悬镜 XMIRROR | ISC | 中国电信 CHINA TELECOM 研究院

软件供应链安全治理与运营白皮书（2022）

法律声明

ABOUT

关于悬镜安全

悬镜安全，DevSecOps 敏捷安全领导者。起源于北京大学网络安全技术研究团队“XMIRROR”，创始人子芽。专注于以代码疫苗技术为内核，通过原创专利级 " 全流程软件供应链安全赋能平台 + 敏捷安全工具链”的第三代 DevSecOps 智适应威胁管理体系，持续帮助金融、车联网、泛互联网、能源等行业用户构筑起适应自身业务弹性发展、面向敏捷业务交付并引领未来架构演进的内生积极防御体系。

悬镜安全官网：https://www.xmirror.cn/

关于 ISC

ISC 是亚太地区乃至当今世界规格高、辐射广、影响力深远的全球性安全峰会。自 2013 年举办首届以来，九年间已围绕网络空间治理、数据安全、威胁情报等前沿领域安全问题，举办超 20 场国际峰会，设立超 300 场分论坛，输出超 2000 个行业前沿议题。吸引来自中国、美国、俄罗斯、以色列、德国等全球 30 多个国家的 2000 余位政要、行业领袖、网络安全专家深度参与，共话全球网络安全生态。已经成为专业性、权威性、全球性的中国网络安全产业名片。

ISC 官网：https://isc.360.com/

关于中国电信研究院

中国电信研究院是中国电信集团公司为适应集团发展和需要而组建的重要科研机构，伴随着通信技术的发展已传承六十余载，一直秉承着中国电信企业发展引擎、技术灯塔、决策智库的定位，承担通信行业创新产品的研发与落地、前瞻技术与应用的研究与攻关、企业运营中业务与技术方向的决策支撑、安全领域全链条的研发与推广的使命，研发创新涵盖 5G、云计算、人工智能、大数据、物联网、安全等多个领域。

参编机构

悬镜安全、ISC、中国电信研究院

导语

INTRODUCTION

数字化时代，软件定义万物，已逐渐成为支撑社会正常运转的最基本元素之一。随着软件开发过程中开源应用的使用越来越多，开源应用事实上逐渐成为了软件开发的核心基础设施，混源软件开发也已成为现代应用主要软件开发交付方式，开源应用的安全问题也已被上升到基础设施安全和国家安全的高度来对待。

软件供应链开源化，导致影响软件全供应链的各个环节都不可避免受到开源应用的影响。尤其是开源应用的安全性问题，将直接影响采用开源应用的相应软件供应链的安全。除了开源应用开发者因疏忽导致的开源应用安全缺陷，还可能存在具有非法目的开发者故意预留的开源应用安全缺陷，甚至还有恶意攻击者伪造的含有隐藏性恶意功能的异常行为代码被故意上传到上游开源代码托管平台，实施定向软件供应链攻击。上述开源应用中存在的众多安全问题，导致软件供应链的安全隐患大大增加，安全形式更加严峻。

而现代软件应用的供应链非常复杂，软件供应链安全管理是一个系统工程，亟需从国家、行业、机构、企业各个层面建立软件供应链安全风险的发现能力、分析能力、处置能力、防护能力，整体提升软件供应链安全管理的水平。为此，需要开展全方位的软件供应链安全检测防御方法和技术研究。第一，开展软件成分动态分析及开源应用缺陷智能检测技术研究，突破高效高准确性的开源应用安全缺陷动态检测技术的瓶颈，解决基于全代码遍历和代码片段级克隆技术比对的应用安全检测难题，进一步实现对全球开源应用的全面安全检测，从源头堵住软件供应链安全隐患的源头。第二，建立全球开源应用的传播态势感知和预警机制，攻克软件供应链中软件来源多态追踪技术，实现对供应链各环节中软件来源的溯源机制。通过软件来源多态追踪技术监控开源应用的使用传播和分布部署态势，全面把握有缺陷的开源应用传播和使用渠道，实现对全球开源应用及其安全缺陷的预测预警。第三，建立国家级／行业级软件供应链安全监测与管控平台，具备系统化、规模化的软件源代码缺陷和异常行为代码分析、软件漏洞分析、开源软件成分及风险分析等关键能力，为关键基础设施、重要信息系统用户提供日常的自查服务，及时发现和处置软件供应链安全风险。第四，严格管控软件供应链上游，尤其重点管控开源应用的使用，积极推动代码疫苗、SCA、区块链和 SBOM 等新技术和标准在软件供应链安全领域的推广和应用，从根本上提供软件供应链安全的可靠保障。

随着 AI 和自动化恶意攻击技术不断升级，专门针对软件供应链的攻击趋势明显加强，软件供应链已经成为网络空间攻防对抗的焦点，直接影响关键基础设施和数字经济安全，这也是为何中国第一届“DevSecOps 敏捷安全大会”（DSO2021）的主题被定为“安全从供应链开始”的主要原因，也是我们本次发布《软件供应链安全治理与运营白皮书（2022）》的主要驱动力。

此外，作为国内 DevSecOps 软件供应链安全的主要推动力量之一，从 2016 年初开始，我和悬镜创始团队就一直希望能有机会结合自身在这几年的前沿技术创新研究和行业应用实践沉淀，可以对软件供应链安全的治理与运营及 DevSecOps 敏捷安全体系的演进做一个系统性的梳理，并分享我们这些年在不同典型用户场景探索的落地实践经验。因此，沉淀了我们近 5 年创新技术研究与全球敏捷安全应用实践经验的《DevSecOps 敏捷安全》书籍应运而生。希望在这个新涌现新变化的前沿技术领域，我们悬镜团队和业界同行一起，凭借长期的技术积累和突破来推动中国自己的安全产业向新的未知空间做更深层次地探索，为产业搭建一个汇集“国家、行业、机构、企业“等综合力量且“同向、同心”的软件供应链安全保障生态体系变得愈发重要。

2022 年 7 月

前言

网络安全关乎着国家安全、企业安全，近年来一直备受重视。尤其是云原生、AI、物联网等技术的不断更新发展与应用，既推进了信息安全产业的快速发展，也衍生了更多的安全威胁。软件供应链安全作为网络安全的重要部分，近年来爆发的安全问题也越来越凸显。了解软件供应链安全的相关内容、探究安全治理与运营解决方案，是企业组织防患于未然的重要举措。

2021 年悬镜安全联合中国信通院发布了《软件供应链安全白皮书（2021）》，详细介绍了软件供应链安全现状、安全风险分析、安全治理办法及安全实践等内容，对软件供应链安全做了框架性的梳理。鉴于近年来软件攻击链安全事件高发，如何治理是众多企业亟需填补的理论洼地，对本模块加深认知才能占领实现软件供应链安全的制高点。基于此，悬镜安全与 ISC 及中国电信研究院联合发布了《软件供应链安全治理与运营白皮书（2022）》，重点讲述软件供应链安全治理体系和开源威胁治理，不断丰富软件供应链安全链条，实现整体安全。

本文的撰写依托于众多参考文献，融合悬镜安全、ISC 与中国电信研究院的专家经验。编撰过程难免有所疏漏，欢迎广大读者批评、指正，共同推进软件供应链安全的发展。

目录

CONTENTS

01 软件供应链安全发展背景

如今，敏捷开发模式下软件的开发以最快的速度将代码从 IDE 或 Git 存储库带到生产环境中，加快了部署速度，提升了业务系统上线的效率。但软件供应链的安全性对组织来说同样重要，因为任何一个安全漏洞都可能造成巨大的损失。然而，近年来攻击者利用软件供应链进行攻击的事件频发，攻击者通过一个安全漏洞就可以轻松访问上下游业务系统，这也是众多攻击者选择此类型攻击的原因。

Gartner 分析指出，“到 2025 年，全球 45% 组织的软件供应链将遭受攻击，比 2021 年增加了三倍。”可见，软件供应链的安全威胁将越来越严重。网络安全关乎着企业是否正常运转，是国民经济能否正常运行的重要前提，而软件供应链安全作为网络安全的重要组成部分，是实现网络安全的重要前提。

1.1 政策驱动下的软件供应链安全

信息技术时代，软件是企业应用程序的重要组成基础，软件无处不在且扮演着越来越重要的角色。一旦软件出现安全问题，将会影响到整个业务系统的正常运行。然而随着软件产业的不断发展，软件供应链的复杂度不断增加，对于信息系统的安全带来了众多安全威胁。不过软件供应链安全意识也在不断完善，国内外政府相继出台了一系列法令法规，也制定了一系列的相关标准，对软件供应链安全做了明示，以指导企业机构等能更好的实现软件供应链安全，规避网络风险的发生。

1.1.1 国内外政策法规

美欧等发达国家和地区在信息技术供应链安全管理领域起步较早，出台了大量政策法规和标准规范，用于加强软件供应链安全管理。我国近年来也在不断出台相关法律规范，逐步完善软件供应链安全管理工作。

1. 美国

表 1-1 美国针对软件供应链安全的规范及措施

序号	时间	主体	措施	内容
1	2022 年 2 月	美国国土安全部（DHS）	成立网络安全审查委员会（CSRB）	成立一个新机构 -- 网络安全审查委员会（CSRB），以调查重大网络安全事件。这个由 15 人组成的委员会将由来自国安局、FBI 和 CISA 等机构及包括国防部和司法部在内的政府部门的高级官员一以及来自 Google、微软和 Verizon 等公司的私营部门高管混合组成。 其成立后的第一项任务，就是针对 Log4j2 供应链漏洞进行调查

2	2022 年 1 月 11 日	美国网络安全和基础设施安全局（CISA）	主导成立的 ICT 供应链风险管理工作组制定了 2022 年的工作规划	工作组计划将软硬件物料清单以及加大对中小企业的影响力作为其供应链风险治理的重点
3	2021 年 5 月 12 日	拜登政府	签署了“关于改善国家网络安全（EO14028）”的行政命令	其中第 4 节“加强软件供应链安全”的（e）条款要求：初步指南发布 90 天内（不迟于 2022 年 2 月 6 日），NIST 应发布加强软件供应链安全实践的指南，并明确了指南需包含的 10 余项具体内容。 NIST 修订了其 2020 年 4 月发布的《安全软件开发框架（SSDF）V1.0》，形成 V1.1 草案，并于 2021 年 9 月 30 日至 11 月 5 日完成公开征求意见
4	2021 年 2 月	拜登政府	签署了第 14017 号《美国供应链行政令》	要求联邦机构对关键领域和行业的全球供应链进行审查，包括在行政令发布后的 100 天内针对包括半导体芯片的四个关键领域的供应链进行审查；在行政令发布后的一年之内对国防、医疗卫生、通信技术、能源、交通和食品生产六个行业进行供应链审查

5	2019 年 5 月	特朗普政府	签署第 13873 号《确保信息通信技术与服务供应链安全行政令》	要求商务部长、国务卿、国土安全部、国家情报总监分别根据自身职责开展持续性评估工作，每年均需发布相关评估报告
6	2018 年 12 月	美国国土安全部	正式组建 ICT 供应链风险管理特别任务组	该任务组主要职责时识别全球 ICT 供应链安全风险挑战，并提供可操作的解决方案
7	2008-2009 年	美国政府	将供应链安全上升至国家战略	早在 2008 年就开始制定相关政策法规，《国家网络安全综合计划（CNCI）》要求在产品、系统和服务的整个生命周期内综合应对国内和全球供应链风险；2009 年发布的《网络空间安全评估报告》，将信息通信技术（以下简称“ICT”）供应链安全纳入国家安全范畴。至此，美国率先完成了 ICT 供应链安全的框架结构设计，明确了其在国家安全中的重要地位

2. 欧盟

表 1-2 欧盟针对软件供应链安全的措施及规范

序号	时间	主体	措施	内容
1	2021 年 7 月	欧洲网络及信息安全局（ENISA）	发布了《ENISA 的供应链攻击威胁情景》	对供应链攻击进行了分类，并对从 2020 年 1 月到 7 月初的 24 起供应链攻击事件进行了研究，对供应链攻击者来源、攻击手段、攻击目标以及应对措施进行了分析
2	2015 年 8 月	欧洲网络与信息安全局（ENISA）	发布《供应链完整性：ICT 供应链风险和挑战概述，以及发展方向愿景》报告	指出 ICT 供应链完整性是国家经济发展的关键因素，提高供应链完整度对公共和私营部门意义重大，并建议供应链安全管理应遵循同一套实践，为评估和管理提供共同的基础，政府应与企业合作建立 ICT 供应链安全风险评估框架

3. 我国相关政策法规

表 1-3 我国针对软件供应链安全的规范及措施

序号	时间	主体	措施	内容
1	2021 年 11 月	国家互联网信息办公室	2021 年第 20 次室务会议审议通过《网络安全审查办法》	为了确保关键信息基础设施供应链安全，维护国家安全，对关键信息基础设施运营者采购网络产品和服务，影响或可能影响国家安全的，应进行网络安全审查
2	2021 年 10 月	人民银行办公厅、中央网信办秘书局、工业和信息化部办公厅、银保监会办公厅、证监会办公厅	联合发布《关于规范金融业开源技术应用与发展的意见》（以下简称《意见》）	《意见》要求金融机构在使用开源技术时，应遵循“安全可控、合规使用、问题导向、开放创新”等原则。《意见》鼓励金融机构将开源技术应用纳入自身信息化发展规划，加强对开源技术应用的组织管理和统筹协调，建立健全开源技术应用管理制度体系，制定合理的开源技术应用策略；鼓励金融机构提升自身对开源技术的评估能力、合规审查能力、应急处置能力、供应链管理能力等；鼓励金融机构积极参与开源生态建设，加强与产学研交流合作力度，加入开源社会组织等
3	2021 年 7 月 30 日	国务院	正式公布《关键信息基础设施安全保护条例》	第十九条明确指出：“运营者应当优先采购安全可信的网络产品和服务；采购网络产品和服务可能影响国家安全的，应当按照国家网络安全规定通过安全审查”

4	2021 年	国家标准化管理委员会	拟通过研究制定《信息安全技术 软件供应链安全要求》	提升国内软件供应链各个环节的规范性和安全保障能力
5	2020 年	中国电子技术标准化研究院	发布国家标准《信息技术产品供应链安全要求》征求意见稿	规定了信息技术产品供应方和需求方应满足的供应链基本安全要求
6	2020 年	电信终端产业协会	发布了《网络产品供应链安全要求》行业标准	对网络产品在管理制度、组织机构和人员、信息系统等以及供应链环节提出了不同等级的安全要求
7	2019 年 7 月	国家互联网信息办公室等四部门	发布《云计算服务安全评估办法》	要求云计算服务安全评估工作中，应重点评估“云平台技术、产品和服务供应链安全情况”。申请安全评估的云服务商应提交“业务连续性和供应链安全报告”
8	2019 年	中国信通院	推动制定《软件供应链安全管理能力成熟度模型》、《软件物料清单建设总体框架》	从软件供应链入口、自身、出口三个阶段多维度保障软件供应链安全，并落地评估测试
9	2016 年 11 月	第十二届全国人民代表大会常务委员会	第二十四次会议通过《中华人民共和国网络安全法》	第三十五条“关键信息基础设施的运营者采购网络产品和服务，可能影响国家安全的，应当通过国家网信部门会同国务院有关部门组织的国家安全审查”和第三

				十六条“关键信息基础设施的运营者采购网络产品和服务，应当按照规定与提供者签订安全保密协议，明确安全和保密义务与责任”，分别从网络安全审查、网络产品和服务安全角度对供应链安全提出要求

1.1.2 国内外标准

除了法令法规之外，国内外也制定了一系列的安全标准，对软件供应链安全提出了更细化的安全要求。

1.1.2.1 国外标准情况

表 1-4 国外标准情况

序号	标准	措施	内容
1	ISO/IEC JTC1/SC27	《信息技术 安全技术供应商关系的信息安全》系列标准	该系列标准包括四个部分，分别是： ISO/IEC27036-1：2021《第 1 部分：概述和相关概念》，提供了 27036 系列标准的总体介绍，对供应商关系的类型、相关安全风险以及风险管理相关概念进行了描述； ISO/IEC 27036-2：2014《第 2 部分：通用要求》，对供应关系中的信息安全进行定义，并对实施、操作、评估、应对措施等提出了通用要求； ISO/IEC 27036-3：2013《第 3 部分：供应链安全指南》，对 ICT 供应链中产品和服务的安全风险进行描述和分析，给出了应对相应风险的措施； ISO/IEC 27036-4：2016《第 4 部分：云服务安全指南》，提出了云计算服务在 ICT 供应链方面存在的安全风险及相关应对措施

2	ISO/IEC 20243	《信息技术 开放可信技术提供商标准 减少恶意和仿冒组件》系列标准	该系列标准包括 ISO/IEC 20243-1：2018《要求和建议》、ISO/IEC 20243-2：2018《O-TTPS 和 ISO/IEC 20243-1：2018 的评估流程》两个部分，针对信息通信技术硬件和软件在产品生命周期内面临的完整性威胁，特别是恶意和仿冒组件带来的安全风险，提供了一套应对准则、方针和建议
3	ISO 28000	《供应链安全管理体系规范》	说明了组织建立、实施供应链安全管理体系的要求，保障供应链安全的重要内容，为各类组织开展供应链安全管理提供了一个较为系统的管理模式
4	NIST SP 800-161	《联邦信息系统和组织的供应链风险管理实践》	针对为联邦信息系统和机构供应链方面面临的安全风险，提出的 ICT 供应链安全风险控制流程和措施。该标准通过提出将供应链安全风险纳入组织整体的风险管理过程，并给出 19 类 ICT 供应链安全控制措施，以减少联邦信息系统和组织面临的供应链安全风险

1.1.2.2 我国相关标准

表 1-5 国内标准情况

序号	标准	措施	内容
1	GB/T 36637—2018	《信息安全技术 ICT 供应链安全风险管理指南》	该系列标准包括四个部分，分别是： ISO/IEC27036-1：2021《第 1 部分：概述和相关概念》，提供了 27036 系列标准的总体介绍，对供应商关系的类型、相关安全风险以及风险管理相关概念进行了描述； ISO/IEC 27036-2：2014《第 2 部分：通用要求》，对供应关系中的信息安全进行定义，并对实施、操作、评估、应对措施等提出了通用要求；

			ISO/IEC 27036-3：2013《第 3 部分：供应链安全指南》，对 ICT 供应链中产品和服务的安全风险进行描述和分析，给出了应对相应风险的措施； ISO/IEC 27036-4：2016《第 4 部分：云服务安全指南》，提出了云计算服务在 ICT 供应链方面存在的安全风险及相关应对措施
2	GB/T 32921—2016	《信息安全技术 信息技术产品供应方行为安全准则》	该标准规定了信息技术产品供应方在提供信息技术产品过程中，为保护用户相关信息，维护用户信息安全应遵守的基本准则，分别从用户相关信息收集和处理的安全、远程控制用户产品的安全和其他行为安全等方面提出相关安全要求，适用于信息技术产品供应、运行或维护过程中的供应方行为管理
3	GB/T 32926—2016	《信息安全技术 政府部门信息技术服务外包信息安全管理规范》	该标准针对政府部门在使用信息技术服务外包时面临的外包服务机构背景复杂、服务人员流动性大、内部管理不规范等问题带来的信息安全风险，建立了政府部门信息技术服务外包信息安全管理模型，明确了服务外包信息安全管理角色和责任，将管理活动划分为规划准备、机构和人员选择、运行监督、改进完成四个阶段，提出具体信息安全管理要求，为政府部门信息技术服务外包的安全管理提供参考
4	GB/T 31168—2014	《信息安全技术 云计算服务安全能力要求》	该标准提出了云服务商应具备的技术能力，适用于对政府部门使用的云计算服务进行安全管理。标准从重要设备安全检测情况，重要信息系统、组件获服务的供应链保护措施情况，供应商情况等方面对云服务商的供应链安全提出要求
5	GB/T 24420—2009	《供应链风险管理指南》	该标准在参考国际航天质量标准、美国机动车工程师协会标准和欧洲航空航天工业协会

			标准《供应链风险管理指南》等标准的基础上制定。标准给出了供应链风险管理的通用指南，包括供应链风险管理的步骤，以及识别、分析、评价和应对供应链风险的方法和工具，适用于各类组织保护其在供应链上进行的产品的采购活动
6	-	《信息安全技术 关键信息基础设施信息技术产品供应链安全要求（报批稿）》	该标准提出了关键信息基础设施、政务信息系统信息技术产品供应链在设计、开发、采购、生产、交付和运维等环节的安全要求，主要从安全漏洞缺陷的防范、安全运营维护、供应商管理、供应来源多样性等方面提出安全要求，适用于关键信息基础设施、政务信息系统加强信息技术产品供应链安全
7	-	《信息安全技术 软件供应链安全（草案）》	该标准规定了软件产品和服务供应链所涉及相关要素的安全要求，包括软件供应链组织管理要求，以及开发、交付、使用等环节的安全要求

1.2 软件供应链安全的重要性

近年来，世界各地出现了越来越多的针对不同国家公共和私人机构的供应链攻击。在某些情况下，攻击是由国家支持的 APT 组织实施的，这些攻击具有区域和全球影响，诸如 SolarWinds 就是此类攻击。除此之外，利用漏洞、后门等不同形式的攻击针对企业的攻击也越来越多，在近年的软件供应链攻击事件中可见一斑。关于不同类型的攻击手段，在《软件供应链安全白皮书（2021）》中也有细数，可再次翻阅。

1. 针对国家层面的软件供应链攻击

软件供应链攻击是复杂的，当软件开发或者第三方代码 / 组件的引入缺乏透明度时，便缺乏了对恶意攻击的抵抗能力。此时，若攻击者通过恶意篡改对供应链展开攻击，一旦攻击成功便容易对“关键基础设施、政府机构”等造成安全威胁，如进行间谍行动窃取国家机密等，对一个国家的危害是巨大的。如 2017 年的 NotPetya 攻击，该攻击使银行、商业、公用事业和物流瘫痪，在全球造成数十亿美元的损失。2020 年全球著名的管理软件供应商 SolarWinds 遭遇国家级 APT 团伙高度复杂的供应链攻击，导致包括美国关键基础设施、军队、政府等在内的超过 18000 家客户全部受到影响，可任由攻击者操控。此类的攻击仍有很多，不一一详述。

2. 针对企业层面的软件供应链攻击

开源软件（Open Source Software, 简称 OSS）让软件开发模式发生转变，为节省开发时间，众多企业在应用系统开发中引入了开源软件，使得开源软件被广泛使用。然而，OSS 项目的指数增长增加了潜在的攻击面，并使审计代码成为更大的挑战。例如，软件开发和源代码管理平台 GitHub 托管的公共存储库的数量从 2009 年 2 月的 4.6 万个激增到 2020 年 1 月的 2800 万个。而且，软件供应链攻击是潜伏的，在软件生命周期的开发和分发阶段使用恶意软件是难以发现的。在某些情况下，攻击者在软件代码编译和签名之前就插入了恶意软件，将其嵌入到标准的安全签名之后，从而降低了被反病毒工具检测到的可能性。在其他情况下，攻击者通过软件发布和升级更新补丁注入恶意代码。从而对企业进行恶意攻击，造成严重危害。

不管是针对国家层面还是企业层面，软件供应链攻击都会带来不可弥补的危害，而且危害一旦发生都是不可逆的。因此，实现软件供应链安全至关重要。

02 软件供应链安全现状

DevOps 软件开发模式的出现，使得现代软件开发框架和方法更侧重于软件交付速度和可靠性的实现。但是，现代开发框架缺乏相关指导，无法帮助企业组织了解其软件的威胁、评估检测和应对威胁以及实施缓解措施。而且，企业组织在 DevOps 软件开发中更关注组织内的代码和进程，经常会忽略可能影响其应用完整性的外部因素，例如，开源软件包的攻击会影响直接或间接依赖于该软件包的任何代码。自 2020 年以来，此类软件供应链攻击急剧增加。除此之外，不同周期阶段发生的软件供应链攻击事件众多，如下重点简述自“棱镜计划”事件以来的典型事件，让大家对供应链攻击有更多了解。

2.1 软件供应链安全事件高发

网络攻防日趋常态化以来，安全威胁的种类也日渐增多。针对软件供应链攻击的事件越来越多，呈频发状态。近年来，出现了一系列典型的软件供应链攻击事件。其中如 SolarWinds Orion、Wind River VxWorks Urgent/11 安全漏洞、Equifax 信息泄露是其中较为典型的几个事件案例，以下本白皮书针对这几个事件做详细描述。

1. SolarWinds Orion 攻击

2020 年 12 月，全球多家网络安全公司发布报告，声称 SolarWinds 公司旗下 Orion 平台遭到黑客入侵。SolarWinds 遭遇的黑客攻击事件被命名为“Sunburst”，这是一次高度复杂的供应链攻击。也是一场全球性的黑客攻击，因为威胁攻击者将 Orion 软件变成了一种武器，可以访问全球多个政府系统和数千个私人系统。由于软件的性质以及扩展的 Sunburst 恶意软件可以访问整个网络，许多政府及企业网络和系统都面临着重大漏洞的风险。

虽然此攻击事件在 2020 年 12 月份被披露，而攻击者的整个攻击过程从 2019 年就已经开始了。

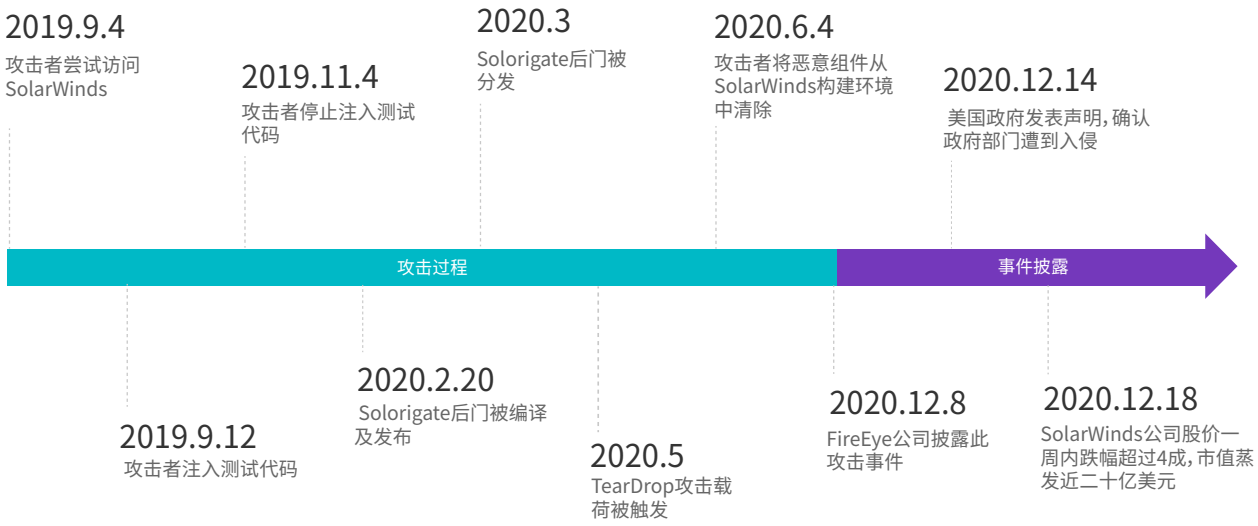

图 2-1 SolarWinds 攻击时间轴

攻击过程

根据权威报告数据显示，此次攻击可以分为三个阶段：

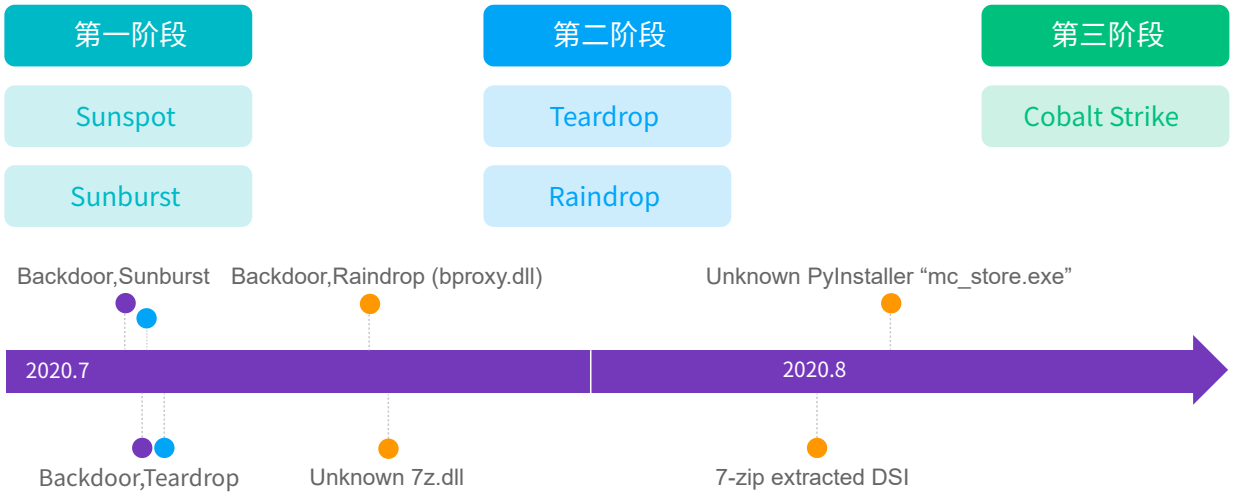

图 2-2 攻击阶段

第一阶段

Sunspot：攻击者使用 Sunspot 将 Sunburst 后门插入到 SolarWinds Orion IT 管理平台的软件版本中，用 Sunburst 代码替换源代码；

Sunburst：一旦安装了 Sunburst 后门，Sunspot 就会持续监视 Orion build 操作并劫持它，以通过 Sunburst 后门将其他恶意代码插入 Orion 库中。这样，攻击者就将恶意代码插入了 Orion 版本更新程序包中，分发给成千上万的用户。

第二阶段

Teardrop&Raindrop 的角色主要为 Cobalt Strike 的 loader。下表是对两个恶意软件的对比情况：

表 2-1 Teardrop 与 Raindrop 对比

	Teardrop	Raindrop
传播方式	由Sunburst分类	-
载荷格式	自定义PE	仅shellcode
载荷嵌入方式	二进制blob	隐写Steganography
加密方式	visualDecrypt combined with XOR	AES、XOR
压缩方式	None	LZMA
混淆方式	插入垃圾代码块	非功能代码延迟执行
导出名称	多变，有时与Tel/Tk projects一致	与Tcl/Tk projects一致

截止 2021 年 1 月 18 日，赛门铁克发现了 4 个 Raindrop 样本。前三个与 Teardrop 类似，Cobalt Strike 被配置为使用 HTTPS 作为通信协议；第四个使用 SMB 作为通信协议。

第三阶段

攻击特点：

该次攻击一个显著特点是，在第二阶段使用的恶意软件是为了引出第三阶段定制的 Cobalt Strike 工具。Cobalt Strike 是一套功能齐全的渗透测试工具，由 Raphael Mudge 于 2012 年创建，是第一批公开的“Red Team”C&C 框架之一，于 2020 年被 HelpSystems 正式收购，成为许多美国政府，大型企业和咨询组织的首选“红队”平台。

攻击影响：

·影响范围广，33000 余名 Orion 客户有一多半都安装了带后门的 Orion 软件，其中不乏美国重要政府部门及各知名企业；

·隐蔽性持久性好，攻击者使用 ATT&CK 上的持久术等技术使得攻击“隐身”，更不易被发现；

·引入了强大的“红队”平台，此渗透测试平台在以往的攻击中也有集成应用，极具代表性。

攻击危害：

受 SolarWinds 攻击影响的重要机构至少 200 家，波及北美、欧洲等全球重要科技发达地区的敏感机构，覆盖了美国、加拿大、日本、比利时、荷兰、澳大利亚等，多为发达国家，其中美国占比超过 60%。在行业分布上，包括国防科技、政府、医疗服务、教育、金融、食品等关键基础商业。

2. Wind River VxWorks Urgent/11 安全漏洞

2019 年，VxWorks 官方发布了安全漏洞公告，在所有的漏洞中有 6 个可导致远程代码执行（RCE）漏洞，其中 CVE-2019-12256、CVE-2019-12255、CVE-2019-12260 CVSS 评分为 9.8 分。其余 5 个漏洞可能导致拒绝服务，信息泄漏或归类为逻辑缺陷。这些漏洞存在于 VxWorks 的 TCP/IP 堆栈（IPnet）中，影响 VxWorks 7 (SR540 and SR610)、VxWorks 6.5-6.9 及使用 Interpeak 独立网络堆栈的 VxWorks 版本。攻击者可以利用其中漏洞实现无需用户交互及认证的远程攻击，最终完全控制相关设备。

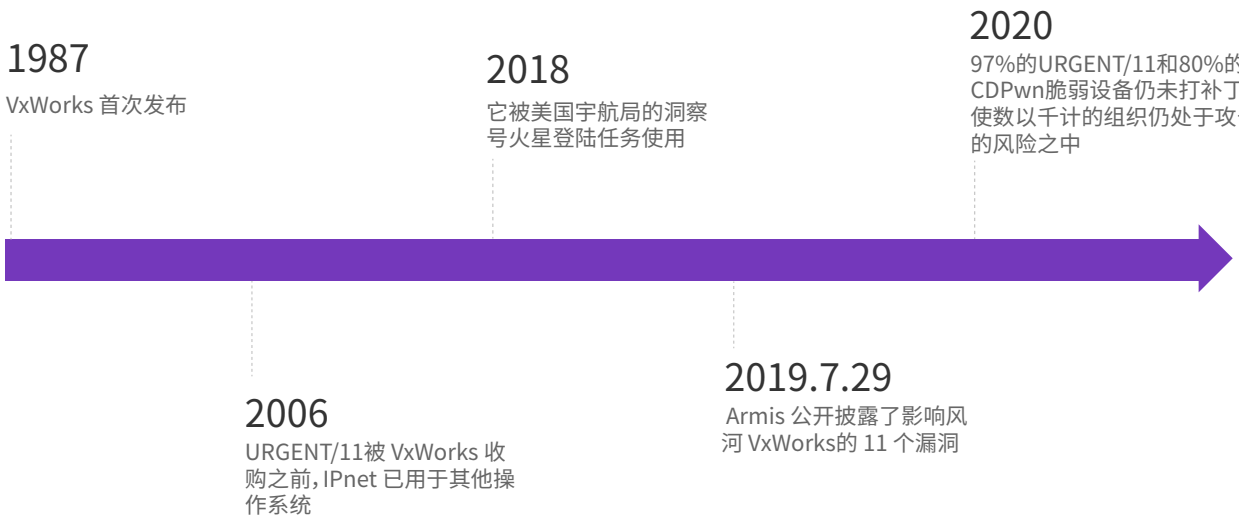

图 2-3 VxWorks 攻击时间轴

影响版本：

VxWorks RTOS（实时操作系统）是全球使用率非常高的嵌入式系统，根据官网的客户列表，从关键基础设施、网络设备、医疗设备、工业系统甚至航天器均在其中，据称被超过 20 亿台设备使用。可以说从 PLC 到 MRI 机器，到防火墙和打印机，再到飞机，火车等等都有广泛应用。它以其良好的可靠性和卓越的实时性被广泛地应用在通信、军事、航空、航天等高精尖技术及实时性要求极高的领域中，如卫星通讯、军事演习、弹道制导、飞机导航等。

具体受影响的版本包括：

CVE	Defect	Component	CVS Sv3	Title	Interpeak	pre-Vx6.5	Vx6.5	Vx6.6	Vx6.7	Vx6.8	Vx6.9.3	Vx6.9.4	Vx7 pre-SR620	Vx7 SR620
CVE-2019-12256	V7NET-2423	TCP/IP-stack	9.8	Stack overflow in the parsing of Ipv4 packets' IP options	N	N	N	N	N	N	Y	Y	Y	N
CVE-2019-12257	VXW6-87101	DHCP Client	8.8	Heap overflow in DHCP offer/ACK parsing inside ipdhcpc	N	N	Y	Y	Y	Y	Y	N	N	N
CVE-2019-12255	VXW6-87100	TCP/IP-stack	9.8	TCP Urgent Pointer=0 leads to integer underflow	Y	N	Y	Y	Y	Y	Y	N	N	N
CVE-2019-12260	V7NET-2425	TCP/IP-stack	9.8	TCP Urgent Pointer state confusion caused by malformed TCP AO option	N	N	N	N	N	N	N	Y	Y	N
CVE-2019-12261	V7NET-2425	TCP/IP-stack	8.8	TCP Urgent Pointer state confusion during connect() to a remote host	N	N	N	N	Y	Y	Y	Y	Y	N
CVE-2019-12263	V7NET-2425	TCP/IP-stack	8.1	TCP Urgent Pointer state confusion due to race condition	N	N	N	Y	Y	Y	Y	Y	Y	N
CVE-2019-12258	V7NET-2426	TCP/IP-stack	7.5	Dos of TCP connection via malformed TCP options	N	N	Y	Y	Y	Y	Y	Y	Y	N
CVE-2019-12259	V7NET-2426	TCP/IP-stack	6.3	DoS via NULL dereference in IGMP parsing	N	N	Y	Y	Y	Y	Y	Y	Y	N
CVE-2019-12262	V7NET-2427	TCP/IP-stack	7.1	Handling of unsolicited Reverse ARP replies(Logical Flaw)	Y	N	Y	Y	Y	Y	Y	Y	Y	N
CVE-2019-12264	V7NET-2428	TCP/IP-stack	7.1	Logical flaw in IPv4 assignment by the ipdhcpc DHCP client	Y	N	Y	Y	Y	Y	Y	Y	Y	N
CVE-2019-12265	V7NET-2428	TCP/IP-stack	5.4	IGMP Information leak via IGMPv3 specific membership report	N	N	Y	Y	Y	Y	Y	Y	Y	N

图 2-4 受影响的版本

URGENT/11 的风险：

URGENT/11 对当前使用的所有受影响的 VxWorks 连接设备构成了重大风险。根据设备在网络上的位置和攻击者的位置，存在三种攻击场景。

场景 1 – 攻击网络防御系统

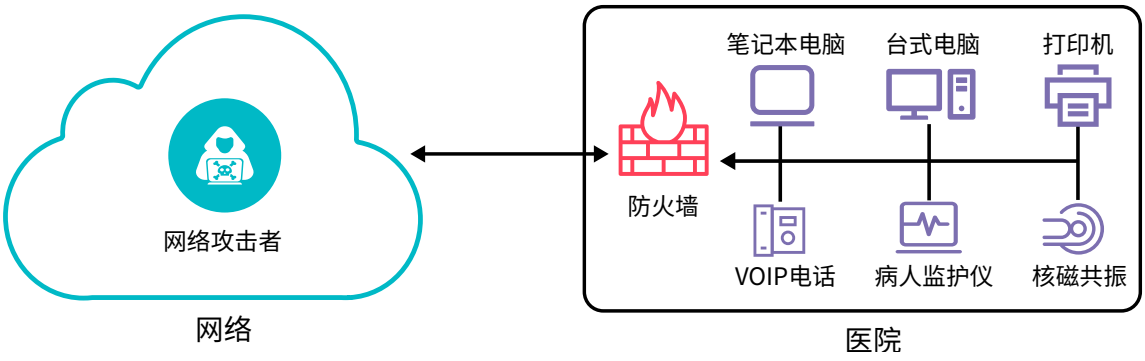

图 2-5 攻击网络防御系统

第一种攻击场景影响位于网络外围的 VxWorks 设备，例如防火墙。使用 URGENT/11 漏洞，攻击者可以对此类设备发起直接攻击，完全控制它们，然后控制它们所保护的网络。

场景 2 – 从网络外部绕过安全防线进行攻击

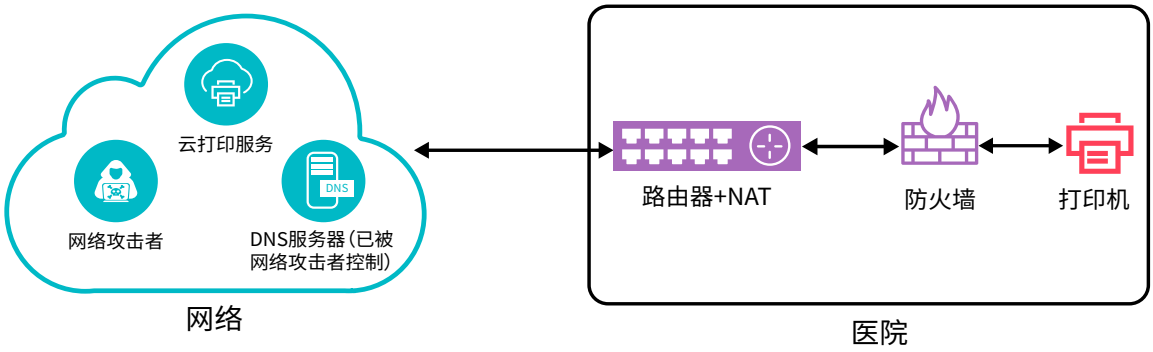

图 2-6 从网络外部绕过安全防线进行攻击

第二种攻击场景会影响任何具有外部网络连接的受影响 VxWorks 设备。URGENT/11 漏洞使攻击者能够接管此类设备，无论在网络外围实施任何防火墙或 NAT 解决方案以抵御攻击。

作为此场景的示例，请考虑对从安全网络中连接到云的 IoT 设备（例如 Xerox 打印机）的攻击。打印机不直接暴露在互联网上，因为它受到防火墙和 NAT 的保护，通过它连接到云应用程序。攻击者可以拦截打印机与云的 TCP 连接（不考虑 TLS）并触发打印机上的 URGENT/11 RCE 漏洞之一，从而完全控制它。为了拦截 TCP 连接，攻击者可以使用诸如 DNSpionage 使用的技术恶意软件，成为组织 Internet 流量的中间人。一旦攻击者接管了网络中的一个设备，他就可以横向扩散控制其中的其它 VxWorks 设备，如下一个攻击场景中所述。

场景 3 – 从网络内部进行攻击

在这种情况下，由于先前的攻击（例如上述情况），攻击者已经位于网络中，可以发送能够完全控制设备的目标 VxWorks 设备数据包，而无需用户交互。此外，攻击者不需要任何有关目标设备的先验信息，因为 URGENT/11 允许他通过在整个网络中广播他的恶意数据包来一次破坏所有易受攻击的设备。

作为这种攻击的一个例子，考虑一个只有内部网络连接的关键设备：医院的病人监护仪。即使它没有连接到 Internet，但通过渗透网络，攻击者仍然可以接管它。虽然人们可能认为将设备隐藏在安全网络中就足够了，但攻击者总有办法进入，正如上面的攻击场景所展示的那样，其中详细说明了攻击者如何使用 URGENT/11 渗透网络。

另一个例子可以在可编程逻辑控制器 (PLC) 中找到，它分布在工厂中。由于它们在受影响的 VxWorks 上运行，使用 URGENT/11 的攻击者可以在网络中广播一次攻击并有效控制整个工厂，无需任何侦察工作，将其关闭以获取赎金或任何其他恶意目的。

影响范围：

全球 VxWorks 操作系统分布情况如下：

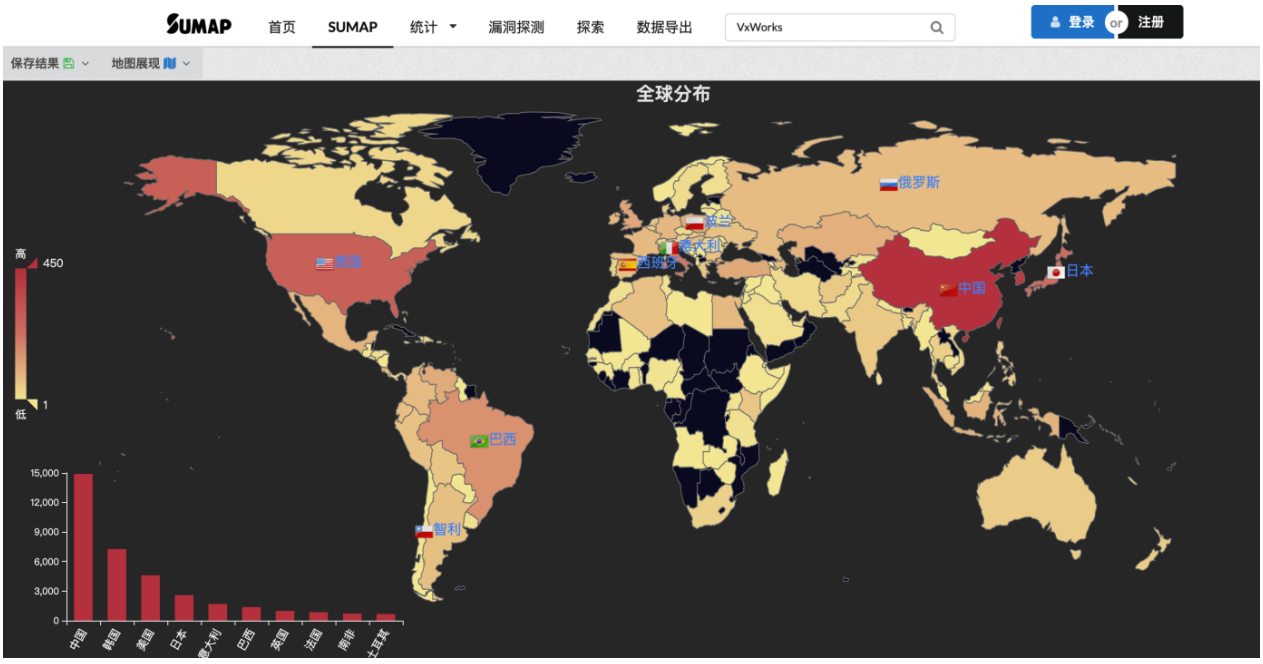

图 2-7 全球 VxWorks 操作系统最新分布情况

全国 VxWorks 操作系统分布情况如下：

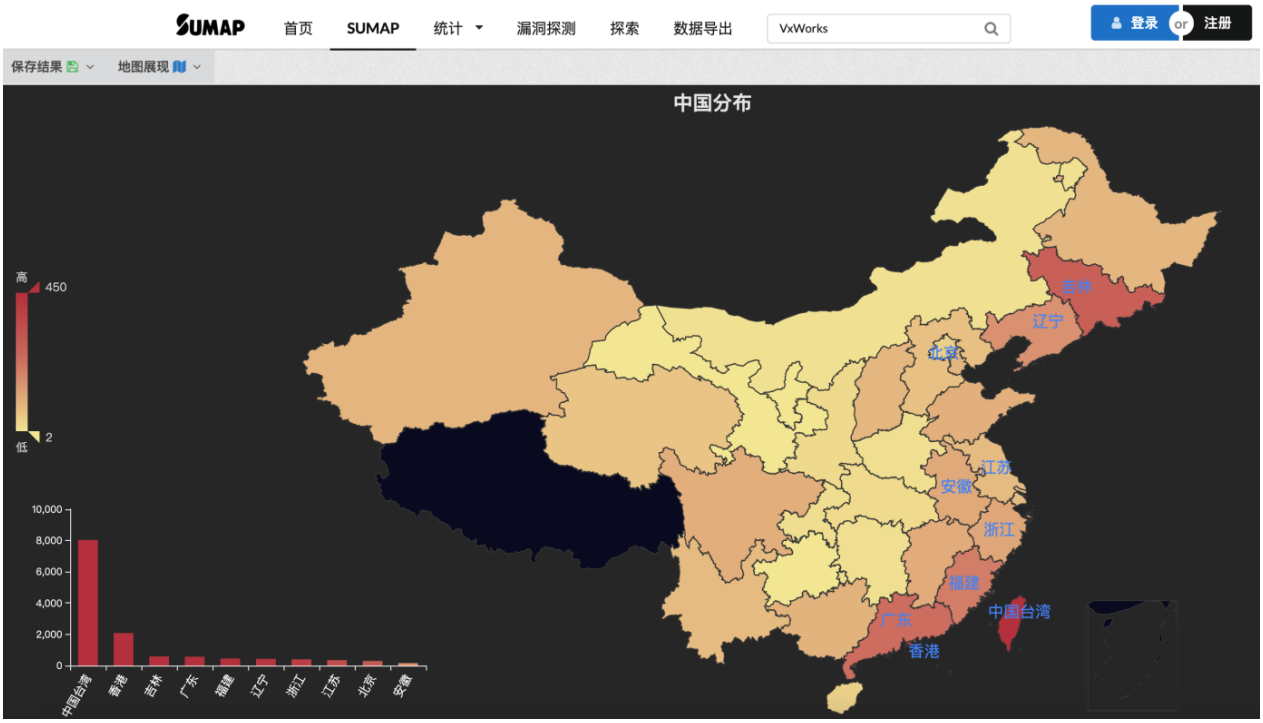

图 2-8 全国 VxWorks 操作系统最新分布情况

3. Equifax 信息泄露事件

2017 年，黑客利用 Equifax 系统中未修复的 Apache Struts 漏洞（CVE-2017-5638）发起攻击，导致了系统中大规模数据泄露，影响极其恶劣。当时这个漏洞的评分为最高分 10 分，Apache 随后发布 Struts 2.3.32 和 2.5.10.1 版本，进行修复。但 Equifax 在漏洞出现的两个月内都没有修复，导致 5 月份黑客利用这个漏洞进行攻击，泄露其敏感数据。

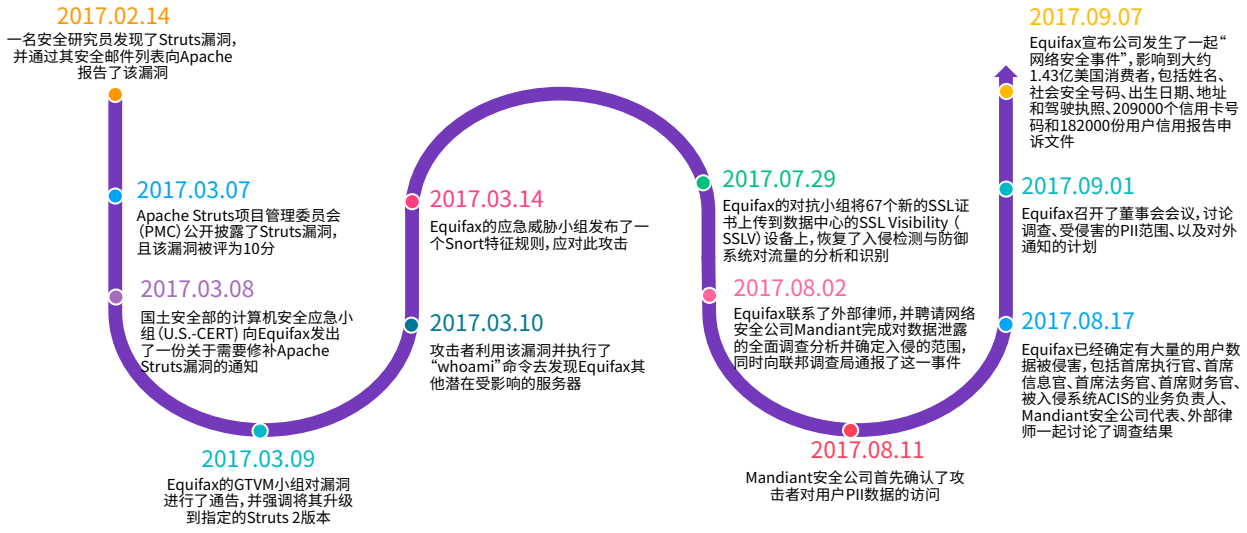

图 2-9 Equifax 信息泄露事件时间轴

攻击过程：

入侵者利用 Struts 漏洞成功进入了 Equifax 的内部网络，攻击的入口是 ACIS 系统（Equifax 为消费者提供用来对信用报告中的不准确信息进行申诉的一个门户网站）。在入侵 ACIS 系统后，攻击者上传了第一个 Webshell 后门程序，用于远程控制。（Webshell 是一种后门程序，攻击者通过它可以随时重新进入这台服务器，可以使用文件系统、进行数据库操作，方便执行系统命令，并提供文件上载 / 下载功能；在后续的攻击过程中，攻击者大约上传了 30 个不同种类的 Webshell）。

随后，攻击者通过挂载 NFS（网络文件系统）共享，由于 Equifax 未对存储中的文件进行访问控制，攻击者获取到了敏感的配置文件，包括未加密的应用程序使用的连接数据库的用户名和密码。攻击者成功使用这些凭证访问了与 ACIS 系统功能无关的 48 个数据库。

攻击者在这些数据库上运行了大约 9000 个查询，这些查询包括对数据库元数据的查询，以发现表中包含的信息类型；以及一旦找到一个带有 PII 信息的表，需要执行额外的查询，用以从表中获取出想要的敏感数据。总共，9000 次查询里的 256 次查询，返回了包含 PII 的数据集，并且所有这些 PII 信息都没有在数据库中被加密。

攻击者将 265 次成功查询的 PII 数据输出存储在文件中，压缩并放置在一个可访问的 Web 目录中。在远程通过 Wget 工具下载了这些文件，成功将数据从 Equifax 窃取。整个攻击过程，攻击者使用了大约 35 个不同的 IP 地址。

攻击持续了 76 天，然后才被 Equifax 的员工发现。原因是 Equifax 有安全设备可以监控到网络里的异常流量，包括它们之前更新了规则的入侵检测与防御系统。但是这些安全设备的前面还有一个 SSL 解密设备 SSL Visibility （SSLV），用于将加密的流量解密后发给检测和防护系统进行分析识别。但是，在事件发生时，SSL 解密设备上的证书已经过期了，根据设备的配置，证书过期不会影响流量的正常传输，但是却会影响入侵检测与防御系统，因为它们无法分析加密的流量。其中 ACIS 的域名 ai.equifax.com 的证书在 2016 年 1 月 31 号就过期了，也就是说在 19 个月之前，入侵检测与防御系统就已经无法检查 ACIS 的流量了。

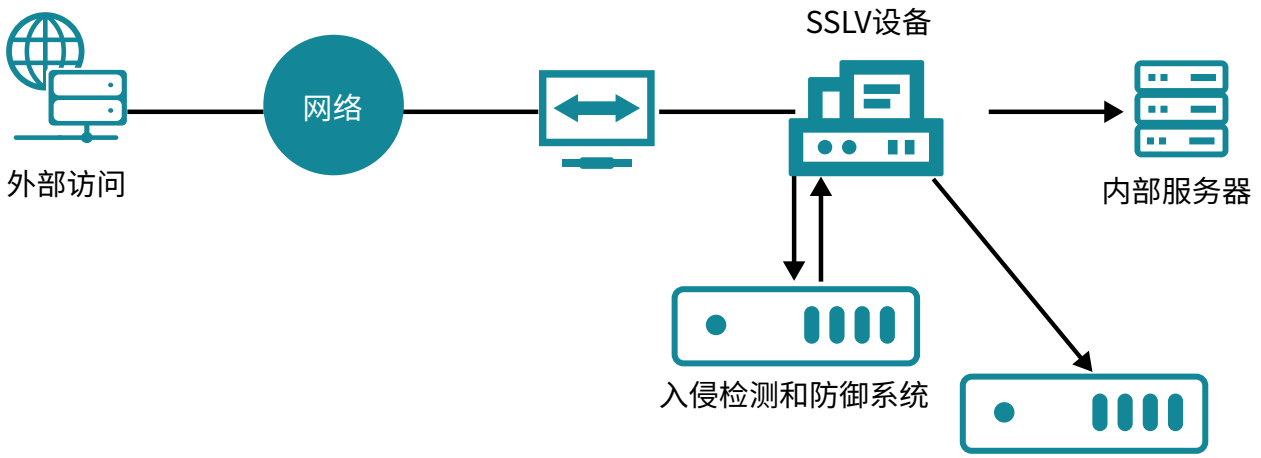

图 2-10 外部访问、SSLV 设备、IDS 设备、以及内部服务器示意图

攻击影响：

Equifax 漏洞攻击事件影响了大约 1.48 亿美国消费者，包括姓名、社会安全号码、出生日期、地址和驾驶执照、209000 个信用卡号码和 182000 份用户信用报告申诉文件。

攻击危害：

Equifax 在泄露事件之后增加了 IT 和网络安全支出，2017 年 11 月，公司临时首席执行官 Paulino do Rego Barros 表示，Equifax 公司自发现泄露事件以来增加了四倍的安全支出。Equifax 报告称 2018 年前 9 个月与这起安全事故有关的费用为 2.215 亿美元，包括技术和数据安全、法律和调查费用、产品赔偿等。

攻除上述三个典型软件供应链安全典型事件之外，还有更多的事件，具体如下 (见下页)：

图 2-11 近年来软件供应链攻击典型事件

下为部分典型事件简述：

2021 年 12 月，Apache Log4j2 远程代码执行漏洞（CVE-2021-44832）。在某些特殊场景下（如系统采用动态加载远程配置文件的场景等），有权修改日志配置文件的攻击者可以构建恶意配置，通过 JDBCAppender 引用 JNDI URI 数据源触发 JNDI 注入，成功利用此漏洞可以实现远程代码执行。攻击者仅需一段代码就可远程控制受害者服务器，危害巨大。

2021 年 9 月，微软 Azure 容器服务跨账户接管漏洞事件。因使用五年前发布的 RunC v1.0.0-rc2，微软 Azure 容器服务爆出跨账户接管漏洞，攻击者可攻陷托管 ACI 的多租户 Kubernetes 集群，接管平台上其他客户的容器，在其中执行代码并访问部署在平台上的数据，该漏洞已经被修复。

2021 年 8 月，Realtek WIFIi 模块 SDK 漏洞事件。台湾芯片设计厂商 Realtek 称，其 WiFi 模块的三款开发包 (SDK) 中存在 4 个严重漏洞，攻击者可利用这些漏洞攻陷目标设备并以最高权限执行任意代码。SDK 用于至少 65 家厂商制造的近 200 款物联网设备中。

2021 年 7 月，REvail 勒索软件攻击事件。攻击者获得 Kaseya 公司后端设施访问权限，在运行于客户现场的安全事件响应工具 VSA 服务器上部署 REvil 勒索软件。通过 VSA 服务器将勒索软件安装到联网工作站，从而感染其它第三方企业网络。攻击发生前，互联网上处于联网状态的 VSA 服务器超过 2200 台。

2021 年 6 月，Linux 漏洞事件。研究人员披露了影响 Linux 平台基于 Pling 的免费和开源软件 (FOSS) 市场的漏洞，攻击者可利用该漏洞进行供应链攻击 XSS 蠕虫并实现远程代码执行 (RCE)。

2021 年 5 月，Visual Studio Code。在流行的 Visual Studio Code 扩展中发现严重安全缺陷。可使攻击者危及本地机器，或通过开发人员 IDE 构建和部署系统，这些扩展的下载量超 200 万。

2021 年 4 月，Codecov 的 bash uploader 脚本被攻击者修改事件。知名代码测试公司 Codecov 宣布其产品的 bash uploader 脚本被攻击者修改，用户在使用 Codecov 产品时，会向攻击者的服务器发送敏感信息，可造成软件源代码等机密信息的泄露。

2021 年 3 月，git.php.ne 攻击事件。攻击者向 git.php.net 服务器上的 php-src 存储库推送了两次恶意提交，在 PHP 代码中植入了一个后门，攻击者可通过后门获得运行 PHP 的网站系统的远程代码执行权限。

2021 年 2 月，一名安全研究人员使用一种新颖的攻击技术攻破了微软、苹果、优步和特斯拉等公司的系统。Alex Birsan 利用依赖 / 名称空间混淆，在不需要社会工程的情况下，成功地向下游数十个引人注目的目标发送了伪造 (但无害) 包。

2021 年 1 月，Mimecast 供应商被攻击事件。云安全公司 Mimecast 报告称，攻击者破坏了供应商用于认证其在 Microsoft 365 Exchange Web 服务上服务的证书。大约 10% 的 Mimecast 用户使用依赖于被泄露证书的应用程序，但 Mimecast 表示只有少数人受到影响。

2020 年，SolarWinds。迄今为止影响最深远的供应链攻击源自一个后门 SUNBURST，该后门被注入到 Orion IT 管理应用程序的更新工具中。在提交给美国证券交易委员会的文件中，SolarWinds 称已有 18,000 名客户下载了该后门程序。该攻击导致包括美国关键基础设施、军队、政府等在内的超过 18000 家客户全部受到影响，可任由攻击者操控。

2018 年，event-stream 攻击。这是针对 GitHub 库的一次攻击，恶意软件被注入到 flatmap-stream 依赖中，这是 event-stream 的一部分。将受损害的依赖关系引入代码的应用程序数量仍然未知。

2018 年 6 月，华硕电脑攻击事件。一场名为 shadowwhammer 的攻击瞄准了华硕电脑的用户。赛门铁克 (Symantec) 的研究人员认为，这次通过华硕自动更新功能发送恶意软件的攻击从 6 月持续到 10 月，影响了多达 50 万个系统 , 其他供应商可能也受到了影响。

2018 年 5 月，ZipperDown 漏洞。攻击者可以通过这个漏洞来破坏手机数据，也可以获取任意代码执行能力。

2018 年 4 月，“寄生推”SDK 病毒。通过预留的“后门”云控开启恶意功能，进行恶意广告行为和应用推广，以实现牟取灰色收益。

2017 年 12 月，Android 平台爆出“核弹级”Janus 漏洞。能在不影响应用签名的情况下，修改应用代码，导致应用的升级安装可能被恶意篡改。同样，随着越来越多的应用采用热补丁的方式更新应用代码，恶意开发者也趁虚而入，在应用更新方式上做手脚，下发恶意代码，威胁用户安全。

2017 年 12 月，友盟 SDK 未导出组件暴露漏洞。攻击者利用该漏洞可以实现对使用了友盟 SDK 的 APP 的任意组件的恶意调用、任意虚假消息的通知、远程代码执行等攻击。

2017 年 11 月，Pujia8 破解网站携带木马。某游戏破解网站上多款游戏应用被植入了 Root 模块，在运行时，利用 CVE-2015-1805 等内核漏洞强行 Root 用户设备，并将无图标恶意应用植入到设备系统目录，长期潜伏用户设备进行恶意广告和流氓推广行为。

2017 年 8 月，Arris 为 AT&T 家庭用户定制版调制解调器内置后门事件。安全研究人员发现知名电信设备制造商 Arris 生产的调制解调器存在 5 个安全漏洞，其中有 3 个是硬编码后门账号漏洞。攻击者利用三个后门账号均可控制设备，提升至 Root 权限、安装新固件，乃至于架设僵尸网络等。

2017 年 8 月，WireX Android Botnet 污染事件。WireX BotNet 僵尸网络通过伪装普通安卓应用的方式大量感染安卓设备并发动了较大规模的 DDoS 攻击，其引发的 DDoS 事件源自至少 7 万个独立 IP 地址，据数据显示，来自 100 多个国家的设备感染了 WireX BotNet。

2017 年 8 月，“隐魂”木马。感染 MBR（磁盘主引导记录）的“隐魂”木马捆绑在大量色情播放器的安装包中诱导下载安装，安装包安装后调用加载读取释放出来的 JPG 图片并解密图片后的 shellcode 代码并执行。“隐魂”木马入侵系统后劫持浏览器主页，并安插后门实现远程控制。短短两周内，“隐魂”木马的攻击量已达上

百万次。

2017 年 8 月，恶性病毒“Kuzzle”。该病毒感染电脑后会劫持浏览器首页牟利，同时接受病毒作者的远程指令进行其他破坏活动。

2017 年 7 月，远程终端管理工具 Xshell 后门事件。2017 年 7 月 18 日发布的软件被发现有恶意后门代码，该恶意的后门代码存在于有合法签名的 nssock2.dll 模块中。2017 年 8 月 7 日，厂商 NetSarang 发布了一个更新通告，声称在卡巴斯基的配合下发现并解决了一个在 7 月 18 日的发布版本的安全问题，提醒用户升级软件。

2017 年 7 月，异鬼 II Bootkit 木马。该木马通过高速下载器传播。隐藏在正规软件甜椒刷机中，带有官方数字签名，导致大量安全厂商直接放行。该木马通过国内几大知名下载站的高速下载器推广，影响百万台机器。

2017 年 7 月，基于域名 bjftzt.cdn.powercdn.com 软件升级劫持攻击。这次事件其实是基于域名 bjftzt.cdn.powercdn.com 的一组大规模软件升级劫持事件。用户尝试升级若干知名软件客户端时，运营商将 HTTP 请求重定向至恶意软件并执行。恶意软件会在表面上正常安装知名软件客户端的同时，另外在后台偷偷下载安装推广其他软件。

2017 年 6 月，“灵隐”木马。该木马主要针对游戏外挂使用者，通过打包修改各种外挂，捆绑木马程序，再通过网盘和各类游戏论坛传播。执行劫持浏览器、删除安全软件、进行软件推广功能。

2017 年 6 月，NotPetya 勒索病毒。病毒攻击的根源是劫持了乌克兰专用会计软件 me-doc 的升级程序，使用户更新软件时感染病毒。政府、银行、电力系统、通讯系统等都不同程度地受到了影响。

2017 年 5 月，惠普笔记本音频驱动内置键盘记录后门事件。来自瑞士安全公司 Modzero 的研究人员在检查 Windows Active Domain 的基础设施时发现惠普音频驱动中存在一个内置键盘记录器监控用户的所有按键输入。研究人员指出，惠普的缺陷代码（CVE-2017-8360）不但会抓取特殊键，而且还会记录每次按键并将其存储在人类可读取的文件中。这个记录文件位于公用文件夹 C:\Users\Public\MicTray.log 中，包含很多敏感信息如用户登录数据和密码，其他用户或第三方应用程序都可访问。

2017 年 1 月，某加固服务被爆出夹带广告。该加固服务在加固应用时，会在开发者不知情的情况下植入代码用于拉取广告、下载拉活其他应用、程序异常上报、获取应用程序信息等行为。

2016 年 11 月，广升被曝向 andriod 设备预装后门。通过此后门会将用户的大量私人信息发送到提供固件的中国公司服务器上，发送的数据包括了手机号码、位置数据、短信内容、呼叫信息、安装和使用的应用等等。

2016 年 11 月，OTA 厂商锐嘉科在 Android 设备中植入 Rootkit。安装该恶意软件的设备可被黑客进行中间人攻击，并且以 Root 权限执行任意代码以此来获得对 Android 设备的绝对控制权，其主要原因是因为设备在 OTA 更新的时候没有采取严格的加密措施导致的。

2016 年 7 月，Toxik 病毒。该病毒将自身伪装成流行软件（游戏修改器、系统周边工具等）在下载站中进行传播，在用户运行后，该病毒会利用国内某知名互联网公司的软件升级程序（WPS 升级程序）下载推广软件，甚至下载病毒驱动进行恶意推广。

2015 年 12 月，Juniper VPN 后门事件。著名网络设备厂商 Juniper 公司发出风险声明，其防火墙、VPN 设备使用的操作系统具有重大安全风险，建议尽快升级相关版本。根据声明，设备的 SSH 登录系统在输入任意用户名的情况下，使用超级密码就可以最高权限登录系统，设备的 VPN 安全通道上传递的数据可以被攻击人解密、篡改和注入。全球上万 NetScreen 设备被攻击。

2015 年 11 月，百度 SDK WormHole 事件。百度 moplus SDK 的一个被称为虫洞（Wormhole）的漏洞被漏洞报告平台 wooyun.org 所披露，研究人员发现 Moplus SDK 具有后门功能，攻击者利用其可以获取访问权限控制权。

2015 年 9 月，多媒体框架 FFmpeg 漏洞。通过该漏洞可在播放漏洞视频或在转码过程中触发本地文件，读取获得指定文件。

2015 年 9 月，Xcode 非官方版本恶意代码污染事件。攻击者通过向非官方版本的 Xcode 注入病毒 Xcode Ghost，它的初始传播途径主要是通过非官方下载的 Xcode 传播，通过 CoreService 库文件进行感染。当应用开发者使用带毒的 Xcode 工作时，编译出的 App 都将被注入病毒代码，从而产生众多携带病毒的 APP。至少 692 种 APP 受污染，过亿用户受影响，受影响的包括了微信、滴滴、网易云音乐等著名应用。

2015 年 2 月，“方程式”组织（Equation Group）拥有一套用于植入恶意代码的超级信息武器库（在卡巴的报告中披露了其中 6 个），其中包括两个可以对数十种常见品牌的硬盘固件重编程的恶意模块，这可能是该组织掌握的最具特色的攻击武器，同时也是首个已知的能够感染硬盘固件的恶意代码。

2.2. 软件供应链风险典型特征

通过悬镜安全实验室对近 5 年的 43 次软件供应链安全事件做的系统性整理、分析和研究，得出以下软件供应链风险的 8 项典型特征，如图 2-12 所示：

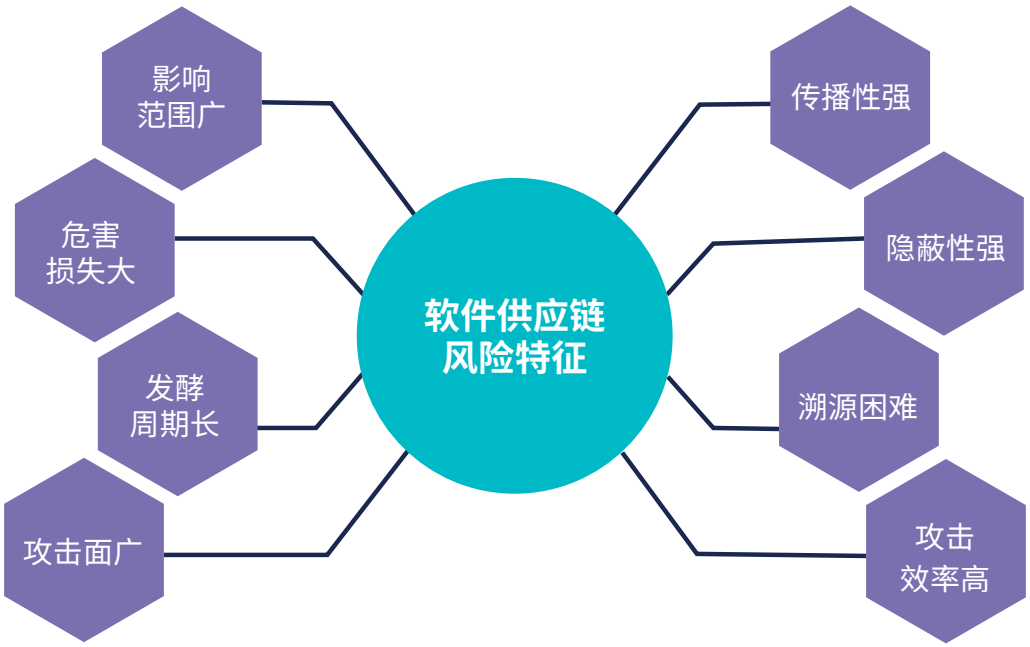

图 2-12 典型特征

影响范围广

软件供应链简单拆分为软件设计开发环节、软件分发和用户下载的交付环节、以及用户的使用环节，这三大环节环环相扣构成其链状结构。这种链状结构中每个环节都面临着安全风险，致使攻击面多，随着软件供应链的暴露面越来越多，攻击者利用其攻击的情况也就越来越多，对于上下游“感染”的机率也会更大，使得传播影响的范围更广泛。

2021 年 12 月 9 日曝出的 Log4j2 安全漏洞事件，是典型的基于开源组件漏洞引入导致的软件供应链安全漏洞事件。该漏洞影响范围极大，据粗略统计该漏洞影响 6 万 + 流行开源软件，影响 70% 以上的企业线上业务系统，IT 通信（互联网）、高校、工业制造、金融、政府、医疗卫生、运营商等几乎所有行业都受到波及。上文提到的 VxWorks Urgent11 事件，单一组漏洞便影响了全球 20 亿台类型设备。

传播性强

软件供应链涉及对象众多，如软件供应商、下游用户等比较多，当软件供应链遭受攻击时，容易对上下游带来不利影响，攻击者可以通过软件供应链的上下游关系，对其它链条的对象发起裂变式恶意攻击。比如从上游开

发环节发生的攻击，一旦成功就会波及软件供应链的中下游，从过往案例看，此类恶意攻击的效果比较明显，且影响深远。

2017 年 NotPetya 攻击和 Equifax 数据泄露影响了数百万用户。同年，勒索软件变种 Petya 攻击乌克兰，使得乌克兰首都机场、国家储蓄银行遭遇重大损失。2020 年爆发的 SolarWinds 事件直接将国家级网络攻击的关注推至新高，无论从规模、影响力和潜在威胁性来看，都堪称过去十年最重大的网络安全事件。这些攻击事件都将攻击行为上升到了国家高度，对被攻击国家的大型机构、基础设施以及大型企业等带来了威胁且规模庞大，这类攻击对国家战略布局都有深远影响，这也是软件供应链攻击的危害之处，也是近年来备受黑客青睐的攻击方式。

危害损失大

观察发现，软件供应链攻击的方式多样，通常不具有一致性，从近年来发生的多起典型软件供应链事件就可以看出这一特性。这样也给此类攻击的防御带来了一定的难度，略有无迹可寻的态势。

例如，2015 年 9 月的 XcodeGhost 开发工具污染事件，造成至少 692 种 APP 受污染，过亿用户受影响，受影响的包括了微信、滴滴、网易云音乐等著名应用。2017 年 9 月的 CCleaner 恶意代码植入事件，感染全球 2000 万用户。WireX Android 僵尸网络攻击事件影响了全球 100 多个国家，7 万多手机受到感染。NotPetya 勒索病毒事件，使得政府、银行、电力系统、通讯系统等都不同程度地受到了影响。部分事件在 2.1 小节中也有提及。

软件供应链的链状结构特点使其每个环节面临不同的安全风险，自然衍生出不同的攻击手法，例如，针对开发环节的源代码和开发工具污染，针对交付环节的下载网站和软件更新网站攻击，针对使用环节的升级更新劫持等。攻击者针对各环节的不同脆弱点实施不同攻击，再依赖供应链上的信任关系以逃避传统安全产品的检查，沿着供应链向后渗透，从而实施对目标网络的渗透及非法攻击。不管是哪种攻击形式，都对企业造成了严重危害。

隐蔽性强

观漏洞事件软件供应链攻击往往是通过骗取机构之间的信任，在用户和开发者毫无感知的情况下实施攻击行为产生巨大的安全威胁。在软件开发环节中，对于源代码、库的篡改或伪造是很难发现的，这些披着“合法”外衣的恶意软件能够轻易规避安全检测，使其能够长期潜伏在目标系统中而不被发现，具有极强的隐蔽性。

2020 年爆发的 SolarWinds 事件，隐蔽性处于其攻击思想绝对首位，攻击者选择源代码阶段，且选择代码仓库为发起感染的位置，并选择非常深层次的调用堆栈，以降低代码重构期被发现的可能性，同时在编码上，高度仿照了 SolarWinds 的编码方式与命名规范和类名等，成功绕过了 SolarWinds 公司复杂的测试、交叉审核、校验等多个环节，在技术和思维上已大大超过常规行动体。

发酵周期长

软件供应链风险典型特征中包括发酵周期长这一典型特征，披露出的安全漏洞修复，但发布的漏洞代码依旧可以被网络攻击者所利用，漏洞利用事件持续发酵，影响着软件供应链的安全。

2017 年，Apache Struts 2 被曝存在远程命令执行漏洞，漏洞编号 S2-045，CVE 编号 CVE-2017-5638，在其发布更新包修复的 48 小时之内，该漏洞的利用代码被发布在 GitHub 上，随后 Cisco Talos 团队监测到了大量漏洞利用事件。此后披露出的 Equifax 漏洞事件、2018 年 3 月的 AADHAAR 数字认证信息泄露事件、2020 年 5 月“8220 团伙”入侵企业内网挖矿事件等，都是利用未修复的 S2-045 漏洞发起的攻击。

溯源困难

由于大量引入开源组件成为了快速响应客户需求、提高软件开发效率、节省开发时间的主流解决方案，安全风险也随之而来。随着开源组件的规模越来越大，软件供应链的复杂度增大，当爆发出安全漏洞事件时，难以溯源到并修复存在安全漏洞的组件。并且一旦软件中一个供应链环节使用了含有安全缺陷的开源软件，并且被编译为可执行代码后，由于代码使用者无法修改这些含有安全缺陷的可执行软件，导致软件供应链的这个环节永远存在无法消除的安全威胁和隐患，SolarWinds 攻击事件就是典型案例。

Kevin Mandia（FireEye CEO）：“我们投入了 100 名左右的人员，总共进行了 10000 小时的详细调查，我们依旧不知道攻击者的入侵路径。因此我们只好更进一步，反编译了 18000 个更新文件，3500 个可执行文件，得到超过一百万行的反编译代码，进行详细的代码审计，工作了 1000 个小时以上，才得出结果，这就是为什么我们这么难以发现它。”

图 2-13 Kevin Mandia（FireEye CEO）

由此针对软件采用供应链溯源技术愈加重要，通过建立全球开源软件的传播态势感知和预警机制，攻克软件供应链中软件来源电子标签技术，实现对供应链各环节中软件来源的溯源机制，以解决软件供应链安全风险难溯源的问题。

攻击面广

相较于传统的针对软件漏洞的攻击，软件供应链攻击从软件本身扩展为软件以及内部的所有代码、模块和服务，以及与这些模块相关的供应链上游供应商的编码过程、开发工具和设备，致使攻击面增多，大大增加了攻击途径，为攻击者提供了便利。

2015 年 9 月 14 日“XcodeGhost”恶意攻击事件对开发工具的污染。2017 年 8 月 WireX BotNet 攻击事件在用户下载环节伪装成普通的安卓程序发动了较大规模的 DDoS 攻击。2021 年 4 月 GitHub Actions 攻击案例，利用 GitHub 的 CI/CD 基础设施非法挖掘加密货币。

攻击效率高

随着软件供应链攻击面的增大，在软件供应链攻击中攻击者仅需对某一薄弱环节进行软件攻击或篡改，就可能引起引起软件供应链安全问题，产生巨大的安全危害。而这种投入产出比高的攻击特点，一直是网络攻击者热衷的渗透打点手段，也是企业、个人及安全保障团队难以完全杜绝的攻击场景。

2017 年多款 Python/Nodejs 库包被爆存在恶意代码，主要利用名称相似性误导用户安装，数万主机误安装受到影响。2021 年堪称“核弹级”的安全漏洞事件 Log4j2.x，其危害程度极高，影响范围巨大，但利用方式十分简单直接，攻击者仅需向目标输入一段代码，不需要用户执行任何多余操作即可触发该漏洞，使攻击者可以远程控制用户受害者服务器。

2.3 开源软件供应链攻击不断增多

软件供应链攻击往往不是针对性的，更多是机会性的，攻击者通常不会通过目标逆向，找到软件的供应商，进而展开一系列攻击。更多是在不同的阶段，寻找合适的机会发起攻击。比如，在开发过程中，软件开发人员经常使用开源代码库来构建应用程序，当攻击者将恶意代码插入到可公开访问的代码库中时，恶意代码将植入到代码片段中。当开发人员使用代码库中的代码时，恶意代码将会被植入，攻击者便可以通过此类恶意代码对目标展开攻击，进而造成严重危害。根据 2021 年开源安全与风险分析（OSSRA）报告显示，84% 的代码库至少存在一个漏洞，每个代码库平均有 158 个漏洞。可见，开源软件供应链攻击产生的机会性是非常高的。

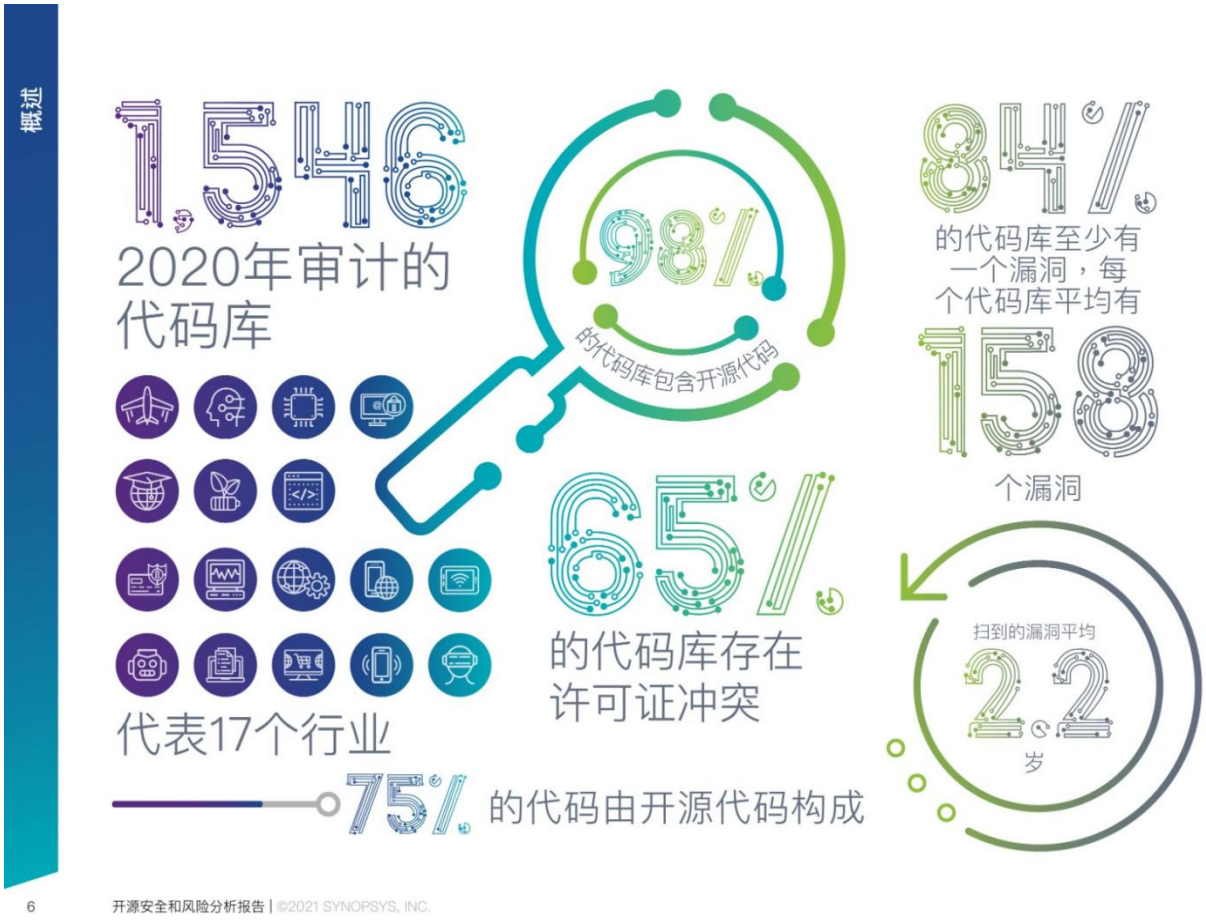

图 2-14 开源安全与风险分析（OSSRA）报告数据

Linux 操作系统的作者 Linus Torvalds 曾这样表达开源项目的优势：众人的关注使 BUG 变少。这也许是一个事实，但是如果同样的关注正在寻找的是可以利用的漏洞呢？我们至今无法逃脱另一个人类社会的事实：并非所有人都怀抱善意。任何拥有 GitHub 帐户的人都可以向关键库贡献代码，这吸引了很多伟大的贡献者，但也掺杂了很多居心叵测的行为。

这些行为通常是隐蔽的。在一些攻击事件中，我们看到攻击者会申请为少数管理员进行维护的开源项目或库贡献自己的代码，这些开源项目虽然维护者不多，但在某一些领域相对流行，甚至会被一些大型企业的开发团队所采用。由于管理人员较少，项目的拥有者通常很愿意接纳新的维护申请，并对提交的代码疏于防范。此时攻击者悄悄植入恶意代码和脚本，就成为了自然而然的事情。这些有意植入的开源漏洞助长了攻击者对开源软件攻击的意愿，同时也大大提升了对下游的终端应用进行渗透的收益。

2.4 软件供应链安全常见风险

在《软件供应链安全白皮书（2021）》中针对软件供应链安全风险进行了分析，从软件供应链风险现状、风险因素、软件供应链的漏洞类型、软件供应链的攻击类型四个维度为大家描述了软件供应链的安全风险。此次基于过往的内容做了更新和补充，以更完整的视角为大家分享软件供应链安全常见的风险问题。

1. 软件依赖进口，源头难以控制

目前，我国在国家信息建设中关键软件技术方面还无法实现完全自主研发可控，诸多核心行业的关键基础软件长期依赖进口，这为我国的软件供应链安全留下了难以估量的安全隐患。软件供应链上的任一环节出现问题，都有可能带来严重的安全风险，在国家关键基础设施中过度使用从国外引进的技术或软件产品，将加大我国软件供应链安全出现威胁的可能性，危害国家信息安全，同时也会给国家经济安全带来严重的隐患。

除此之外，西方大国还借助其在部分 IT 领域的技术优势，在相关软硬件产品内预置后门，通过我国进口其软件产品将安全威胁带入国内，非法获取我国重要的信息数据。因此，长期依赖软件进口，将难以从软件生产的源头进行安全治理，会加剧我国软件供应链所面临的安全风险，影响我国推进软件供应链安全的进程，甚至会威胁到国家安全。

2. 开源存在缺陷，引入安全风险

为了加快业务创新，应用开源技术提高开发效率已经成为企业的主流选择，但也导致企业对复杂的软件供应链的依赖日益增加。尽管开放源码组件具有许多优点，但它的广泛应用也带来了新的安全挑战，一方面由于开发者自身安全意识和技术水平不足容易产生软件安全漏洞，另一方面也无法避免恶意人员向开源软件注入木马程序进行软件供应链攻击等安全风险的存在。

据 Sonatype 发布的《2021 State of the Software Supply Chain》报告数据显示，供应链攻击呈指数级增长，2021 年全球软件供应链攻击增加了 650%。开源漏洞在流行项目中最为普遍，29% 的流行项目至少包含一个已知的安全漏洞。开源组件的使用增加，其安全性已经成为影响软件供应链安全的重要一环。

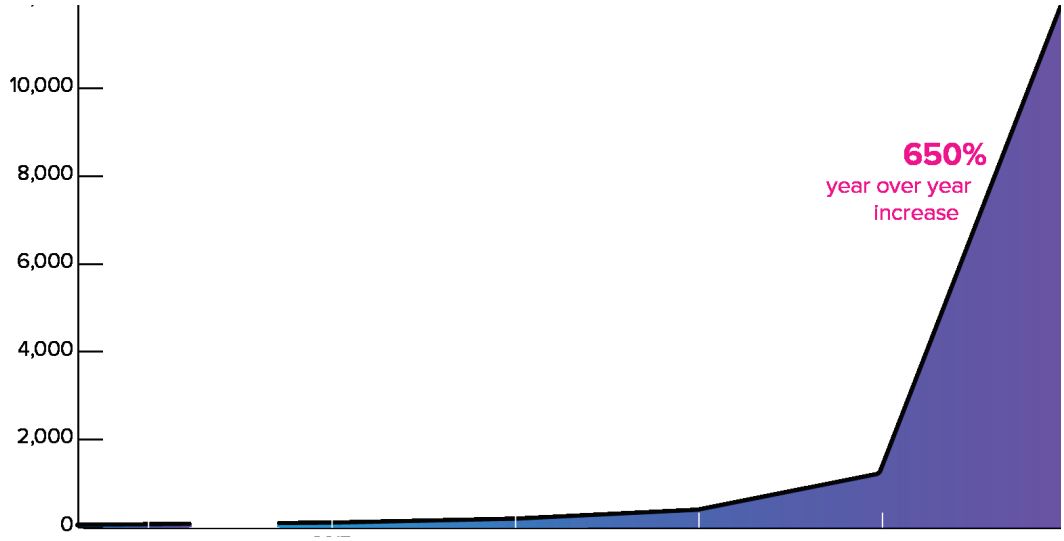

图 2- 15 软件供应链攻击增加了 650%

3. 重视程度不够，安全防护不足

由于安全与敏捷开发往往呈对立关系，开发者为了提高效率，往往会忽视掉软件供应链的安全性，安全保障过程被孤立，实行“业务先行”的模式。 然而，业务系统从设计、编码、测试到上线运行各个环节都有可能出现安全漏洞，给业务系统带来安全风险。业务系统中水平 / 垂直越权、批量注册、业务接口乱序调用等业务逻辑漏洞和第三方开源组件漏洞频发，高危的安全漏洞一旦被利用，可能会造成严重的信息泄露或系统中断问题。

而且，企业对软件供应链安全技术研发的投入远远不够，敏捷开发与快速迭代导致企业往往在加快开发进度的同时忽略掉部分安全隐患和响应效率，来弥补开发过程中时间的匮乏。高速运转的敏捷开发运营模式将传统的安全工作甩在身后，仅在上线运行阶段开展安全风险控制工作，已无法防御由网络技术发展带来的各项安全威胁。

4. 管理制度不完善，安全评估缺失

企业软件供应链管理制度不完善，缺乏针对软件生产等重要环节的管控措施。而且，企业软件供应链透明度不高，安全评估缺失，难以依据安全风险划分供应商的安全等级，从而进行针对性的安全管理。包括部分企业开源代码管理机制尚不完善，在软件开发过程中，随意使用开源组件的现象屡见不鲜，管理者和程序员无法列出完整的开源组件的使用列表，对软件供应链安全问题严重缺乏重视，给软件供应链管控带来极大的安全挑战。

5. 网安意识薄弱，缺少安全培训

网络安全归根结底是“人”的安全。公司的员工是抵御网络攻击的第一道防线，当对员工没有进行定期的网络安全意识培训时，员工在缺乏安全意识的情况下缺少对敏感数据潜在攻击的识别能力，极易通过人而造成攻击的产生。攻击者一旦突破防线，进入业务系统，便可以实现上下游的攻击，对企业组织造成严重后果。

03 软件供应链安全面临的挑战

开源技术及云原生技术的应用正逐渐形成主流趋势，不断推动着深度信息技术的创新发展，驱动着产业数字化转型进程的不断加速，但与此同时，软件供应链也越来越趋于复杂多样化，安全风险加剧，针对软件供应链薄弱环节的网络攻击随之增加。除了国际层面上软件供应链竞争环境加剧，以及安全管理制度不完善等外部因素外，由云原生和开源带来的下一代软件供应链安全正面临着新的安全挑战。

3.1 开源技术的使用，安全风险加剧

作为业务应用程序的重要组成部分，开源软件已成为网络空间的重要基础设施。作为创新的基础，开源软件需求量正呈爆炸式增长，根据 Sonatype 发布的《2021 State of the Software Supply Chain》报告可以了解到，2021 年世界各地的开发者请求超过 2.2 万亿个的开源包，意味着开源组件的下载量同比增长 73%。

为实现企业业务的快速开发和科技的创新，从根本上要求开发人员频繁而有效地重用代码，从而又导致了开发人员对第三方开源组件的严重依赖。开源软件的大量使用帮助企业加速了软件开发的生命周期，降低了开发成本，但同时也带来了两大安全风险：嵌入到开发流程中的安全漏洞风险和构成法律或知识产权（IP）风险的许可证问题。

3.1.1 安全漏洞风险

因其开放、自由、共享等特性，开源技术可以从代码托管平台、技术社区、开源机构官方网站等渠道获取，或通过合作研发、商业采购等方式引入开源代码、开源组件、开源软件和基于开源技术的云服务等。开源社区参与者广泛分布于全球各地，没有中央权威来收录漏洞信息，保证开源软件的质量和维护。开发人员不了解开源软件信息，缺少漏洞跟踪能力，漏洞修复时间滞后，增加了软件供应链安全治理难度。

本质上来讲，开源软件并不比自定义代码更安全，与任何软件一样，它可能包含导致安全问题的漏洞。而且，大部分软件开发人员在引入第三方库的时候，并没有关注引入组件是否存在安全隐患或者缺陷，并且由于开源软件之间的关联依赖关系错综复杂，一旦开源软件存有恶意代码或病毒，将会产生蝴蝶效应，导致所有与之存在关联依赖关系的其他软件系统出现同样的漏洞，漏洞的攻击面由点及面呈现出爆炸式的放大效果，给用户带来严重危害。图为信通院 2021 年发布的《2021 年开源软件供应链安全风险研究报告》中组件漏洞依赖层级传播范围，相关数据表明一级组件漏洞传播影响范围扩大了 125 倍，二级传播影响范围扩大了 173 倍。

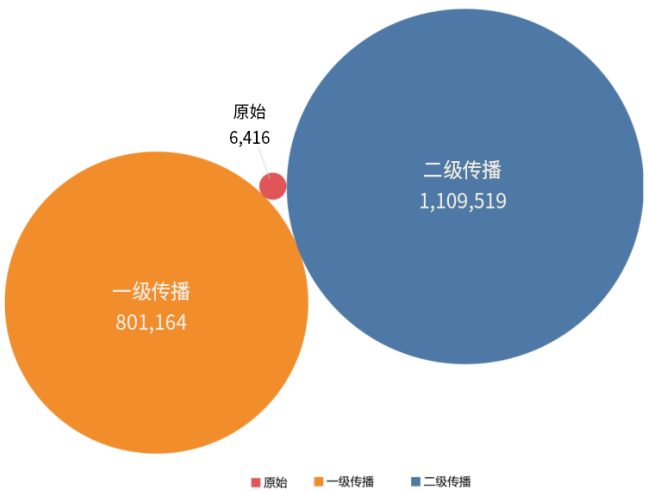

图 3-1 组件漏洞依赖层级传播范围

3.1.2 许可证合规及兼容风险

开源软件对用户是免费的，但并不意味着可以在不遵守其它义务的情况下使用。随着开源技术的发展，结合各自需要，开源软件一般都有对应的开源许可证（Open Source License），用以对软件的使用、复制、修改和再发布等进行限制。常见的开源软件许可证根据开放程度可以大致分为两大类：宽松自由软件许可协议（“Permissive Free Software Licence”）和著作权许可证（“Copyleft License”）。宽松自由软件许可协议包括 MIT、Apache、BSD 等，允许用户自由复制、修改、许可和再许可代码，开源软件源代码变更或衍生软件可以变为专有软件；著作权许可证包括 GPL 许可证、MPL 许可证和 LGPL 许可证等，强制要求公开源代码变更或衍生软件开源。

Contrast Security 发布的《2021 State of Open-Source Security Report》报告中相关数据表明，几乎所有（99%）的组织都至少有一个高风险的 Java 许可证，69% 的 Java 应用程序和 33% 的 Node 应用程序中至少存在一个高风险许可证，95% 的 Java 应用程序和 70% 的 Node 应用程序至少存在一个未知或可变风险的许可证。如果不了解开源软件的知识产权或未按照开源许可证使用开源软件，很可能侵犯他人知识产权，引起法律纠纷，面临重大风险。同时开源软件可能存在层层依赖关系，企业组织在使用开源过程中不断加入新的开源组件，可能存在许可证不兼容的风险。

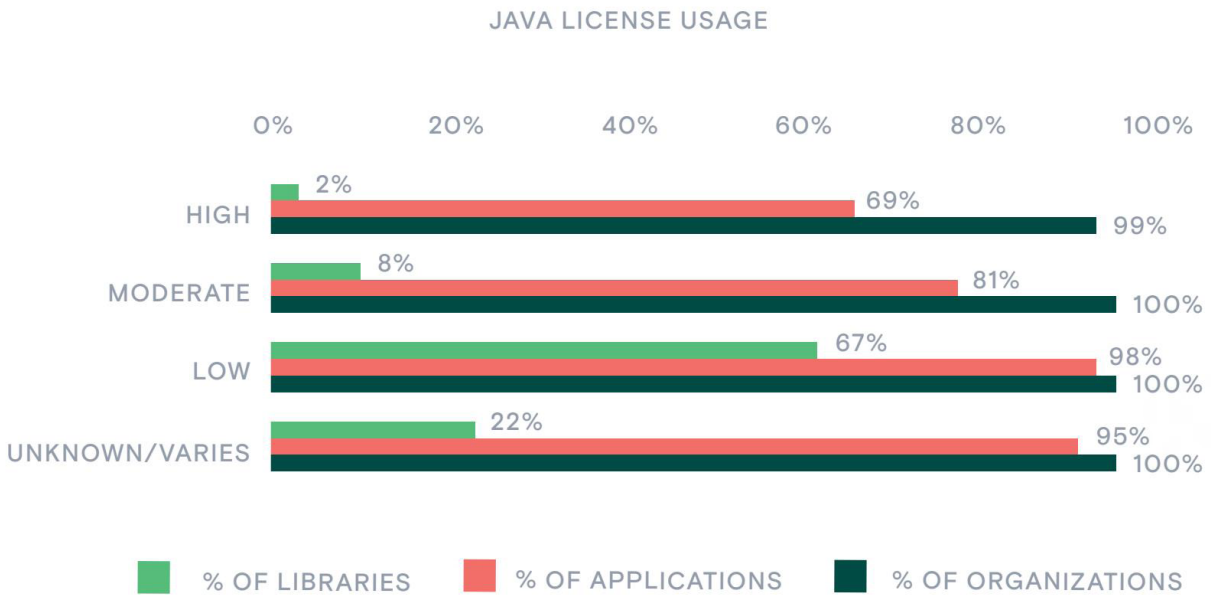

图 3-2 JAVA 许可证使用

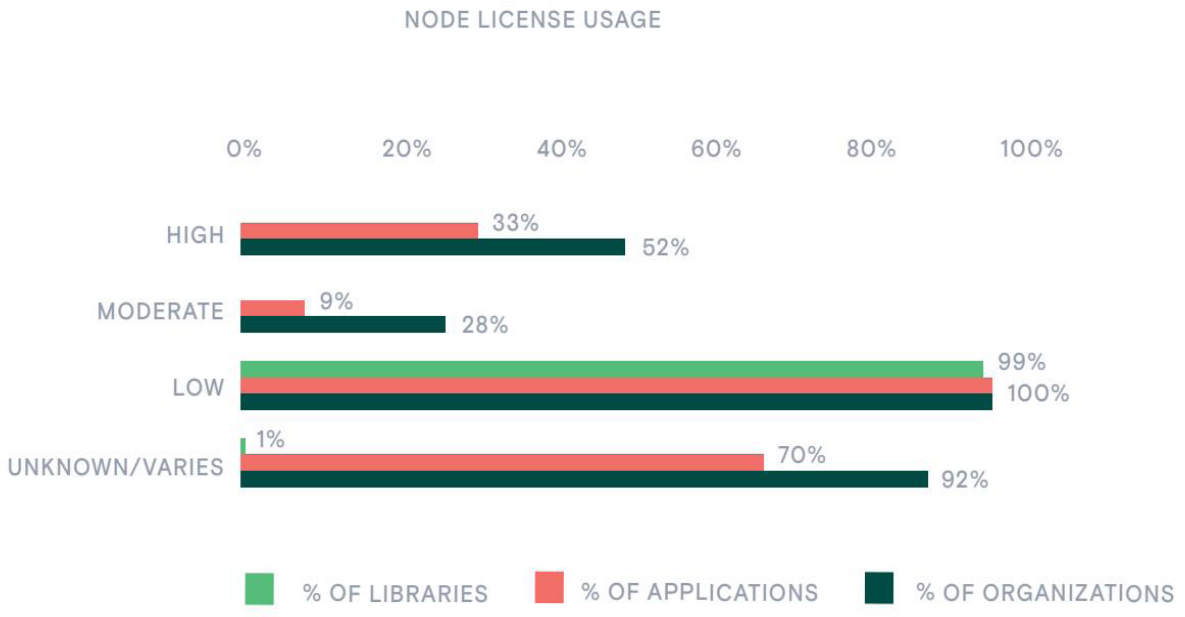

图 3-3 NODE 许可证风险

3.2 云原生技术的兴起，复杂度增加

以容器、微服务、DevOps、不可变基础设施为基础的云原生技术加快了业务响应速度，有效缩短了交付周期，降低了运营成本，然而云原生技术的发展极度加大了软件供应链的复杂性，为软件供应链带来了新技术风险和新安全组织模式的双重挑战。

其中 DevOps 和持续交付打破开发与运营之间的壁垒，实现软件开发的自动化、敏捷性，帮助组织实现了应用程序更快更可靠的大规模交付，同时开发人员出于对速度和效率的追求，经常会利用开源项目的框架进行二次开发或引用开源组件，在云环境中，这些组件常以基础镜像的方式在软件供应链里传递，并上传至镜像仓库，从而被容器所使用。若开源组件和框架中存在安全漏洞，那么镜像软件就有可能存在漏洞，镜像文件完整性被破坏，镜像文件遭受恶意配置或者更改（比如上传或者下载过程被修改，植入后门）导致容器被利用势必会引入所开发的应用中，导致大面积的风险漏洞的存在。

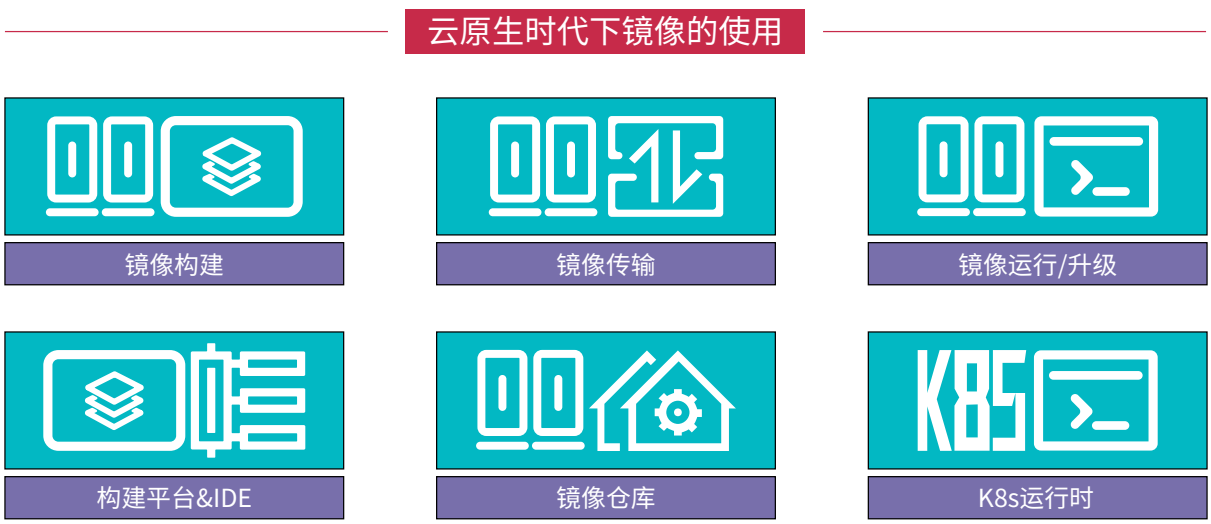

图 3-4 云原生时代下的软件供应链

据 Gartner 预测，随着越来越多的企业步入云原生化的进程，更多地采用云原生应用程序和基础设施，到 2025 年，成熟经济体中 85% 的大型企业将更多地使用容器管理，远高于 2022 年的 30%。容器化的使用，使得应用部署和运行非常便利，但通过引入容器和 K8s 给软件供应链带来了更多不可控的第三方依赖，导致漏洞利用、容器逃逸、拒绝服务等风险的存在，从而将安全漏洞大面积引入到云环境中，以致威胁整个云生态。

3.3 软件供应链安全治理痛点

近年来，全球范围内有关软件供应链安全的攻击事件层出不断，对个人、企业，甚至国家安全都造成了严重威胁。典型的 Apache Log4j2 漏洞被业内称为“核弹级”的安全漏洞，其严重性、广泛性对各个领域的波及面非常大，让企业组织意识到做好软件供应链威胁治理迫在眉睫。但在软件供应链治理的过程中，存在一些难点会让实践裹足不前。软件供应链安全治理通常具有如下四个痛点：

1. “理不清”

企业不清楚在系统中使用了多少第三方软件和组件。目前很多企业的现状是，在每一次快速迭代和快速发布过程中，都会引入一部分第三方组件，但是由于团队多、涉及的业务范围广，导致无法清楚地知道具体使用了哪些。而且，第三方组件通常又会依赖其它更多组件，多级依赖关系使得整个组件结构更加复杂，这种结构的安全性对于应用的研发和使用来说，很多时候也是未知的，不可控的。

2. “看不见”

企业在使用第三方组件的过程中，不知道它们中有的已产生过安全漏洞和知识产权风险。很多企业会使用非常老的组件和软件，其中很多爆发过安全漏洞，但没有及时去更新。对于这些已知漏洞的风险隐患，企业无法获悉，这种不可见性增加了危险系数。

3. “找不到”

企业在第三方组件出现漏洞的时候，无法快速地定位受影响的组件以及评估影响的范围。例如在 Log4j2 事件和 Spring 事件爆发时，安全工程师希望能够立即找到自身业务系统中受影响的软件和组件，进而设法进行治理。但是漏洞数据源的回溯需要时间，且随着软件成分的日渐复杂，类似的安全问题更加严峻。

4. “治不了”

当企业明确漏洞影响的范围以及受影响的组件并定位到具体项目后，就需要进行相关治理工作，对组件进行相应的评估、缓解和修复。当前很多企业缺乏软件供应链安全治理相关知识、技能和工具，对风险缓解的方法粗犷且效率低下。

04 软件供应链安全治理体系

由于软件供应链安全面临着严峻的挑战和治理痛点，很多机构推出了针对软件供应链的安全治理体系，以求系统化地、基于完整的视角解决软件供应链安全威胁问题。

4.1 软件供应链安全治理框架

4.1.1 Google SLSA 框架

针对软件供应链攻击问题，谷歌提出了 SLSA（Supply Chain Levels for Software Artifacts，软件制品的供应链级别）解决方案，旨在确保软件开发和部署过程的安全性，专注于缓解由于篡改源代码、构建平台或构件仓库而产生的威胁。

在其当前状态下，SLSA 是一套被逐渐采用的安全准则，由业界一致认可。在最终形式中，SLSA 在可执行性方面与最佳实践列表有所不同：它将支持自动创建可审核元数据，这些元数据可以输入到策略引擎中，以便为特定包或构建平台提供“SLSA 认证”。SLSA 被设计成持续的和可操作的，并且在每一步都提供最优措施。一旦一个组件达到了最高级别，用户就可以确信它没有被篡改，并且可以安全地追溯到源代码。

SLSA 侧重于以下两个主要原则：

（1）非单方面：
任何人都不能修改软件供应链中任何地方的软件制品，除非经过至少一个其他“受信任的人”的明确审查和批准。目的是预防和尽早发现风险的变化。

（2）可审计：
软件制品可以安全透明地追溯到原始的、人类可读的来源和依赖项。主要目的是自动分析来源和依赖关系，以及临时调查。

尽管并不完美，但这两个原则为广泛的篡改、混淆和其他供应链攻击提供了实质性的缓解。为了根据上述两个原则衡量供应链的保护程度，Google 提出了 SLSA 级别，SLSA 由四个级别组成，其中 SLSA 4 表示理想状态。较低级别表示具有相应完整性保证的增量里程碑。这些要求的定义如下表所示。

表 4-1 SLSA 框架要求

		Required at			
Requirement		SLSA 1	SLSA 2	SLSA 3	SLSA 4
Source	Version Controlled		✓	✓	✓
	Verified History			✓	✓
	Retained Indefinitely			18 mo.	✓
	Two-Person Reviewed				✓
Build	Scripted	✓	✓	✓	✓
	Build Service		✓	✓	✓
	Ephemeral Environment			✓	✓
	Isolated			✓	✓
	Parameterless				✓
	Hermetic				✓
	Reproducible				○
Provenance	Available	✓	✓	✓	✓
	Authenticated		✓	✓	✓
	Service Generated		✓	✓	✓
	Non-Falsifiable			✓	✓
	Dependencies Complete				✓
Common	Security				✓
	Access				✓
	Superusers				✓

○ = *required unless there is a justification*

SLSA 1

要求构建过程完全脚本化 / 自动化，并生成出处。出处是关于如何构建中间件的元数据，包括构建过程、顶级源代码和依赖关系。了解出处可以让用户做出基于风险的安全决策。尽管 SLSA 1 的源代码不能防止篡改，但它提供了基本级别的源代码识别，有利于漏洞管理。

SLSA 2

需要使用版本控制和生成经过身份验证出处的托管构建服务。这些额外的要求使用户对软件的来源有了更大的信心。在这个级别上，源代码在构建受信任服务过程中可以防止被篡改。SLSA 2 还提供了到 SLSA 3 的简单升级途径。

SLSA 3

进一步要求源代码和构建平台分别满足特定标准，以保证源代码的可审计性和完整性。设想有这么一个认证过程，通过这个过程，审计人员可以证明平台符合要求，用户就可以直接依赖它了。SLSA 3 通过防御特定类型的威胁（如交叉编译污染），提供了比低级别更强大的防篡改保护。

SLSA 4

目前是最高级别的，需要双人对所有变更进行审查，并且需要一个封闭的、可复制的构建流程。双人评审是一种行业最佳实践，可以发现错误并阻止不法行为。封闭的构建保证了源代码的依赖列表是完整的。可复制的构建，虽然不是严格要求的，但是提供了许多可审计性和可靠性的优势。总的来说，SLSA 4 让用户对软件没有被篡改有绝对的信心。

SLSA 是一个用于端到端软件供应链完整性的实用框架，基于一个在世界上最大的软件工程组织中被证明可以大规模工作的模型。对于大多数项目来说，实现最高级别的 SLSA 可能很困难，但是较低级别的 SLSA 所认可的持续改进将在很大程度上提高开放源代码生态系统的安全性。

4.1.2 CNCF in-toto 框架

2022 年 3 月 11 日，CNCF 发文表示，CNCF 技术监督委员会（TOC）已投票接受 in-toto 作为 CNCF 的孵化项目。in-toto 是一个用于保护软件供应链完整性和可验证性的系统。它通过使用户能够收集到关于软件供应链行为的信息，并允许软件消费者和项目经理发布关于软件供应链实践的政策来保护软件供应链的完整性和可验证性，这些政策可以在部署或安装软件之前进行验证。简而言之，它有助于捕获软件供应链中发生的事情，并确保它按照定义的策略发生。

in-toto 的实现主要包含三个组件：

• 用于生成和设计供应链布局的工具。项目所有者将使用此工具生成所需的供应链布局文件。其中供应链布局（也称布局、布局元数据）是指一个签名文件，规定了在软件供应链中创建最终产品需要执行的一系列步骤。布局包括有序步骤、这些步骤的要求以及负责执行每个步骤的参与者（或工作人员）列表。供应链中的步骤由项目所有者布置。

• 一种工具，工作人员可以使用该工具创建有关步骤的链接元数据。例如“in-toto-run.py”。其中，链接是指执行供应链步骤或检查时收集的元数据信息，由执行该步骤的官员或执行检查的客户签名。此元数据包括材料、产品和副产品等信息。

• 客户用来对最终产品进行验证的工具。此工具使用上述工具生成的所有链接和布局元数据，它还按照布局的指示执行检查步骤。此工具通常包含在包管理器或系统安装程序中。例如“in-toto-verify.py”。

下面我们将描述一个简单的场景，对框架系统的工作流进行一个说明：

项目所有者 Alice 希望 Diana 写一个 Python 脚本（foo.py），且希望 Bob 将脚本打包成一个 tarball（foo.tar.gz）。该压缩包将作为最终产品的一部分发送给客户 Carl。在向 Carl 提供压缩包时，Alice 将创建一个布局文件用来说明以下内容：

- 脚本是 Diana 写的；
- 将脚本打包是由 Bob 完成的；
- 压缩包中包含的脚本与 Diana 编写的脚本匹配。

因此如果 Bob 是恶意的，或者打包程序有错误，Carl 为了能够检测到问题，在最终产品中要求四个文件：（1）Alice 的布局文件，描述上面列出的要求；（2）一段能够对应于 Diana 编写脚本的链接元数据；（3）一段 Bob 打包脚本步骤的链接元数据；（4）目标文件（foo.tar.gz）也必须包含在最终产品中。

当 Carl 验证最终产品时，他的安装程序将执行以下检查：

(1) 布局文件存在并使用受信任的密钥（在本例中为 Alice 的密钥）进行签名；

(2) 布局文件中的每个步骤都有一个或多个由预期函数签名的相应链接元数据文件，如布局文件中所描述的信息（在本例中为 Bob 和 Diana 提供的链接元数据）；

(3) 链接元数据中列出的所有材料和产品都能够一一对应起来，正如布局文件描述的信息一样，防止包在没有记录的情况下被更改，或在传输过程中被篡改。（在本例中 Diana 和 Bob 的材料应相匹配）；

(4) 最后，检查步骤在客户端运行，检查压缩包，以验证提取的 foo.py 文件是否与 Diana 编写的文件相匹配。如果这些所有验证都通过，则安装将照常继续。

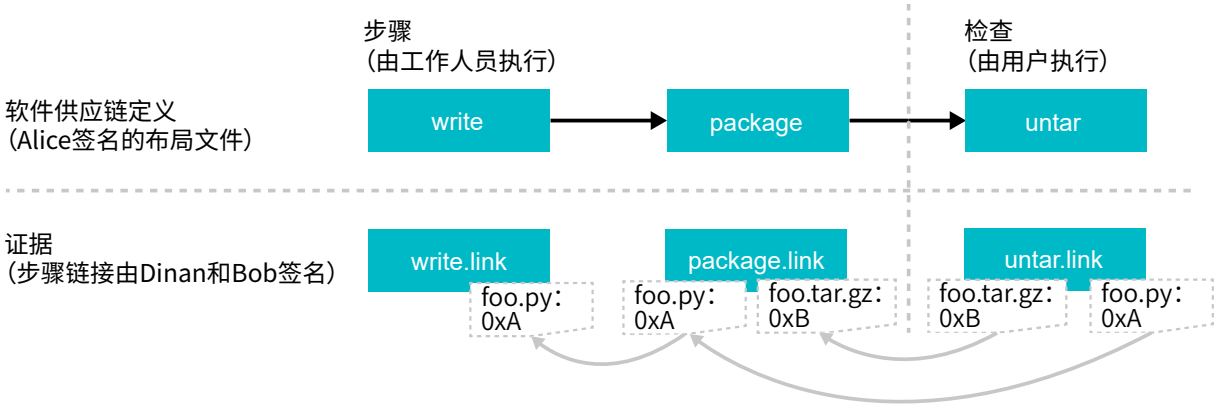

图 4-1 此示例的流程

4.1.3 Microsoft SCITT 框架

Microsoft 的供应链完整性、透明度和信任 (SCITT，The Supply Chain Integrity，Transparency and Trust) 计划是一套已提议的行业标准，用于管理端到端供应链的产品和服务的合规性。它支持对产品和服务的持续验证，其中可以确保实体、证据、政策和制品（软件或硬件）的真实性，并且可以保证实体的行为是经过授权的、不可否认的、不可变的和可审计的。

Microsoft 在之前将 SCITT 称为 SCIM（Supply Chain Integrity Model，供应链完整性模型），包括在去年提交给 NIST 的，与 EO 14028 相关的提交中。SCIM 与开发和实施供应链完整性要求的迭代方法保持一致，允许根据不断变化的威胁模型和实践随着时间的推移进行增强。分阶段推出需求以提高供应商规划和工程的清晰度，并最大限度地减少中断。其工作流程如图所示，描述了供应链完整性模型中实体之间的制品、证据和策略流。

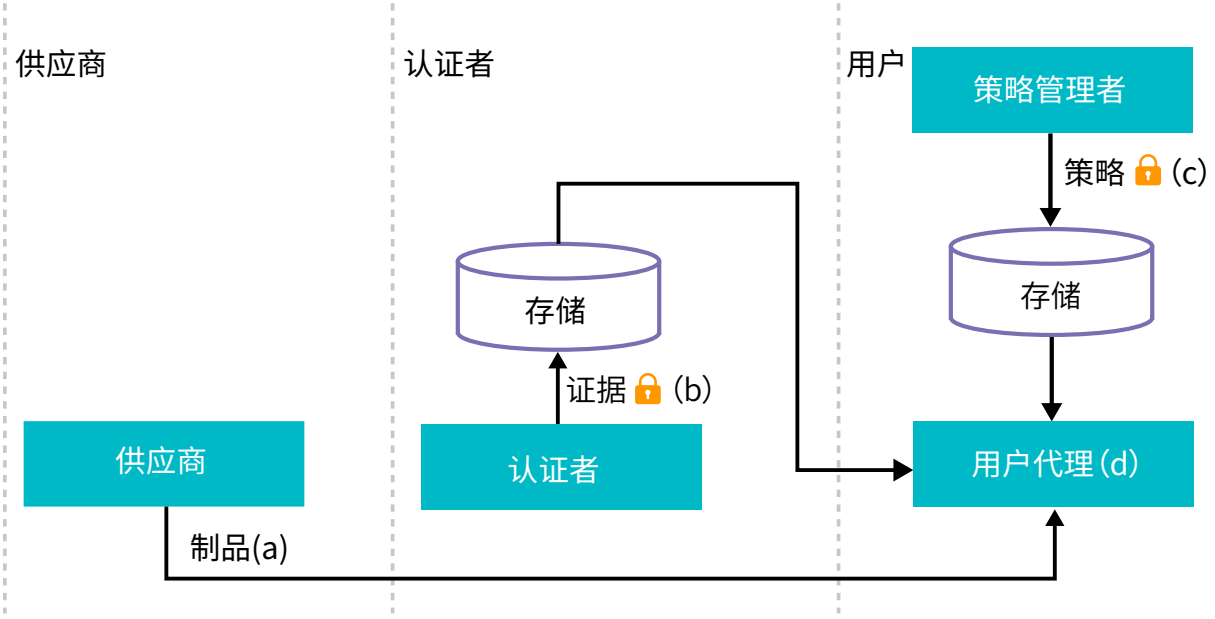

图 4-2 SCIM 的工作流程

其中，供应商创建制品（a），认证者创建证据（b）并提交到应用商店进行日志记录、查询和检索。供应商和认证者可能是同一实体。策略管理器创建策略（c）并提交到存储区，在其中记录策略并可供查询和检索。用户代理（d）接收制品（a），检索证据（b）和策略（c），并验证制品。

下图显示了 SCIM 在软件开发生命周期（SDLC）中的应用示例。

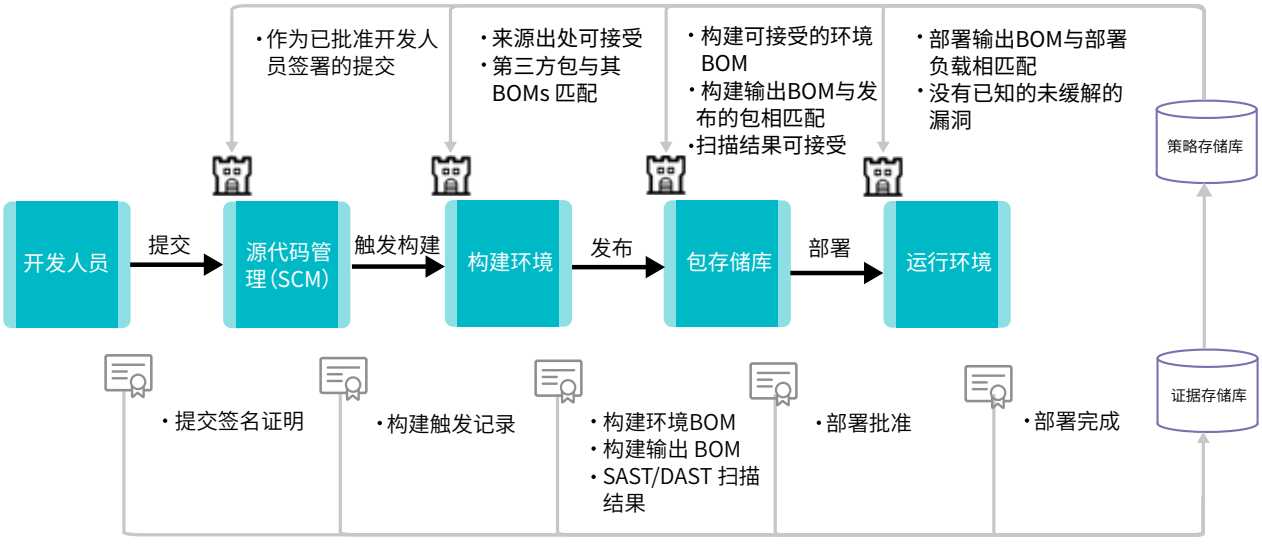

图 4-3 SCIM 在软件开发生命周期（SDLC）中的应用

当前 SCITT 行业标准还在制定中，随着 NIST 对其不断地完善，相信之后 SCITT 能够发挥其更大的作用。

4.1.4 软件供应链安全框架对比分析

软件供应链安全框架和模型的建立可以帮助企业组织理解软件供应链安全活动的关键要素，从而帮助企业组织了解当前自身安全现状，在企业构建的软件供应链安全治理与运营体系中选择适合自身企业的安全活动或措施，制定相应的解决方案，进而使组织的软件供应链安全逐渐成熟。上述三种软件供应链安全治理框架对比分析如下：

表 4-2 软件供应链安全治理框架

	软件供应链安全治理框架		
	SLSA框架	in-toto框架	SCITT框架
框架类型	安全准则	安全项目	行业标准
机构	Google	CNCF	Microsoft
适用范围	软件供应链	软件供应链	供应链
可操作性	SLSA框架遵循四个保证级别以确保完整性	in-toto框架遵循低、中、高三个保证级别以确保完整性	SCITT框架描述了传达证据的原则和建议的模型，未指定实施要求
框架应用情况	由业界一致认可且逐渐应用的实用性模型	框架落地，但未广泛应用	还在制定提议中
相同点	应用上述三种软件供应链安全治理框架都可以依靠某种政策或策略从端到端的保障软件供应链的完整性、不可否认性以及可审计等，提高了软件供应链的透明度及可追溯性。		

4.2 治理体系构建

软件供应链作为一个复杂、庞大的系统，难免存在薄弱环节，这就为攻击者创造了一个契机，攻击者无需进行代码审计或漏洞挖掘，而是以软件供应链中的薄弱环节为攻击点进行软件攻击或篡改，发起软件供应链攻击，引起软件供应链安全问题，从而产生巨大的安全危害。因此，保障软件供应链安全，若仅仅对某单一环节进行安全防护是远远不够的，需从软件供应链的供应过程及软件自身安全出发，基于 4.1 节所介绍的软件供应链安全治理框架，本报告构建了一套较为全面系统的软件供应链安全治理体系，借助安全自动化工具，实现对软件供应链全链路的安全风险治理，如 4-4 所示。

软件供应过程风险治理主要是指对软件产品由供应商运输给最终用户的这一供应过程的风险治理以及供应过程中人员（包括供应商、管理人员、开发人员、测试人员、运营人员、终端用户等）的安全管理；软件开发生命周期安全风险治理主要指软件开发过程中从需求设计、开发测试、发布运营、下线停用等环节进行全流程的闭环安全风险监控及安全管理，确保从软件生命周期的源头保障软件供应链安全，实现开发运营的安全闭环。

图 4-4 软件供应链安全治理体系

4.3 软件供应过程风险治理

软件供应过程风险治理，主要包括软件来源管理、软件安全合规性管理、软件资产管理、服务支持及安全应急响应等，以提升软件供应链可追溯性和透视性，如图 4-5 所示。

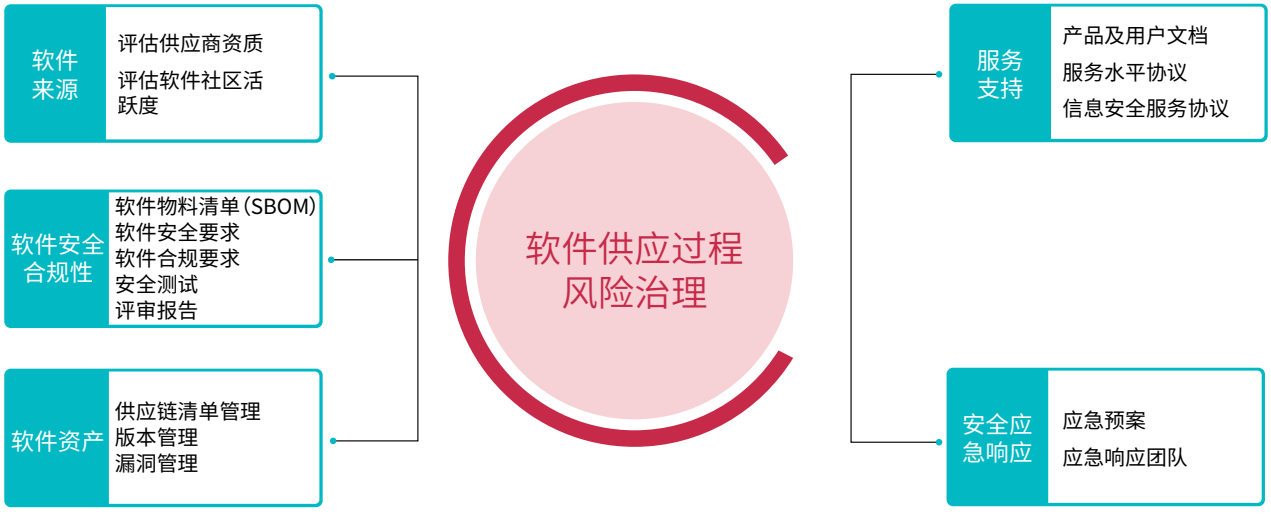

图 4-5 软件供应过程风险管理

4.3.1 软件来源管理

企业对软件来源进行管理以确保软件的来源安全可信，包括评估软件供应商资质、软件社区活跃度等。其中软件供应商作为当前较为重要的软件来源，对软件供应商的企业资质、能力资质等进行评估至关重要，接下来主要从软件供应商风险管理流程、软件供应商评估模型、供应商风险评估流程这三方面详细介绍软件供应商风险治理。

1. 软件供应商风险管理流程

基于 2021 年悬镜安全出版的《软件供应链安全白皮书（2021）》报告中提出的软件供应商风险管理流程，我们对其进行了流程上的优化，如下图所示。

标准确立：结合企业自身需求的实际情况，规划供应商管理计划，构建软件供应商评估模型，创建供应商库存并跟踪业务定义的关键属性，制定软件供应商考核的评估标准及安全框架；

供应商资质评估：根据制定的软件供应商评估模型和相关标准，对初步符合要求的软件供应商进行尽职调查，从如企业资质、市场影响力、技术储备、创新能力、软件质量、故障修复能力等多角度多维度进行综合性资格评估，选出匹配度最高的供应商；

供应商风险评估：对通过资格评估的软件供应商进行安全风险评估，针对软件供应商面临的潜在的安全风险、存在的弱点以及有可能造成的影响综合分析其可能带来的安全风险进行评估；

供应商风险监控：对软件供应商实施长期性的安全风险监控，持续识别和管理软件供应商的安全风险，根据监测结果实施更新软件供应商的风险管理策略；

供应商退出：在供应商期限结束时，应再次审查合同，以确保所有义务都已得到履行，随后进入退出程序，仔细考虑数据保存和处置，最后复盘，对软件供应商风险管理系统及供应商准入机制进行持续的优化和改进。

图 4-6 软件供应商风险管理流程

2. 软件供应商风险评估模型

供应商风险评估是指企业通过对供应商的产品分析及企业过程的诊断，用于了解和降低与产品或服务的第三方供应商合作所带来的风险。在购买或与第三方供应商合作时，组织容易受到各种风险的影响，例如安全漏洞，业务中断，供应商不遵守法规和行业标准而造成的声誉损害等。为了防止这些情况发生，企业应对其进行评估，对潜在风险进行识别、衡量和确定优先级，以确定潜在风险带来的影响。

《软件供应链安全白皮书（2021）》报告中提出了一种较为系统化、结构化的供应商风险评估模型，如下图所示。

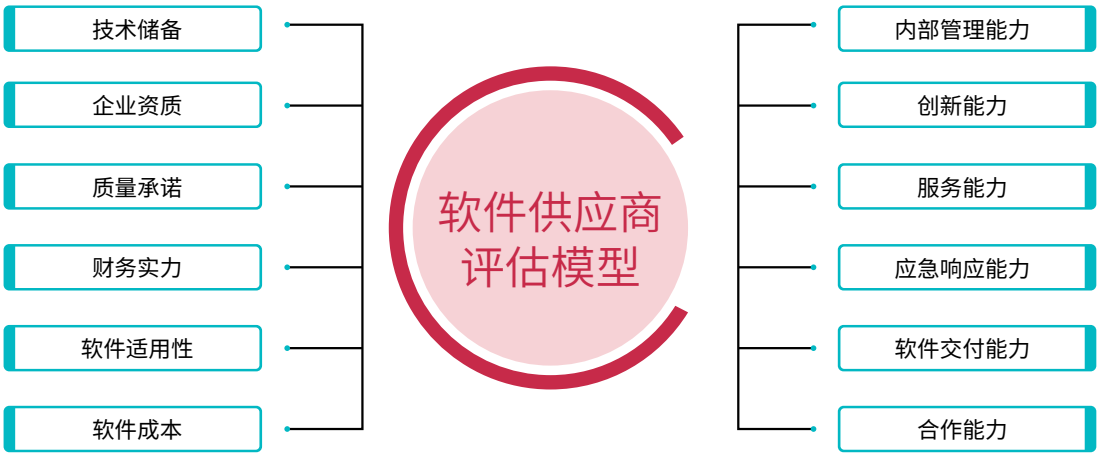

图 4-7 供应商风险评估模型

·技术储备：评估软件供应商是否拥有自主研发能力以及自主技术知识产权，对科技知识是否有进行不断的积累和及时更新，对企业提高技术水平、促进软件生产发展是否有开展一系列的技术研究；

·企业资质：评估软件供应链上的第三方供应商是否能够提供软件安全开发能力的企业级资质，是否具备国际、国家或行业的安全开发资质，在软件安全开发的过程管理、质量管理、配置管理、人员能力等方面是否具备一定的经验，具有把安全融入软件开发过程的能力；

·质量承诺：评估软件供应商的相关软件产品是否符合国家及行业标准要求，信息安全和数据保护控制流程必须遵守法律、监管或合同义务以及任何行业标准的信息安全要求；

·财务实力：评估软件供应商的财务能力以及稳定性，确保供应商具有稳定性和可靠性来提供业务所需要的服务；

·软件适用性：评估软件在开发部署以及动态运行时的适用性，是否可以持续满足新的需求；

·软件成本：评估软件供应商所提供的软件成本是否存在虚报等现象，审查产品及相关服务是否可以按照合理的价格交付；

·内部管理能力：评估软件供应商是否拥有完善的内部管理制度流程、有效的风险防范机制以及是否对员工定期进行安全培训等，对供应商内部安全开发标准和规范进行审查，要求其能够对开发软件的不同应用场景、不同架构设计、不同开发语言进行规范约束，审查软件供应商对其自身信息安全保密程度；

·软件适用性：评估软件在开发部署以及动态运行时的适用性，是否可以持续满足新的需求；

· 软件成本：评估软件供应商所提供的软件成本是否存在虚报等现象，审查产品及相关服务是否可以按照合理的价格交付；

· 内部管理能力：评估软件供应商是否拥有完善的内部管理制度流程、有效的风险防范机制以及是否对员工定期进行安全培训等，对供应商内部安全开发标准和规范进行审查，要求其能够对开发软件的不同应用场景、不同架构设计、不同开发语言进行规范约束，审查软件供应商对其自身信息安全保密程度；

· 创新能力：评估软件供应商的综合创新能力，包括技术创新能力、研究开发能力、产品创新能力以及生产创造力等；

· 服务能力：评估软件供应商的售前服务能力、培训服务能力以及售后维护服务能力是否满足企业的要求，在合作期间内是否可以始终如一的提供高水平的质量和服务；

· 应急响应能力：评估软件供应商从软件开发到运营阶段是否持续实行实时监控机制，是否有利用适当的网络和基于端点的控制来收集用户活动、异常、故障和事件的安全日志，是否具有适当的业务连续性和恢复能力，以防止或减轻业务中断和相关影响；

· 软件交付能力：评估软件供应商在整个软件及信息服务交付的过程中，是否能满足软件持续性交付的要求；

· 合作能力：评估软件供应商是否拥有高效的沟通渠道以及全面的解决方案，拥有共同的价值观和工作理念有助于建立长期的合作关系。

3. 软件供应商风险评估流程

进行软件供应商风险评估是任何成功的软件供应商风险管理计划中最关键的方面之一。软件供应商风险评估流程如下：

第一步：进行背景调查，以确保供应商能够制定并保持高质量的标准，而不会对公司及其客户造成任何风险。主要工作包括但不限于列出当前所有供应商并对其进行排名，记下每个供应商的主要工作、哪些供应商有权访问关键数据等信息，同时确认出最关键的软件供应商，评估每个供应商的意外风险是否会对公司造成重大中断，造成多大损失以及这种损失是否会影响客户，并且考虑修复这些损失可能需要花费的时间。

第二步：衡量供应商的可靠性和准确性。充分了解软件供应商所有可能产生的风险类型，如 IT 中断或故障风险、数据和隐私风险、合规风险、供应商更换风险等。并结合市场状况、企业的供应成本以及本报告提出的软件供应商风险评估模型等实际情况，制定相应的软件供应商风险评估标准，并量化其潜在风险，公式如下：

（风险因素）的可能性 X（风险因素）的影响成本 = 风险

如下图所示，我们使用风险评估矩阵模板来衡量潜在风险的严重程度和可能性，分为低、中、高和严重，为每个条件选择选项后，使用矩阵值评估每个风险的严重性级别。

风险矩阵

风险评级

低	中	高	严重
0-接受	1-勉强接受	2-不可接受	3-无法忍受
可以正常运行	在可控合理的范围内采取缓解措施	在规定时段内降低风险	暂停活动

严重性 →

可能性 ↓	可以接受 对结果几乎没有影响	勉强接受 有影响但不影响最终结果	不能接受 严重影响活动的过程和结果	无法忍受 可能导致灾难性后果
不可能 风险不大可能发生	低 —1—	中 —4—	中 —6—	高 —10—
可能性较小 可能会发生风险	低 —2—	中 —5—	高 —8—	严重 —11—
可能性较大 发生风险	中 —3—	高 —7—	高 —9—	严重 —12—

图 4-8 风险评估矩阵模板

第三步：根据风险评估标准，选择合适的软件供应商，并对其进行供应商深入评估，包括但不限于评估供应商如何在整个项目周期中管理数据文档计划，以及供应商在发生意外安全事件（如数据丢失、攻击事件、系统瘫痪）时的安全事件应急响应计划。

第四步：审查报告完成后，企业须与供应商管理层一起审查审计报告的结果，并允许供应商代表就审计中包含的绩效数据提供反馈，进一步建立供应商绩效目标和目的，以实现良好的合作关系。

总而言之，供应商评估是为了确保良好的合作关系，同时这也是一个适用于当前供应商自身评估的过程，用于衡量和监控其绩效，以降低成本，降低风险并持续优化和改进。

4.3.2 软件安全合规性评审

软件安全合规性评审是指针对软件自身的安全及合规进行管控，确保软件自身不存在安全及合规风险，包括提供软件物料清单（SBOM）、软件安全要求、软件合规要求、安全测试及评审报告和安全监控防护等。

SBOM 可以通过提高软件供应链透明度，以达到降低网络安全风险和总体成本的目的，目前已成为国际公认的重要方法论。Gartner 在《Innovation Insight for SBOMs》报告中提出，预计到 2025 年，60% 负责开发关键基础设施相关软件的组织将在其软件工程实践中强制使用标准化的 SBOM，比 2022 年（不到 20%）大幅上升。事实上，对于快速定位分析并解决新出现的漏洞和攻击来说，生成 SBOM 并快速地获取其信息已变得至关重要。政府和行业已经逐渐认识到 SBOM 对于软件网络安全的重要性。对于软件供应商来说，构建 SBOM 提升了软件供应链透明度，可以帮助企业组织全面洞察每个应用软件的组件情况，从而更高效地定位并响应漏洞问题。

此外，SBOM 的标准规范化有助于组织间的友好协作，进而提高客户信任度；对于采购方来说，可以清晰直观地看到其感兴趣的产品信息，例如基线组件信息或许可信息等。对于运营商来说，SBOM 增加了软件及其组件的可见性，有助于他们在其生产业务之前验证软件的状态。

4.3.3 软件资产管理

软件像其它 IT 硬件设备一样，作为企业的资产需要加以综合管理。软件资产管理旨在帮助企业降低成本并防范风险。软件资产管理指针对软件的供应链清单、软件及源代码版本、风险漏洞等信息进行统一管理。

软件供应链清单管理，指建立供应链清单信息，包括供应商相关信息、软件类型、SBOM、采购时间、可使用版本、许可期限等内容。软件及源代码版本管理，指针对交付的软件产品、源代码及其配置库信息等各个不同版本进行统一管理，自研代码与第三方组件也应要进行独立存放、目录隔离等操作。漏洞管理，指对软件供应链所涉及的软件产品及服务及其所包含的第三方开源组件中所有涵盖的漏洞进行统一的管理。

4.3.4 服务支持

服务支持是用来支持软件产品交付部署过程中的服务性活动，需供应商明确出软件的服务支持方式以及服务水平、安全服务协议等内容。服务支持方式大多以产品、用户文档以及培训咨询等方式展示，主要用于说明软件产品功能、对用户使用进行指导、系统异常问题的解决处理等相关信息；服务水平协议，主要用于明确软件交付部署方式、后续维保、新增需求的后续开发及服务支持方式等内容；信息安全服务协议，主要用于明确保密协议、知识产权归属及软件交付安全标准等相关信息。

4.3.5 安全应急响应

供应过程是软件产品或服务由供应商转移到软件用户的过程，这个过程带有传播性，若其过程发生捆绑下载、下载劫持、物流链劫持等攻击，风险传播影响也将是不可小觑的。因此，在该过程无论是企业内部还是供应商，建立成熟的应急响应机制是至关重要的。主要包括：1）应急预案，对于软件供应链安全事件应急响应机制进行说明，包括事件分类分级处理、处理时效、响应流程等内容；2）应急响应团队，明确软件供应链安全事件应急响应处理团队，明确人员职责及分工。

4.4 人员管理

软件供应链任一环节的干系人，如果出于方便等原因，引用未经审核的开源组件、下载非官方的软件、工具等，都有可能带来安全隐患。如 2015 年 9 月“XcodeGhost”恶意攻击事件，开发者为了方便从非官网渠道获取 Xcode 开发工具，致使开发出的多款知名 APP 受到污染，对平台下用户隐私的安全造成了巨大的威胁。由此，增强软件供应链上所有干系人的安全意识是至关重要的。

最小特权原则和零信任原则本身并不是控制措施，但在复杂和动态环境中，无论是供应商、管理人员、开发人员、测试人员、运营人员、还是终端用户，都可以为其业务流程提供一个灵活的框架。在人员管理中，根据不同的角色，授予其不同级别资源访问权限的身份和凭据，在企业组织中，为其所有员工定期进行相关培训，如帮助开发人员了解编码过程中的安全设计原则、安全开发标准等；帮助所有企业终端用户明确企业安全策略，包括有关下载应用程序、使用工作 / 个人设备、分享数据等的公司策略，以及帮助其了解如何识别网络钓鱼电子邮件、常见的凭据盗窃战术与当前的攻击方法（如可疑附件、虚假账单请求等）等，以提高其安全意识，养成良好的安全习惯。

1. 最小特权原则

最小特权原则是一种信息安全模式。在这种模式中，将仅向访问者提供在给定时间段内完成工作职能所需的最低级别的访问权限。此模式还可以应用到执行所需任务时需要访问权限的应用程序、系统或互联设备中。强制实施最小权限，仅在需要的时间提供所需的访问权限，可大大减少攻击面。

2. 零信任原则

零信任是一种安全模型，基于访问主体身份、网络环境、终端状态等尽可能多的信任要素对所有用户进行持续验证和动态授权，旨在通过严格的访问控制框架来保护现代业务基础架构。在当今如公共云、私有云、SaaS 应用程序等复杂的环境中，访问路径越来越多样化，边界变得越来越模糊，传统的基于边界防御的网络安全策略已无法应对频发的现代安全风险，如何授予访问权限是保护安全的关键。

零信任的理念十分简单：组织不得自动信任来自任何通信的任何服务、组件、访问请求，无论是在网络边界的“内部”还是“外部”。基于零信任的系统要求尝试连接的任何人和任何事物都必须先通过认证、身份验证和授权，然后才可授予访问权限。

4.5 软件开发生命周期安全风险治理

软件开发生命周期安全风险治理强调安全前置，通过组织管理手段及安全工具，在软件开发全生命周期的各个阶段（需求分析阶段、研发测试阶段、发布运营阶段、停用下线阶段）进行安全防护，从源头提升安全质量，实现开发运营安全闭环，保障软件应用全生命周期安全。

软件开发安全应从组织文化、安全开发流程、安全技术及工具等多角度、多维度考虑以构建出安全的软件产品。

组织文化上，建设安全文化，加强部门之间沟通，提高安全团队对基础设施和工具的可见性，以便安全团队在分配访问权限和选择安全解决方案时作出最佳选择，同时加强开发人员安全意识，实现人人都为安全负责。

安全开发流程上，实现软件开发、运营的全流程安全覆盖和安全风险管控，以确保软件开发全生命周期的安全。软件安全开发流程的优化促进跨团队的有效沟通和协作，采用统一的 IT 治理模型和可度量的安全指标，在软件安全开发流程中实现可跟踪性、可审计性和可视性，同时进行数据反馈，形成安全闭环，不断优化流程实践，以建立更安全的环境。

安全技术及工具上，结合组织安全管理工作，对软件全生命周期的每个阶段，即需求分析阶段、开发测试阶段、发布运营阶段和停用下线阶段，采取必要的安全考虑及相应的安全防护措施。安全技术和工具方面聚焦于保障和提升软件的安全技术能力，在软件开发过程中应尽可能地实现各个阶段安全工具自动化，支持 CI/CD 集成，从而实现持续交付的自动化和敏捷化，提升软件交付速率。

4.5.1 建设全流程安全开发管控

4.5.1.1 需求分析阶段

需求分析阶段考虑安全，实现安全左移，安全团队宣贯安全评估流程，提出安全质量要求，引导输出安全需求，帮助开发人员提高其安全意识和安全能力，以保障业务安全。主要内容包括：

（1）安全需求分析，包括安全合规需求以及安全功能需求；

（2）安全设计原则；

（3）确定安全标准，规范安全要求，满足业务机密性、完整性和可用性需求；

（4）攻击面分析，分析系统各个模块可能会受到的攻击；

（5）威胁建模，确定安全目标、确定威胁列表，进行漏洞储备；

（6）安全隐私需求设计知识库，构建安全需求知识分享平台，根据安全需求，得出安全设计解决方案。

4.5.1.2 开发测试阶段

为了尽可能避免软件上线前存在安全风险，需要在开发及测试过程中进行全面的代码安全审查，主要内容包括：

（1）安全编码，建立安全编码规范，帮助开发人员避免引入安全漏洞问题；

（2）管理开源及第三方组件安全风险，选用第三方组件时评估其风险级别，对第三方组件进行安全检查，提出解决风险方案；

（3）变更管理，对于变更操作进行统一管理；

（4）代码安全审查，制定安全审查方法和审核机制，确定代码安全审查工具；

（5）开源及第三方组件确认，确认第三方组件的安全性、一致性，根据许可证信息考虑法律风险，根据安全漏洞信息考虑安全风险；

（6）配置审计，制定配置审计机制，包括配置项与安全需求的一致性，配置项信息的完备性等；

（7）安全隐私测试，基于安全隐私需求设计测试用例并进行验证；

（8）漏洞扫描；

（9）模糊测试；

（10）渗透测试。

4.5.1.3 发布运营阶段

发布阶段确保可以交付安全的软件产品，主要内容包括：

（1）发布管理，制定相应的安全发布流程与规范，包括对发布操作的权限管控机制，发布流程的监控机制、告警机制等；

（2）安全性检查，进行安全漏洞扫描，以及校验数字签名的完整性等；

（3）事件响应计划，制定事件响应计划，包括安全事件应急响应流程，安全负责人与联系方式等。

运营阶段保障软件产品可以稳定运行。主要内容包括：

（1）安全监控，制定运营阶段安全监控机制，构建统一的安全监控平台，持续监控并上报；

（2）安全运营，定期进行常规安全检查与改进，如若发现潜在安全风险则应及时告警，根据漏洞信息、业务场景等智能化推荐安全解决方案，保证全生命周期安全；

（3）风险评估，制定和实施安全风险评估计划，定期进行安全测试与评估；

（4）应急响应，制定明确的应急事件响应流程，具备专门的应急响应安全团队，对于应急事件进行全流程跟踪、可视化展示、自动化处理、及时复盘形成知识库、量化风险指标；

（5）升级与变更管理，制定明确的升级与变更操作制度流程、权限管控机制、审批授权机制等，确保升级变更操作有明确的操作信息记录，与版本系统信息保持一致；

（6）服务与技术支持，有明确的服务与技术支持方式，对监管部门、运营商、用户等提出的问题进行反馈和及时响应，并对反馈问题进行系统性分类整理，确保安全问题的及时响应；

（7）运营反馈，定期收集运营过程中的安全问题，通过反馈平台统一收集、分类、全流程跟踪、解决、复盘等，以优化研发运营全流程。

4.5.1.4 停用下线阶段

软件生命周期是指软件的产生直到报废或停止使用的生命周期。由此软件开发生命周期安全风险治理需要考虑软件停用下线阶段的安全性，实现研发运营安全闭环。软件停用下线前需提供满足客户使用的更新软件，确保客户使用过程衔接。软件停用下线后需保护云服务用户的隐私与数据安全，具体包括制定服务下线方案与计划，明确隐私保护合规方案，确保数据留存符合最小化原则。

4.5.2 构建完善的开发运营安全工具链

安全工具链是实现全面安全保障能力的前提条件，是对软件开发运营的安全赋能，因为在软件开发流程中嵌入安全工具链既可以提升安全运营效率，也能进一步延伸安全管理能力。考虑到安全开发体系建设工作推动的便利性和效果的显著性，在对现有安全技术进行了分析和对比后，筛选出了适合进行实践落地的安全技术，包括但不限于软件成分分析 SCA 工具、交互式应用安全测试 IAST 工具、运行时自免疫 RASP 工具、容器安全工具、入侵与攻击模拟 BAS 工具。

4.5.2.1 软件成分分析 SCA

软件成分分析 SCA 技术是通过对二进制软件的组成部分进行识别、分析和追踪的技术。SCA 的目标是第三方基础组件 / 可执行程序 / 源代码等类型的以二进制形式存储的文件，包括但不限于源代码片段或 Package，可执行的二进制组件 / 程序，基础 lib，tar/tgz 压缩文件，镜像 / 镜像层，广义的软件构建过程等等。

其中制品扫描是重要的质量关卡，同时也是运营、开发过程重要的安全信息来源，当前绝大部分企业已经建立制品库，且对制品晋级管理关注度上升，然而仅 13.55% 的企业将所有交付制品纳入制品库，实现制品晋级管理，且具备完善的开源合规的制品管理。

SCA 可以生成完整的 SBOM，SBOM 作为制品成分清单，同时建立软件构成图谱，为后续分析提供基础，即分析开发人员所使用的各种源码、模块、框架和库，以识别和清点开源软件（OSS）的组件及其构成和依赖关系，并精准识别系统中存在的已知安全漏洞或者潜在的许可证授权问题，把这些安全风险排除在软件的发布上线之前，也适用于软件运行中的诊断分析。SCA 允许组织在整个软件供应链中对开源软件的使用进行安全风险管理，保护最终用户免受安全漏洞的影响，在保证组织能够利用开源软件带来的优势的同时，也能保持安全性和合规性。其检测流程如图 4-9 所示。

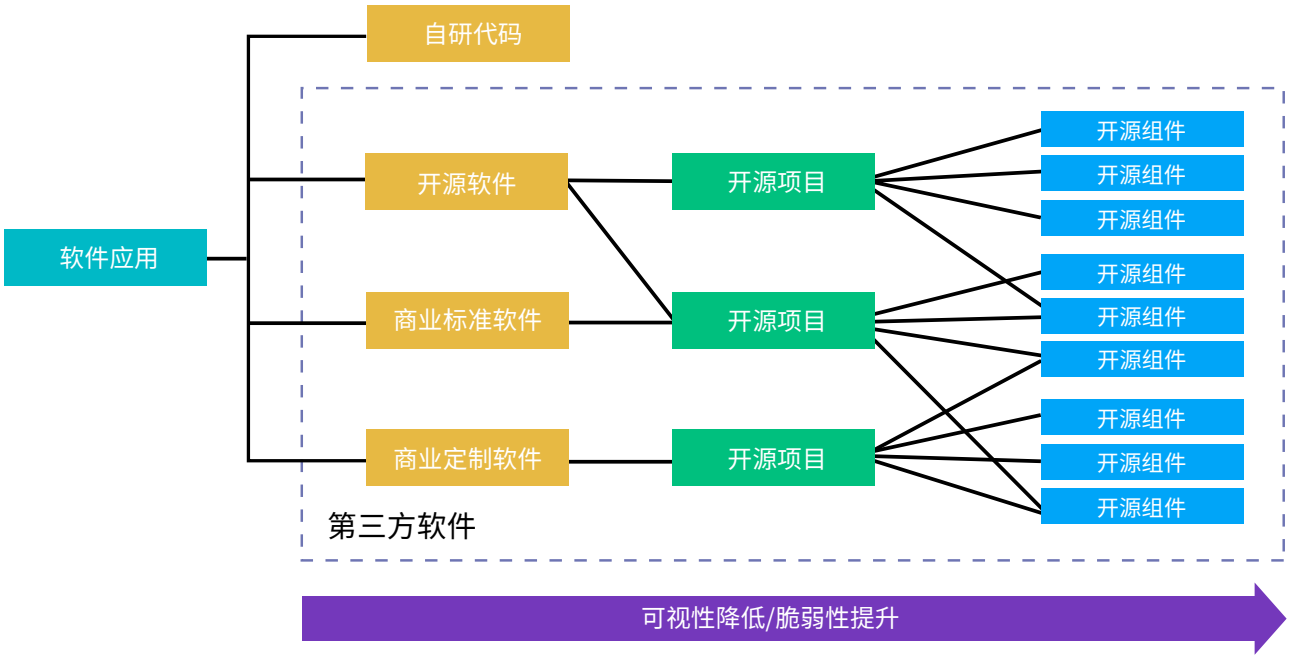

图 4-9 SCA 检测范围

随着开源软件的日益普及，SCA 的重要性逐年增长。SCA 工具也正在成为应用程序安全的必备工具，SCA 工具可以精准识别应用开发过程中，软件开发人员有意或违规引用的开源第三方组件，并通过对应用组成进行分析，多维度提取开源组件特征，计算组件指纹信息，深度挖掘组件中潜藏的各类安全漏洞及开源协议风险。同时，SCA 工具还可以对应用程序源代码、支持库、所有相关组件以及它们之间的间接和直接依赖项执行扫描，通过代码扫描可以及早发现漏洞和许可证合规问题并降低修复成本，允许自动扫描以更少的成本发现和修复安全漏洞。

通过使用基于多源 SCA 开源应用安全缺陷检测技术的安全审查工具，在整个软件供应链中对开源软件的使用进行安全风险管理，为组织提供具有可操作性的洞察力和详细的补救措施：

(1) 创建准确的软件物料清单（SBOM）：
SCA 工具生成软件物料清单（SBOM），其中包括已识别的源代码，然后针对包括 NVD 在内的多个数据库进行检查。SBOM 可帮助安全专业人员和开发人员更好的了解应用程序中使用的组件并深入了解潜在的安全和许可证问题，同时可以快速识别和修复任何关键的安全和法律问题。

(2) 发现并跟踪所有开源组件：
SCA 工具可以帮助组织跟踪当前运行的程序、源代码、构建依赖项、子组件等所依赖组件的使用情况，检测扫描代码中与知识库中跟踪的已知漏洞相对应的任何安全漏洞和依赖项中的安全漏洞。

(3) 制定和执行相关开源政策：
SCA 工具可以制定细粒度的策略来定义和自动执行关于开源软件组织可接受的安全性和合规性指南，实现快速响应各类安全事件。

(4) 启用主动和持续监控：

为了更好地管理工作负载并提高生产力，SCA 工具继续监控安全和漏洞问题，通过持续监控、扫描和来自强大漏洞数据库的最新 CVE 知识，实现更深入的洞察。同时对关键警报进行优先级排序和自动修复，以帮助开发人员在不中断工作流程的情况下快速解决问题。

(5) 无缝集成到构建环境中：

在 DevOps 环境中集成操作系统安全和许可证扫描，以便扫描代码并识别构建环境中的依赖性。

4.5.2.2 交互式应用安全测试 IAST

交互式应用安全测试 IAST 技术是近几年较为火热的一项应用安全测试新技术。IAST 最显著的特征就是通过运行时部署插桩 Agent 来收集安全信息，当外部扫描器发起扫描测试触发这个 Agent 时，IAST 便持续监测应用程序中的流量信息，一旦检测到漏洞，便实时提供报警并进行漏洞定位，被 Gartner 列为网络安全领域的 TOP 10 技术之一。

IAST 技术实现原理主要包括以下几种技术：

(1) 主动插桩技术

主动插桩技术需要在被测试应用程序中部署插桩探针，使用时需要外部扫描器去触发这个 Agent。一个组件产生恶意攻击流量，另一个组件在被测应用程序中监测应用程序的反应，由此来进行漏洞定位和降低误报，其主动插桩技术原理如下图所示。

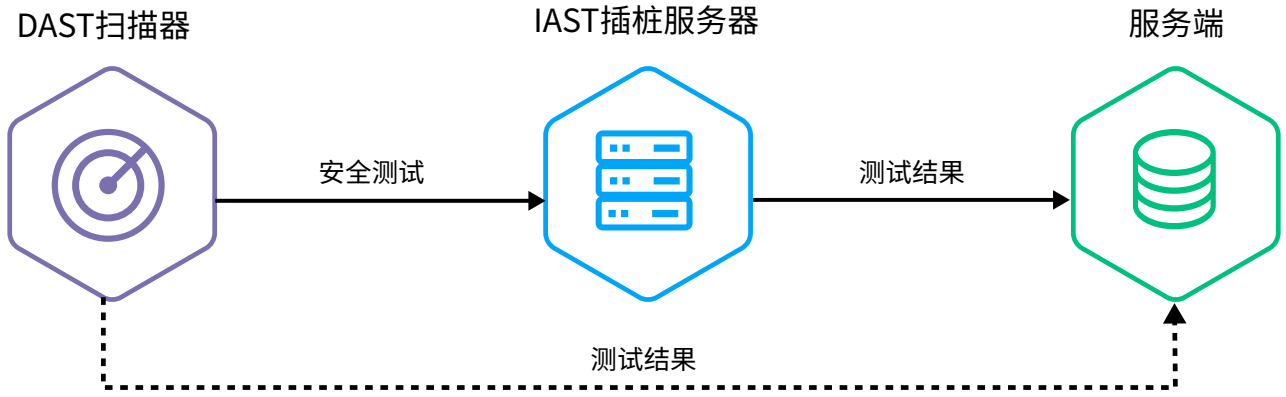

图 4-10 IAST 主动插桩技术原理

1) 被测试服务器中安装 IAST 插桩探针；
2) DAST Scanner 主动发起扫描测试；
3) IAST 插桩探针追踪被测试应用程序在扫描期间的反应，并动态分析测试覆盖率和上下文。当定位到具体漏洞信息时，将有关信息发送给管理控制台，控制台展示安全测试结果。

主动插桩技术结合了 DAST 的部分能力，需要在被测试应用程序中部署插桩探针进而在底层函数 Hook 点实时监听，并借助外部扫描器主动构造重放 Payload 数据来触发这个探针。其中，在整套主动插桩系统中，一个组件主动产生定向攻击重放流量，另一个组件在被测应用程序中监测应用程序的反应，由此来进行精准漏洞定位。

(2)被动插桩技术
1)被测试服务器中安装 IAST 插桩探针；
2)插桩探针在应用程序运行时获取请求和代码数据流、代码控制流,进行动态污点追踪；
3)当定位到具体漏洞信息,插桩探针将获取的信息发送给管理控制台,控制台展示应用安全测试结果。

被动插桩技术在程序运行时监视应用并分析代码,它不会主动对 Web 应用程序执行攻击,而是纯粹被动地分析检测代码。因此,这是一个巨大的技术优势,它将不会影响同时运行的其他测试活动,并且只需要业务测试(手动或自动)来触发安全测试,有正常测试流量过来就可以实时的进行漏洞检测,其被动插桩技术原理如图 4-11 所示。

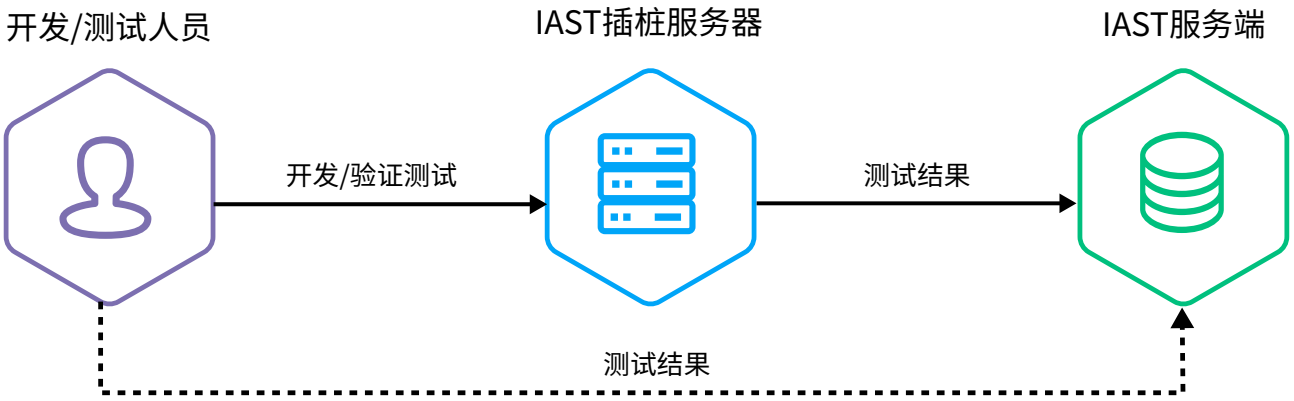

图 4-11 IAST 被动插桩技术原理

(3)终端流量代理技术
1)功能测试人员在浏览器或者 APP 中设置代理,将 IAST 检测平台地址填入；
2)功能测试人员开始功能测试,测试流量经过 IAST 检测平台,IAST 检测平台将流量复制一份,并且改造成安全测试的流量；
3)IAST 检测平台利用改造后的流量对被测业务发起安全测试,根据返回的数据包判断漏洞信息。

运行时插桩技术需要在服务器中部署 Agent,如图 4-12 所示,不同的语言不同的容器需要适配不同的 Agent,这对某些用户或一些用户的某些场景来说是难以接受的。而流量代理技术不需要服务器中部署 Agent,如图 4-13

所示，只需测试或使用人员简单配置代理即可，且能检测业务逻辑漏洞。该技术局限的地方在于安全测试会产生一定的脏数据，漏洞的详情无法定位到代码片段。当用户的某些业务场景无法部署插桩 Agent 时，流量代理和流量镜像技术是一个很好的补充。

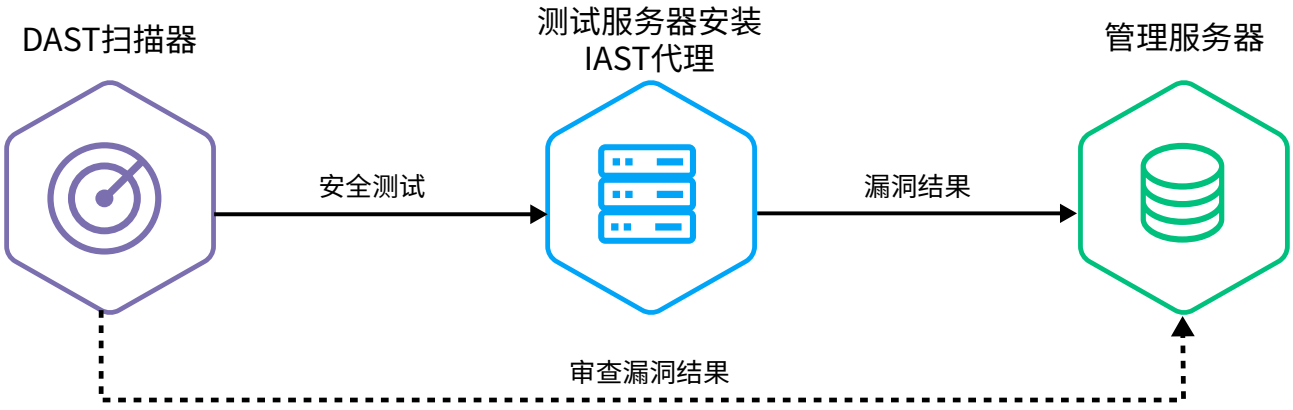

图 4-12 运行时插桩模式

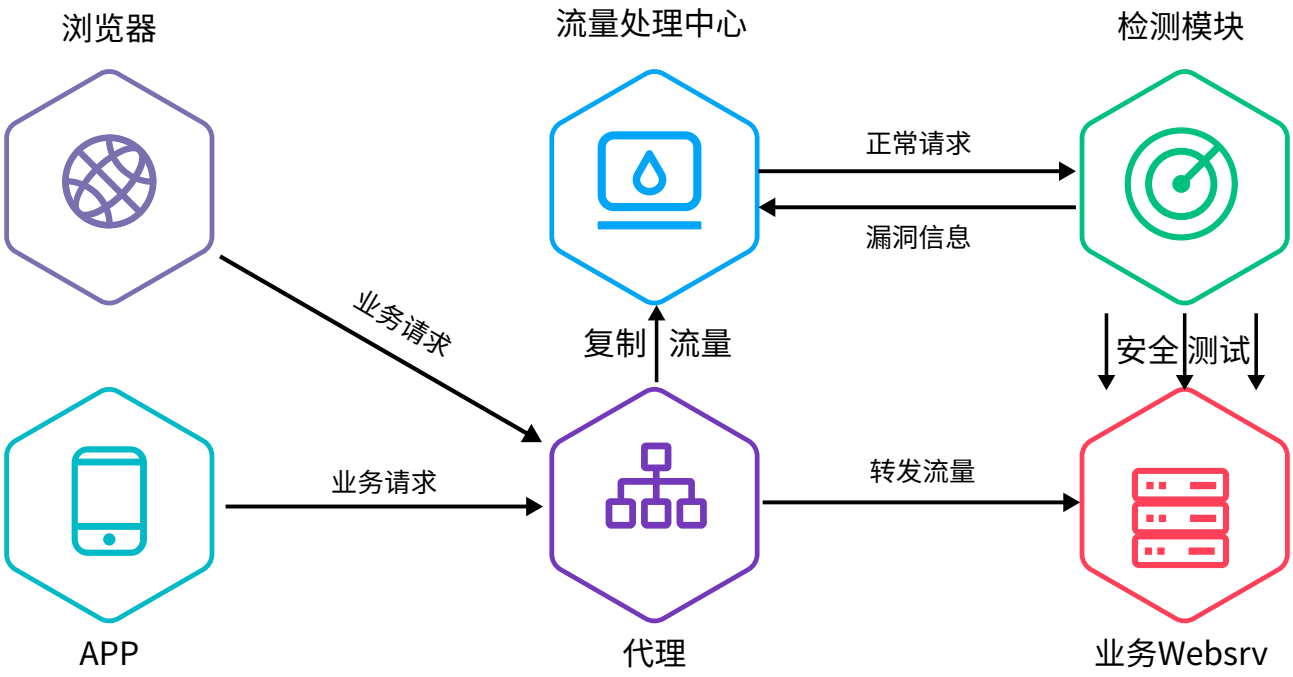

图 4-13 流量代理模式

4.5.2.3 RASP 运行时自免疫

随着针对应用程序的黑客攻击威胁的不断发展以及“HW”行动的兴起，仅靠 Web 应用程序防火墙（WAF）已不再是抵御威胁的有效方法。传统 WAF 通过在网络流量到达应用程序服务器之间对其进行分析，完全独立于

随着针对应用程序的黑客攻击威胁的不断发展以及“HW”行动的兴起，仅靠 Web 应用程序防火墙（WAF）已不再是抵御威胁的有效方法。传统 WAF 通过在网络流量到达应用程序服务器之间对其进行分析，完全独立于应用程序进行工作。在门外处理的方式，使其无法真正核实请求的合法性，难以保证其准确性，因此管理员只能使其处于“日志模式”。

为了更好地预防针对应用程序的攻击威胁以及应对“HW”等超大流量超大规模的实战验证，RASP（Runtime application self-protection，运行时应用自我保护）作为 Gartner 提出的一种新型 Web 防护手段，将保护代码注入到应用程序中，与应用程序融为一体，使应用程序具备自我保护能力，通过分析应用程序行为和上下文，对访问应用系统的每一段代码进行检测，保护实时运行的应用程序。通过使用该应用程序持续监控其自身行为，当应用程序遭受到实际攻击和伤害时，RASP 可以实时检测和阻断安全攻击，无需人工干涉。

市面上绝大多数 Java 版 RASP 技术都是在运行时阶段字节码加载前进行插桩，其优点是无需对源代码进行修改真正做到对应用程序的无侵入。其实现原理：Java 源程序在运行时需要通过编译器编译成为 .class 文件，即字节码文件。Java 的跨平台特性是 Java 字节码文件可以直接在 JVM（Java 虚拟机）运行，而 JVM 提供的 Java Agent 参数可以使得在字节码加载前修改字节码，完成 RASP Agent 的注入，当应用程序受到攻击时，就会触发 HOOK 函数，此时 RASP Agent 就可以获取到函数参数，实现对 Java 运行程序的检测。

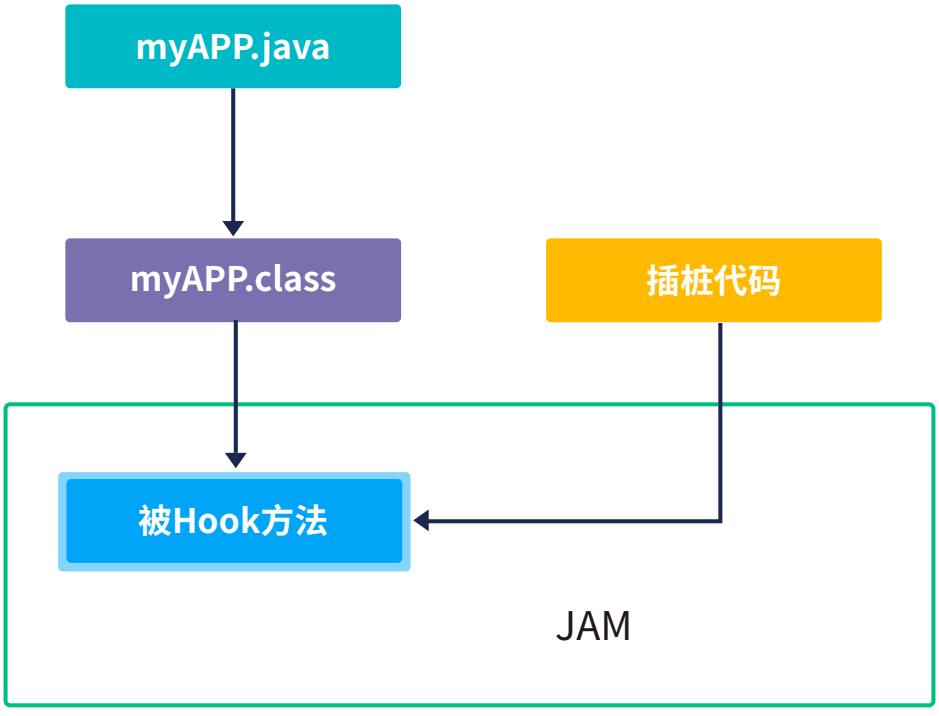

图 4-14 Java 代码中实现 RASP 注入防护

面对应用现代化，安全防护需要“左移”，推动安全战略实现从“传统基于边界防护的安全”向“面向应用现代化的内生安全”模式转变。RASP 作为降低应用风险的一项关键创新技术，弥补了传统边界安全防护产品的先天性防护不足，可应对无处不在的应用漏洞与网络威胁，为应用程序提供全生命周期的动态防护和内生主动安全免疫能力，其必将加快企业数字化转型，推动软件供应链的创新发展。

4.5.2.4 容器安全工具

随着云原生时代的到来，容器作为云原生的代表技术，简化了应用程序或服务及其所有依赖项的构建、封装和推进，为重要的应用程序和工作负载提供了灵活性和隔离，帮助开发人员实现了高效部署，然而容器的灵活性和实用性也带来了一定的安全风险。由于容器的相对不透明性，使得很难识别安全风险以及容器交替运行时的不断变化，也使得在 Kubernetes 阶段进行扫描时检测当前未使用的镜像或容器更具挑战性，因此在开发的早期阶段确保容器安全至关重要。

容器安全软件和工具在开发和运行时环境中为容器中的所有资源提供持续性保护，自动执行漏洞扫描，并通知开发人员和 IT 团队容器环境中可能存在的威胁。容器安全软件和工具不仅要保护承载容器的宿主机的安全、容器软件自身的安全、容器镜像安全、容器集群安全，同时还要监视生成管道的完整性以及容器内应用程序的基础设施层。图 4-15 为某容器安全工具架构。

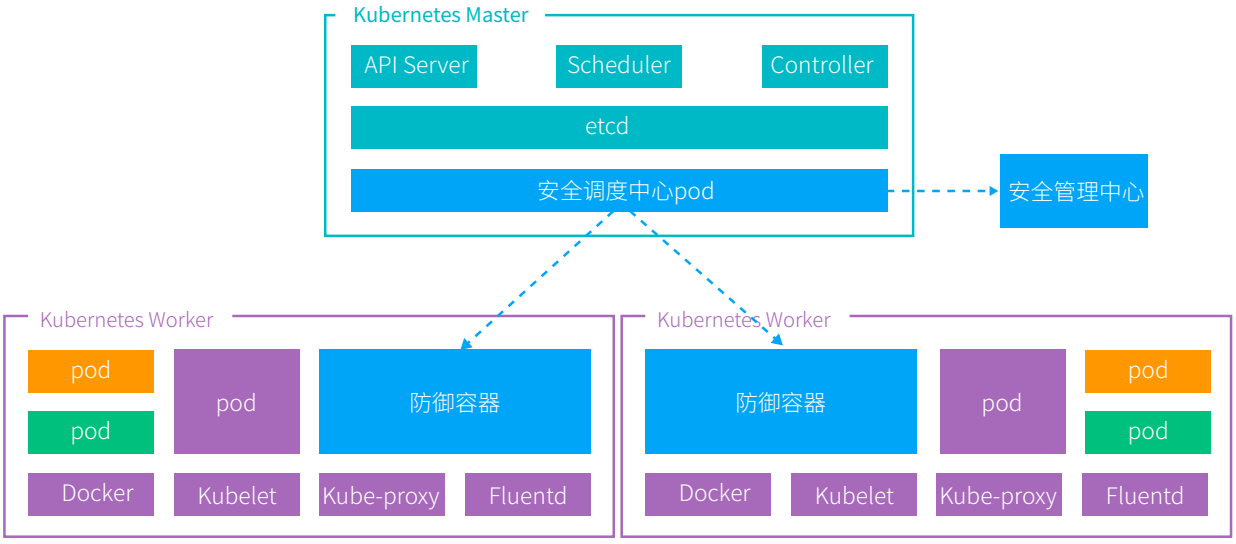

图 4-15 某容器安全工具架构

2020 年 Gartner 在《预测分析：全球容器管理（软件和服务）》报告中预测到，到 2024 年，所有应用中将有 15% 的应用运行在容器中，远高于目前的 5%；此外，将有 75%的大型企业会在生产中使用容器技术，而目前这一比例还不到 35%。在开发和部署阶段实现容器化的企业比例正在不断增加，但容器安全性已经落后，虽然有几种安全技术可以帮助企业改善其安全状况，但许多组织都没有全面的容器安全策略，企业在应用云原生技术时，应整体考虑容器安全，让安全与云原生相融合，以更好的保护应用系统。

4.5.2.5 入侵与攻击模拟技术 BAS

传统的安全威胁检测一般会利用漏洞扫描、渗透测试以及红蓝对抗等方法进行定期测试。然而，这些方法都有一定的局限性。如漏洞扫描报告通常只列出发现的漏洞，并且还可能产生误报并标记某些可能对安全性影响不大的问题。渗透测试和红蓝对抗是资源密集型活动，对安全专业人员的要求较高，且成本相对也比较大。时至今日，越来越多的企业开始采用入侵和攻击模拟 BAS 工具进行安全测试，以更高效的方式实现业务安全。

与传统安全验证方式相比，入侵和模拟攻击 BAS 具有以下几个优势：

· BAS 允许防御者采用攻击者的思维方式进行主动防御，而不是被动地等待扫描结果或发布补丁。

· 除了提高效率和降低成本之外，BAS 规避了人的因素，避免因人或团队不同带来不同的测试结果。

· BAS 模拟了一个完整的攻击，沿着网络杀伤链，以自动和连续的方式模拟大量的攻击，以查看它们是否预防和 / 或检测安全威胁，有效验证了安全措施。例如，BAS 会检查发起的攻击是否通过防火墙、IPS 甚至反病毒程序到达目标，而不是检查目标应用程序中是否存在给定的漏洞。

· BAS 可以进行持续测试，自动探测隐藏的安全漏洞，及时捕捉到新发现的安全漏洞。

· 加速了 DevOps 中持续部署和持续交付的速度。CI/CD 管道是一组流程，需要遵循这些流程才能成功地将应用程序的一组软件功能可靠且频繁地交付给不同的云环境，例如开发、QA 和生产。这些流程集的有效自动化（在软件工具的帮助下）将决定 CI/CD 管道的功效。CI 自动化涉及软件开发过程的每个单元，如设计、代码、构建和测试，并集成每个功能所需的适当软件插件，以便无缝执行。使用 BAS 可以缩短交付周期，让业务更安全的快速上线运行。

漏洞、缺陷等安全威胁并不能避免，且一直处于增长比较迅速的态势。在 DevOps 效率提升的前提下，如何保证安全也是防守者面临的重要课题。BAS 相对传统安全检测手段，具有独特优势，可以更好的保护业务系统安全运行，对企业来讲毋庸置疑是较好的选择。

4.6 软件安全成熟度模型

结合实际情况，企业在构建好软件供应链治理体系后，可以借助软件安全成熟度模型对其体系进行成熟度评估。软件安全成熟度模型作为一类基于规范或描述的模型，在一定程度上也可以度量企业构建的软件供应链安全治理体系成熟与否，起到了一定的指导性或辅助性作用。

4.6.1 可信研发运营安全能力成熟度模型

随着软件供应链安全风险加剧，实现主动式安全防御，构建覆盖软件应用服务全生命周期的安全体系框架势在必行。在此背景下，中国信通院牵头制定《可信研发运营安全能力成熟度模型》标准，强调安全左移，规范企业研发运营全生命周期安全体系，从源头提升软件质量，加固应用安全。

《可信研发运营安全能力成熟度模型》行业标准规定了面向云计算的可信研发运营安全能力成熟度模型参考框架，分为管理制度以及涉及软件应用服务全生命周期的要求阶段、安全需求分析阶段、设计阶段、研发阶段、验证阶段、发布阶段、运营阶段和下线阶段九大部分，每个部分提取了关键安全要素，规范了企业研发运营安全能力的成熟度水平，自低向高依次分为基础级、增强级和先进级。该标准适用于企业在落地实践研发运营安全时进行参考与评价，也可为第三方机构对于企业的研发运营安全能力审查和评估提供标准依据，如下图所示。

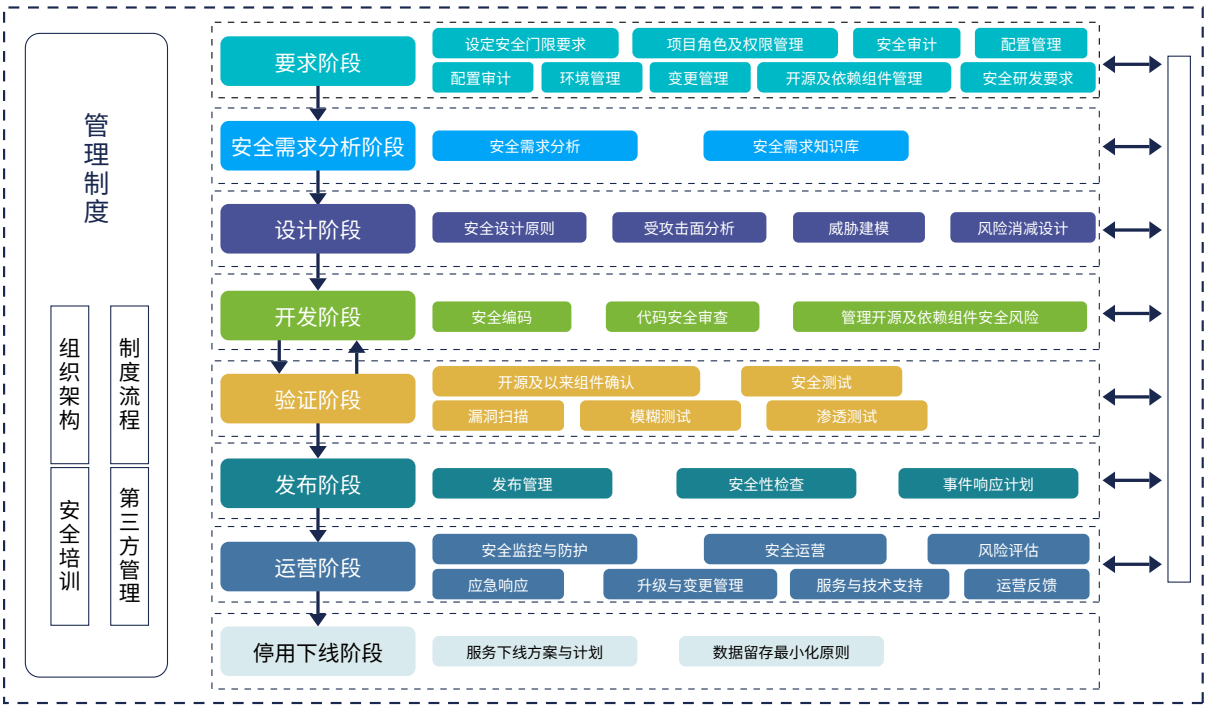

图 4-16 可信研发运营安全能力成熟度模型

数字化时代，随着企业 IT 基础架构云化及外部网络威胁环境持续恶化，原有的平衡正在被打破，云安全越来越受到企业重视。作为我国云计算安全信任体系的权威评估，可信云以完整的衡量标准和评估体系，为云安全的发展提供有力规范。中国信通院依据《可信研发运营安全能力成熟度模型》标准，于 2021 下半年开展了首批面向供应链研发运营安全能力的试点评估。

4.6.2 研发运营一体化（DevOps）能力成熟度模型

《研发运营一体化（DevOps）能力成熟度模型》是中国信息通信研究院牵头组织的全球首个 DevOps 标准。DevOps 标准体系旨在帮助组织：（1）明确概念、框架、能力。DevOps 的概念、框架和能力到达什么程度，做一个非常详细明确的说明。（2）对于 DevOps 涵盖的流程、组织、实施，有明确的指引，企业可以按照这个标准去提升自己 DevOps 各个环节的能力。

DevOps 标准评估体系主要包括敏捷开发管理、持续交付、技术运营、应用设计、安全及风险管理、系统和工具等部分，如下图所示。

<table>
<tr><td colspan="14">一、研发运营一体化（DevOps）过程</td></tr>
<tr><td colspan="3">敏捷开发管理</td><td colspan="6">持续交付</td><td colspan="5">技术运营</td></tr>
<tr><td>需求管理</td><td>计划管理</td><td>过程管理</td><td>配置管理</td><td>构建与持续集成</td><td>测试管理</td><td>部署与发布管理</td><td>环境管理</td><td>数据管理</td><td>度量与反馈</td><td>监控服务</td><td>数据服务</td><td>容量服务</td><td>连续性服务</td><td>运营反馈</td></tr>
<tr><td>需求收集</td><td>需求澄清和拆解</td><td>迭代管理</td><td>版本控制</td><td>构建实践</td><td>测试分级策略</td><td>部署与发布模式</td><td>环境供给方式</td><td>测试数据管理</td><td>度量指标</td><td>应用监控</td><td>数据收集能力</td><td>容量规划能力</td><td>高可用规划</td><td>业务知识管理</td></tr>
<tr><td>需求分析</td><td>故事与任务排期</td><td>迭代活动</td><td>版本可追踪性</td><td>持续集成</td><td>代码质量管理</td><td>持续部署流水线</td><td>环境一致性</td><td>数据变更管理</td><td>度量驱动改进</td><td>质量体系管理</td><td>数据处理能力</td><td>容量平台服务</td><td>质量体系管理</td><td>项目管理</td></tr>
<tr><td>需求与用例</td><td>计划变更</td><td>过程可视化及流动</td><td></td><td></td><td>测试自动化</td><td></td><td></td><td></td><td></td><td>事件响应及处置</td><td>数据告警能力</td><td>运营成本管理</td><td></td><td>业务连续性管理</td></tr>
<tr><td>需求验收</td><td></td><td>度量分析</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>监控平台</td><td></td><td></td><td></td><td>运营服务管理</td></tr>
<tr><td colspan="14">二、研发运营一体化（DevOps）安全架构</td></tr>
<tr><td colspan="14">三、研发运营一体化（DevOps）安全管理</td></tr>
<tr><td colspan="14">四、研发运营一体化（DevOps）组织结构</td></tr>
</table>

图 4-17 研发运营一体化（DevOps）能力成熟度模型

其中，安全管理在 DevOps 标准体系中至关重要，因为 DevOps 在一定程度上降低了安全性，也是很多企业落地 DevOps 遇到的最大阻力之一。相比于传统开发模型，该部分明确规定了 IT 软件或相关服务在采用 DevOps 开发模式下，将安全融入每个阶段过程，使得开发、安全、运营各部门可以更好地紧密合作。

安全管理分为四个板块，包括：控制通用风险、控制开发过程风险、控制交付过程风险、控制运营过程风险，如图 4-18 所示。

图 4-18 安全及风险管理分级技术要求

其中，在安全管理中：

（1）控制通用风险，在 DevOps 模式下，安全内建于开发、交付和运营过程中，其中通用风险覆盖三个过程中的共性安全要求，包括：组织建设和人员管理、安全工具链、基础设施管理、第三方管理、数据管理、度量与反馈改进；

（2）控制开发过程风险，从应用的开发过程开始实施安全风险管理工作，可以保障进入交付过程的代码是安全的，降低后续交付、运营中的安全风险，保障研发运营一体化的整体安全，包括：需求管理、设计管理和开发过程管理；

（3）控制交付过程风险，交付过程是指从代码提交到应用发布给用户使用，那么安全交付则是将安全嵌入到交付过程中，包括：配置管理、构建管理、测试管理、部署与发布管理；

（4）控制运营过程风险，技术运营过程是指应用发布给用户后的过程，将安全嵌入于运营过程中，通过监控、运营、响应、反馈等实现技术运营的安全风险闭环管理，包括：安全监控、运营安全、应急响应、运营反馈。综上所述，安全管理更加侧重考虑全流程端到端的全局安全管理及服务，包括人员的管理、工具、代码和第三方合作的安全，涵盖了开发过程、交付过程以及技术运营过程中的风险等，贯穿 DevOps 能力成熟度模型的整体过程，旨在安全风险可控的前提下，帮助企业有效提升 IT 效能，更快速且更准确地实现研发运营一体化。

4.6.3 BSIMM

BSIMM（Building Security In Maturity Model，软件安全构建成熟度模型）是一个基于软件安全框架（SSF），由十二个安全实践组成并针对多个软件公司的软件安全项目进行研究的模型，该模型对不同软件安全项目所采取的措施和实践进行量化，描述其共性和各自的特点。

BSIMM 研究始于 2008 年，出版了 BISMM 1，当时对来自软件厂商、技术公司和金融服务业界的 9 个顶尖软件安全项目进行了数据收集和分析，随着近年来的快速发展，BSIMM 已经逐渐覆盖了软件构建的整个过程。

2021 年 BSIMM12 发布，旨在帮助企业规划、执行、评估和完善其软件安全计划。BSIMM12 报告中提出了新的趋势：

（1）影响广泛的勒索软件和软件供应链中断促使人们更加关注软件安全；
（2）企业开始学习如何将风险转化为数据；
（3）增强的云安全功能；
（4）安全团队正在借调资源、人员和知识用于 DevSecOps 活动；
（5）从“左移”变成“无处不移”的趋势；
（6）软件物料清单活动增加了 367%。

当前最新的 BSIMM 12 的 SSF 共有 4 个领域，分别为管理、情报、SSDL 触点和部署，每个领域又各分为三项实践，共形成 12 项安全实践模块，如下图所示。

区域			
管理	情报	SSDL触点	部署
用于协助组织、管理和评估软件安全计划的实践。人员培养也是一项核心的管理实践。	用于在企业中汇集企业知识以及开展软件安全活动的实践。所汇集的这些知识既包括前瞻性的安全指导，也包括组织机构威胁建模。	与分析和保障特定软件开发工件（artifacts)及开发流程相关的实践。所有的软件安全方法论都包含这些实践。	与传统的网络安全及软件维护组织机构打交道的实践。接按键配置、维护和其他环境问题对软件安全有直接影响。
实践			
管理	情报	SSDL触点	部署
1. 战略和指标 (SM) 2. 合规与政策 (CP) 3. 培训 (T)	4. 攻击模型 (AM) 5. 安全功能和设计 (SFD) 6. 标准和要求 (SR)	7. 架构分析 (AA) 8. 代码审查 (CR) 9. 安全性测试 (ST)	10. 渗透测试 (AA) 11. 软件环境 (CR) 12. 配置管理和安全漏洞管理 (CMVM)

图 4-19 BSIMM12 的框架

05 软件物料清单 SBOM

软件供应链安全始于对关键环节的可见性，SBOM 做为软件供应链治理的核心技术之一，可以包含应用程序的多种关键信息，通过这些信息可以追溯软件的原始供应链，极大提高开发人员对软件安全风险的理解，帮助企业在网络安全风险分析、漏洞管理和应急响应过程中提高效率，在软件供应链安全治理中起到了重要作用。

5.1 SBOM 的重要性

SBOM（Software Bill of Material，软件物料清单）的概念源自制造业，其中物料清单 BOM 是用来详细说明产品中包含的所有项目的清单。例如在汽车行业，制造商为每辆车提供一份详细的物料清单，列出原始设备制造商制造的部件以及来自第三方供应商的部件。当发现有缺陷的部件时，汽车制造商可以准确地知道哪些车辆受到影响，进而通知车主维修或更换。

如今，绝大部分应用程序都是组装而成而非从零开始构建。通常软件供应商并不了解其产品具体使用了哪些开源软件，也无法确定产品是否使用了不安全或存在漏洞的软件组件，因此供应商无法在开发阶段及时规避这些风险，从而导致软件发布后产生的安全风险较大与修补成本较高。此外，当有漏洞被揭露时，很少有企业可以快速、准确地定位并响应一些关键性问题，例如，产品是否受到该漏洞的影响，该漏洞影响了哪些产品线，哪些软件的版本是否存在这些问题等。实际上这一情况的根源在于未管理产品的软件组成所造成的，从而大幅增加了开发、采购、维护与处理的成本。由此，企业需要维护准确、最新的 SBOM，以确保其代码质量高、合规且安全。

2021 年发生的几起包括 Codecov、Kaseya 和 Apache Log4j 等备受瞩目的软件供应链攻击事件，促使美国总统拜登发布了一项网络安全行政命令（EO），其中明确提出要增强美国联邦政府的软件供应链安全，要求向美国联邦政府出售软件的任何企业，不仅要提供软件本身，还必须提供软件物料清单（SBOM），以明确该软件的组成成分，提高软件供应链透明度，确保联邦政府使用的软件应用程序的安全性和完整性。

5.2 建立通用的 SBOM

美国国家电信和信息管理局（NTIA）发布的《构建软件组件透明度：建立通用软件物料清单（SBOM）》第二版中提出，SBOM 是一个包含软件组件列表和层次依赖信息且机器可读的规范性清单。这些清单应当是全面的，或者应当明确说明不全面之处。SBOM 可能包括开源软件或专有软件，它可以是公开可见或设置限制访问，SBOM 还应能够识别并列出软件组件、提供这些组件的相关信息以及它们之间的供应链关系。但特定 SBOM 中所包含的组件信息的数量和类型可能会根据行业或部门以及 SBOM 消费者需求等因素而有所不同。对于这一情况，关键是要形成一个基线 SBOM 的最低预期，只要求该基线包含可以支持基本组件及其特性所需的最少信息量和流程。在最低基线 SBOM 的基础上，随着各部门进一步完善和实践日益成熟，SBOM 可以逐步融入到各部门的开发流程中，成为企业与生俱来的 DNA 的一部分。

通用 SBOM 属性主要包括基线属性集、未确定的属性值、映射到现有的格式、组件关系以及附加元素，如图 5-1 所示。建立通用 SBOM 有助于我国软件安全领域实现 SBOM 标准统一化，进一步加快修复漏洞的速度，提升安全漏洞修复能力。

基线属性集	未确定的属性值	映射到现有的格式	组件关系	附加元素
·作者姓名 ·时间戳 ·供应商名称 ·组件名称 ·版本字符串 ·组件哈希值 ·唯一标识符 ·关系	·对于未确定的属性需明确确定义区分 ·无断言（即数据缺失） ·没有值（即该属性不适用于此特定的SBOM）	·基线属性信息映射到SPDX、CycloneDX和SWID等现有格式	·包含 ·包含于	·除了基线属性外，SBOM可能还需要额外的元素和组件属性以支持不同的用例。

图 5-1 通用 SBOM 属性

1. 基线属性集

SBOM 的基线属性集包括：

1) 作者姓名：指 SBOM 的作者，SBOM 的作者并不总是供应商，也就是说除了供应商，SBOM 还可以由作者创建；

2) 时间戳：指上次更新 SBOM 的日期和时间；

3) 供应商名称：指 SBOM 条目中组件供应商的名称或其他标识符；

4) 组件名称：指组件的名称或其他标识符。组件名称应该具备处理多个名称或别名的功能；

5) 版本字符串：指组件的版本，版本信息有助于进一步识别组件；

6) 组件哈希值：指组件的加密哈希值；

7) 唯一标识符：指帮助唯一标识组件的附加信息。唯一标识符可以通过与全局唯一层次结构或命名空间对比或参考已有的全局坐标系来生成。一些可用作唯一标识符的系统包括通用平台枚举（CPE）、包 URL（PURL）、通用唯一标识符（UUID）（也称为全局唯一标识符 [GUID]）、软件遗产 ID（SWHID）和组件哈希值等都可以作为有效的唯一标识符；

8) 关系：指 SBOM 组件之间的关联。组件关系是 SBOM 模型设计中固有的组成部分，关系类型有包含（includes）和包含于（include in），只要选定并贯彻一个方向，方向的选择不会影响使用 SBOM 模型。

2. 未确定的属性值

在某些情况下，某些属性可能不可用、没有意义，或对组件识别没有实质性贡献，其中一个重要原因是缺乏组件组成的第一手知识。当 SBOM 作者不是软件组件的供应商时，作者可能会缺乏生成某些属性所需的信息。另一个原因在于创建 SBOM（和组件）的时间点，大致为：预构建、构建或打包时以及构建后。例如，由（非供应商）作者在构建后执行的二进制软件组成分析可能会检测到组件，但不会提取二进制组件以生成哈希值。SBOM 必须恰当地处理缺失或不适用属性的情况。为此 NTIA 提出了一个小建议：总是提供所有的基线属性，但需明确定义区分“无断言（no assertation）”(即数据缺失) 和“没有值（no value）”(即该属性不适用于此特定的 SBOM)。或者，SBOM 格式可以允许缺少基线属性，并将缺失某几项属性视为默认值（例如“无断言”或“没有值”）。

3. 映射到现有的格式

SBOM 生成方式可以分为手动生成或使用自动化技术生成。手动生成方法可能适合较小的项目，通过表格或文件等形式手动列出所有软件组件，以及每个组件的版本、许可证、依赖项和任何其他相关信息，但为具有上千个直接和传递依赖项的项目手动生成 SBOM 清单是不现实的，且手动方法也容易出现人为错误，更新依赖项也容易忘记更新 SBOM 清单。而自动化技术则是将生成 SBOM 的自动化工具整合到持续集成和持续交付 (CI/CD) 管道中，以扫描所有软件项目，列出专有和开源软件组件以及相关属性，例如许可证和对第三方库的依赖关系，并根据需要更新其相关组件。

SPDX、CycloneDX 和 SWID 等 SBOM 标准的使用也帮助 SBOM 实现了其自动化支持，SPDX 是一个由 Linux 基金会运营的项目，旨在标准化组织共享和使用软件物料清单中的信息的方式，SPDX 主要捕获与来源、许可 CycloneDX 是一种轻量级的 SBOM 标准，设计用于应用安全环境和供应链组件分析。SWID 是一种标准化的 XML 格式，用于标识软件产品的组件并将其上下文化。如表 1(来自《现有 SBOM 格式调查》表 1) 所示，映射了 SPDX、CycloneDX 和 SWID 中的基线属性。

表 5-1 将基线组件信息映射到现有格式

属性	SPDX	CycloneDX	SWID
作者姓名	(2.8) Creator:	metadata/authors/author	<Entity> @role (tagCreator), @name
时间戳	(2.9) Created:	metadata/timestamp	<Meta>
供应商名称	(3.5) PackageSupplier:	Supplier publisher	<Entity> @role (softwareCreator/ publisher), @name
组件名称	(3.1) PackageName:	name	<softwareIdentity> @name
版本字符串	(3.3) PackageVersion:	version	<softwareIdentity> @ version
组件哈希值	(3.10) PackageChecksum: (3.9) PackageVerificat ionCode:	Hash “alg”	<Payload>/../<File> @ [hash-algorithm]:hash
唯一标识符	(2.5)SPDX Document Namespace (3.2) SPDXID:	bom/serialNumber component/bomref	<softwareIdentity>@tagID
关系	(7.1) Relationship: DESCRIBES CONTAINS	(Inherent in nestedassembly/ subassembly and/or dependency graphs)	<Link> @rel, @href

(4) 组件关系

开发 SBOM 的第一步是列举供应商直接列入主要组件中的一级组件。然而，为了有效推广 SBOM，SBOM 还需要捕获组件之间已知的嵌套供应链关系。物理组件的物料清单通常将这些关系描述为“多级 BOM”。虽然

SBOM 可能会支持多种类型的关系，但在“基线属性集”中描述的基线关系属性定义了一种单一类型的关系：包含（或包含于），上游组件（通常称为依赖项）包含在下游组件或组件中。其他类型的关系可能是必要的或是有用的，而现有的 SBOM 格式支持不同类型的关系。由此可以进一步细化“包含于”关系，例如明确下列三种情况的差异：

- 直接包含（未修改的）一个上游二进制组件；
- 通过链接或编译包含（未修改的）一个上游源代码组件；
- 选择一个上游的源代码组件，对其进行修改（新建分支），然后通过链接或编译将其包含在内。修改一个组件会创建一个新组件 (例如，一个分支)，修改者会成为该新组件的供应商。重要的是保留修改组件的修改信息并传递其修改记录。例如，SPDX 支持 GENERATED_FROM 和 DESCENDANT_OF 关系类型。虽然上游组件的信息通常是 SBOM 功能的一部分，但不使用组件的某些部分的情况也很常见。软件程序（组件）可能包含一个库（组件），但只调用库提供的一些函数。或者，组件的某些特性可能会在构建或打包期间被禁用。这在一些 SBOM 用例中很重要，特别是在漏洞管理用例中。如果一个漏洞影响到上游组件，则该漏洞可能会影响也可能不会影响下游组件。漏洞可利用性交换 (VEX) 旨在传达组件中漏洞的状态。

(5) 附加元素

除了基线属性外，SBOM 可能还需要附加元素和组件属性以支持不同的用例。并不是所有附加的元素或属性都支持每个用例，所需的具体信息取决于用例的具体情况，包括但不限于：

- 组件的使用寿命结束日期或技术支持结束日期；
- 体现组件实施或支持技术能力的信息；
- 对组件进行分组的机制，可能是通过生产线或已实现的技术进行分组。

5.3 SBOM 的生成

在成熟的体系下，SBOM 的生成可以通过软件生命周期每个阶段所使用的工具和任务流程化地完成，这些工具和任务包括知识产权审计、采购管理、许可证管理、代码扫描、版本控制系统、编译器、构建工具、CI/CD 工具、包管理器和版本库管理工具等，如图 5-2 所示。

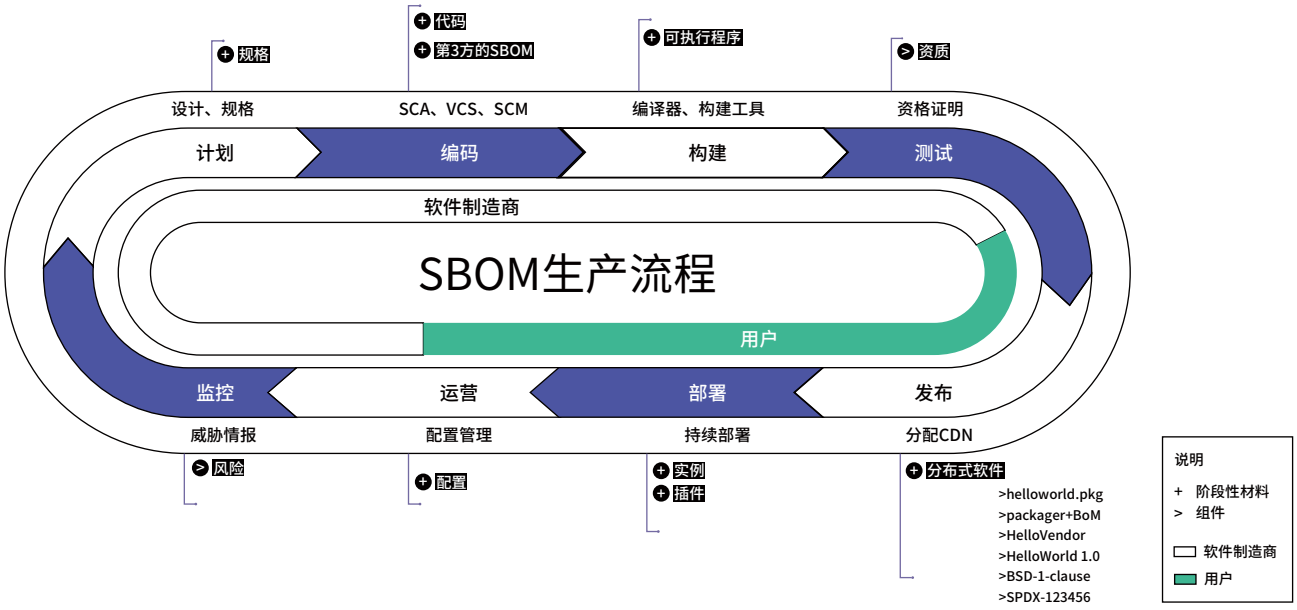

图 5-2 SBOM 的生成流程

SBOM 中应该包含软件组件与此组件所依赖的任何其他基础软件组件之间的关系。软件产品在发布任何版本时，SBOM 都应作为产品文档的一部分被提供，在 CI/CD 的标准实践中，SBOM 中包含的信息将不断更新。它从在需求中集成安全性需求开始，或者是 SBOM 中的一些元素已经在需求阶段被添加到用例中，这样安全性和 SBOM 就可以成为 DevOps 过程中标准和结构化的一部分。

软件开发阶段的不同，SBOM 的生成方法也是不同的，主要以软件构建为阶段划分，分为三个阶段：软件构建前、软件构建时和软件构建后。

软件构建前，SBOM 主要作为版本控制系统一部分，或由挖掘产品构建管道输入的工具创建源码级 SBOM，该阶段的生成的 SBOM 有助于关键组件的识别及在创建产品之前查找漏洞，也便于从构建到成为源文件的整个过程的追溯跟踪，但构建前的 SBOM 可能会有易受到攻击的源代码，只是它们并不会包含在最终可执行文件中。

软件构建中，SBOM 生成方法有三种：

1）作为构建制品自动化生成 SBOM，生成的 SBOM 通常不会出现人工输入错误，而且会包含更具权威性的软件组件特征，实现 SBOM 的签名自动化，这也为供应商和下游消费者提供了更多的可审计性。但该方法中 SBOM 供应商必须确定构建过程中将生成的 SBOM 格式，如 SPDX、CycloneDX 和 SWID 等。

2）在构建管道中通过构建插件生成 SBOM，这些插件与底层构建和依赖关系管理系统集成，生成一种支持的 SBOM 格式。这种方法很简单，但可能需要额外资源来集成所有构建管道。

3）从容器化过程中导出容器镜像 SBOM，随着越来越多的软件以容器镜像的形式交付，例如 Docker/Open Container Initiative (OCI) 格式，但值得注意的是容器镜像有分层的概念，每层可能都会包含各种软件应用程序，包括操作系统 (OS)、操作系统包、应用程序及其关联库，还有其他可能已集成到各层的制品。SBOM 描述容器镜像时，应该会汇总并标识来自所有层的所有软件。

软件构建后，并非所有交付软件的 SBOM 都能在构建时生成。许多软件功能，尤其是在老旧系统中使用的功能，一开始并不是用目前软件版本控制或者持续集成的方法开发的，构建后的 SBOM 可能是来自各种不同供应商、过程以及工具的 SBOM 集合。此类生成的 SBOM 应尽可能补充接近工程过程的组件数据、已知的未知情况等。通常情况下，一般应用 SCA 工具可扫描上游供应商组件以及构建后的 SBOM。

5.4 SBOM 的优势

如今，SBOM 已成为安全业界公认的遏制软件供应链风险的最佳方案之一，实施 SBOM 有助于揭示整个软件供应链中的漏洞与弱点，提高软件供应链的透明度，减轻软件供应链攻击的威胁，驱动软件供应链的安全。通过使用 SBOM 可以帮助企业进行漏洞管理、应急响应、资产管理、许可证和授权管理、知识产权管理、合规性管理、基线建立和配置管理等，如图 5-3 所示。

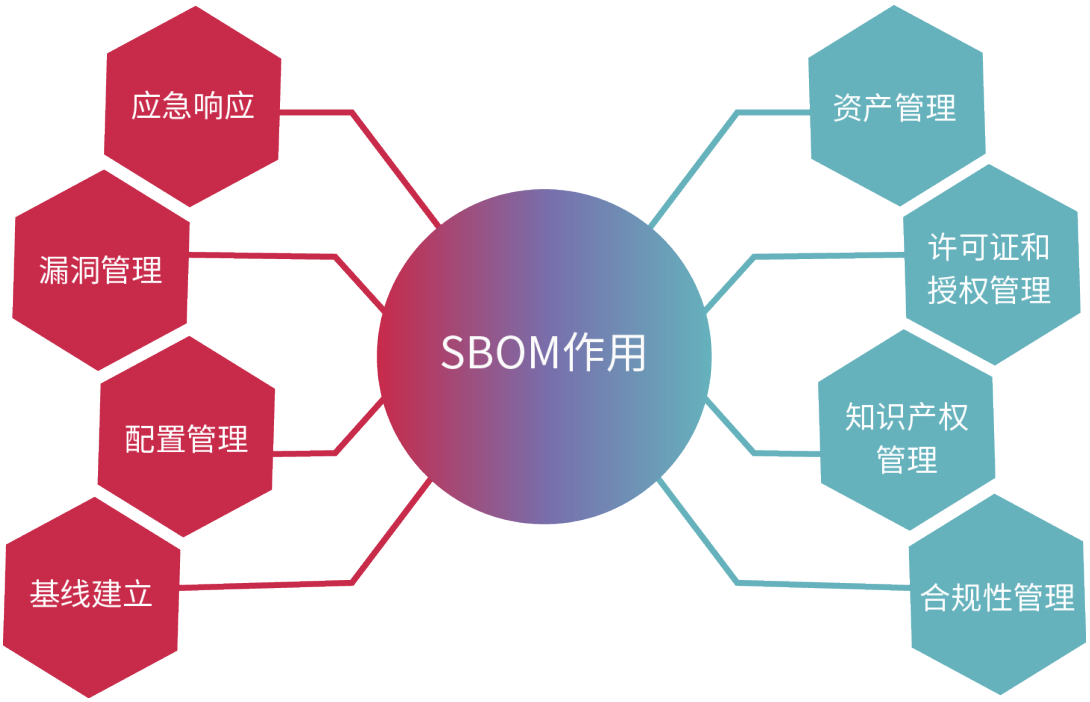

图 5-3 SBOM 的作用

SBOM 作为独立的实体，并非是所有安全问题的解决方案。但 SBOM 有助于获得做出正确选择的信息，是团队做出智慧安全决策的信息来源。

(1) 对于开发和运营团队来说，在软件开发过程中，开发人员可以使用 SBOM 监视和修复不同组件中存在的软件漏洞。例如，开发团队可能会收到有关 Ramda 漏洞的警报，了解这一点后，开发人员可以使用 SBOM 查找使用 Ramda 库的每个软件，并立即修补问题。同时，通过使用 SBOM，开发人员能够创建和实施允许列表和拒绝列表，从而能够更好地控制哪些组件和版本可在软件中使用，哪些组件和版本是被禁止的。在软件交付运营过程中，运营团队如果收到安全风险预警，并及时通知开发人员时，开发人员利用 SBOM 能够追踪溯源，发现存在漏洞的组件，及时进行漏洞修复或组件的替换等工作。

(2) 对于软件供应商来说，构建 SBOM 提升了软件供应链透明度，可以帮助企业组织全面洞察每个应用软件

的组件情况，从而更高效的定位并响应漏洞问题。同时，支持提供 SBOM 可以帮助供应商在竞争时，利用 SBOM 作为差异化因素，从而在市场中脱颖而出。

(3) 对于采购方来说，通过 SBOM 可以清晰直观地看到其感兴趣的产品信息，例如基线组件信息或许可信息等，也可以通过 SBOM 了解承包商 / 供应商管理系统、第三方风险的合规管理及报告等。

(4) 对于运营商来说，SBOM 增加了软件及其组件的可见性，有助于他们在其生产业务之前验证软件的状态。

SBOM 的出现为保护软件供应链奠定了重要基础，成为软件供应链的有利抓手，通过提供一组附加信息把软件组成成分和依赖关系以及作者等信息可视化并统一记录管理，大大提升了软件供应链整体透明度，对于降低软件使用和维护成本，保障软件供应链安全具有重要意义。当下，我们正在见证 SBOM 被认可和发挥作用的过程，未来，在 SBOM 逐渐实现标准化、服务产业化、生成功能集成化、生态化下，将会有越来越多的组织机构采用 SBOM 方法以满足行业内的更多需求。

06 开源威胁治理

Perforce 的 Open Source Initiative (OSI) 和 OpenLogic 联手开展了一项关于开源软件状态的全球调查，数据显示，在过去 12 个月中，77% 的受访者在其组织中增加了对开源软件的使用，36.5% 的受访者表示他们的使用量显著增加。

也有数据显示，有 90% 的组织都依赖于开源软件，而且开源软件也经常被嵌入到其他开源项目中。但是，尽管开源为企业的创新和快速发展提供了源动力，但与之而来的还有安全风险问题，为攻击者提供了潜在的安全威胁入口点。这一点在 3.2 小节为大家进行了概述。

在软件供应链的构建中，由于开源社区贡献者众多，当攻击者将恶意软件植入开源软件中，一旦开发中利用了这些开源软件，攻击者便会基于此对企业组织发起恶意攻击。此外，开源软件的引入还存在合规性问题，如若没有强意识做好前期的规划和预防，当违反开源许可协议时，从法律角度讲，都会给企业带来安全风险。针对众多开源威胁问题，亟需找寻合适的威胁治理工具，才能从根本解决此类安全威胁问题。

6.1 开源软件安全风险

通常情况下，常见的开源安全威胁有如下几种：

1. 漏洞利用

在 DevOps 快速发展的推动下，从公共存储库中提取和使用代码的开发人员可能会在不知不觉中将易受攻击、有风险、未经许可或过时的组件整合到他们的项目中。据 Synopsys 网络安全研究中心 (CyRC) 发布的 2020 年 OSSRA 报告显示，75% 的商业代码库包含开源安全漏洞，近一半包含高风险漏洞。由于开源的分布式特性，漏洞可能会在很长一段时间内未被检测到，但是攻击者可以在很长一段时间利用此漏洞。

例如前文提到的例子：2017 年信用报告机构 Equifax 的重大漏洞，其中暴露了 1.43 亿人的个人信息。事件发生的原因是攻击者注意到 Equifax 使用了一个开源 Apache Struts 框架的版本，该版本有一个高风险的漏洞，而攻击者利用了这些信息。

2. 许可证管理困难

单个专有应用程序通常由多个开源组件组成，其项目根据多种许可类型中的任何一种发布，例如 Apache 许可、GPL 或 MIT 许可。考虑到企业开发和发布软件的频率以及存在超过 200 种开源许可证类型的事实，这导致管理开源许可证变得困难。

组织必须遵守不同许可的所有单独条款，否则许可条款会使企业组织面临法律诉讼的风险，损害公司的财务安全。

3. 潜在的侵权问题

开源组件可能会引入知识产权侵权风险，因为这些项目缺乏标准的商业控制，从而为开源项目引入商业代码提供了可能。这种风险在 SCO Group 的真实案例中很明显，该案例声称 IBM 窃取了 UnixWare 源代码的一部分并将其用于他们的 Project Monterey 以寻求数十亿美元的赔偿。

4. 操作风险

在企业中使用开源组件时，其主要风险来源之一是运营效率低下。从操作的角度来看，主要关注的是未能跟踪开源组件并在新版本可用时更新这些组件。这些更新通常会解决高风险的安全漏洞，延迟可能会导致灾难，就像 Equifax 事件一样。

因此，对所有开发团队的开源使用情况进行清点至关重要，不仅要确保可见性和透明度，还要避免不同团队使用同一组件的不同版本。库管理需要成为开源使用专用策略的一部分，软件成分分析工具提供了一种以自动化、易于管理的方式实施这种做法的方法，而无需手动更新电子表格。

另一个问题是被遗弃的项目，这些项目可能从开源社区的有力支持逐渐走向消亡，直到没有人再更新它们。如果此类项目以库或框架的形式进入应用程序，开发人员有责任修复未来的漏洞。跟踪不经常更新的项目是良好的库管理的工作之一。

5. 开发人员的不当行为

一些安全风险是由于开发人员的不当行为而产生的，例如从开源库复制和粘贴代码。复制和粘贴是一个问题，首先是因为复制的项目代码中可能存在漏洞，其次是因为一旦将代码片段添加到代码库中，就无法跟踪和更新代码片段，从而使应用程序容易受到未来可能出现的漏洞。企业组织可以通过创建专门的禁止将代码段直接从项目复制中粘贴到应用程序代码库的开源策略来避免此问题。

另一个可能发生的不当行为是通过电子邮件或 IM 软件在团队之间手动传输开源组件。这与推荐的最佳实践相反，即使用存储库管理器来同步组件。

6.2 开源威胁治理技术

当今的软件开发环境下，开发人员往往采用开源软件以加快开发速度。然而随着开源组件的引入，开源组件漏洞、许可证法律风险问题都是潜在的安全威胁，如何做好安全态势检测，在软件开发生命周期的每个阶段做好跟踪成为必然。开源威胁治理技术中，软件成分分析（SCA）是其中最为重要的基础性核心工具，其提供了对集成到开发团队创建的软件中的开源组件和库的可见性，帮助管理安全性和许可证相关的风险，确保嵌入到应用程序中的任何开源组件都符合某些标准，以避免引入可能导致数据泄露、知识产权受损或法律纠纷的风险。随着开源使用的持续增长，显然需要 SCA 通过检测软件许可证、依赖项以及代码库中的已知漏洞和潜在漏洞来分析开源组件，使开发和安全团队能够管理其安全风险和许可证合规性。

2019 年，Gartner 在报告中把 SCA 纳入 AST 技术领域范围，从而形成了包含 SAST、DAST、IAST 和 SCA 的应用软件安全测试技术体系，并正式发布了有关 SCA 的技术洞察报告。报告中，对软件成分分析技术进行了准确定义：软件成分分析产品通常在开发过程中对应用程序进行分析，以检测开源软件组件是否带有已知的漏洞，例如具有可用安全补丁程序的过期库，以及需要相应授权许可（法律风险）的商业软件或第三方产品。

在开源威胁治理中，SCA 工具通常应用“清单、分析和控制”框架，让团队全面了解开源使用情况以便给出相应的解决方案。

清单

管理 OSS 漏洞的道路始于全面而准确的依赖项清单。毕竟，如果不了解其代码库中的所有组件，组织就无法解决漏洞或许可合规性问题。

分析

如果 SCA 工具确实检测到一个易受攻击的库，它会披露重要的上下文信息，如漏洞描述、受影响的库版本、CVSS 评分和严重性、CVE 和 CWE 标识、关系路径 (该攻击是如何引入代码的)。它还将突出显示任何许可证遵从性问题，并显示代码中发现许可证的位置。

控制

最后，SCA 解决方案通过提供补救指导帮助组织控制开源风险。这包括有关如何更好地升级有问题的 OSS 组件的信息。此外，用户可以轻松地为不同的组件实施拒绝 / 标记 / 批准策略，以确保最佳风险状态。

正因如此，SCA 技术在开源威胁治理中得到广泛应用，并大放异彩。

6.3 开源威胁治理的前提

通常情况下，开源威胁治理前应做好如下几点。

6.3.1 树立开源风险意识

近几年从国家层面针对开源也从政策法规的角度做了重要指示，明确了软件供应链安全建设、及对关键信息基础设施建设的重要性，将它提升到了新高度，在行业中也针对其出台了一系列标准，指出了从软件设计的不同阶段做好安全防范，包括组件漏洞、安全运营等不同维度都有了明确要求。这些都是从比较高的维度对开源 / 软件供应链安全做了明确的政策引导。反观之下，站在企业角度，在开源组件的引入和利用之前，从最初的软件规划设计阶段就应做好开源风险意识的认知，加强开源相关专业人才的培养、同时做好相关专业知识的培训，从本质上对其重视，让相关技术人员明确认知到开源风险，提升风险意识。

6.3.2 明确开源治理规范

对于企业而言，应该参照国家相关标准规范等，明确要求，同时结合自身制定内部的开源威胁治理规范，便于提前做好开源威胁治理，事先规避相关开源风险。推动内部形成共识，促进开源能力提升。

1. 漏洞

通常情况下，漏洞信息一般都会兼容 OWASP TOP10、国家信息安全漏洞库（CNNVD）、国家信息安全漏洞共享平台（CNVD）及 CWE 标准。当发现漏洞时可以对这些漏洞数据进行搜集，通过数据的清洗、关联和匹配，再结合企业形成的组件 SBOM 清单和引擎进行实时分析。

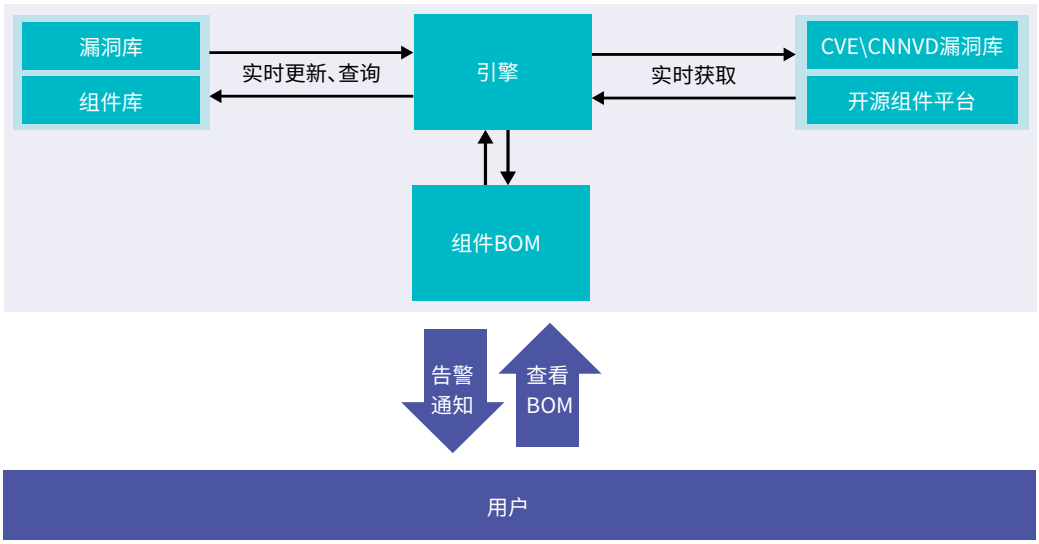

图 6-1 漏洞分析

2. 许可证

开源促进组织 OSI 颁发了上百个开源许可证协议。对这些开源协议进行风险梳理，尤其当引用了强传染性的协议时，要特别注意通过智能化的分析去找到相应的组件。当然通过组件也可以找到相应的许可证协议，这样就能够及时确定组件或者项目的知识产权风险。

3. 组件分析

组件分析需要支持开发阶段、CI/CD 阶段以及测试阶段，全流程对开源组件的检测，梳理并管控开源组件的信息，并从企业、部门、项目及任务等多角度分析组件的影响范围和依赖关系。基于多维度的依赖关系分析，通过自动化和智能化的工具协助企业形成 SBOM 清单，从而在追溯的过程中，能够快速地定位受影响的组件。

6.3.3 建立开源治理制度体系

企业需要构建自己的开源管理制度，从企业内部做好开源软件从最初的引入到最后的输出管理，形成开源管理的闭环，规范化地做好每个阶段的内容。

（1）建立开源软件的审查管理机制

无规矩不成方圆，企业和软件供应商应建立对开源软件的审查管理机制，严格执行对开源软件全生命周期的安全使用和风险控制。

- 建立针对开源软件的顶层审查管理机制。设立相应的组织机构，明确管理职责，结合企业的实际特点，制定审查管理制度和流程；

- 建立开源软件安全准入机制，制定评估要点，在其生命周期内持续评估其完整性、安全性和法律风险，实时掌握开源软件资产和与其关联的软件系统的安全风险状态；

- 借助于开源软件的自动化工具持续检测和发现开源软件资产，持续跟踪开源软件漏洞情报；

- 建立开源软件的漏洞缓解工作机制，研究开源软件的漏洞缓解措施，开展应急响应处置，及时规避网络安全风险；

- 建立开源软件的运营管理平台，按照审查管理制度流程，执行开源软件全生命周期的运营管理，跟踪并记录开源软件动态，提升开源软件治理效率。

（2）推进源代码安全保障实践方法落地

企业使用开源软件面临的首要问题仍然是代码的安全性问题。在软件源代码安全保障实践方面具有代表性的是微软提出的安全开发生命周期 (Security Development Lifecycle，SDL) 和 Gartner 基于敏捷开发推出的 DevSecOps 的方法。其中，SDL 将安全要素嵌入培训、需求分析、系统设计、编码实现、测试验证、发布和

响应等软件开发和运营的 7 个阶段。DevSecOps 通过设计一系列可集成的控制措施，增大安全监测、跟踪和分析力度，将安全融入敏捷过程中，并将安全能力赋予各个团队，同时保持“敏捷”和“协作”的初衷。

无论 SDL 还是 DevSecOps，都可以看到，安全的位置实际上都进行了左移，并且贯穿于所有的过程中。安全活动如下：

- 安全需求分析：结合业务实际需求，依据行业标准规范，如等级保护、分级保护，识别出软件系统的安全需求，作为后续安全设计的输入；

- 安全设计：依据安全需求分析文档，以减少攻击面为原则，围绕用户输入验证、异常处理、身份鉴别、密码管理、会话安全、访问控制、安全审计、数据脱敏、防 Web 攻击等方面对软件系统进行安全设计；

- 安全编码：使用指定的安全开发工具，严格依据安全设计文档对软件系统进行编码实现；

- 安全测试：借助于源代码审计、渗透测试和压力测试等综合测试方法，充分验证软件系统的安全性、可用性和完整性；

- 安全发布：结合实际网络拓扑，合理部署，减少来自网络的攻击面；

- 安全响应：制定网络安全事件响应计划和处置流程，有效响应网络安全事件。

（3）借助专业力量持续评估开源软件安全风险

企业应引入或借助专业的技术力量持续加强对开源软件的安全风险评估。通过常态化渗透测试、代码安全性分析或者建设网络靶场，持续、动态地验证、评估开源软件的可靠性、可用性和安全性，提升开源软件的安全应用能力。

（4）合作建立开源软件的漏洞情报库

现有的 CVE/CPE 体系以及 NVD 等漏洞库，已经无法满足开源软件安全治理的需求，需要创新开源软件的漏洞情报研究、共享、共治机制，合作建立开源软件漏洞情报库。

6.4 开源威胁治理阶段

开源治理工具还要与开源治理方案的三个阶段也就是安全开发、安全测试和安全管理相结合。

在安全开发阶段，首先，工具需要结合企业的 IDE，做到安全前置，帮助开发者快速定位漏洞并提供修复方案。其次，工具具备的企业级核心引擎要支持二次开发，因为每家企业的情况是不一样的，需要对接不同的工具和平台。第三，工具对于开发人员要友好，不能增加他们的工作量，还要给他们带来文化和价值的赋能，提升他们相应的能力。

在安全测试阶段，工具要具备对第三方开源组件的安全性测试功能和系统上线前开源组件的安全审查功能，帮助提高软件产品的安全性，防止应用带“病”上线。

在安全管理阶段，第一点是对第三方组件和供应商软件的安全准入，要制定相应的准入标准和评估体系，这与软件供应链安全也是相关的。第二点是在企业内部建立安全组件库，比如 maven 私服库等，好处在于有一个统一管理的组件引入来源。第三点是软件和组件的可视化清单分析。有了这样一个清单，就能够及时地进行相应的定位。最后一点就是要助力安全部门的合规审查及相关开源治理工作。我们会发现安全合规以及法律部门的介入，都会逐步推进整个开源治理体系的建设。

6.5 开源的 SCA 工具

为了减轻开源威胁带来的安全威胁，国内外安全企业研发设计了相应的治理工具。从使用成本分类，可分为开源（免费）的开源威胁治理工具和商业化的开源威胁治理工具。SCA 是开源威胁治理的核心工具，开源的 SCA 工具有 Snyk Open Source（Snyk）、OpenSCA（悬镜安全）、Veracode SCA（Veracode）、DependencyCheck（OWASP）等几个代表性的工具。每个开源威胁治理工具各有不同，各具特色。

6.5.1 Snyk Open Source

Snyk Open Source 允许在应用程序使用的开源库中查找和修复漏洞。它还允许查找和解决这些开源库中（或由这些库引起的）许可问题。该平台主动、无缝地查找和解决 Docker 映像和开源依赖项中的许可证违规和漏洞。

1. 产品优势

1. 查找漏洞

- 映射完整的应用程序依赖树；
- 检测所有开源依赖项中的薄弱环节；
- 利用 API、集成或 CLI 添加要测试的项目；
- 不断测试新发现的漏洞；
- 针对平台庞大的漏洞数据库检查依赖关系。

2. 报告（标准计划及以上）

- 可见性：在一个位置查看所有许可证问题和安全漏洞的状态，概览设计用于在大屏幕上显示；
- 问责制：查看团队解决问题的速度；
- 可审计：项目中使用的所有依赖项的清单，可以导出为 CSV。

3. 许可证（标准计划及以上）

• 审查合规性：获取项目中使用的许可证及其依赖项的清单；

• 保持合规：防止在发出 GitHub 拉取请求时使用有风险的许可证；

• 自定义策略：为企业制定定制的许可策略。定义特定许可证的严重性级别，并在项目使用有问题的许可证时收到警报。

4. 团体（专业计划及以上）

• 团队灵活性：为团队定义区域，以专注于与他们相关的项目；

• 超级强大的报告：了解公司中的薄弱环节状态；

• 快速过滤器：在报告中包含过滤器，以便快速访问重要数据。

2. 检测能力

Snyk Open Source 支持这些开发语言、构建工具和包管理器。

表 6-1 Snyk 检测能力

支持语言	包管理器或工具
.Net (C#, F#, Visual Basic)	Nuget, Paket
Bazel	See API docs.
C/C++	N/A
Elixir	hex
Go	Go Modules, dep, govendor
Java	Gradle, Maven

JavaScript	npm, yarn
PHP	Composer
Python	pip, Poetry, pipenv
Ruby	Bundler
Scala	sbt
Swift and Objective-C	CocoaPods

6.5.2 OpenSCA

OpenSCA 作为悬镜安全旗下源鉴 OSS 开源威胁管控平台的开源版本，继承了源鉴 OSS 的多源 SCA 开源应用安全缺陷检测等核心能力，通过软件成分分析、依赖分析、特征分析、引用识别、合规分析等方法，深度挖掘组件中潜藏的各类安全漏洞及开源协议风险，保障应用开源组件引入的安全。

(1) 产品优势

1. 丰富的语言支持，海量知识库支撑

- 支持主流编程语言的软件成分分析，如：Java、JavaScript、PHP、Ruby；
- 云平台实时的组件库 / 漏洞库 / 许可证库 / 特征库等海量知识库支撑。

2. 组件依赖解析，可视化 SBOM 分析

- 组件的直接依赖及间接依赖解析分析；

- 组件安全漏洞分析，可快速定位漏洞影响范围并及时修复；
- 透明化 SBOM（软件物料清单），助力快速梳理内部软件资产。

3. 许可合规分析，知识产权安全保障

- 支持主流许可证的检出；
- 分析开源许可证的合规性及兼容性风险。

4. 企业级核心引擎，更高检出更低误报

- 拥有企业级 SCA 核心检测引擎及分析引擎；
- 基于海量知识库，多源 SCA 开源应用安全缺陷检测等算法，对特征文件进行精准识别，提高组件的检出率。

(2) 检测能力

OpenSCA 现已支持以下编程语言相关的配置文件解析及对应的包管理器，后续会逐步支持更多编程语言，丰富相关配置文件的解析。

表 6-2 OpenSCA 检测能力

支持语言	包管理器	解析文件
Java	Maven	pom.xml
	Gradle	.gradle
PHP	Composer	composer. lock composer.json

Ruby	Gem	gemfile.lock gems.locked
Golang	Gomod	go.mod go.sum
Rust	Cargo	cargo.lock
Erlang	Rebar	rebar.lock
Python	pip	pipfile pipfile.lock setup.py

(3) 数据可视化

OpenSCA 具备 HTML 格式报告导出功能，将检测结果以可视化形式统计、展示，提升了检测报告的可读性，便于组件依赖清单及漏洞信息的管理维护。

报告结果概览：

1. 统计检出的组件、漏洞数量及占比情况。

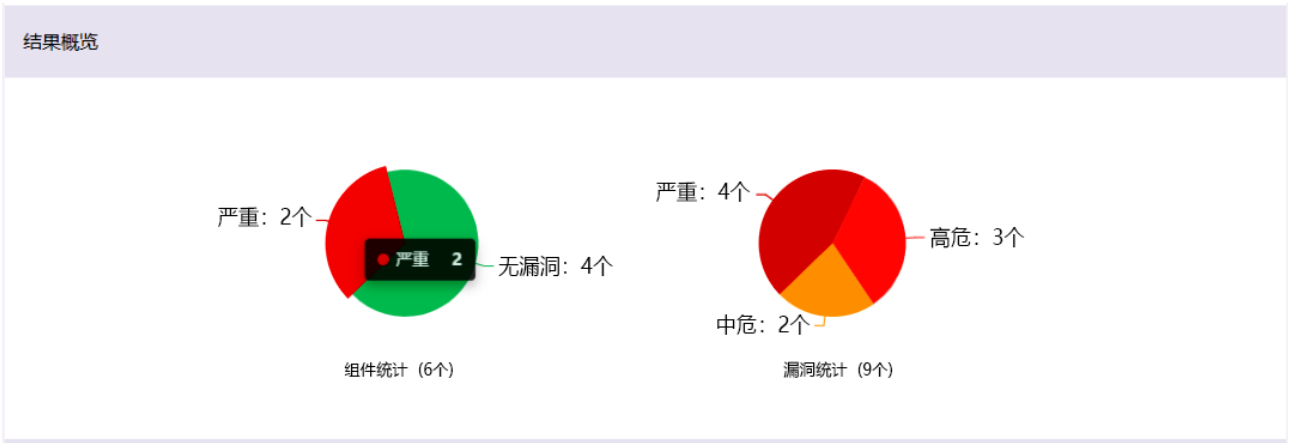

图 6-2 结果概览

2. 在检出的组件依赖列表，会明确展示组件的风险等级、检出的漏洞数、依赖方式等信息。

图 6-3 依赖列表

3. 点击 < 依赖列表 > 内的组件，可查看组件的详细信息，包括检出路径、组件漏洞信息等。

图 6-4 漏洞详情

(4) 应用场景

1. 安全开发

- OpenSCA 包含 IDE 开源风险检测插件，帮助个人 / 企业开发者快速定位并修复漏洞；
- 开发人员友好，轻量级低成本零门槛安装；
- 企业级 SCA 核心引擎，支持二次开发。

2. 安全测试

- 产品第三方开源组件的安全测试；
- 提高软件产品安全性，防止应用带病上线。

3. 安全管理

- 第三方组件及供应商软件的安全准入；
- 企业内部安全组件库的建立；
- 软件或组件资产可视化清单梳理；
- 安全部门合规审查及相关开源治理工作。

6.5.3 Veracode SCA

Veracode SCA 用于构建第三方组件清单，包括开源和商业代码，以识别漏洞。Veracode SCA 主要通过扫描编译应用程序中的库列表，然后识别每个库中的已知漏洞。同时，Veracode 也可以通知新公布的影响应用程序的漏洞，而无需执行新的扫描。Veracode SCA 支持两种扫描方法，可以在开发生命周期的不同阶段运行它们：扫描上传的应用程序和基于代理的扫描。

(1) 产品优势

1. 在开发环境中进行测试

可以直接从命令行启动扫描，以便在管道和 IDE 中快速反馈。在软件开发生命周期的早期查看并修复代码错误。

2. 修复时间短

自动修复功能规定了智能修复，生成自动拉取请求，并最大限度地减少中断，以获得更高的准确性和更快的修复率，修复时间从几小时缩短到了几分钟。

3. 自动化开源策略和治理

通过持续监控、广泛的分析和灵活的策略轻松管理开源使用。

(2) 检测能力

基于代理的扫描语言支持矩阵。

表 6-3 Veracode SCA 检测能力

支持语言	包管理器
Java	Maven
	Gradle
	Ant
	Jars
Scala	SBT
Kotlin	Maven
	Gradle
Go	Trash

	Glide
	GoVendor
	GoDep
	go get
	Go modules
	Dep
Python	pip
	Pipenv
JavaScript	NPM
	Yarn
	Bower
Objective-C	CocoaPods
Swift	CocoaPods
Ruby	Bundler
PHP	Composer
C/C++	Make
C#/.NET	NuGet
	DLL

6.5.4 Dependency-Check

Dependency-Check 是 OWASP 的一个实用开源程序的 SCA 工具，用于识别项目依赖项并检查是否存在任何已知的，公开披露的漏洞。执行依赖项检查时将收集到的有关依赖项的信息与下载到本地的 CPE&NPM 库数据进行对比，以此确认该依赖项中是否存在漏洞，如果检查发现扫描的组件存在已知的易受攻击的漏洞则标识出来，最后生成报告进行展示。

(1) 产品优势

Dependency-Check 支持面广（支持多种语言）、可集成性强，作为一款开源工具，在多年来的发展中已经支持和许多主流的软件进行集成，比如：命令行、Ant、Maven、Gradle、Jenkins、Sonar 等；具备使用方便，落地简单等优势。

(2) 检测能力

表 6-4 Dependency-Check 检测能力

支持语言	包管理器
Java	Maven、Ant、Gradle
.NET	
Python	
Ruby	
PHP	
Node.js	
Swift/OC	
JavaScript	

6.5.5 不同工具对比

通过对几个开源工具的调研，从多个细分维度进行统计、对比，结果如下表所示：

表 6-5 工具对比表

平台名称	Snyk Open Source	OpenSCA	Veracode SCA	Dependency-Check
发布时间	2019 年	2021 年 12 月	2021 年 10 月	2012 年
厂商	Snyk	悬镜安全	Veracode	OWASP
背景	英国	中国	美国	美国
官网地址	https://snyk.io/product/open-source-security-management/	https://opensca.xmirror.cn/	https://www.veracode.com/products/software-composition-analysis	https://owasp.org/www-project-dependency-check/
项目库地址	**Github:** https://github.com/snyk/cli	**Github：** https://github.com/XmirrorSecurity/OpenSCA-cli **Gitee:** https://gitee.com/XmirrorSecurity/OpenSCA-cli/releases	**Github:** https://github.com/marketplace/actions/veracode-software-composition-analysis	Github: https://github.com/jeremylong/DependencyCheck

产品介绍		Snyk Open Source 作为 Snyk SCA 产品的开源版本，允许在应用程序使用的开源库中查找和修复漏洞。其中还包括 Snyk 许可证合规性，以帮助管理开源许可证使用	OpenSCA 作为源鉴 OSS 开源威胁管控平台的开源版本，继承了源鉴 OSS 的多源 SCA 开源应用安全缺陷检测等核心能力，通过软件成分分析、依赖分析、特征分析、引用识别、合规分析等方法，深度挖掘组件中潜藏的各类安全漏洞及开源协议风险	Veracode SCA 用于构建第三方组件清单，包括开源和商业代码，以识别漏洞。在扫描时确定库和漏洞列表，同时也可以通知新公布的影响应用程序的漏洞，而无需执行新的扫描	Dependency-Check 是一种软件组成分析（SCA）工具，它试图检测项目依赖项中包含的公开披露的漏洞。它通过确定给定依赖项是否存在通用平台枚举（CPE）标识符来完成此操作。如果找到，它将生成一个报告，链接到相关的 CVE 条目
适用人群		安全、开发、测试	安全、开发、测试	开发、安全、测试	开发、安全、测试
检测能力	检测语言支持	JavaScript、PHP、python、.Net、C/C++、Elixir、Go、Java、Swift and objective-c、Ruby、Scala 等	Java、JavaScript、PHP、Ruby、Golang、Rust、Erlang、Python 等	Java、Scala、Kotlin、Go、Python、JavaScript、Objective-C 等	Java、.NET、Python、Ruby、PHP、Node.js、Swift/OC 等

检测能力	包管理器支持	Npm、yarn、Composer、pip、 Poetry、pipenv、Nuget、Paket、 Gradle、Maven、CocoaPods、Bundler 等	Maven、npm、Composer、Gradle、Gem、go mod、Cargo、Rebar、pip 等	Maven、Gradle、Ant、Bundler NPM、Composer、Pip、Gradle 等	Maven、Ant、Gradle 等
	软件成分分析	支持	支持	支持	支持
	软件特征分析	支持	支持	不详	不详
	软件直接依赖分析	支持	支持	支持	支持

检测能力	软件间接依赖分析	支持	支持	不支持	不支持
	实时检测	支持	支持	支持	支持
	容器镜像检测	支持	支持	支持	支持
	代码安全检测	支持	支持	支持	支持
检测场景	需求阶段	支持	支持	支持	支持
	设计阶段	支持	支持	支持	支持

检测场景	编码阶段	支持	支持	支持	支持
	测试阶段	支持	支持	支持	支持
	安全管理阶段	支持	支持	支持	支持
平台/工具链集成	IDE插件	支持	支持	支持	不支持
	代码仓库	支持	支持	不支持	支持
特征库及知识库	漏洞库兼容	CVE	NVD、CNNVD、CNVD	CVE	NVD
	组件库更新	不详	支持	不详	不详

特征库及知识库	漏洞库更新	支持	支持	不详	不详
	许可证库更新	支持	支持	支持	不详
	漏洞库查询	支持	不支持	不支持	不支持
	组件库查询	支持	不支持	不支持	不支持
	特征库规则库更新	不详	支持	不详	不详
	漏洞修复建议	支持	支持	支持	不支持

特征库及知识库	漏洞利用难度	支持	支持	不支持	不支持
	报告	支持	支持	支持	支持
总结	1. 各开源威胁治理工具都有丰富的语言及包管理器支持，可以更好地满足用户对工具的需求； 2. 每个工具都支持软件成分分析、实时检测、容器镜像检测、代码安全检测； 3. 每个工具都覆盖需求阶段、设计阶段、编码阶段、测试阶段、安全管理阶段的检测场景，可以更好地支持检测功能； 4. 每个工具都支持检测结果以不同的报告形式展现，可以更好地对安全态势分析； 5. Snyk Open Source、OpenSCA 支持软件特征分析，其它不详； 6. Snyk Open Source 漏洞库更新、许可证库更新；OpenSCA 支持组件库更新、漏洞库更新、许可证库更新、特征库规则库更新；Veracode SCA 支持漏洞库更新；其它不详。				

备注：由于某些信息未对外公布，以“不详”为结论展示。

6.6 商业化的 SCA 工具

在落地实践中，商业化的 SCA 工具更适合在大型企业的场景中使用。针对 SCA 的专项商业工具往往功能比较齐全，包含针对第三方组件的完整性、安全性和合规性检测的同时，能提供较多的接入方式，例如与组件库的对接，与代码仓库的对接，同当前开发流程的对接等等。国内代表厂商工具如：源鉴 OSS（悬镜安全）、开源卫士（奇安信）、FOSSCheck(棱镜七彩) 等；国内不管是源代码文件的 SCA 检测工具还是二进制文件的 SCA 检测工具，种类还是比较齐全的。国外代表厂商工具如：Black Duck（Synopsys）、Veracode SCA（Veracode）、CxSCA（Checkmarx）、Contrast SCA（Contrast Security）。

不管是国内还是国外的应用安全厂商，都有自己成熟的 AST 工具链、甚至还有平台和服务体系，也衍生了更丰富的工具布局和规划设计，以适应云原生、物联网、产业互联网等不同应用场景的需求。在开源治理中，企业应该选择具有如下能力的 SCA 工具：

1. 有丰富的语言支持和海量的知识库支撑。每一个企业都会用到很多语言，除了前端和后端，还包括随着市场需求变化引入的新技术和新框架。这就要求工具的检测能力要有非常大的覆盖面。知识库包括组件库、许可证库和特征库等，有大量的知识库支撑才能使分析结果比较完善。

2. 具备组件依赖解析和可视化 SBOM 分析的能力。从组件定位到相关部门和企业，从相关组件的漏洞定位到受影响的组件，这实际上是多维度关联分析。同时通过工具化的方式，生成可视化的 SBOM 清单去帮助企业快速梳理内部的资产。

3. 具备许可证分析能力。要能够做到对主流许可证的检出以及对合规性和兼容性的检测。

4. 拥有企业级的核心引擎。整个引擎能够独立出来，那么基于海量的知识库，对特殊的特征文件能进行精准识别，提高组件的检出率。如果知识库的体量不够大，不是企业级的检测和分析引擎，检出的结果一般是比较差的，误报率会比较高，同时给出的修复建议和方案也不能够满足企业需求。

5. 具有丰富的检测能力。对开源组件、二进制文件、第三方软件、漏洞、容器镜像等具有检测能力，可以针对未知威胁做好检测工作。

6. 拥有动态分析能力。随着技术的进步和发展，开源检测工具也需要有与时俱进的分析能力。比如传统静态分析技术包含基于依赖关系分析、基于文件级别的分析、基于代码片段的分析。由于使用者对结果精准度要求的逐渐提升，动态组件的实时分析能力至关重要。它是在测试环境，依赖 IAST 插桩技术原理，在用户进行功能测试和性能测试的同时，发现运行中相应组件的信息。这就能够避免数量巨大的误报和检出无法应用的漏洞。

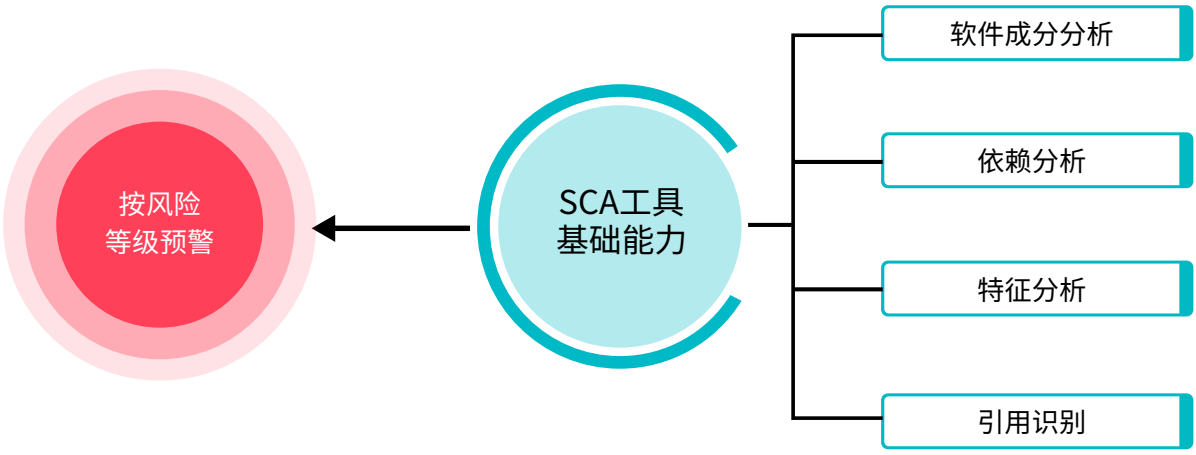

图 6-5 SCA 工具基础能力

SCA 工具选项众多，对于企业组织机构而言，面对众多选择往往会无所适从。第三方调研咨询机构根据多维度的评判，对 SCA 工具做过汇总式整理，可以作为企业组织选择工具的参考。

1. 云计算开源产业联盟

2022 年 7 月，由中国信通院发起成立的云计算开源产业联盟对外正式发布了《中国 DevOps 现状调查报告（2022）》。本次调查由中国信息通信研究院联合包括中国农业银行、中国工商银行软件开发中心、建信金科、招商银行、中信银行、华泰证券、百度、腾讯、华为云 DevCloud、中兴 RDCloud、京东、中国联通软件研究院在内的超 50 家企业共同发起。问卷以中国信息通信研究院牵头编制的《研发运营一体化（DevOps）能力成熟度模型》系列标准为参考，聚焦中国 DevOps 实践成熟度现状，对 DevOps 转型现状、未来 DevOps 的发展、企业对政策 / 资质的需求等情况进行了调查。

报告中对开源软件安全工具的使用情况做了详尽的调查和统计，展示出主流的开源软件安全工具市场使用率。如下图所示：

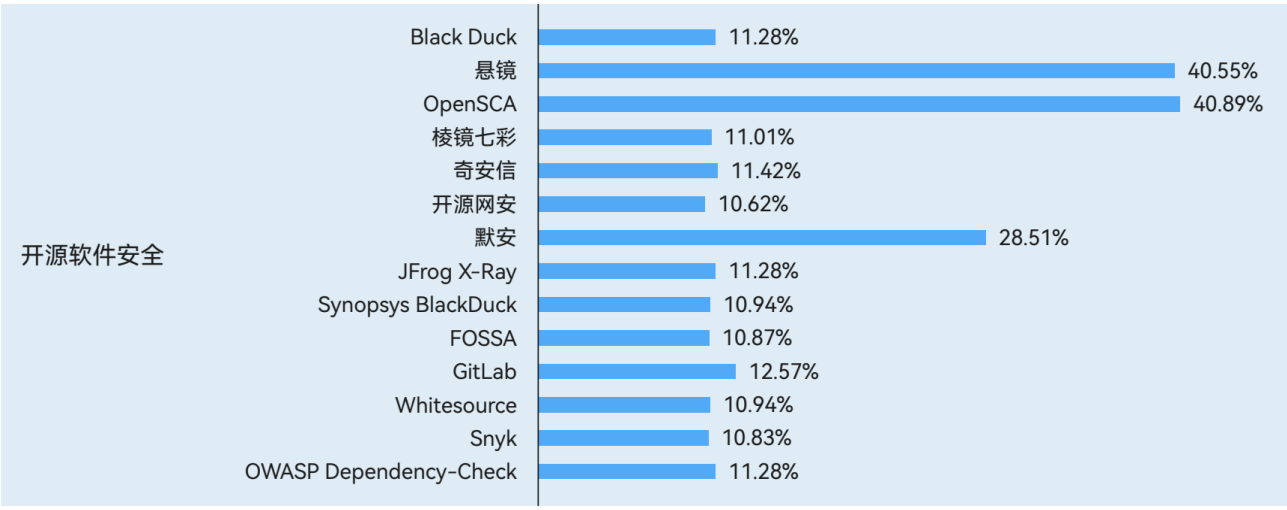

图 6-6 开源软件安全工具市场占比

2. 嘶吼安全产业研究院

2022 年 7 月 13 日，嘶吼安全产业研究院联合国家网络安全产业园区（通州园）正式发布《嘶吼 2022 网络安全产业图谱》。据悉，此次图谱是嘶吼安全产业研究院结合企业营收、客户类别、涉及领域、创新能力、投入情况等数据进行深度剖析，精制出品。可以说是在强有力数据支撑的前提下，为每一个细分领域进行分类，为进一步规划网安产业布局。这也是各个安全厂商工具产品能力的强有力体现。

图 6-7 嘶吼安全产业研究院软件组件安全分析与监测能力图谱

3. 数世咨询

国内数字化产业第三方调研与咨询机构数世咨询正式发布了《2022 年中国数字安全百强报告》，百强报告调研了国内 700 余家经营网络安全业务的企业，基于 2021 年度的上百项评价指标结合多种角度、不同维度的企业相关数据进行梳理和评价。基于此，根据安全领域进行划分，分类，形成了此图谱。在开发安全的分类中，对于能力者和创新者也做了展示，也是企业作为工具选型的强有力依据。

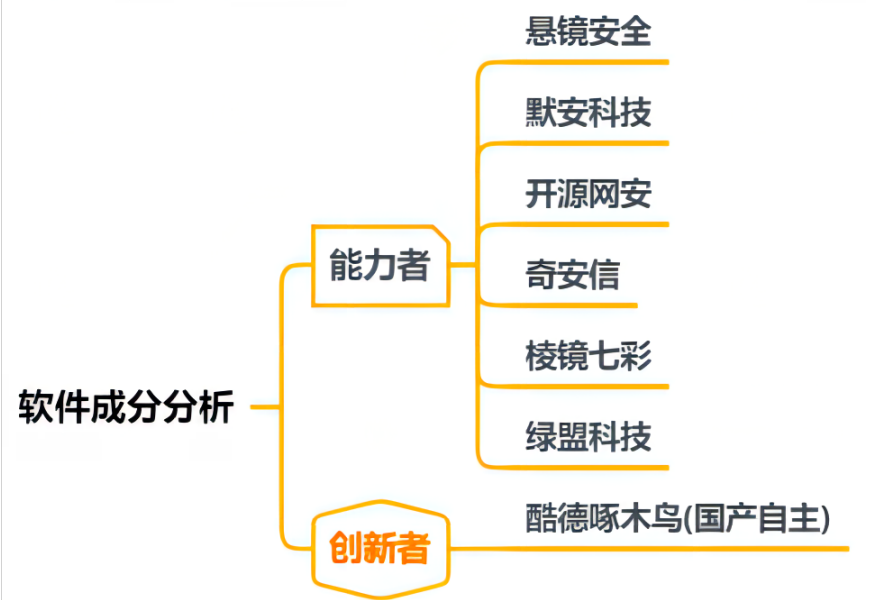

图 6-8 数世咨询软件成分分析能力图谱

4. 斯元商业咨询

2022 年，斯元商业咨询对外正式推出了《Emerging Technology Vendor Index·网安新兴赛道厂商速查指南》。该指南主要是助力企业安全负责人、渠道合作伙伴和安全从业者及时了解网安行业的新兴赛道及前沿产品，在项目产品选型时，高效检索细分赛道和代表性厂商。

图 6- 9 斯元商业咨询软件成分分析 SCA 分类厂商

07 软件供应链安全治理发展趋势

近年来，诸如 SolarWinds、MIMECAST、HAFNIUM 和 Log4j2 漏洞事件已经将软件供应链相关风险的现实问题摆在了人们的面前，让企业组织意识到了减少软件供应链安全风险的重要性。据数据显示，软件供应链攻击影响了 62% 的企业组织，可见对企业的危害之大。而且，DevSecOps、云原生、微服务等新技术应用，都为软件供应链的安全带来了新的安全威胁，软件供应链安全治理在未来会有以下趋势。

7.1 软件物料清单（SBOM）将得到更多实践

SBOM 是安全和合规最佳实践的基础。在美国总统拜登签署的“关于改善国家网络安全的行政命令”中，SBOM 是关键部分。根据《Anchore 2022 软件供应链安全报告》，尽管 SBOM 在提供对软件供应链的可见性方面发挥着基础性作用，但只有三分之一的组织遵循 SBOM 最佳实践，因此还有很大的实践增长空间。

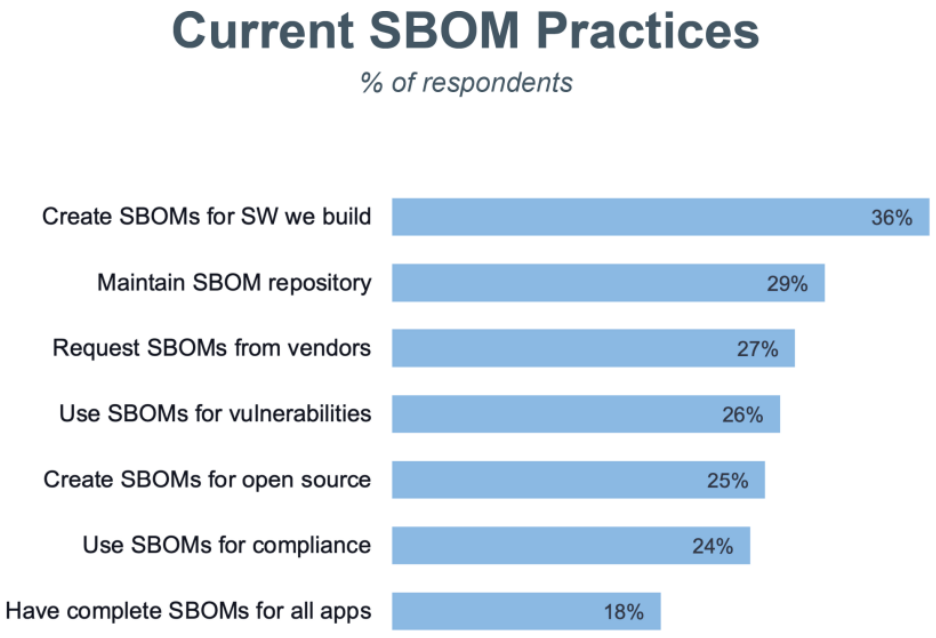

图 7-1 SBOM 实践使用率

当前，SBOM 在国内尚缺乏广泛的认知和了解，顶层的规范、标准与行业要求也未完成建立。造成当下 SBOM 落地较难、使用率较低的现状。但随着以中国信通院《软件物料清单建设总体框架》为代表的 SBOM 标准落地，以及 SBOM 价值的逐步体现，相信 SBOM 会得到越来越多的应用实践。

7.2 供应链和开源安全将成为容器安全中的新热点

据《Anchore 2022 软件供应链安全报告》显示，由于开发人员在他们构建的容器化应用程序中加入了大量的开源软件 (OSS)，在“最重要的容器安全挑战”的调查中，24% 的受访者将 OSS 容器的安全性列为第一大挑战，近一半 (45%) 受访者将其列为三大挑战之一。

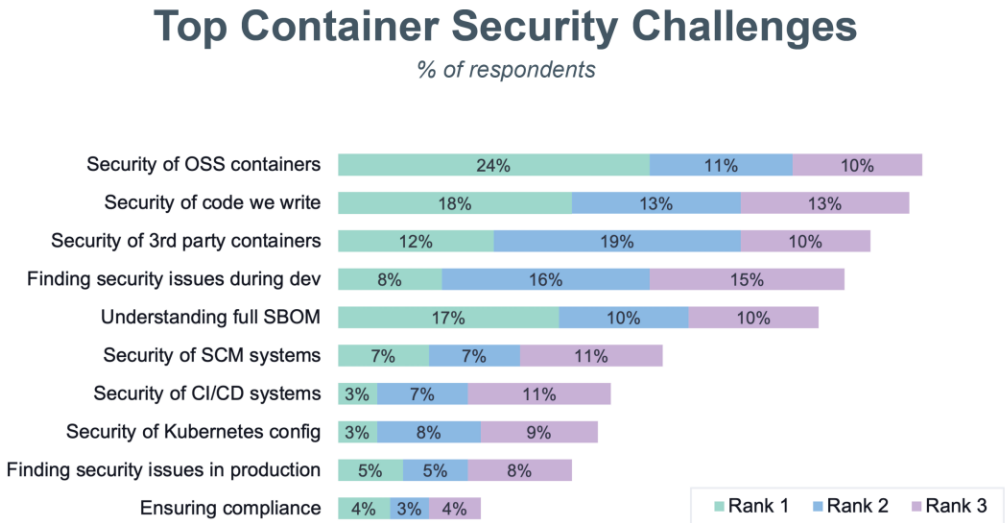

图 7-2 容器安全挑战

在“未来 18 个月优先容器工作计划”的调查中，“改善软件供应链安全”高居榜首，甚至超过了“提升容器使用率”这一底层诉求。

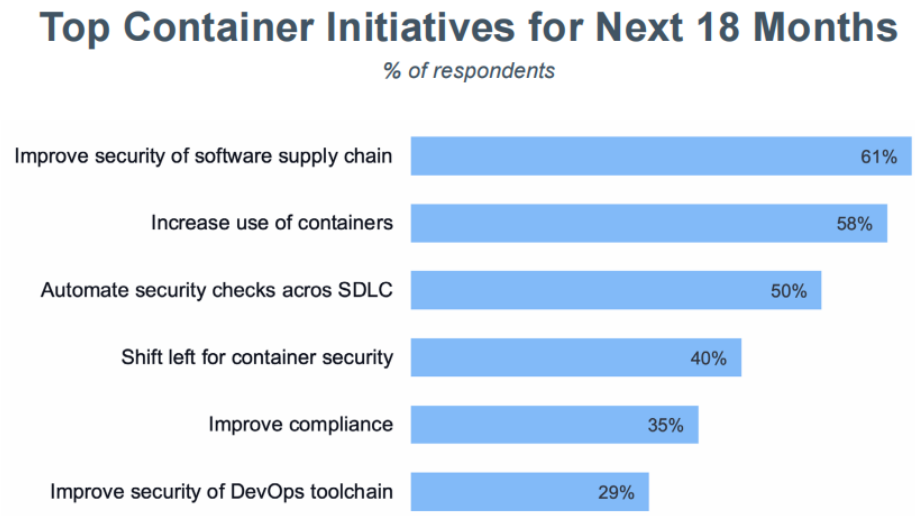

图 7-3 未来 18 个月优先容器工作计划调查

可见，未来随着容器化使用率的不断提升，开源和供应链安全面临的威胁也逐渐增多。做好容器中的开源和供应链安全是未来重点关注的方向。

7.3 开源许可证风险将获得高度关注

从 2019 年起，我国就成为了世界第一大专利申请国以及第一大论文发表国，且与第二名的差距逐年拉大。这意味着我国在各行业核心技术上，从以学习借鉴为主的跟随者变成了以创新创造为主的引领者。在这种态势下，强化对知识产权的重视和尊重是必然结果。

2021 年，GPL3.0 许可证侵权案的首例判决，法院认定被告方因使用了原告方的系统代码（该代码后续开源版本中已将“适用 GPL3.0 协议”的表述删除），因此违反了 GPL3.0 协议导致 GPL3.0 协议自动解除，被告失去了 GPL3.0 协议下的源代码授权保护，进而构成侵权，且侵权事实成立，最终被判罚 50 万元。

图 7-4 GPL3.0 许可证侵权案的判决书

本次判例，让开源许可证风险真正走到了开源生态相关方的面前。为避免侵犯他人知识产权，引起法律纠纷，开源许可证管理将成为企业在未来开源软件使用中必不可少的工作。

7.4 RASP 会成为软件供应链安全运营的核心工具

随着第三方组件应用率的提升，组件漏洞的影响愈发广泛，单一组件的漏洞可能会影响成千上万的用户和设备，诸如 Log4J2 漏洞（CVE-2021-44228）、Spring 框架漏洞（CVE-2022-22965）等严重级软件基础设施漏洞的爆发事件引发了社会广泛的关注和热烈讨论。随之而来的问题是，在业务系统的安全运营过程中，要如何降低不断爆发的组件漏洞所造成的风险。

传统安全运营思路更多关注修补过程，而对缓解程序重视不足。在严重漏洞爆发时，在漏洞的公布到官网修复补丁发布这段真空期，安全工程师通常采用下线业务系统这种简单粗暴的方式避免漏洞被攻击者利用，但这种方式会让正常业务也无法运行，这在金融、能源、政府等民生类业务系统上是难以接受的。

RASP 可以有效地解决这一窘境，RASP 运行在应用内部的特性，使之拥有函数级的威胁拦截能力。RASP 可以做到只拦截漏洞相关的风险函数，对业务系统的运行造成最小的影响。从使用场景看，RASP 解决了传统安全运营模式在组件漏洞爆发后的缓解期内缺乏精准性手段的问题，之后会成为软件供应链安全运营的基础性工具。

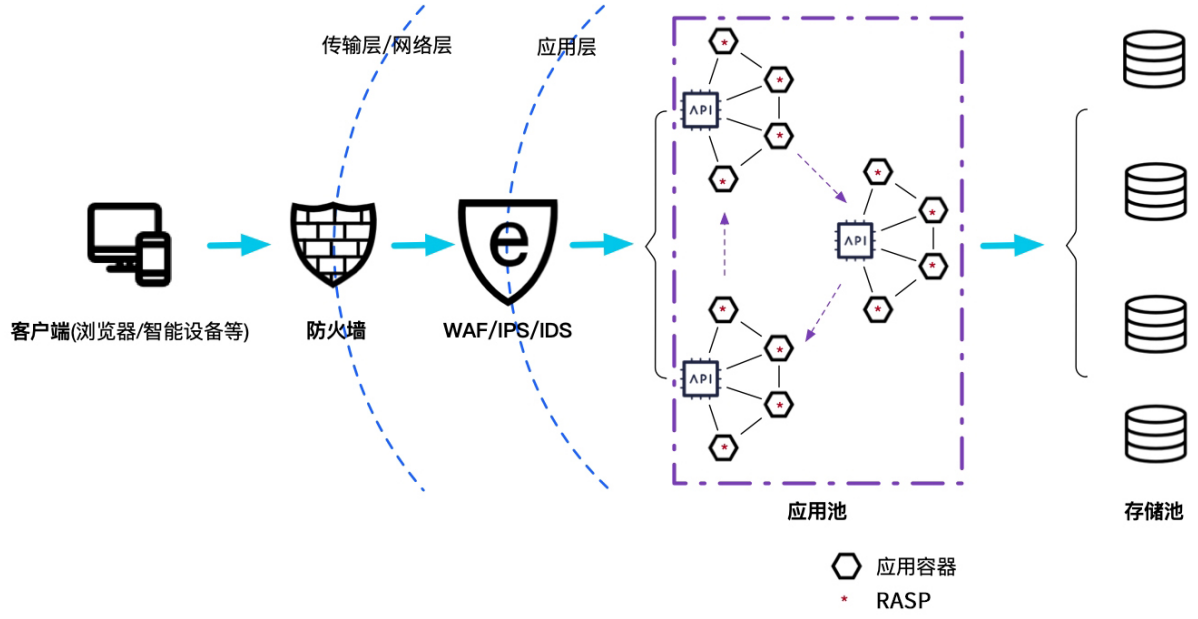

图 7-5 RASP 使用场景

08 总结

科技的发展让社会协作变得更加高频和具有深度，在信息技术行业亦是如此。我们自己的业务系统会集成第三方的代码、使用第三方提供的开发环境、应用第三方生产的构建工具、运行在第三方开发的微服务和容器之上，这种深度协作方式让我们的业务生产和运营效率指数级提升，但同时带来的，也有软件供应链风险史无前例的增长。

SolarWinds Orion 事件让人们见识了供应链攻击能做到什么，Apache Log4J2 漏洞让人们了解了开源代码的千疮百孔，Equifax 信息泄露事件让人们知道了大型企业的安全建设在软件供应链安全漏洞前是多么的脆弱不堪。这一切，促成了我们对本报告的编撰工作，我们希望用系统性的思维和方式，收集、分析、整理、阐述软件供应链安全治理与运营的方方面面。由于篇幅所限，本文提及的工具、方法和措施只是软件供应链安全领域的沧海一粟，更多的还需要读者在实践中探寻。

结语

CONCLUSION

数字化的发展，极大依赖于信息技术的发展，其中最重要的就是软件。DevOps、Scrum 等敏捷开发的理念，促进了软件开发的效率，开源软件等大大促进了软件的繁荣。但是，数字化时代，软件系统和经济利益相关性越来越高，也吸引了更多的恶意攻击者，通过发现、利用系统潜在漏洞获取利益，因此对软件系统自身的安全性提出了更高的要求。

信息系统的安全，不仅仅是运营阶段的安全监测防护，而是需要全生命周期的安全保障，尤其是开发阶段输出的系统软件自身的健壮性，包括自身代码的安全、依赖的组件的安全、使用配置的安全性，这也是“安全左移”或者“DevSecOps”的核心要义，要把安全覆盖到开发运营的全生命周期。

软件供应链安全是保障系统软件安全的关键因素，软件供应链安全的治理需要所有从业者从理念上提高意识；在技术上需要同行共同潜心研发，突破卡脖子技术；加强能力共享，共同推进软件供应链安全公共基础设施建设。白皮书旨在希望能推动行业共识，促进软件供应链安全的健康发展。

何国锋

2022 年 8 月

引用参考

[1]Aqua.Software Supply Chain Attacks[EB/OL]. https://www.aquasec.com/cloud-native-academy/supply-chain-security/software-supply-chain-attacks/

[2] 美国“加强软件供应链安全实践的指南”解读 [EB/OL].
https://www.secrss.com/articles/36760.

[3] 上官晓丽 , 孙彦 , 李彦峰 . 信息通信技术供应链安全政策法规与标准研究 [J]. 中国信息安全 ,2021 No.143 10 45-48.

[4]REVERSINGLABS.A (Partial) History of Software Supply Chain Attacks[EB/OL].
https://blog.reversinglabs.com/blog/a-partial-history-of-software-supply-chain-attacks

[5]Cloudflare.The software supply chain is under attack[EB/OL]. https://www.cloudflare.com/zh-cn/learning/insights-supply-chain-attacks/

[6]I-scoop.Software supply chain attacks and security: state and outlook[EB/OL].
https://www.i-scoop.eu/cybersecurity/software-supply-chain-security/

[7]OWASP.Free for Open Source Application Security Tools[EB/OL].
https://owasp.org/www-community/Free_for_Open_Source_Application_Security_Tools

[8]ITProPortal.The threats of open source software in cloud native[EB/OL].
https://www.itproportal.com/features/the-threats-of-open-source-software-in-cloud-native/

[9]Xfive.5 Open Source Security Risks You Should Know About[EB/OL].
https://www.xfive.co/blog/5-open-source-security-risks/

[10]Bytesafe.SLSA: A Novel Framework For Secure Software Supply Chains[EB/OL].
https://bytesafe.dev/posts/slsa-introduction/

[11]nnovecs.TOP 10 SOFTWARE DEVELOPMENT TRENDS TO ADD MASSIVE VALUE TO YOUR BUSINESS[EB/OL].
https://innovecs.com/blog/software-development-trends/

[12]Anchore.2022 Security Trends: Software Supply Chain Survey[EB/OL].
https://anchore.com/blog/2022-security-trends-software-supply-chain-survey/

[13]Chetan Conikee.3 Critical Software Development Security Trends and Best Practices[EB/OL].
https://www.darkreading.com/vulnerabilities-threats/3-critical-software-development-security-trends-and-

best-practices

[14]Argn.5 Common Risks for Supply Chain Cyber Attacks and What to Do About Them[EB/OL]. https://www.argon.io/blog/risk-for-supply-chain-cyberattacks/

[15] David Shavgulidze.Supply Chain Security Analysis of the Georgian Ecosystem[EB/OL]. https://www.isaca.org/en/resources/news-and-trends/isaca-now-blog/2022/supply-chain-security-analysis-of-the-georgian-ecosystem

[16]Stephen Pritchard.Software supply chain attacks – everything you need to know[EB/OL]. https://portswigger.net/daily-swig/software-supply-chain-attacks-everything-you-need-to-know

[17]FOSSA.The Complete Guide to Software Composition Analysis[EB/OL]. https://fossa.com/complete-guide-software-composition-analysis

[18] 悬镜安全 .IAST 技术进阶系列（一）： 关键语言支持 [EB/OL]. https://mp.weixin.qq.com/s/Q81ZVRX2yUkeMW5O7Af4IA.

[19] 悬镜安全 .IAST 技术进阶系列（二）： 全场景多核驱动 [EB/OL]. https://mp.weixin.qq.com/s/8JTQt-LiRi7DTug7Mx5RPg.

[20] 悬镜安全 .IAST 技术进阶系列（三）： 高并发 & 高可用场景支持 [EB/OL]. https://mp.weixin.qq.com/s/Q_dQgfA6LlZcXtI0z8YSCg.

[21] 中国信息通信研究院 . 研发运营安全白皮书（2020）[R]. 北京 : 中国信息通信研究院 ,2020.

[22] 中国信息通信研究院 . 研发运营一体化（DevOps）能力成熟度模型 [R]. 北京 : 中国信息通信研究院 ,2018.

[23] 中国信息通信研究院 . 软件供应链安全发展洞察报告（2021 年）[EB/OL]. 北京 : 中国信息通信研究院 ,2021. http://doc.opensourcecloud.cn/2021/1228.pdf.

[24] 何熙巽 , 张玉清 , 刘奇旭 . 软件供应链安全综述 [J]. 信息安全学报 ,2020,5(1).

[25] 国家保密科技测评中心 . 开源软件风险分析及治理措施研究 [EB/OL].2020[2021]. http://www.gjbmj.gov.cn/n1/2020/1127/c411033-31947270.html.

[26] 国家保密科技测评中心 . 开源软件漏洞安全风险分析 [EB/OL].2020[2021]. http://www.gjbmj.gov.cn/n1/2020/1127/c411033-31947310.html.

[27] 阿里云云栖号 . 对 SolarWinds 事件更深的思考 : 如何防御供应链攻击 [EB/OL]. 2021[2021]. https://blog.csdn.net/yunqiinsight/article/details/112601468.

出品方