，19项措施提供软件制品完整性保护。

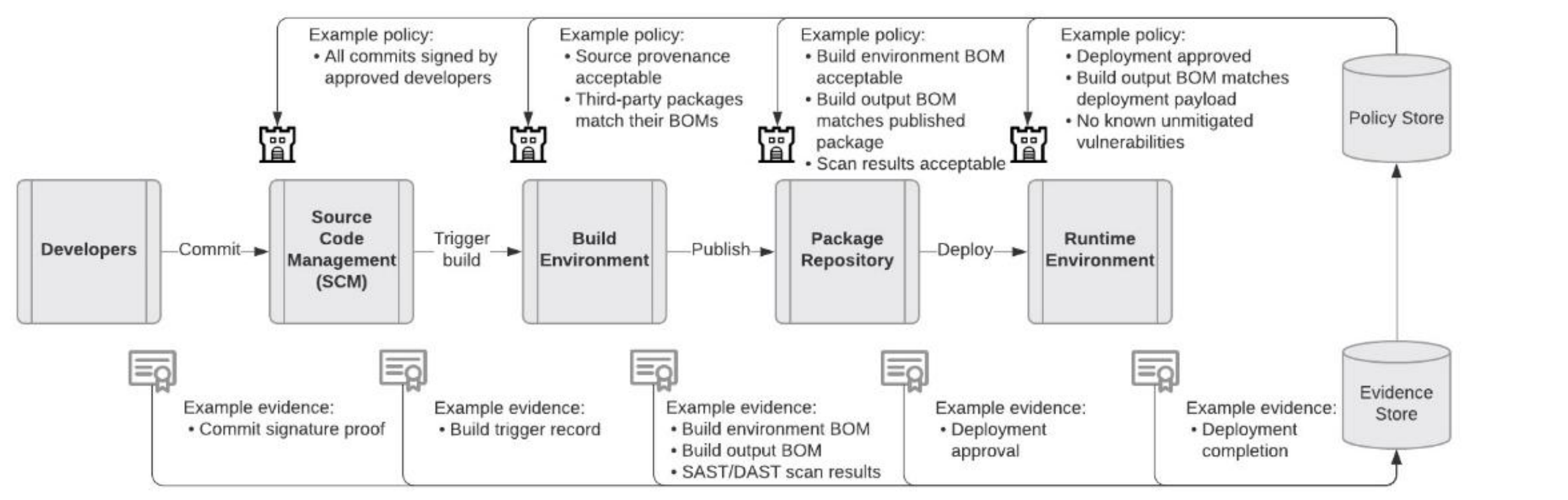


需求层级	需求细则	SLSA 1	SLSA 2	SLSA 3	SLSA 4	描述
源代码	版本控制		需要	需要	需要	对源代码的每一个更改都会在版本控制系统中进行跟踪。版本控制系统会识别谁进行了更改、更改是什么以及更改发生的时间。
	已验证的历史记录			需要	需要	历史上的每一次更改都至少有一个经过强身份验证的参与者身份（作者、上传者、审阅者等）和时间戳。
	无限期保留			18个月	需要	中间件及其更改历史被无限期保留，不能删除。
	双人审核				需要	至少有两个值得信赖的人（"可信人"是由拥有/生产软件的组织或开发人员定义的。消费者最终必须相信他们会做正确的事情。非单方面原则防止组织内的个人破坏组织的目标。）同意历史上的每一个变化。
构建	已编写脚本	需要	需要	需要	需要	所有的构建步骤都是在某种"构建脚本"中完全定义的。唯一的手动命令（如果有的话）是调用构建脚本
	生成服务		需要	需要	需要	所有的构建步骤都是使用一些构建服务运行的，比如持续集成（CI）平台，而不是在开发人员的工作站上。
	临时环境			需要	需要	生成步骤在临时环境（如容器或VM）中运行，这些环境仅为此生成设置，而不是从以前的生成中重用。
	环境隔离			需要	需要	构建步骤在独立的环境中运行，不受其他构建实例的影响，无论是先前的还是并发的。构建缓存（如果使用）是纯内容寻址的，以防止篡改。
	无参数				需要	生成输出不受生成入口点和顶级源位置以外的用户参数的影响。
	封闭性				需要	所有的构建步骤、源和依赖项都是用不可变的引用预先声明的。构建步骤在没有网络访问的情况下运行。所有依赖项都由构建服务控制平面获取并检查完整性。
	可复制				需要	使用相同的输入中间件重新运行构建步骤会导致逐位相同的输出(无法满足此要求的生成必须提供理由。)

需求层级	需求细则	SLSA 1	SLSA 2	SLSA 3	SLSA 4	描述
起源	可用的	需要	需要	需要	需要	出处可供中间件的使用者或验证策略的任何人使用，它至少标识中间件、执行构建的系统和顶级源。所有中间件引用都是不可变的，例如通过加密散列。
	需要认证		需要	需要	需要	出处的真实性和完整性可以通过数字签名等方式进行验证。
	服务生成		需要	需要	需要	出处是由构建服务本身生成的，而不是在服务之上运行的用户提供的工具。
	不可伪造性			需要	需要	生成服务的用户不能伪造出处。
	完全依赖				需要	出处记录了所有构建依赖项，这意味着构建脚本可以使用的每个中间件。这包括生成工作进程的计算机、VM或容器的初始状态。
常规特性	安全性				需要	该系统符合一些TBD基线安全标准，以防止泄露(修补、漏洞扫描、用户隔离、传输安全、安全引导、机器标识等（可能是NIST 800-53或其子集。）
	可访问性				需要	所有物理和远程访问都必须是罕见的、有日志记录的，并在多方批准后关闭。
	特权账户				需要	只有少数平台管理员可以覆盖此处列出的保证。这样做必须得到第二个平台管理员的批准。

	威胁	案例	SLSA 的消减措施
A	将恶意代码提交到代码仓	Linux 恶意提交: 研究人员通过邮件列表补丁的方式，故意将缺陷引入到Linux内核.	双人审核可以大部分缺陷，但并不能完全避免.
B	源码管理平台泄漏	PHP: 攻击者攻破了PHP的git托管服务器，并注入了两个恶意提交.	一个更安全的保护源码平台将使攻击者更难攻击.
C	篡改参与构建的源码	Webmin: 攻击者篡改了构建的基础源码.	一个符合SLSA的构建服务器会识别实际使用的源代码，从而允许使用者检测此类篡改.
D	构建平台泄漏	SolarWinds: 攻击者攻破构建平台，安装了一个植入程序在每次构建过程中注入恶意行为.	更高的SLSA级别需要对构建平台进行更强大的安全控制，使得被攻陷和得到控制更加困难.
E	使用错误的依赖关系（即会对A到H的依赖关系，进行递归查找）	event-stream: 攻击者添加了一个无害的依赖项，然后更新该依赖项以添加恶意行为。更新与提交给GitHub的代码不匹配（即攻击F）.	SLSA会递归地应用到所有依赖项来阻止这个特定的问题，依赖的起源能够指示出问题的依赖不是由正确的生成器构建的，或者源代码不是来自GitHub.
F	上传不是由CI/CD系统生成的中间件	CodeCov: 攻击者使用泄漏的凭据将恶意中间件上传到GCS存储桶，用户可以从中直接下载.	GCS bucket中中间件的出处会显示中间件不是以预期的方式从预期的源repo构建的.
G	篡改包托管到仓库	攻击包镜像: 研究人员为几个流行的包存储运行镜像，这些库被篡改后可能被用来提供恶意包服务.	与上面的（F）类似，恶意中间件的出处会显示它们不是按照预期构建的，或者不是从预期的源repo构建的.
H	欺骗开发者使用恶意包	Browserify typosquatting: 攻击者上传了一个与原始文件同名的恶意软件包.	SLSA不能直接解决这个威胁，但是源代码管理的源代码链接可以启用和增强其他解决方案.

软件供应链安全框架实践 – 微软SCIM

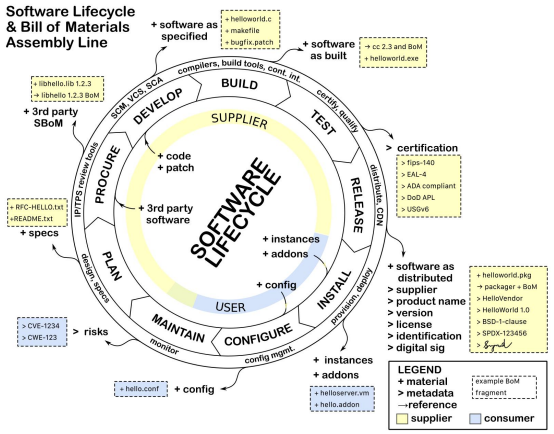


Supply Chain Integrity Model 供应链完整性模型

SBOM: 解决软件供应链风险的基石

SBOM定义

SBOM是一种正式标准化的、机器可读的元数据，唯一的标识软件组件及其内容，同时包括版权及许可证等成分数据。SBOM旨在**跨组织共享**，有助于提供软件供应链成分清单，提升软件透明度。



SBOM数据标准：SPDX、CycloneDX、SWID

SBOM标准	SPDX	SWID	Cyclone DX
组织	Linux Foundation	ISO&IEC	OWASP
标准化	ISO/IEC 5962 (UD)	ISO/IEC 19770-2	No Plan
当前版本	2.2.1	2	1.3
软件标识	Pakage SPDX ID, Hash	SWID	GAV, PURL, CPE, SWID, Hash
内容	Document Creation Information Package Information File Information Snippet Information Licensing Information Relationships Annotations	SWID Entity Payload	BOM Metadata Component Services Dependencies Compositions Vulnerability Signatures

SBOM标准化

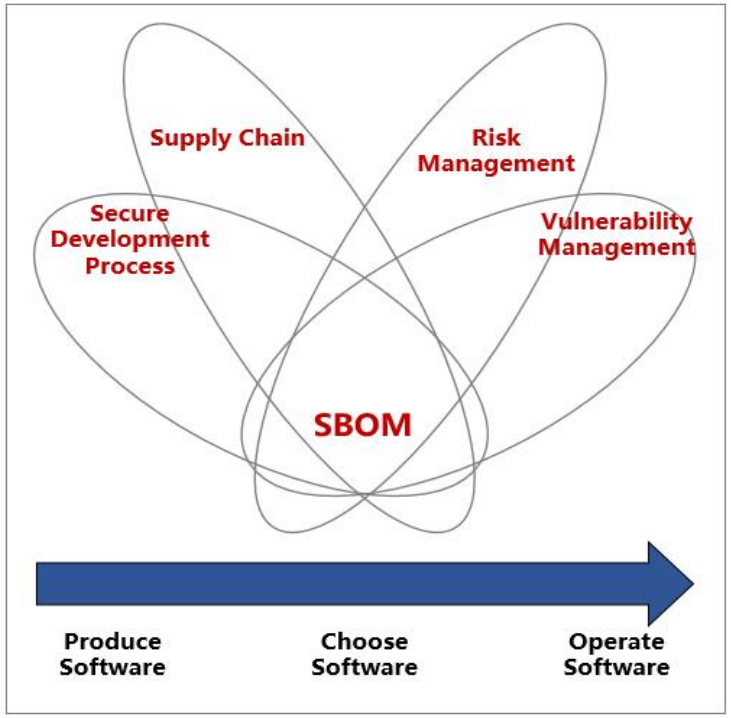
- 发起组织 NITA（美国国家电信与信息管理局），2018/6/6起，启动软件组件透明度计划
  - 目标：定义SBOM基础格式达成行业共识
  - 四个工具组：框架组、实践组、格式与工具组、医疗健康POC
- 主要协议组织
  - Linux Foundation
  - ISO&IEC
  - OWASP

行业实践

- 2021年9月欧盟发布网络弹性法案，要求软件制造商提供至少需包括产品的顶层依赖关系的软件材料清单SBOM；
- NITA于2021年6月征集意见，Google、微软等88家企业提供了积极反馈
- 美国2021年5月发布14028号行政命令，要求美国商务部会同NITA发布SBOM的“最小元素”
- Snyk、Sonartype、Fossology以及国内安全软件开始支持生成SBOM或利用SBOM作为漏洞分析的输入
- ENISA（欧洲网络及信息安全局）于2020年11月发布《Guideline for Securing IOT》，建议针对物联网设备使用SBOM

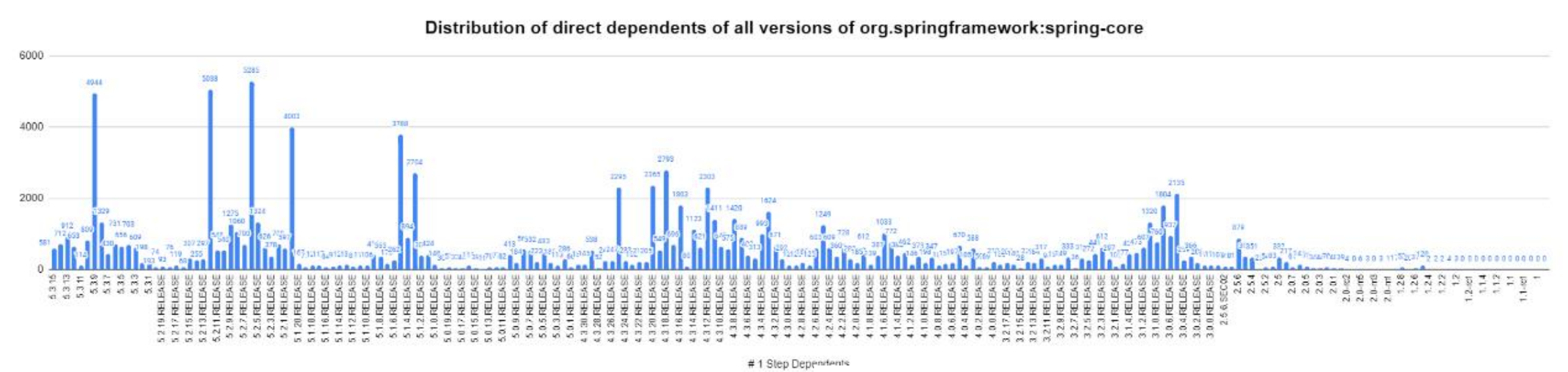
SBOM应用场景与用途

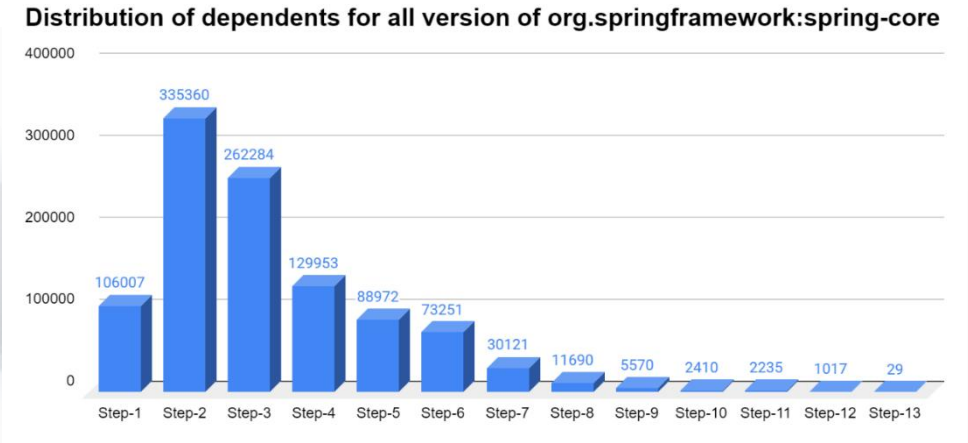
- **场景**：软件供应链安全管理、安全漏洞管理、安全应急响应，高可信安全应用管理
- **用途**：能帮助软件生产商、购买者和运营商更高效的识别软件成分、排查License风险/合规风险/安全漏洞影响风险、履行义务声明等

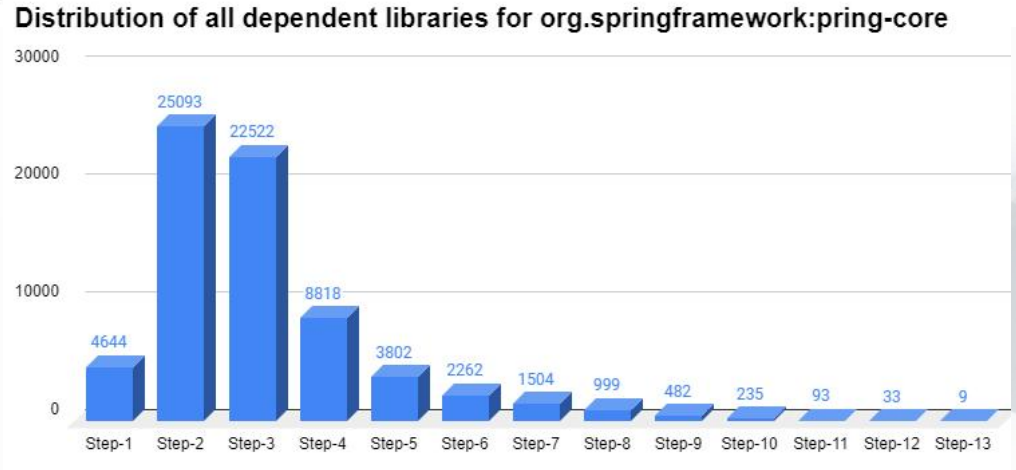


Benefits	Produce	Choose	Operate
Cost	Less unplanned, unscheduled work	A more accurate total cost of ownership	More efficient administration
Security Risk	Avoid known vulnerabilities	Easier due diligence	Faster identification and resolution. Know if and where specific software is affected.
License Risk	Quantify and manage licenses and associated risk	Easier due diligence	More efficient, accurate response to license claims
Compliance Risk	Easier risk evaluation. Identify compliance requirements earlier in lifecycle	More accurate due diligence, catch issues earlier in lifecycle	Streamlined process
High Assurance	Make assertions about artifacts, sources, and processes used	Make informed, attack-resistant choices about components	Validate claims under changing and adversarial conditions

SBOM的应用举例 – spring-core漏洞传播影响分析







104万个第三方包版本直接或间接地依赖了spring-core，占整个Maven生态中所有组件包的12.35%，绝大部分的这些下游组件主要分布在距离较短1-4级的依赖中

以SBOM为基础，建立开源软件依赖关系图谱数据库

统一数据标准

覆盖自研/开源/第三方软件

- 统一软件包标识
- 统一SBOM元素
- 统一多标准转换规则

生成SBOM及依赖关系数据库

组织级

- 构建精准成分及依赖关系解析能力（包管理器、非包管理器、源代码、二进制）

国家/行业级

- 基于开源数据生成宏观开源生态依赖关系图谱数据库

数据利用

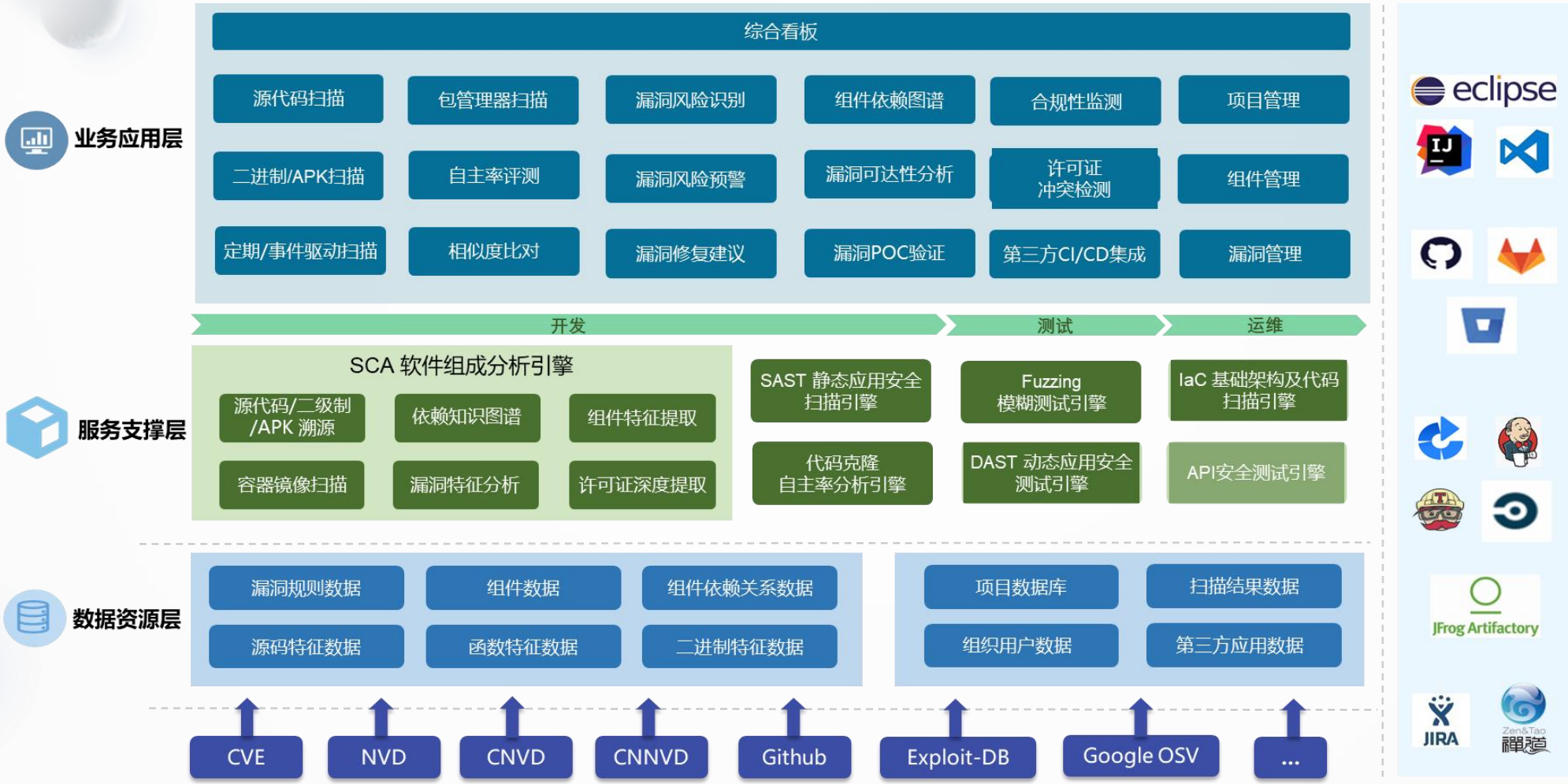
- 供应及脆弱点风险分析
- 安全传播影响分析及预警
- 成分/依赖关系查询
- 溯源审计
- 开源可信管理

组织级平台 组织级平台 组织级平台 组织级平台 组织级平台 组织级平台

行 业 级 平 台 行 业 级 平 台 行 业 级 平 台

国 家 级 平 台

思探明软件供应链安全综合检测平台V4.0 全新能力升级



GOTC

THANKS

全球开源技术峰会

THE GLOBAL OPENSOURCE TECHNOLOGY CONFERENCE