悬镜
XMIRROR

软件供应链安全白皮书（2021）

法律声明

ABOUT

关于悬镜安全

悬镜安全，DevSecOps 敏捷安全领导者。由北京大学网络安全技术研究团队“XMIRROR”发起创立，致力以 AI 技术赋能敏捷安全，专注于 DevSecOps 软件供应链持续威胁一体化检测防御。核心的 DevSecOps 智适应威胁管理解决方案包括以深度学习技术为核心的威胁建模、开源治理、风险发现、威胁模拟、检测响应等多个维度的自主创新产品及实战攻防对抗为特色的政企安全服务，为金融、电信、政务、能源、教育等行业用户提供创新灵活的智适应安全管家解决方案。

悬镜安全官网：https://www.xmirror.cn/。

前言

INTRODUCTION

随着容器、微服务等新技术日新月异，开源软件成为业界主流形态，软件行业快速发展。但同时，软件供应链也越来越趋于复杂化和多样化，软件供应链安全风险不断加剧，针对软件供应链薄弱环节的网络攻击随之增加，软件供应链成为影响软件安全的关键因素之一。近年来，全球针对软件供应链的安全事件频发，影响巨大，软件供应链安全已然成为一个全球性问题。如何更加全面、高效地保障软件供应链的安全对于我国软件行业发展、数字化进程推进具有重要意义。

软件供应链安全作为国家近几年新提出的网络安全理念，不再是针对软件供应链上单一环节进行安全防护，而是针对软件供应链全链路进行安全监控防护，薄弱环节的安全预防更是重中之重。

本白皮书着重分析了软件供应链安全，梳理了软件供应链的安全现状，透过现状全面剖析软件供应链的安全风险及面临的安全挑战，有针对性地提出如何对软件供应链的安全风险进行防范与治理，系统阐述了软件供应链安全的防护体系及软件供应链安全的应用实践以供参考，最后白皮书结合现在软件供应链安全的发展趋势进行了全面的分析及展望。

目录

CONTENTS

4 软件供应链安全治理方法

5 软件供应链安全应用实践

01 软件供应链概述

1.1 软件供应链与传统供应链之间的共性及差异性

行业内对软件供应链有多种理解，其中包括三种主流理解方式，第一种是将软件供应链单纯视为以开源组件为主的第三方组件供应链条；第二种是将其视为软件生产过程供应链条，包括第三方组件、开发工具和开发环境等要素；第三种是将其视为从原材料开始加工成消费者手中的最终产品并实施运营过程的全流程链条。由于前两种理解存在不同程度的片面性，因此本白皮书中以第三种理解作为阐述基础。

传统供应链的概念可以理解为一个由各种组织、人员、技术、活动、信息和资源组成的将商品或服务从供应商转移到消费者手中的过程，这一过程从原材料开始，将其加工成中间组件乃至最终转移到消费者手中的最终产品。软件供应链是根据软件生命周期中一系列环节与传统供应链的相似性，由传统供应链扩展而来。

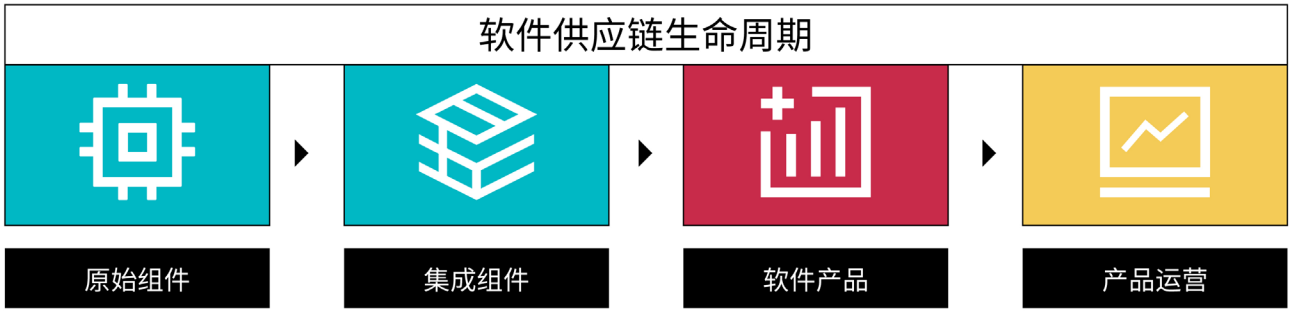

图 1 软件供应链生命周期

软件供应链的生命周期包括原始组件、集成组件、软件产品及产品运营四个环节（如图 1 所示）。在软件供应链中，原始组件是原材料，集成组件是中间组件，软件产品是交到消费者手中的商品，产品运营是为消费者提供的服务保障产品的正常运行。因此，软件供应链可以理解为软件和系统的从生产到交付全过程，是一套自动化、标准化及规模化的持续交付的流水线。通过设计和开发阶段，将生产完成的软件产品通过软件交付渠道从软件供应链运输给最终用户。

原始组件、集成组件、软件产品及产品运营四部分涵盖保障软件供应链安全涉及的诸多关键安全要素，了解软件供应链的每一个阶段及流程中出现的源代码、工具及集成组件对于构建安全可靠的软件供应链至关重要。

根据软件供应链的定义，软件供应链安全可以被理解为软件生产的整个过程中软件设计与开发的各个阶段来自编码过程、工具、设备、供应商以及最终交付渠道所共同面临的安全问题。

对软件从业者来说，实际需要关注的是自身的软件产品开发过程和运营过程，也就是软件产品和产品运营这两个阶段，软件供应链中的原始组件和集成组件阶段的安全问题应由相应的组件供应商解决。做个类比，假如我们需要生产一部手机，我们需要关注的是手机生产和上市运营。在生产手机过程发生前，我们需要选择可信的零部件供应商，从之购买合格的零部件。而并不需要接管供应商的生产流程，帮助供应商做质量管理。软件产品阶段和产品运营阶段的加总，实际上就是传统概念中的软件生命周期。软件生命周期可以划分为 4 个主要阶段：

设计阶段、编码阶段、发布阶段、运营阶段（如图 2 所示）。这也是软件供应链安全治理中真正需要关注的部分。

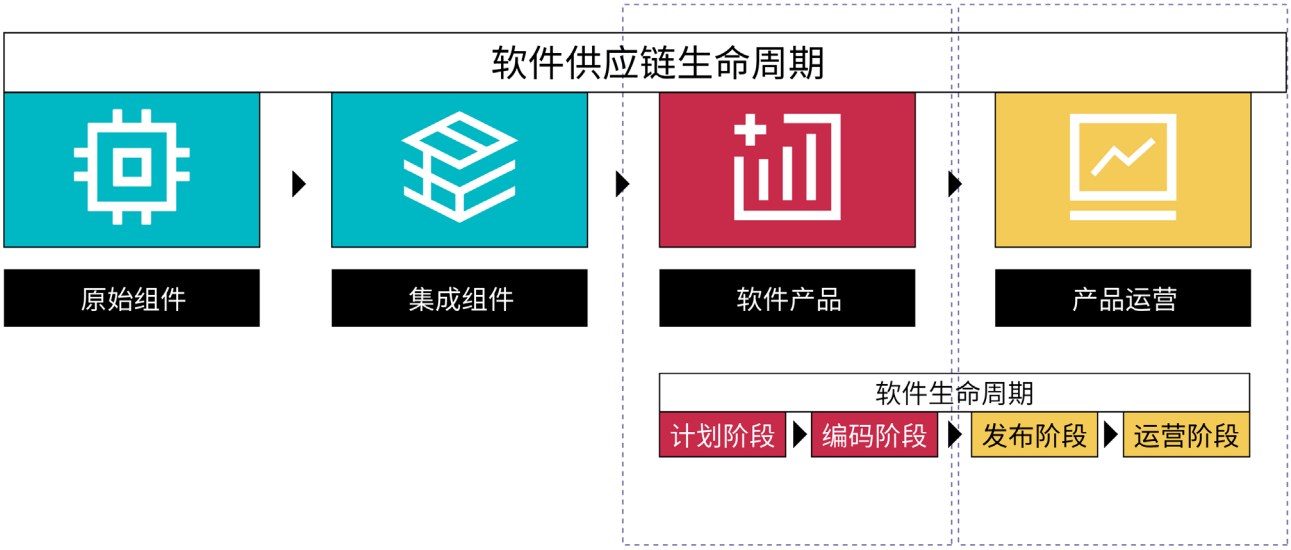

图 2 软件供应链生命周期与软件生命周期的关系

软件供应链和传统供应链的安全性之间存在显著的共性及差异性，软件供应链与传统供应链之间的共性是攻击者通过攻击目标供应链中较弱的组成部分，导致产业链上游被污染，大量下游厂商的产品或服务作为上游组件的集合，产生大量潜在的攻击目标，而且基于对产品和服务的潜在信任，导致在安全问题出现时，难以彻底断根筛查。同时，召回问题产品的代价巨大，周期漫长，显著增加了供应链攻击的影响程度。

差异性是软件供应链所受到的攻击与传统供应链相比，软件攻击面的边界由产品本身扩大到软件生产过程中代码、服务及组件的边界，导致软件供应链的受攻击面不断扩大，显著降低了攻击者攻击的难度，这就导致软件供应链攻击可能发生在软件供应链生命周期的任何阶段，同时更聚焦在威胁的高传播性和强隐蔽性，而非仅仅聚焦在瞬间的破坏力上。

1.2 软件供应链的发展历程

1984 年，UNIX 创造者之一的 K.Thompson 在 A.M Turing Award （ACM）图灵奖的获奖演讲中提到如何通过三步构造一个通过编译器污染所有通过此编译器编译并发布软件的攻击方式，这种攻击可以在人们难以发现的情况下修改编译器，并在其中设置后门。基于此，一个通过攻击软件开发过程中薄弱环节的攻击方式暴露在人们的视野中，这就是令人头疼的 Advanced Persistent Threat （APT）攻击，在著名的震网事件中，非法组织利用震网病毒破坏世界各国的计算机系统，进而破坏国家重要的基础设施。

1995 年，软件供应链的概念出现在大众的视野中，之后在 2000 年，M.Warren 和 W.Hutchinson 提出了利用网络攻击破坏软件供应链的可能性。

2004 年，微软公司提出了著名的软件安全开发生命周期流程，这一流程将软件开发流程划分为多个阶段，并在各个阶段中引入不同的安全措施，保障软件开发以及最终用户的安全性，并建立了漏洞发现和漏洞处理框架机制。同时提出了多种安全测试工具以及软件发布后的相关运营管理规范，至今仍是各大企业采用处理软件开发安全问题的重要手段之一。

2010 年，R.J.Ellsion 和 C.Woody 两人针对当时软件开发过程中直接采购商品化的产品和技术以及产品外包服务逐渐增加的趋势，出于对软件供应链的安全考虑，针对软件开发过程中提出了软件供应链风险管理这一思想，并介绍了相关的风险来源、种类以及风险的分析方法，并讨论了如何应对软件供应链风险的相关措施。

2014 年出现著名的 HeartBleed 漏洞，作为被基于 SSL\TLS 的软件和网络服务所广泛使用的开源软件包的漏洞，感染了软件和服务的开发阶段中上游代码和模块，并沿着软件供应链，对供应链下游造成了不可磨灭的负面影响，因此这一事件被广泛认为是一起典型的软件供应链安全事件。由于近年来，一系列具有极高相似性的安全事件频频发生，这些事件中所使用的网络攻击方式与传统攻击方式相比具有更加鲜明的高隐蔽性、高传播性、低成本和高效率的特点，使软件供应链安全这一概念受到产业界和学术界的极大关注。

2017 年，微软公司旗下的一个研究团队发出声明，声明中表示微软旗下的安全软件阻挡了一起精心策划的，通过攻击软件更新渠道，将插入了恶意代码的第三方软件传输给使用该软件的多家知名机构的高级持续性威胁攻击。在这篇声明中，微软针对本次安全事件，首次提出了“针对软件供应链的网络攻击”这一概念。

近年来，由于软件开发日益全球化，导致软件供应链的安全风险愈发严峻，已成为人们不可忽视的存在。软件供应链像是一个充满了未知且不可确定危险因素的地带，需要人们给予更多的关注和思考。

1.3 开源和云原生时代改变了传统的软件供应链

开源是指公开源代码的软件，任何人都可以进行自由修改和共享。源代码是程序员可以创建和编辑以改变软件工作方式的代码。通过访问程序的源代码，开发人员或程序员可以通过添加功能或修复有缺陷的部分来改进软件。

在当今快节奏的商业世界中，越来越多的软件团队采用了 DevOps 等敏捷开发实践以跟上业务的需求，这些做法给开发人员带来了很大的压力，要求他们能够更快地构建和部署应用程序。为了在较短的软件发布周期内成功实现此目标，开发人员经常通过对开源软件组件进行修改添加所需要的功能以加快软件开发的速度。

开源组件已成为企业实现快速开发和科技创新的必要条件。据统计，大部分的商业程序中都包含开源软件，开源为企业节省了大量的时间和金钱、提高了软件生产质量、提供了业务敏捷性并降低了某些业务风险。若没有开源组件，企业的软件生产将成倍地迟缓。据 Forrester 2021 年发布的报告数据显示（如图 3 所示），开源代码占软件代码的比例从 2015 年到 2019 年的五年时间内几乎翻了一倍。随着开源组件的不断增多，大量的第三方开源组件被插入到产品中，大量开源组件的使用也导致软件供应链变得越来越复杂。

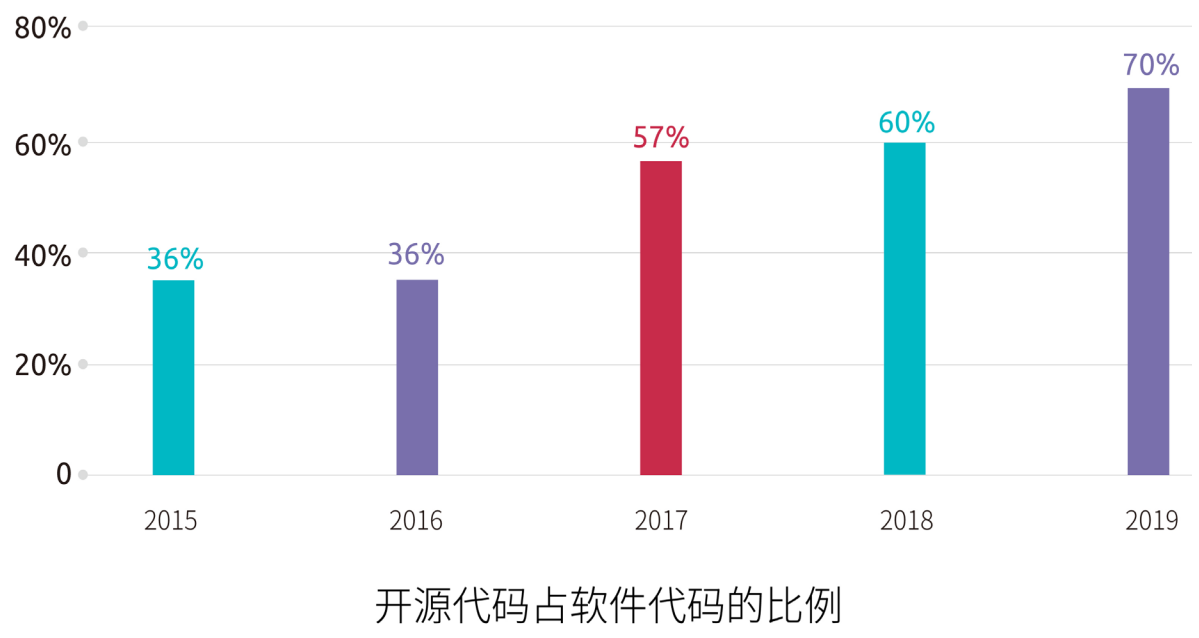

图 3 开源代码占软件代码的比例不断增长

与开源密切相关并改变传统供应链的还有云原生时代的出现，容器、Docker 和 Kubernetes（K8s）等概念的出现，改变了传统软件交付的方式，进而影响到软件供应链。容器和 K8s 的引入导致软件供应链的复杂程度加剧。（如图 4 所示）通过引入容器和 K8s 给软件供应链带来更多不可控的第三方依赖，不同于传统的软件供应链，镜像作为软件统一交付的标准在容器场景下被大规模的应用，这极度加大了软件供应链的复杂性。

云原生时代下镜像的使用

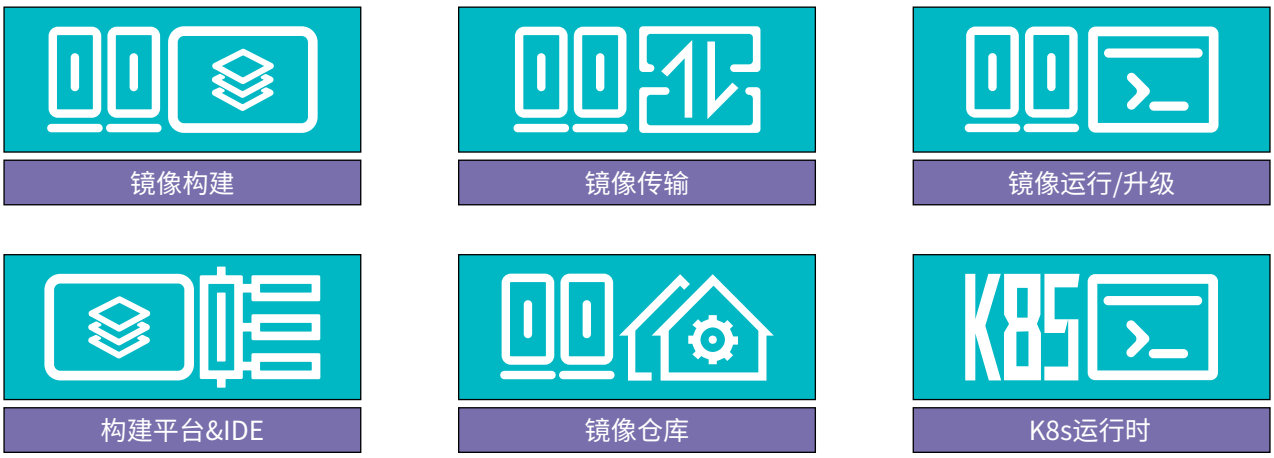

图 4 云原生时代下软件供应链

K8s 是一个复杂的平台，需要多个部分组合才能完成工作，但是将其各个部分集成为一个整体后又变得难以管理，当需要安装或更新 K8s 时，必须分别对每个单独的组件进行配置；大量使用容器虽然优化了软件应用程序架构，并且标准化了软件开发和生产环境，但同时也增加了软件部署管理的复杂度。

02 软件供应链安全现状

2.1 国内外软件供应链安全发展现状

开源和云原生时代的到来导致软件供应链越来越趋于复杂化和多样化，安全风险不断加剧，针对软件供应链薄弱环节的网络攻击也随之增加，软件供应链安全已然成为一个全球性的问题。虽然，国际上对加强软件供应链安全管理也早已成为共识，但仍需详细了解软件供应链的背景及发展情况，以便更好的应对软件供应链的安全风险。

2.1.1 国际软件供应链安全发展现状

自软件供应链的概念被提出以来，国际上对软件供应链安全有着高度的重视，国际软件供应链安全的发展情况可从国家层面和企业层面两个方面进行讨论。

国家层面，以美国为例，出于对软件供应链的安全性及脆弱性的担忧，早在多年前就开始着手布局国家级软件供应链的安全战略，陆续出台一系列相关政策和重点项目来加强软件供应链的安全管控。

2015 年 4 月，美国国家标准与技术研究院（NIST）正式发布软件供应链制定规范《ICT 供应链风险管理标准》（NIST. SP800-161），帮助组织机构管理软件供应链安全风险，清楚地界定了软件供应链中相关的标准和要求，建立了适当的政策和流程控制，在一定程度上规避了软件供应链面临的诸多风险。目前，NIST 的标准系列文件，已成为美国和国际安全界广泛认可的事实标准和权威指南。

近年来随着管理措施的陆续落地和管理范围的扩大，美国软件供应链安全防御体系不断加强。2017 年、2018 年更是针对关键信息基础设施相关的供应链安全提出了明确的要求，其中包括促进供应链风险态势及相关信息共享、加强供应链风险审查评估、推动相关标准的实施应用等。

美国政府正在不断将软件供应链安全问题深化和细化，关注点正在聚焦到特定的 IT 产品和服务上。2021 年 5 月 12 日，美国总统拜登签署发布了《改善国家网络安全行政令》，该行政令是美国联邦政府试图保护美国软件供应链安全采取的最强劲措施，要求向联邦正式出售软件的任何企业不仅提供应用程序，而且还必须提供软件物料清单，提升组成该应用程序组件的透明度，构建更有弹性且安全的软件供应链环境，确保美国的国家安全。
除美国之外，2016 年，英国国家互联网应急中心（CERT-UK）发布供应链网络安全风险白皮书，其中，介绍了各类软件供应链的风险案例和规避建议。2018 年 1 月，英国国家网络安全中心（NCSC）发布供应链安全专题和指导文件，其中包括 12 条安全原则，供应链攻击示例和安全性评估方法以及管理实践。

在企业层面，开源软件作为软件供应链中最重要的一个部分，国际上的头部企业也进行了相关的合作：以 Google 为首的 7 家技术公司在 2017 年合作推出了一个名为 Grafeas 的开源计划，旨在为企业定义统一的方式，审计和管理其使用的开源项目。

同时，国际上许多知名企业正不断加大针对软件供应链的安全风险治理工作，针对开源软件采用软件成分分析技术，确保第三方开源组件的安全性；针对软件开发这一过程，微软提出的软件安全开发生命周期（SDL）及 Gartner 提出的 DevSecOps 理念，均旨在帮助企业降低在软件开发过程中所面临的安全问题。

2.1.2 国内软件供应链安全发展现状

随着网络安全形势的不断变化发展，在严峻的网络安全环境下，我国对软件供应链安全给予了高度的重视。2017 年 6 月，我国发布实施《网络产品和服务安全审查办法》，将软件产品测试、交付、技术支持过程中的供应链安全风险作为重点审查内容，并推动开展了云计算服务网络安全审查。在 2020 年 4 月 27 日，国家互联网信息办公室等 12 个部门联合发布了《网络安全审查办法》，要求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，应当进行网络安全审查，此政策的发布代表已明确将软件供应链安全带入到国内大众的视野中。

我国针对开源软件的发展也已出台了相关政策，特别是 2021 年在中国《“十四五”规划和二零三五年远景目标纲要》文件中，明确提出“支持数字技术开源社区等创新联合体发展，完善开源知识产权和法律体系，鼓励企业开放软件源代码、硬件设计和应用服务”，这是开源首次被写入国家总体规划纲要之中。

企业层面，我国头部的互联网企业和安全厂商均开始投入到软件供应链的安全建设中，围绕保障软件供应链安全的重大需求，充分发挥创新技术在软件供应链网络安全保障中的作用，加大软硬件安全检测及分析、攻防渗透技术、源代码安全审计、漏洞挖掘技术、大数据分析技术等创新技术的研发及投入，切实落实对软件供应链安全的保障，构建一套动态的安全防护体系。

2.2 软件供应链的安全挑战

软件可以定义世界，数据可以驱动未来，软件正逐渐成为影响世界快速发展的重要因素。软件快速发展需要安全保驾护航，因此软件供应链安全也成为世界广泛关注的重点，根据上述对我国软件供应链现状的描述，结合现阶段国际网络安全形势，可以总结出当前我国软件供应链所面临的安全挑战。

2.2.1 国际竞争环境加剧，软件供应链完整性遭遇挑战

软件供应链的竞争和保障，不仅影响着企业的生存和发展，同时也成为世界各国之间互相制约与竞争的重要手段。某些西方国家通过实行严密的技术封锁，建立完善的出口管制法律制度体系，将本国的软件、硬件和技术列入出口管制清单，这导致国际软件供应链竞争环境加剧，软件供应链的完整性遭遇严重的挑战。

在全球软件供应链安全问题日益突出，国际 IT 技术竞争激烈的背景下，对我国自主研发生产相关软硬件、研发创新技术及提高 IT 技术实力提出了新要求，同时需要做好软件供应链战略计划，确保我国软件供应链自主可控、安全高效。

2.2.2 软件开源化趋势增强，安全风险加剧

软件供应链开源化趋势增加，导致影响软件供应链的各个环节都不可避免的受到开源环境的影响。作为创新的基础，开源正不断推动深度信息技术的创新发展，因此开源软件正在呈现指数级增长。例如：根据美国国家计算机安全中心（NCSC）公开的数据显示由流行的软件开发和源代码管理平台 GitHub 托管的公共存储库数量从 2009 年 2 月的 46000 个激增到 2020 年 1 月的 2800 万个。

由于开源软件之间的关联依赖关系非常复杂，一旦出现安全问题所带来的蝴蝶效应将带来十分严重的影响，若一款开源软件出现未知的安全漏洞，将会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，漏洞的攻击面将会由点及面呈现出爆炸式的放大效果。根据 Sonatype 发布的《2020 State of the Software Supply Chain》报告可以了解到（如图 5 所示），开发人员往往不能在漏洞披露的第一时间进行漏洞修复工作，51% 的开发人员至少需要一个星期以上的时间才会对漏洞采取修复措施，这意味着攻击者有充分利用漏洞进行攻击的时间，极大地增加了软件供应链的安全威胁。

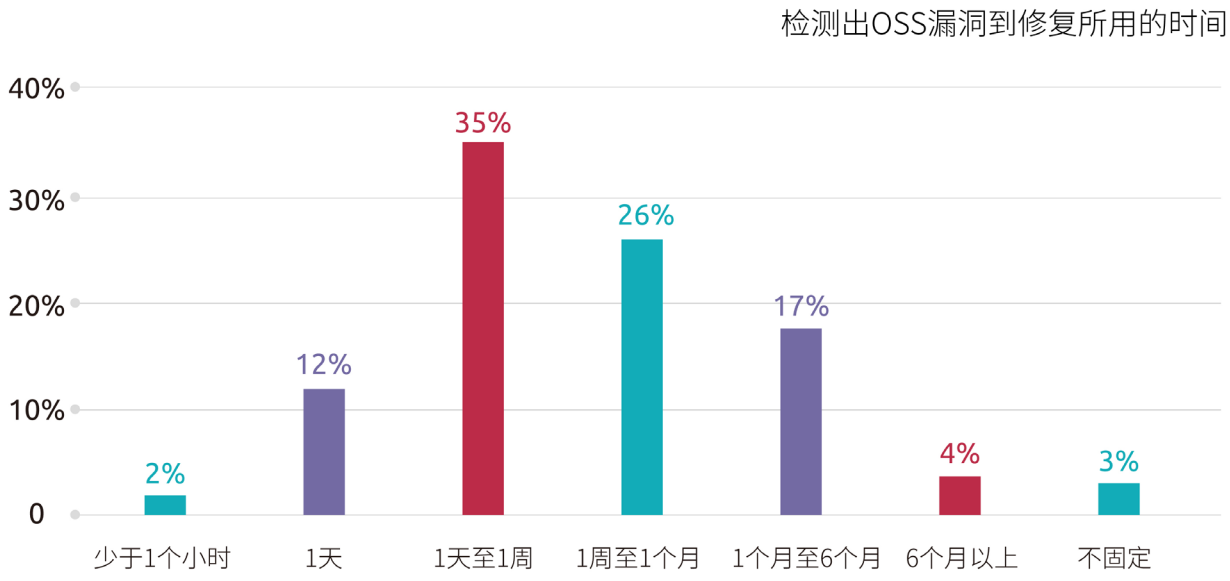

图 5 从漏洞发现到修复所用的时间

开源软件出现漏洞时，若不能及时获取应用程序中漏洞的详细信息，则会导致开发人员修复漏洞难度的增加。根据 Veracode 发布的 2021 软件安全状况报告可以了解到（如图 6 所示），当开发团队了解应用程序中漏洞信息时，3 个星期的时间就可以修复 50% 的漏洞，若开发团队不了解其信息，则这一时长大幅增长到 7 个月以上。

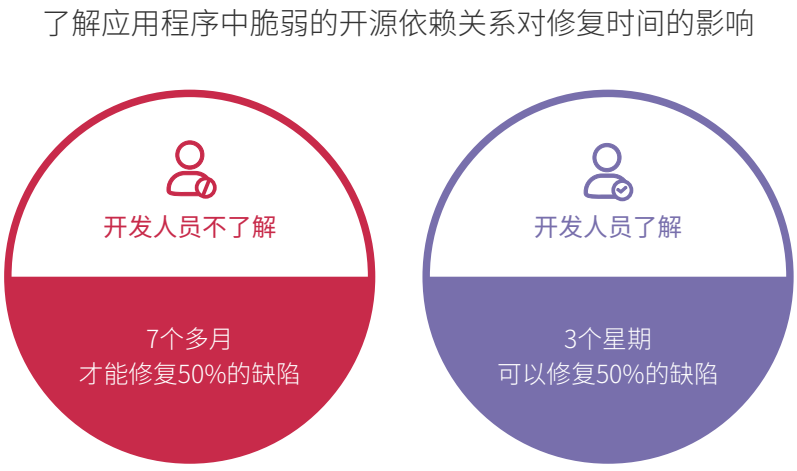

图 6 了解漏洞信息对修复时间的影响

除此之外，还可能存在具有非法目的开发者故意预留后门的安全缺陷，攻击者通过将恶意代码注入为全球软件供应链提供组件的开源项目中，借助开源软件的“高信任度”和影响力，通过感染软件供应链的“上游”组件加速向“下游”扩散，产生的破坏性进一步加大。

同时，国内部分开源软件和开源组件还涉及到开源许可证的冲突风险和知识产权的风险，推进开源标准，完善开源知识产权和法律体系是我国实现开源发展体系化规模化的重要手段之一。不仅如此，开源软件提高了企业软件供应链的暴露程度，国内大多数企业缺少对技术风险、法律风险和供应链风险的认知，极度缺乏专业知识和应对经验，在大量使用开源软件的情况下，可能使其成为攻击者的切入点。因此，软件开源化的趋势导致软件供应链面临极大的安全挑战。

2.2.3 软件复杂度增加，软件供应链每一环节均存在风险

现阶段，软件的复杂度随着用户对软件功能需求的不断提高而增加，由于信息技术产业的不断发展以及其它产业对软件依赖的加深，这一趋势将在未来几年持续增加。软件复杂度的增加会导致软件开发的难度加大，同时也会增加后续软件维护的难度。

随着软件规模越来越大，程序逻辑越来越复杂，对软件完整语义的理解及操作逻辑的把握越来越困难，设计缺陷、深层次漏洞更难以发现，后门更易于隐藏。在软件扩展过程中新增的组件需要与原组件进行交互，导致了软件开发的组件化、复用化，以及软件供应链的产生。

软件供应链可以理解为一个链条，那么在链条上的每一个环节都可能产生安全威胁，由于软件设计缺陷和软件开发过程中产生的漏洞将沿着软件供应链的树状结构由上游向下游扩散，加大了软件供应链的安全风险，同时这也对软件的业务逻辑和安全质量控制及标准提出了更高的要求。

2.2.4 难以处理敏捷开发与安全成本之间的平衡

随着互联网时代的高速发展，信息得以快速传递，这对软件开发管理带来前所未有的冲击。用户对于软件的功能性、实用性和易用性等方面的要求越来越高，软件市场的竞争压力不断加大，互联网企业面临更加复杂和快节奏的外部环境，这要求软件开发者在较短的时间内完成大量功能开发的任务，并在后续持续不断地高速迭代以满足市场的需求。由于要节省时间成本，开发者在开发过程中必然会大量使用开源库与外包供应商提供的非公开库。但是，在使用外部库时会存在大量安全隐患，一旦出现数据泄露等安全问题，将导致极其巨大的经济损失。

在敏捷开发模式中，因为安全影响开发效率导致安全与开发严重割裂，传统安全团队上线前介入的模式已经严重滞后，不仅不能有效地进行系统的安全防护，同时也会影响应用的交付速度。如何在较短的开发周期内尽量完善软件功能，同时完成相对完备的安全性测试，实现质量、安全和速度的平衡是现阶段各企业及开发部门所面临的重要挑战。

03 软件供应链风险分析

3.1 软件供应链风险概述

随着我国科技的高速发展，软件供应链正不断延伸，由于针对软件供应链的攻击成本低且回报高，不法分子逐渐将攻击目标转换到软件供应链的薄弱环节上。我国软件供应链面临的安全风险不断加大，遭受破坏而引发的网络安全事件数量逐步上升，因此，有必要针对导致我国软件供应链产生安全风险的主要因素进行深入探讨分析。

3.1.1 软件供应链风险现状

3.1.1.1 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。

“棱镜门”事件的发生让世界人民意识到：网络监听、远程控制、数据窃取等并不是危言耸听而是真实存在的。中国互联网新闻研究中心发布的报告《美国全球监听行动记录》显示，经过长达几个月的查证，中国发现美国的“棱镜门”秘密项目中有针对中国机密的监听行为，此次监听行动涉及中国政府和领导人、中资企业、科研机构等，给中国的信息安全带来严重的影响。

除此之外，西方大国还借助其在 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。

3.1.1.2 开源存在缺陷，引入安全风险

开源软件的安全形势已愈发严重。根据 RiskSense 发布的研究报告得知，2019 年已公开的开源软件 CVE 漏洞总数为 968 个，比之前数量最高年份的两倍还多，2020 年前三个月新增的 CVE 漏洞数量也处于历史高位。从美国国家漏洞库（NVD）对于开源软件的漏洞管理情况来看，其收录开源软件漏洞的滞后性比较严重，从漏洞首次公开披露到收录进 NVD 平均耗时 54 天，最长耗时 1817 天。攻击者完全有可能利用这段时间开发和部署漏洞利用程序，而正在使用存在漏洞开源软件的企业由于无法及时收到 NVD 的安全警报，企业将完全暴露在安全风险之中。

2020 年 2 月，国家信息安全漏洞共享平台（CNVD）发布了关于 Apache Tomcat 存在文件包含安全漏洞的公告，该漏洞可以造成 Tomcat 上所有 webapp 目录下的重要配置文件或源代码等敏感数据的泄露，若同时存在文件上传功能，则可能会进一步实现远程代码执行（RCE），直接控制服务器。Tomcat 是 Apache 软件基金会下一个重要的开源软件，在全球范围内被广泛使用，根据调查数据显示，国内受影响采用 AJP 协议的 IP 数量大约是 4 万个。

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。

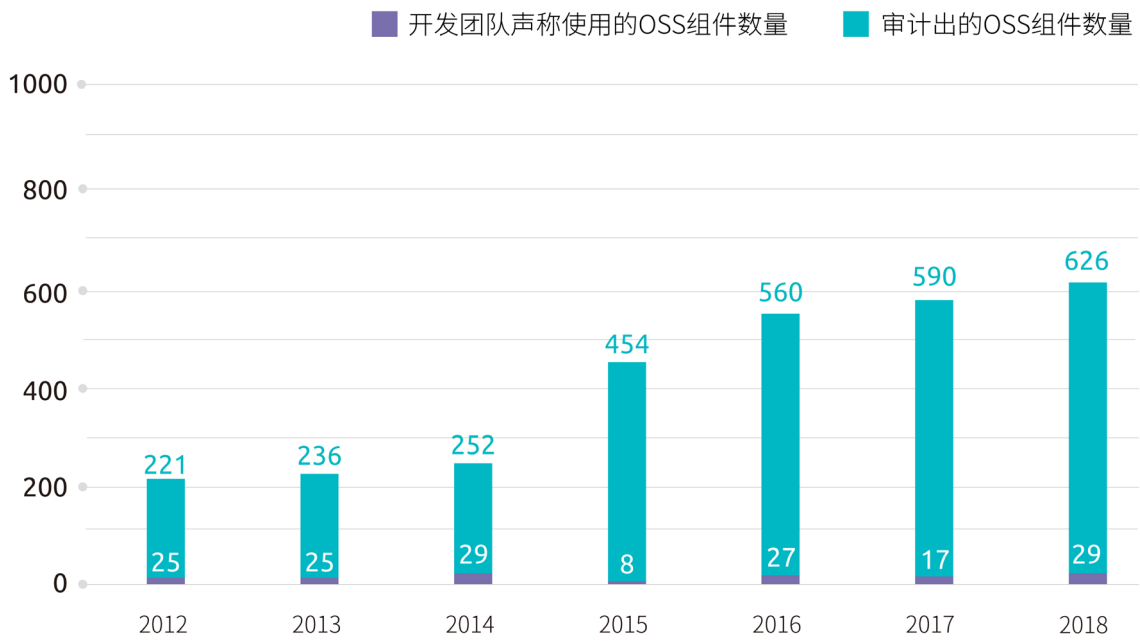

图 7 OSS 组件实际使用数量随着时间推移不断增多

由于开源软件使用与获取的便利性，几乎每个商业软件都会大量使用开源组件，但企业并没有对开源组件的来源和质量给予足够的重视，甚至会忽略掉软件组件所带来的安全风险。据 Sonatype 发布的《2020 State of the Software Supply Chain》报告数据显示，从 2012 年到 2018 年开发团队声称使用的 OSS 组件数量和实际审计出的 OSS 组件数量的差距逐渐加大（如图 7 所示）。企业在软件开发的过程中，对开源软件的使用比较随意，管理者通常不清楚开发团队在开发过程中是否有使用开源软件，具体使用了哪些开源软件，这些开源软件是否存在安全漏洞等，这无形中给软件供应链带来了难以预知的安全隐患，从而导致软件供应链攻击呈现上升趋势。

3.1.1.3 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。

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用

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。
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。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。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。

3.1.2 软件供应链风险因素分析

通过对导致软件供应链安全风险主要因素的分析，可以发现软件供应链的各环节都有可能被攻击进而产生安全风险，攻击者一旦对软件供应链中的任一环节进行攻击行为，都有可能引起软件供应链的连锁反应，进而造成严重的安全威胁，甚至危害国家网络安全。

下文根据软件供应链的相关特征，针对软件生命周期主要的四阶段分析其安全风险。

3.1.2.1 设计阶段

由于开发人员对安全认知的缺失和忽略，往往导致软件产品在功能需求、架构设计和编译的程序代码中存在天然的安全缺陷，而这些往往是无法通过后期软件开发环境加固和安全应急响应进行根本解决的。

若在软件开发的设计阶段开发团队没有进行全面的威胁分析，那么很有可能因为没有充分对安全架构和设计进行验证、明确安全目标、识别相关威胁和漏洞及制定相关应对政策，从而加大软件的攻击面，造成软件开发后期需要成本高昂的补救措施。

由于应用环境的复杂性，软件可能会存在许多安全漏洞和威胁，但在软件的实际应用中，并不是所有安全漏洞和威胁都会造成软件安全问题，因此，若在软件设计阶段不能识别出软件所需要重点注意的安全威胁，就不能合理的投入资源进行安全防护，从而很有可能会导致不能及时消除有可能造成安全问题的漏洞。

3.1.2.2 编码阶段

在编码阶段，软件编译器是常见的攻击目标，若编译器被植入恶意代码，那么通过该编译器编译出的目标代码也同样会受到病毒感染。编译器在自身编译的过程中被恶意插入“后门”，那么在通过该编译器编译任何源码时都可能将后门插入到目标代码中，对代码安全造成极为恶劣的影响。

开源软件已经成为软件开发人员开发代码的重要来源，所有软件开发者均参与到开源软件源代码的开发与维护中，随着开发者将自己的代码提交到开源软件平台的项目库中，这就加大了被攻击者的恶意代码感染的可能性。随着软件开发人员对开源软件的不断使用和传播扩散，一旦开源软件中存在有意或无意引入的安全缺陷，将进步影响软件供应链更多环节的安全性，同时项目集成和代码复用也会带来额外的安全风险，未对第三方代码与软件进行安全检测和审查，是软件供应链安全风险的重大隐患来源。

3.1.2.3 发布阶段

现代软件开发厂商多以敏捷开发、DevOps 开发方式为主，具有“版本迭代快、开发时间短”的特征。开发团队在版本迭代过程中占用大部分时间执行编码工作，而测试团队则往往没有充足的时间开展测试工作，更无法做到每个版本都执行安全测试。因此，错误的架构设计或程序编码因安全检测不足而未被识别，导致软件自带安全缺陷进行上线部署，并通过不断迭代集成，最终变成软件供应链的安全隐患。

除此之外，在软件发布阶段我国遭受最为广泛的软件攻击手段是软件的捆绑下载。各大应用发布平台在软件应用上线前缺少对软件应用的安全审核，导致软件从上传至平台到最后的下载环节，都有被引入安全风险的可能。
在软件发布的环节，许多软件厂商出于推广的需求，往往会进行捆绑安装下载，对此已形成一条完整的灰色产业链，导致用户在毫不知情的情况下，下载存在恶意代码或后门的捆绑软件。

同时，对于某些软件产品来说，软件开发人员在最初编码过程中为了方便后续的修改和管理，可能会预留一些具有高级管理权限的账号，而当软件产品正式发布时由于开发人员遗忘或故意留下的“后门”，将导致软件产品在发布阶段被攻击者利用进而造成巨大的危害。

3.1.2.4 运营阶段

软件产品在发布之后到达用户手中时，就进入了软件产品的运营阶段，而用户使用过程中，除了产品本身的安全缺陷所带来的安全威胁之外，还可能遭受用户使用环境带来的安全威胁，针对软件运营阶段最常见的攻击方式主要是软件的升级劫持。

软件产品在整个软件开发生命周期中几乎都会进行更新，常见的有功能升级更新、软件缺陷更新等等。攻击者可以通过劫持软件更新的“渠道”，比如通过预先植入用户机器的病毒木马更新下载链接、运营商劫持更新下载链接、软件产品更新模块在下载过程中被劫持替换等方式对软件升级过程劫持进而植入恶意代码。

在运营阶段，尤其是在更新或升级过程中，多数软件并未对软件升级过程进行严格检查，从而使攻击者有机可乘。在软件升级或更新过程中，攻击者可能通过中间人攻击替换升级软件或补丁包，或诱骗用户从非官方渠道下载，以达到网络攻击的目的。

3.2 软件供应链的漏洞类型

由于互联网信息技术产业的高速发展和软件开发需求的急速扩增，导致软件开发的难度与复杂程度不断加大，软件开发生命周期中的每一个环节都存在引入漏洞的可能性，导致软件供应链的安全风险无处不在，非法攻击者一旦对软件供应链中的任意环节进行攻击、篡改等，都将会引起最终软件供应链安全风险的连锁反应，产生巨大的安全危害。可以将软件供应链的漏洞大致分为以下几种类型进行分析（如图 8 所示）。

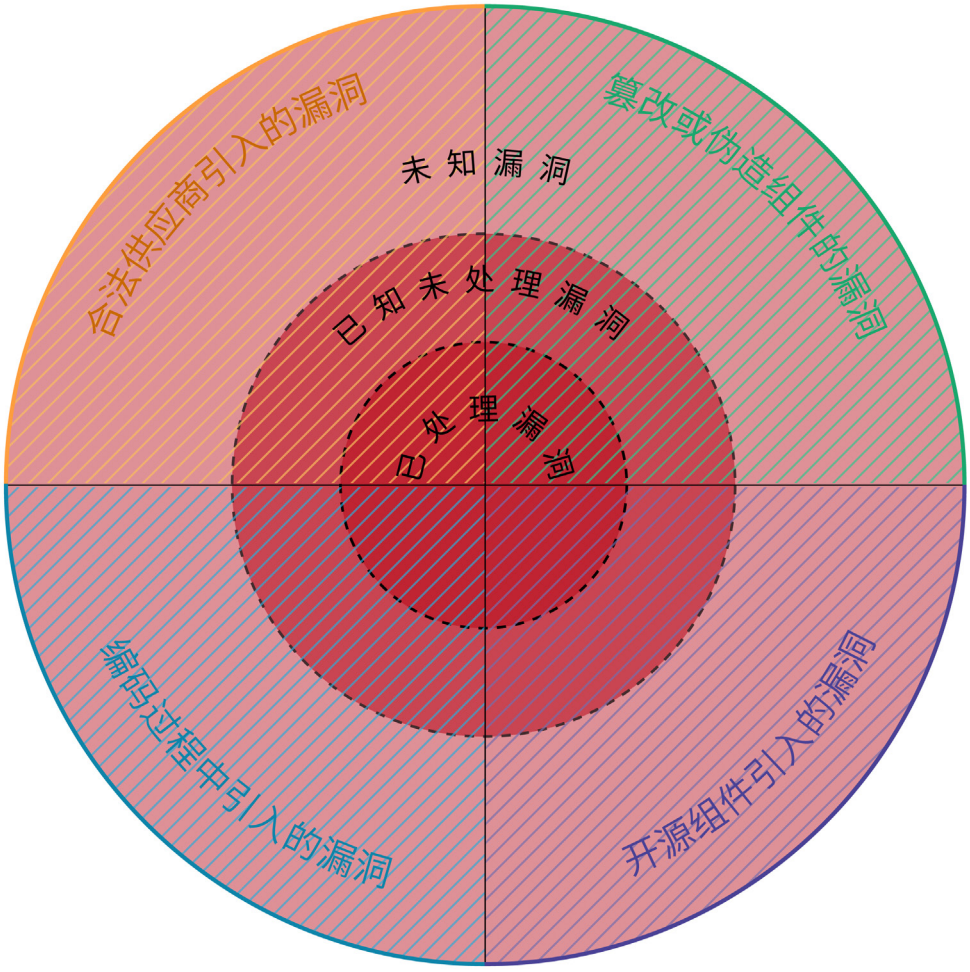

图 8 软件供应链漏洞类型

3.2.1 漏洞来源类型

3.2.1.1 合法供应商引入的漏洞

随着开发日益全球化，大规模的软件开发导致软件供应链发生了很大的变化，其中最重要的变化是如今的软件供应链中包含引入供应商这一关键环节。全球的互联网企业现如今越来越依赖成百上千的第三方供应商、承包商等外部机构的帮助来支持完成软件生产的过程。

通常一个产品的产生需要引入一个或多个第三方供应商，企业往往需要与这些供应商之间进行数据信息共享，但这些第三方供应商的安全性通常难以得到保障，使之成为软件供应链中一个薄弱的环节，给软件供应链带来前所未有的安全威胁。一个典型的通过合法供应商引入漏洞的例子是 VxWorks 所引发的安全事件。

VxWorks 是一种使用最广泛的实时操作系统，全世界范围内有超过 20 亿台设备正在使用，包括工业、电力、能源和航天航空等行业关键基础设施。多年前，网络安全公司 Armis 在 VxWorks 中发现了 11 个 0day 漏洞被称为 URGENT/11，其中 6 个漏洞为严重漏洞并且可以远程执行代码，其中 5 个漏洞包含拒绝服务、信息泄露和逻辑缺陷漏洞，这些漏洞存在于 TCP/IP 网络堆栈中，允许攻击者绕过传统边界，且无需任何用户操作即可远程接管设备，包括 SCADA 设备、工业控制器、病人监护仪、MRI 机器、防火墙和打印机等关键设备。

3.2.1.2 篡改或伪造组件的漏洞

通过对近年来的网络攻击行为进行深入分析发现，恶意代码的传播和扩散占据很大的比例，尤其是伪造的恶意组件，其主要特征之一是披着正规厂商“合法”的外衣，使伪造恶意代码组件在传播速度与产生的恶劣影响方面有了大幅度的提升。

伪造的组件可以使所有信任被攻击厂商证书的机构都陷入被入侵的风险，攻击者可能通过此伪造的组件绕过安全检查进而进行更大范围的攻击活动。披着“合法”外衣的伪造组件，拥有极高的隐蔽性和传播性，能在用户和开发者无感知的情况下产生巨大的安全威胁。

利用伪造的组件进行软件供应链安全攻击的典型例子之一是 SolarWinds Orion 攻击事件。攻击者在成功入侵 SolarWinds 后，将其官网提供的 Orion 软件安装包替换成植入后门的版本，获取正规厂商的证书并利用其对自身进行签名，使其披上“合法”的外衣，躲避官网对软件的安全检查，从而实现软件供应链攻击。

3.2.1.3 编码过程引入的漏洞

随着软件已成为日常生活中不可或缺的一部分，保证底层源代码的安全性和完整性至关重要。但开发人员在编写代码时，由于缺乏安全意识以及不良的编码习惯、实践和策略往往会引入一些安全漏洞。随着软件规模的持续扩大，功能越来越复杂，编码过程引入的安全漏洞也越来越多；开发人员为了赶研发进度，很有可能会忽视自己使用的框架库中存在的安全缺陷，从而制造出潜在的安全风险，同时开发人员往往会大量复用软件模块，将导致安全漏洞不断延续。

除此之外，开发人员在编码过程中为了后续的测试工作往往会留下一些具有高级权限的账号，而在后续的编码过程中故意或误将这些具有高级权限账号留下，这种情况带来的软件漏洞极有可能被攻击者利用，读取软件中重要的敏感信息。

3.2.1.4 开源组件引入的漏洞

现代软件大多数是被“组装”出来的，而不是“开发”出来的。现在的软件开发过程类似于早期的工业生产活动，是以开源软件为基础原料，在此基础上，再结合实际的业务需求和应用场景补充添加相对独立的业务代码，最后“拼装”出一套软件系统。这种开发方式虽然在一定程度上提高了软件系统开发的效率，却并未充分考虑其使用的基础开源组件是否安全可靠，为软件系统的安全性和可控性带来了巨大的安全挑战。

随着开源技术快速形成生态，开源热度的持续升高，企业引入开源软件已成为一种趋势。Sonatype 发布的《2020 State of the Software Supply Chain》报告中提到现代应用程序中 90% 的组件都是开源的（如图 9 所示），其中 11% 的开源组件中存在已知的安全漏洞。随着开源组件使用的增加，风险面也在不断膨胀，使用含有已知安全漏洞的开源组件很有可能将安全缺陷引入到软件产品中，并随着软件的使用而扩散，进而对软件供应链产生巨大的安全威胁。

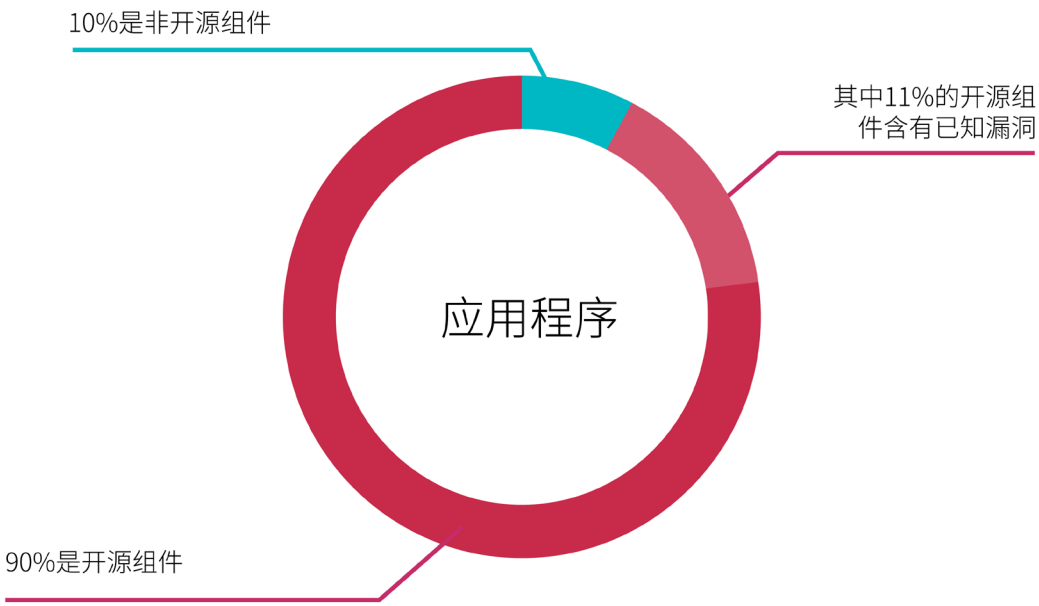

图 9 应用程序中组件的比例

通过不安全的开源组件所引发的典型安全事件是开源软件 Linux 中一个被频繁使用的程序“Bash”，被发现存在安全漏洞，其对计算机用户造成的威胁可能要超过“Heartbleed”漏洞。Bash 是用于控制 Linux 计算机命令提示符的软件，黑客通过利用 Bash 中的一个漏洞，对目标计算机系统进行完全控制。Bash 漏洞的严重程度在历史上被评为 10 级，意味着它具有着极大的影响力，而其利用的难度被评为“低”级，意味着攻击者可以比较容易地利用之发起网络攻击。利用这个漏洞，攻击者可能会接管计算机整个操作系统的权限，进而得以访问机密信息，并对系统进行篡改等操作，对整个软件供应链产生巨大的安全威胁。

3.2.2 漏洞状态类型

3.2.2.1 未知的漏洞

软件供应链中未知的漏洞往往指的是 0day 漏洞，是指负责应用程序的开发人员或供应商所未知的软件缺陷，因为该漏洞未知，所以没有可以利用的补丁程序。在所有的软件漏洞中，0day 通常能够造成最大的安全风险，针对

0day 漏洞的攻击很少立即被发现，发现这些缺陷通常需要数天到数年的时间，这使得这类漏洞往往会造成十分严重的安全威胁。

随着移动互联网、互联网 +、数字经济的快速发展，信息安全变得愈发重要，0day 自然也成为攻击者首选的目标，0day 漏洞可以通过售卖或悬赏获取，而出于自身利益，买卖双方都不会将漏洞信息公开，这进一步助长了 0day 漏洞的不可知性和危害性。

3.2.2.2 已知未处理的漏洞

当企业处于敏捷开发模式下时，为了提升效率节省时间成本，往往会引入大量的开源软件、第三方库和第三方供应商。在开发过程及后续软件运营的过程中，开发者会发现之前使用的开源软件或第三方供应商中存在历史漏洞，且这些漏洞中还存在部分并未发布补丁。部分企业出于时间成本的考虑，会放任这些漏洞不给予处理，这相当于埋下了一颗随时有可能被引爆的炸弹，带来安全隐患，一旦发生意外，将会产生不可挽回的影响。

出于利益考虑，攻击者往往会选择成本最低的攻击方式，攻击者会开发出各类自动化利用工具，寻找已经通过各种途径曝光的历史漏洞进行大规模的攻击行为，对攻击者而言，只要存在一定比例未进行漏洞修复的用户，那么这种攻击就是有收益可言的。因此，这种低成本、高收益的攻击往往会得到攻击者的青睐。

近些年，攻击者利用未修复的安全漏洞成功实现安全攻击的事件数不胜数，其中一个典型的通过历史漏洞进行大规模网络攻击的事件是在 2017 年，WannaCry 蠕虫通过 MS17-010 漏洞感染事件在全球范围内大爆发，全球范围内近百个国家遭到大规模的网络攻击，攻击者利用 MS17-010 漏洞，向用户机器的 445 端口发送精心设计的网络数据包，实现远程代码执行，被攻击者的电脑中大量高价值的文件被加密，同时被要求支付比特币以实现解密文件。

虽然 WannaCry 事件造成的影响正在逐渐被人们所遗忘，但 MS17-010 漏洞却一直存在于网络世界中，在 WannaCry 事件之后，又发生多起与 MS17-010 相关的安全事件，WannaMine、PowerGhost、Satan 等恶意软件均利用了历史漏洞 MS17-010 进行传播，造成十分恶劣的影响。

3.2.2.3 已处理的漏洞

当软件开发者或运营者完成了发现漏洞、缓解漏洞、治理漏洞的整个过程时，漏洞便进入了已处理的状态。现代软件常见的漏洞治理方法是使用供应商提供的漏洞补丁或自有编码人员进行漏洞代码修复。

需要注意的是，在漏洞处理完成后，软件开发者或运营者应及时使用漏洞检测类工具进行全面的漏洞检测。其目的之一是验证漏洞处理方法的有效性，确认之前发现的漏洞已被实际修复。同时，需要此步骤验证软件系统中是否因为修复过程而被引入了新的漏洞。

一个实际的例子是，2016 年 9 月 OpenSSL 官方为了修补 CVE-2016-6307 低危漏洞释放出了 1.1.0a 版本补丁，但这一版本在修复 CVE-2016-6307 漏洞的同时，却同时引入了等级为严重的 CVE-2016-6309 漏洞。

3.3 软件供应链的攻击类型

近年来，随着软件系统安全性的不断加强，网络攻击者开始采用软件供应链攻击作为击破关键基础设施的的重要突破口，从而导致软件供应链的安全风险日益增加。软件供应链攻击是指利用软件供应商与最终用户之间的信任关系，在合法软件正常传播或升级过程中，利用软件供应商的各种疏忽或漏洞，对合法软件进行劫持或篡改，从而绕过传统安全产品检查而达到非法攻击的攻击类型。

对于软件供应链攻击而言，主要可分为厂商预留后门、开发工具的污染、升级劫持、捆绑下载和源代码污染五种类型。（如图 10 所示）以下是根据软件供应链攻击的五种类型列举了近几年发生的典型软件供应链安全事件并进行分析。

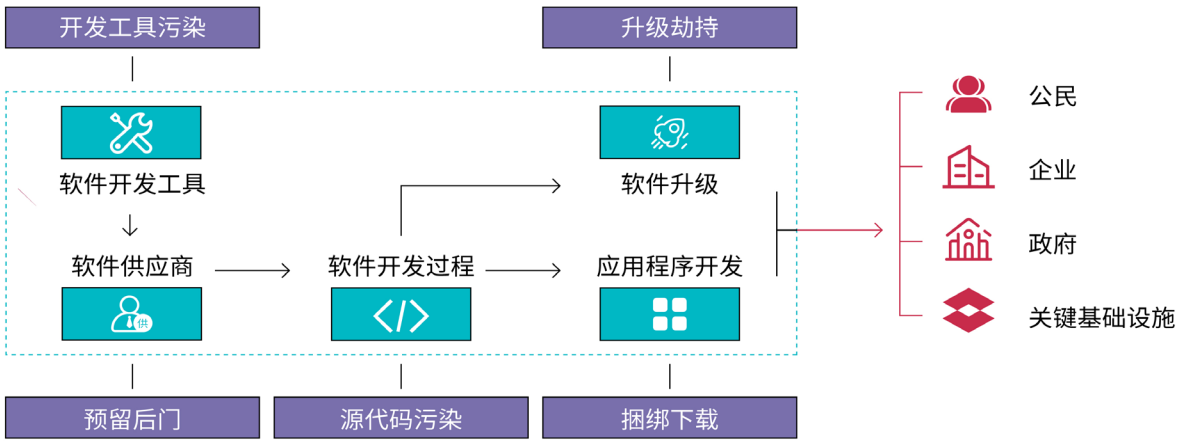

图 10 软件供应链攻击类型

3.3.1 预留后门

多数软件厂商在开发过程中出于为了方便后续测试或技术支持的考量，可能会预留一些具有高级权限的管理员账户，而当软件正式发布时忘记或故意留下“后门”。导致产品在发布之后被网络犯罪者所利用造成难以挽回的危害。某些厂商因为国家的需要，为国家安全部门特意预留一些暗藏的“接口”，方便国家获取用户敏感信息数据，进行监视。

棱镜门事件：2013 年 6 月，棱镜计划（PRISM）正式暴露在人们视线中，它是一项由美国国家安全局（NSA）自 2007 年开始实施的绝密电子监听计划，该计划的正式代号为“US-984XN”，直接进入美国国际网络公司的中心服务器里挖掘数据、收集情报，包括微软、雅虎、谷歌和苹果等在内的 9 家国际巨头均参与其中。其中以思科公司为代表的科技巨头利用其占有的市场优势在科技产品中隐藏“后门”，协助美国政府对世界各国实施大规模的信息监控，随时获取各国最新动态。思科公司多款主流路由器产品被爆出在 CPN 隧道通讯和加密模块存在预置式“后门”，即技术人员在源码编写过程中已经将“后门”放置在产品中，利用“后门”可以获取秘钥等核心敏感数据，几乎涵盖所有接入互联网使用的人群。

3.3.2 开发工具污染

攻击者对开发者常用的代码开发编辑器进行攻击，对开发工具进行篡改以及增加恶意模块插件。当开发者进行代码开发的时候，恶意模块将在代码中植入后门，经过被污染过的开发工具编译出的测试程序，或部署到生产业务中的程序，都将被植入恶意代码。

XcodeGhost 事件：2015 年 9 月 14 日，一例 Xcode 非官方版本恶意代码污染事件被披露，受到广泛关注，多数专业分析者称这一事件为“XcodeGhost”（如图 11 所示）。Xcode 是一款由苹果公司发布的运行在操作系统 macOS X Version 上的集成开发工具，是开发 OSX 和 iOS 应用程序的最主流的工具。攻击者利用通过官方渠道难以获取 Xcode 官方版本的情况，向非官方版本的 Xcode 注入病毒 XcodeGhost。其具体方法为修改 Xcode 软件的用于加载动态库的配置文件，在其中添加了非官方的具有恶意功能的 Framework 软件开发工具包，同时利用 Xcode 开发环境中使用的 Object-C 语言的扩展类功能这一特性重写系统应用启动时调用的函数，使得恶意代码能够随着应用的启动而自启动。多款知名的 APP 受到污染，受感染的 APP 被用于各种潜在的非法用途，对平台下用户隐私的安全造成了巨大的威胁。

图 11 XcodeGhost 攻击过程

3.3.3 升级劫持

软件产品在整个生命周期中几乎都要对自身进行更新，攻击者可以通过劫持软件更新的“渠道”，比如通过预先植入用户机器的木马病毒重定向更新下载链接、在运营商劫持下载链接、在下载过程中劫持网络流量等方式对软件升级过程进行劫持进而植入恶意代码。

NotPetya 勒索病毒事件：2017 年 6 月 27 日晚，据外媒报道，欧洲多个国家遭遇了 Petya 勒索病毒变种的 NotPetya 的袭击，数万台机器受到了感染。在这次事件中，病毒感染用户计算机系统的行为与传统的

勒索病毒大体一致，但是具有独特的网络感染手段。攻击者通过劫持乌克兰专用会计软件 me-doc 的升级程序，使得用户在更新软件时感染病毒。NotPetya 事件中劫持软件更新渠道的做法使其成为一次典型的软件供应链安全事件，同时攻击者利用大型组织供应链中的薄弱环节进行攻击的做法体现了供应链攻击的思想。

3.3.4 捆绑下载

攻击者可轻易通过众多未授权的第三方下载站点、云服务、共享资源以及破解版软件等方式对软件进行植入恶意代码，不仅如此，就连正规的应用商场由于审核不严等多种因素也会将被攻击者植入过含有恶意代码的“正规”软件分发给用户，造成重大安全威胁。

WireX BotNet 事件：2017 年 8 月 17 日，WireX BotNet 作为一种典型的僵尸网络通过控制大量安卓设备发动了较大规模的 DDoS 攻击。WireX 僵尸网络中的僵尸程序病毒通过伪装成普通的安卓程序，成功避过了谷歌应用商店的检测，伪装的安卓程序通过谷歌官方渠道被用户下载安装后将安卓主机感染成为僵尸主机。据相关报道，来自 100 多个国家的设备感染了 WireX BotNet。

3.3.5 源代码污染

软件产品如果在源代码级别就被攻击者植入恶意代码将难以被发现，并且这些恶意代码将在软件厂商的合法渠道下躲避对安全产品的检测，或许会长时间潜伏于用户设备中不被察觉。

SolarWinds Orion 攻击事件：2020 年 12 月 13 日，美国著名网络安全公司 FireEye 发布分析报告称，SolarWinds 旗下的 Orion 基础设施管理平台的发布环境遭到一名为 UNC2542 的 APT 黑客组织的入侵。黑客通过获取 SolarWinds 内网高级权限，创建了高权限账户，对源代码包进行了篡改，植入恶意代码添加了后门，该文件具有合法数字签名，其会通过该公司的官方网站进行分发，后门伪装成 Orion OIP 协议的流量进行通信，将其恶意行为融合到 SolarWinds 的合法行为中，可进行信息收集、读写删除文件等恶意行为。FireEye 称已在全球多个地区检测到攻击活动，包括北美、欧洲、亚洲和中东的一些政府、咨询和技术公司。

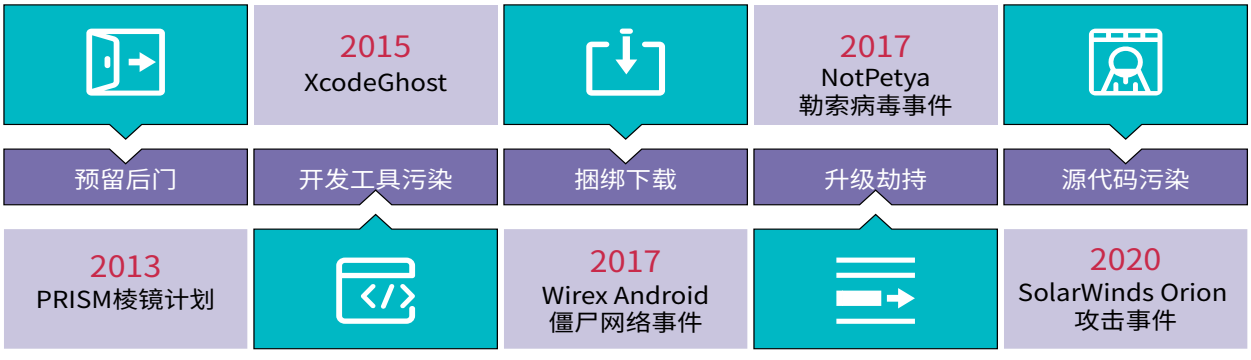

图 12 软件供应链典型攻击事件

除了上述提及的事件（如图 12 所示）之外，进一步根据软件供应链的攻击类型整理出部分典型安全事件（如表 1 所示），通过这些安全事件可以看出软件供应链安全攻击造成的影响远大于传统漏洞的影响。

针对软件供应链各环节的攻击在近几年都在呈上升趋势，在趋于逐渐复杂和多样化的互联网环境下，软件供应链所暴露给攻击者的攻击面和脆弱点越来越多，当这些漏洞被攻击者所利用时，将有可能造成严重的安全威胁。因此，针对安全形势日益严峻的软件供应链问题，需要得到更多的重视，开展进一步深入的研究。

时间	事件名称	攻击类型	事件描述
2015.11	mobiSage广告库被植入后门	预留后门	FireEye披露，mobiSage上千个APP被植入后门代码，通过服务端的控制，这些广告库可以做到录音和截屏、上传GPS信息、增删改查app数据、发送数据到服务器等功能。
2015.12	Juniper VPN后门事件	预留后门	Juniper公司发出声明，其防火墙、VPN设备使用的操作系统具有重大安全风险，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攻击、解密、篡改和注入。
2017.06	Fireball事件	捆绑下载	Fireball通过利用野马浏览器、Deal Wifi等多款流氓软件进行传播，从而劫持Chrome浏览器首页及新标签页。
2017.07	异鬼事件	捆绑下载	异鬼 II Bootkit木马通过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。
2017.09	PyPI第三方软件存储库被污染事件	开发工具污染	攻击者通过将软件包的名称拼写错误将恶意软件库上传到PyPI中，通过用户下载软件对用户信息进行收集。
2020.05	针对GitHub中Java项目的定向攻击	源代码污染	通过对开源项目植入恶意代码，导致开发人员开发的所有应用程序均受感染。
2020.10	NPM包管理工具攻击	源代码污染	攻击者通过向NPM注入四个名为Plutov-slack、Nodetest199、nodetest1010和npmpubman的恶意软件包，实现对受感染计算机的远程控制以及收集受感染计算机的系统信息。
2020.11	WIZVERA VeraPort攻击事件	升级劫持	APT组织Lazarus攻陷某些WIZVERA VeraPort网站，并修改了XML配置文件，分发带有签名的恶意文件，最后在用户机器上执行后门程序。
2020.12	SignSight攻击	源代码污染	ESET披露，“SignSight”攻击了越南政府的证书颁发机构，破坏了该机构的数字签名工具包，在合法软件的安全程序中插入名为PhantomNet的间谍软件工具。

表 1 软件供应链安全事件

04 软件供应链安全治理方法

随着国家对网络安全的愈发重视，软件供应链安全已经成为国家和业界重点关注的安全领域之一。软件供应链安全主要包括软件开发生命周期和软件生存运营周期，且与软件开发过程中的开发人员、环境、工具等因素密切相关。由于软件供应链中拥有太多不确定的因素，因此，软件在生产过程中很容易带有安全缺陷，并最终被非法分子恶意利用进而出现网络安全事件。

目前，业界已充分认识到造成网络安全事件出现的主要原因之一，是由于软件开发者在开发过程中对开发工具、开发团队、开发生命周期和软件产品自身管理不当，致使软件存在着安全缺陷，破坏或影响最终用户的信息安全。

通过推进针对软件生命周期进行全流程安全管控的落地实践，有助于从软件生命周期的源头保障软件供应链安全。

通过建立软件开发过程中保证软件供应链安全的体系化方法，为软件开发过程中尽可能避免和消除软件的安全缺陷、保证软件供应链安全奠定重要基础。从软件安全开发生命周期角度分析软件供应链安全的应用实践方法，主要有以下几个阶段。

4.1 体系构建阶段

4.1.1 SDL 软件安全开发生命周期

微软在 21 世纪初期的软件产品开发实践中，意识到无法通过技术层面彻底解决软件面临的安全风险。因此，微软尝试从流程和管理的角度解决这个问题，并探索在各个软件开发环节中加入安全过程、把控安全风险，确保每个环节交付到下一环节的交付物都安全可控。于是，针对传统的瀑布式模型微软提出了“SDL 软件安全开发生命周期”这一概念。

软件安全开发生命周期（SDL），是一个在帮助开发人员构建更安全的软件和解决安全合规要求的同时降低开发成本的软件开发过程。SDL 将软件开发生命周期划分为 7 个阶段（如图所示），并提出了 17 项重要的安全活动，旨在将安全集成在软件开发的每一个阶段，以减少软件中漏洞的数量并将安全缺陷降低到最小程度。SDL 更侧重的是软件开发的安全保障过程，旨在开发出安全的软件产品。

图 13 SDL 软件安全开发生命周期

在 SDL 的 7 个阶段中（如图 13 所示），SDL 要求前 6 个阶段的 16 项安全活动，为开发团队必须完成的安全活动。同时，SDL 认为开发团队应该保持灵活性，以便选择更多合适的安全活动，如人工代码分析、渗透测试、相似应用程序的漏洞分析，以确保对某些软件组件进行更高级别的安全分析。SDL 重视各种工具的使用，重心在从需求阶段到测试阶段的工具集，如威胁建模、静态源代码分析等工具。

SDL 的 7 个阶段主要的含义如下：

培训：针对开发团队进行安全意识与能力的培训，以确保 SDL 能有效实施并落地，同时针对新的安全问题与形势持续提升团队的安全能力；

需求：通过安全需求分析，定义软件产品安全实现过程中所需要的安全标准和相关要求；

设计：通过分析攻击面，设计相对应的功能和策略，降低和减少不必要的安全风险。同时通过威胁建模，分析软件的安全威胁，提出缓解措施；

实施：按设计要求，实现对应功能和策略，以及缓解措施涉及的安全功能和策略。同时通过安全编码和禁用不安全的 API，减少实施时导致的安全问题，尽量避免引入编码级的安全漏洞，并通过代码审计等措施来确保安全编码规范的实行；

验证：通过安全测试检测软件的安全漏洞，并全面核查攻击面，各个关键因素上的威胁缓解措施是否得以验证；

发布：建立相应的响应计划，进行最终安全检查，确认所有安全活动均完成后将最终版本发送给客户；

响应：当出现安全事件与漏洞报告时，及时实施应急响应和漏洞修复。在此过程中新发现的问题和安全问题模式，可用于 SDL 的持续改进过程中。

4.1.2 DevSecOps

DevSecOps 是一种全新的安全理念和模式，由 DevOps 的概念延伸和演变而来。其核心理念是安全是整个 IT 团队每个人的责任，需要贯穿从开发到运营整个业务生命周期每一个环节才能提供有效保障。

DevSecOps 覆盖编码阶段、测试阶段、预发布阶段、发布阶段、线上运营阶段，强调自动化与平台化，由 CI/CD 平台推动整个开发和运营流程自动化。DevSecOps 依赖于 DevOps 流程工具链（如图 14 所示），将威胁建模工具、安全编码工具、安全测试工具、容器安全检测工具、基线加固工具、漏洞管理工具等自动化工具无缝集成到 DevOps 流程中，进而实现开发 - 安全 - 运营一体化。

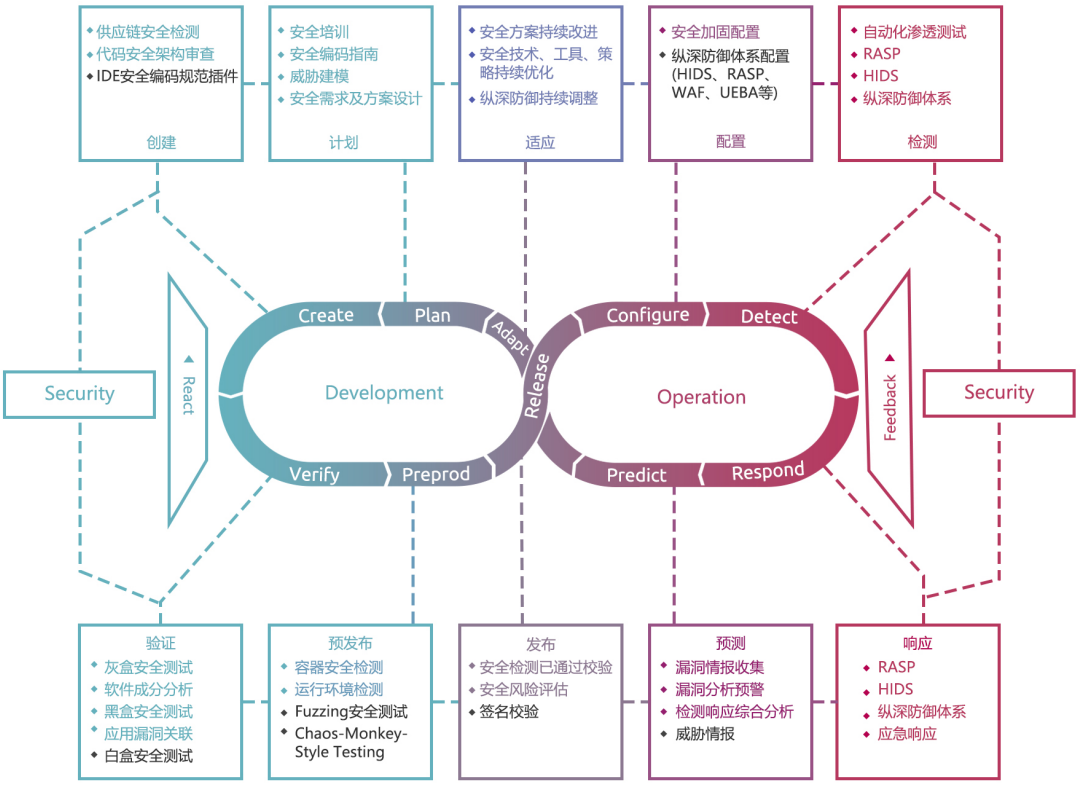

图 14 DevSecOps 工具链

很多企业在向 DevSecOps 转型时，会发现很多传统的安全工具在集成性和实用性上都难以满足 DevSecOps 的需求，因此，在 DevSecOps 的不同阶段需要采用不同的 DevSecOps 安全工具（如图 15 所示），这些安全工具主要的共同特点是高度自动化以及可集成性。

	集成阶段及自动化程度				
DevSecOps工具链	设计	开发	构建	测试	运营
SCA-OSSA （软件成分分析-组件分析）	◐	◑	●	●	n/a
SCA-OSG （软件成分分析-威胁治理）	n/a	●	●	●	n/a
SAST （静态应用安全测试）	n/a	◔	◔	n/a	n/a
DAST/Automated-PT （动态应用安全测试/自动化渗透测试）	n/a	n/a	n/a	◔	◔
IAST （交互式应用安全测试）	n/a	●	●	●	n/a
TM （威胁建模）	◐	n/a	n/a	n/a	n/a
RASP （运行时应用自保护）	n/a	n/a	n/a	n/a	◔

图 15 DevSecOps 安全工具在不同阶段的自动化程度

在软件供应链中每个阶段都在面临不同的安全风险，为了更好的对软件供应链进行风险治理，在 DevSecOps 模式下，安全开发工具链需要尽量覆盖软件生命周期中的所有阶段（如图 16 所示）。

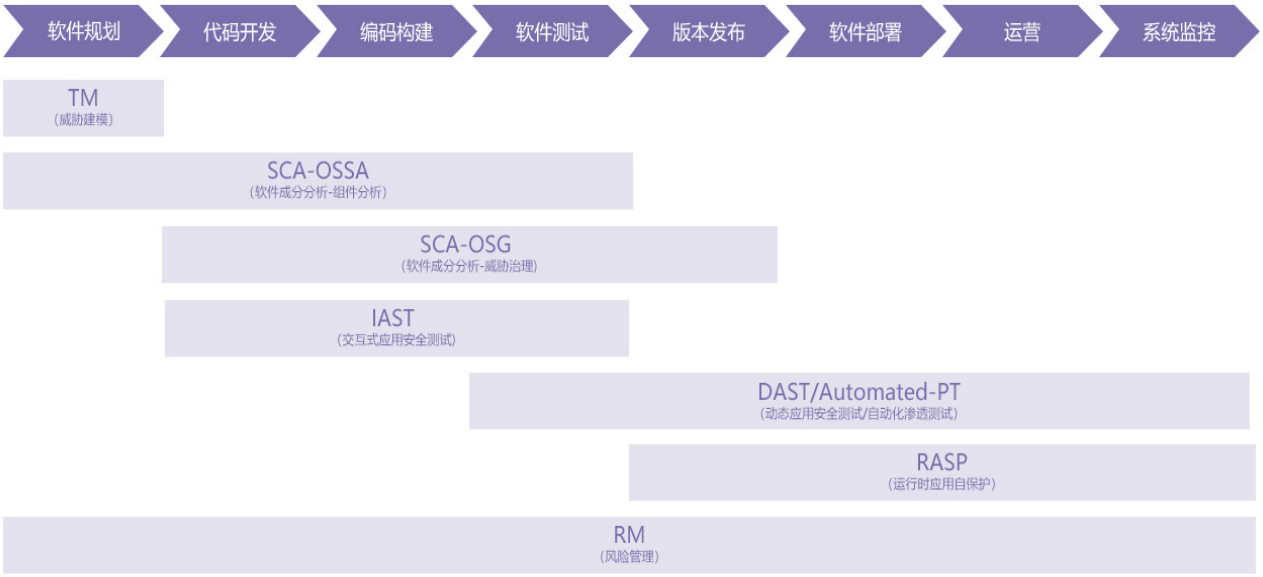

图 16 DevSecOps 模式下软件生命周期

除了以上所提到的工具之外，在 DevSecOps 的应用实践过程中，还有一些更为前瞻性的安全工具，具体可参考 DevSecOps 敏捷安全技术金字塔（如图 17 所示）。该金字塔在《2020 DevSecOps 行业洞察报告》中首次被提出，目前已发展到 2.0 版本。其描述了一组层次结构，金字塔底部是基础性技术，越上层的技术对 DevSecOps 实践成熟度的要求越高。

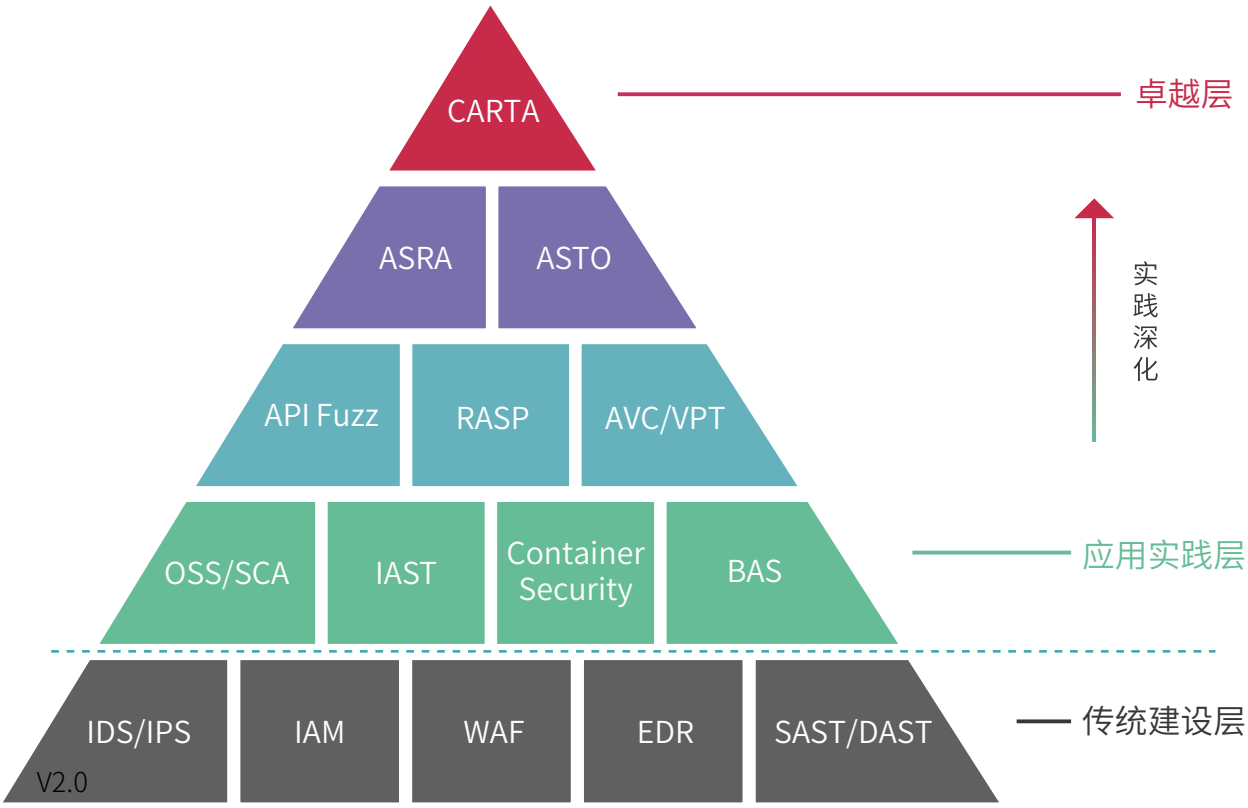

图 17 DevSecOps 敏捷安全技术金字塔 V2.0 版

DevSecOps 敏捷安全技术金字塔的不断升级，是为了给业界更好的预测和参考，关于 DevSecOps 敏捷安全技术未来演进方向的预测是开放且持续性的，随着 DevSecOps 实践的不断发展，其中的安全工具会进行调整和更新。

4.2 设计阶段

4.2.1 软件供应商风险管理流程

软件供应商风险是指与第三方供应商相关的任何可能影响企业利益或资产的固有风险。在选择第三方软件供应商时，为了避免因引入第三方供应商而带来众多潜在的安全风险，需要稳健的流程来识别和管理日益增加的软件供应商风险。因此，企业亟需构建有效且稳健的软件供应商风险管理流程。

构建完整的软件供应商风险管理流程可以提高软件供应链的透明度，同时帮助企业实现降低采购成本、识别和减轻供应商相关风险以及对软件供应商风险管理系统的持续优化改进。以下是针对软件供应商基本风险管理流程（如图 18 所示）。

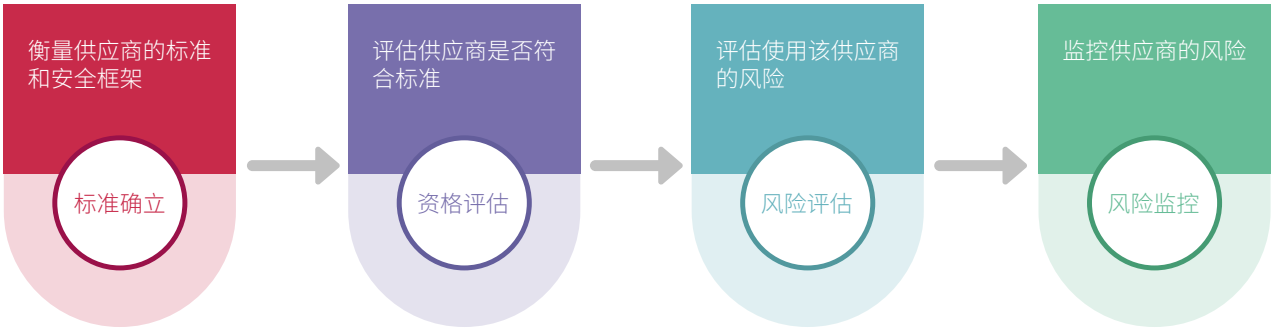

图 18 软件供应商风险管理流程

标准确立：结合企业的实际情况，构建软件供应商评估模型，制定软件供应商考核的评估标准及安全框架；

资格评估：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多维度的综合性资格评估，选出匹配度最高的供应商；

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；

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。

4.2.2 软件供应商评估模型

为了确保企业可以拥有较为稳定的供应链，提高企业的综合竞争力，经过多方面的综合考察分析，以下构建了一个系统化、结构化的软件供应商评估模型以供参考。软件供应商评估模型的关键意义在于从不同的维度对软件供应商进行评估，通过考察软件供应商的综合实力，以选择最合适的合作伙伴。以下是通过十二种不同维度对软件供应商评估的全过程（如图 19 所示）。

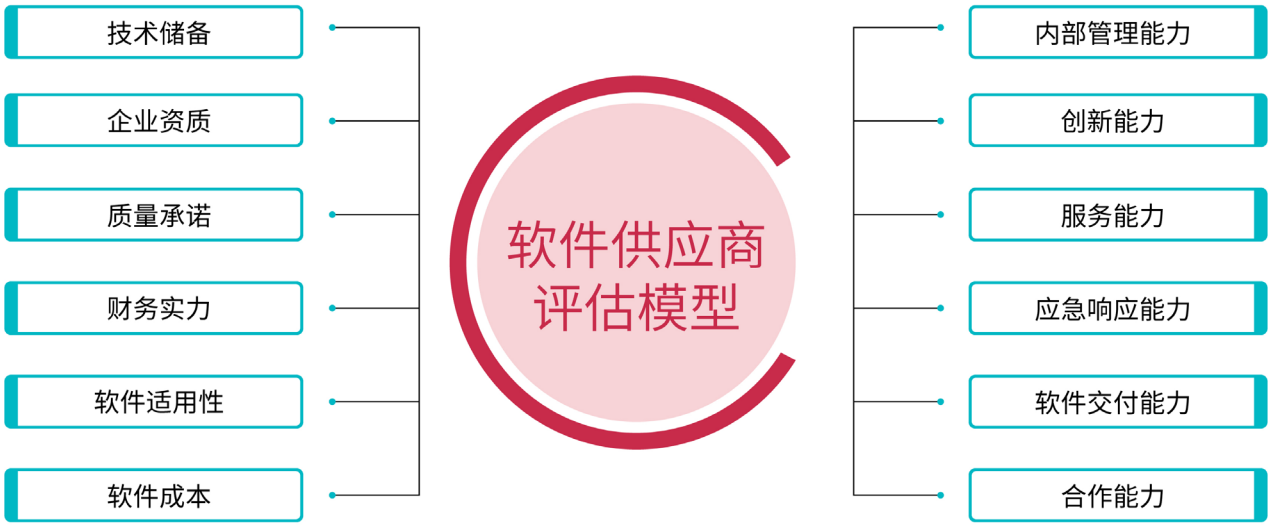

图 19 软件供应商评估模型

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；

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；

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；

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；

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；

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；

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；

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；

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；

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；

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。

4.2.3 软件供应商风险管理的作用

通过对软件供应商风险管理有助于企业更加高效准确地控制软件供应商带来的新的安全风险，以下是软件供应商风险管理的具体作用。

降低风险：通过对软件供应商进行彻底的审查可以识别其可能对业务构成的安全威胁的任何潜在弱点，根据软件供应商对企业业务的影响确定漏洞的重要性；

降低成本：通过对软件供应商风险进行充分的评估，可以以主动而非被动的方式应对安全威胁，减少潜在的安全风险，避免网络安全攻击或其他数据泄露等问题给企业造成的财务损失；

提高效率：通过对软件供应商进行实时风险监控，可以提前预知风险并及时做出响应，提高企业处理安全风险的效率；

培养长期合作关系：通过对供应商风险的管理、评估和跟踪监控，加强对供应商健康状况的监督，有助于培养可靠的长期合作关系。

4.3 编码阶段

4.3.1 构建详细的软件物料清单

软件供应链安全始于对关键环节的可见性，企业需要为每个应用程序持续构建详细的 SBOM（Software Bill of Material，软件物料清单）（如表 2 所示），从而全面洞察每个应用软件的组件情况。SBOM 是描述软件包依赖树的一系列元数据，包括供应商、版本号和组件名称等多项关键信息，这些信息在分析软件安全漏洞时发挥着重要作用。

字段	SPDX值	SWID值
供应商	(3.5) PackageSupplier:	<Entity> @role (softwareCreator/publisher), @name
组件	(3.1) PackageName:	<softwareIdentity> @name
唯一标识	(3.2) SPDXID:	<softwareIdentity> @tagID
版本	(3.3) PackageVersion:	<softwareIdentity> @version
组件散列值	(3.10) PackageChecksun:	<Pay1oad>/../<File> @[hash-algorithm] :hash
相互关系	(7.1) Relationship: CONTAINS	<Link>@re1, @href 个
SBOM编辑人	(2.8) Creator:	<Entity> @role (tagCreator), @name

表 2 软件物料清单示例

上表是一份软件物料清单示例，其中 SPDX 和 SWID 是两种国际通用的 SBOM 字段标准。SPDX（The Software Package Data Exchange，软件包数据交换）是 Linux 基金会下的开放性标准，其用于交流软件物料清单信息，包括组件、许可证、版权等信息。SPDX 通过为公司和社区共享重要数据提供通用格式来减少冗余工作，从而简化流程并提高合规性。SWID（Software Identification，软件标识）标签旨在为组织提供一种透明的方式来跟踪在他们的托管设备商安装的软件，它于 2012 年由 ISO 提出，并于 2015 年更新为 ISO/IEC 19770-2:2015。SWID 标签文件包含有关软件产品特定版本详尽的描述性信息。除表格中的两种应用最为广泛的 SBOM 字段标准外，还有 CycloneDX、CoSWID、CPE、Grafeas 等其他较为常见的标准，各标准的应用场景存在一定的区别。

SBOM 的概念源自制造业，其中物料清单是详细说明产品中包含的所有项目的清单。例如：在汽车行业，制造商会为每辆车维护一份详细的材料清单。此 BOM 列出了原始设备制造商自己制造的零件和第三方供应商的零件。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，并可以通知车主维修或更换。

同样，构建软件的企业也需要维护准确、最新的 SBOM，其中包括第三方和开源组件的清单，以确保其代码质量高、合规且安全。企业通过要求软件供应商提供 SBOM，以发现潜在的安全和许可证问题，或者应用程序是否使用过时的库版本。当发现此类问题时，管理员可以要求供应商使用较新版本重建应用程序，在等待更新的软件期间，安全人员有机会采取临时缓解措施来保护应用程序免受攻击者利用该漏洞进行攻击，还可以帮助安全人员在漏洞被披露或核心库发布新版本时 , 对应用程序和代码进行抽查以避免出现安全问题。

举个例子：如果安全人员手中有一份在其环境中运行的每个应用程序的物料清单，那么早在 2014 年 4 月，当 Heartbleed 漏洞最初被披露时，安全人员就无需测试每个应用程序中是否包含 OpenSSL，而是可以通过检查列表就立即知道哪些应用程序依赖于易受攻击的版本和需要采取的措施。

4.3.1.1 SBOM 生产流程

在成熟的体系下，SBOM 的生产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 20 所示）。

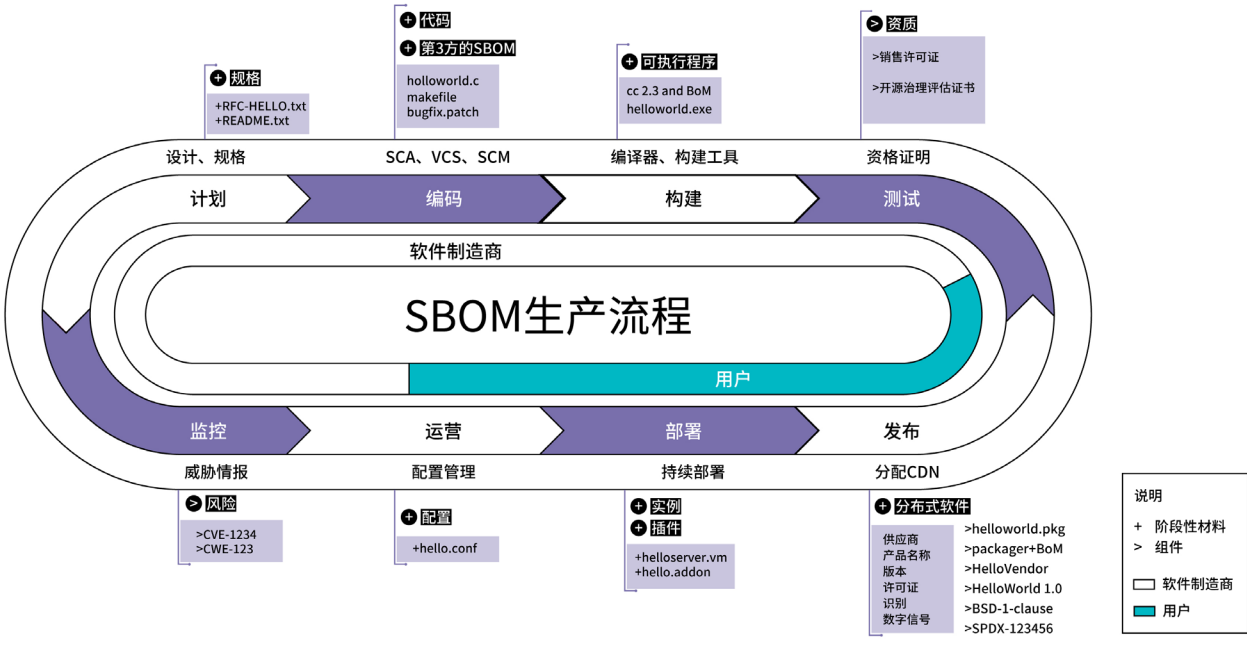

图 20 SBOM 生产流程

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 过程的标准和结构化的一部分。

为了确保持续一致性，应在测试工作中将 SBOM 作为测试用例的一部分，同时 SBOM 信息应随着使用工具和组件的更新而更新，使 SBOM 信息自动更新成为 CI/CD 管道中每个构建周期标准的一部分。在发布运营阶段使用 SBOM 可以在使用的库或组件存在漏洞时，以更快的时间检测出有哪些应用程序中存在这些漏洞，并及时进行修复

工作。

4.3.1.2 SBOM 可提高软件供应链的透明度

尽管 SBOM 对许多人来说依然很陌生，但其需求却不断呈现增长态势。Gartner 在其 2020 年的“应用程序安全测试魔力象限”中预测到：“到 2024 年，至少一半的企业软件买家要求软件供应商必须提供详细的、定期更新的软件物料清单，同时 60% 的企业将为他们创建的所有应用程序和服务自动构建软件材料清单，而这两组数据在 2019 年都还不到 5%。”

现代软件是使用第三方组件组装而成的，它们以复杂而独特的方式粘合在一起，并与原始代码集成以实现企业所需要的功能。在现代多层供应链中，单个软件可能有成百上千的供应商，从原材料来源到最终组装系统的全套供应商中找出某一组件的来源需要花费大量的时间和精力。因此，为所有组件构建详细准确的 SBOM，帮助企业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，检测软件组件是否带有已知的安全漏洞或功能漏洞。

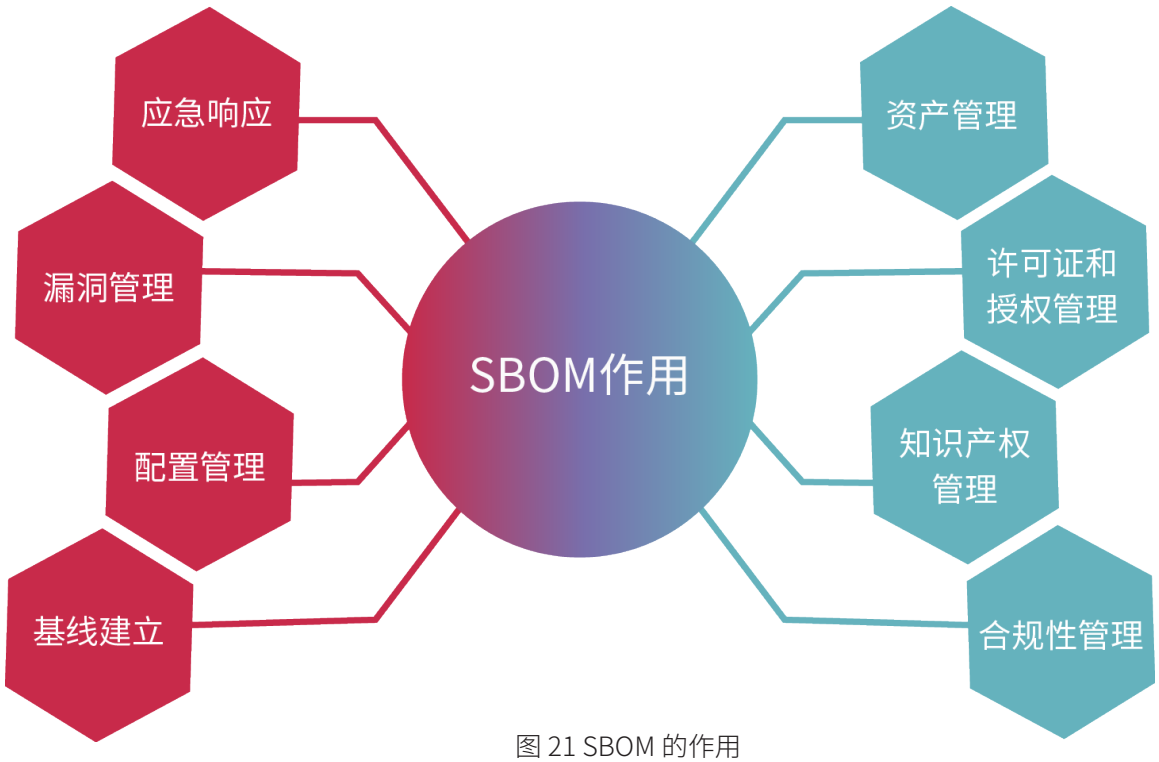

图 21 SBOM 的作用

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。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 21 所示）。

通过自动化创建 SBOM 可以在漏洞披露时及时地进行响应排查以及快速的安全修复，最小化软件供应链的安全风险；在开源组件和版本快速迭代的情况下，从风险管理的角度跟踪和持续监测闭源组件和开源组件的安全态势；同时 SBOM 列举了管理开源组件的许可证，可以保护企业免受不当使用相关的法律或知识产权的风险，保护应用程序在软件供应链中的合规性，避免将已知缺陷传递到软件供应链的下游。

4.3.1.3 SBOM 为漏洞风险治理节省大量时间

SBOM 的使用可以为软件供应链的漏洞治理节省大量时间。及时性对于企业在漏洞修复时是非常重要的。以往，企业在修复已部署系统的漏洞缺陷时往往需要几个月甚至是数年的时间，其重要原因之一是企业无法在漏洞出现的第一时间知晓该信息。软件供应链下游的企业需要等待上游软件供应商完成软件补丁，才可以进行漏洞修复，在等待的时间内，下游企业往往会面临无法预知的安全风险。而构建详细准确的 SBOM 则可以避免这一现象的发生，允许所有利益相关者在漏洞发现时立即开始评估漏洞，并开始制定相关的补救措施。以下通过一张对比图来说明 SBOM 对漏洞风险治理时间的影响（如图 22 所示）。

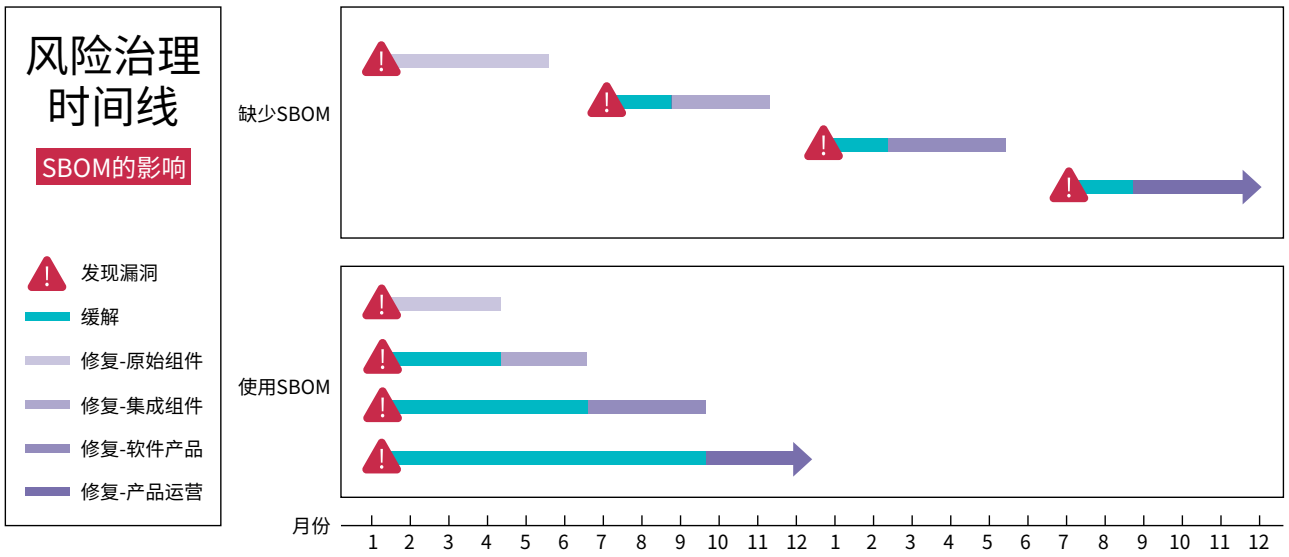

图 22 SBOM 对漏洞风险治理时间的影响

受感染的开源组件在软件中未被修复的每一分钟都会增加潜在被利用的风险，SBOM 有助于企业在漏洞披露的早期对漏洞进行识别，通过 SBOM 提供受感染开源组件和依赖项的准确位置，采取适当的步骤进行修改，为企业在风险分析、漏洞管理和补救过程中节省数百小时至数月的时间。

4.3.2 使用基于 SCA 技术的工具

企业需要谨慎、合理地选择、获取和使用第三方闭源组件和开源组件。软件安全团队或研发团队通过必要的技术手段确保所使用的第三方组件的安全性，及时获取所使用第三方组件和开源组件的漏洞情报，并适时做出响应。

软件成分分析（Software Composition Analysis，SCA）是一种对二进制软件的组成部分进行识别、分析和追踪的技术。SCA 可以生成完整的 SBOM，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。

软件成分分析分为两种模式，静态和动态。静态模式是使用工具对目标工程文件进行解压，识别和分析各个组件的关系；动态模式则是依赖于执行过程，在程序执行的同时收集必要的活动元数据信息，通过数据流跟踪的

方式对目标组件的各个部分之间的关系进行标定。

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，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。

某金融企业的业务团队无法接受速度的迟滞，在研发效率和编码速度的考量下，大量的软件应用都基于第三方的组件、开源代码、通用函数库实现，随之而来是绝大多数应用程序都包含开源组件的安全风险，为企业带来了许多未知的安全隐患。

为了更好地进行开源组件治理工作，该企业引入基于 SCA 技术的工具，与 DevOps 流程无缝结合，在流水线的测试阶段自动发现应用程序中的开源组件，提供关键版本控制和使用信息，并在 DevOps 的任何阶段检测到漏洞风险和策略风险时触发安全警报。所有信息都通过安全和开发团队所使用的平台工具实时发送，实现及时的反馈循环和快速行动。

在不改变该企业现有开发测试流程的前提下，将 SCA 工具与代码版本管理系统、构建工具、持续集成系统、缺陷跟踪系统等无缝对接，将源代码缺陷检测和源代码合规检测融入到企业开发测试流程中，帮助企业以最小代价落地源代码安全保障体系，降低软件安全问题的修复成本，提升软件质量。

4.4 发布运营阶段

4.4.1 建立成熟的应急响应机制

在软件的发布运营阶段，企业需要具备安全应急响应能力（如图 23 所示），能在软件发布后对发生在软件和软件补丁获取渠道的软件供应链安全事件、软件安全漏洞披露事件进行快速的安全响应，控制和消除安全事件所带来的安全威胁和不良影响，进而追溯和解决造成安全事件的根源所在。

发布运营阶段包括监测告警、应急响应、事件处置、持续跟进等关键活动。在日常的运营管理中，企业可以通过采用自动化分析技术对数据进行实时统计分析，发现潜在的安全风险，并自动发送警报信息。在有突发事件出现时，通过监测预警，安全人员可以迅速地进行安全响应，在最短的时间内确定相关解决方案并进行事件处置，在解决之后进行经验总结并改进。通过监测预警技术对软件系统进行实时自动监测，当发现安全问题时，立即发出警告，同时实现信息快速发布和安全人员的快速响应。

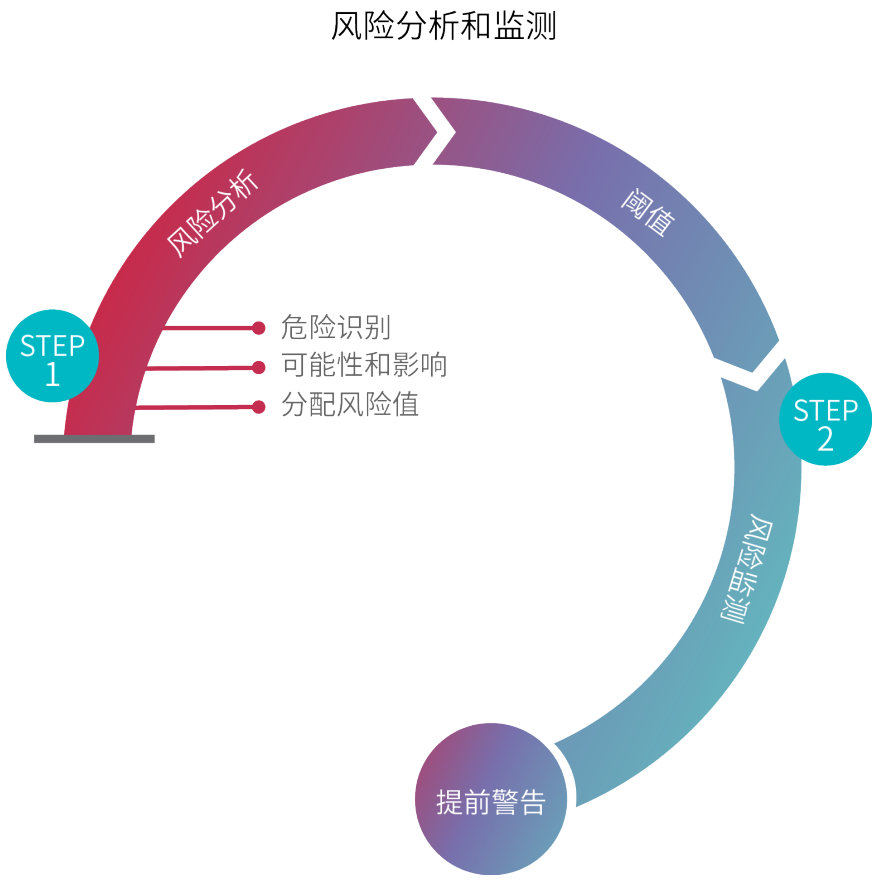

图 23 安全风险监测分析及响应

在发布运营阶段发生突发事件之后的应急响应与对安全事件进行处理的管理能力相关，因此，企业需要加强检测预警能力、提高应急响应速度、加快应急处置效率，从事后被动救火转化为主动应急管理。充分预估突发事件的场景，通过管理活动与技术手段避免突发事件的发生，在突发事件发生时能够及时监测预警，并有序进行处理行为。

由于在应用程序发布很久之后，仍有可能在其中发现新的安全漏洞，这些漏洞可能存在于构成应用程序的底层开源组件中，导致“零日”漏洞的数量不断增加。因此，企业需要制定事件响应和漏洞处理策略，与领先的漏洞研究机构进行合作，积极监控大量漏洞信息来源。同时，进行持续性的安全检查，定期的安全检查可以保护应用程序免受新发现的安全漏洞的影响。

4.4.2 构建完善的运营保障工具链

4.4.2.1 BAS

2017 年，Gartner《面向威胁技术的成熟度曲线》中首次提及入侵与攻击模拟（Breach and Attack Simulation，BAS）工具（如图 24 所示），并将其归到新兴技术行列。在 2021 年，Gartner 将 BAS 纳入“2021 年顶级安全和风险管理趋势”。

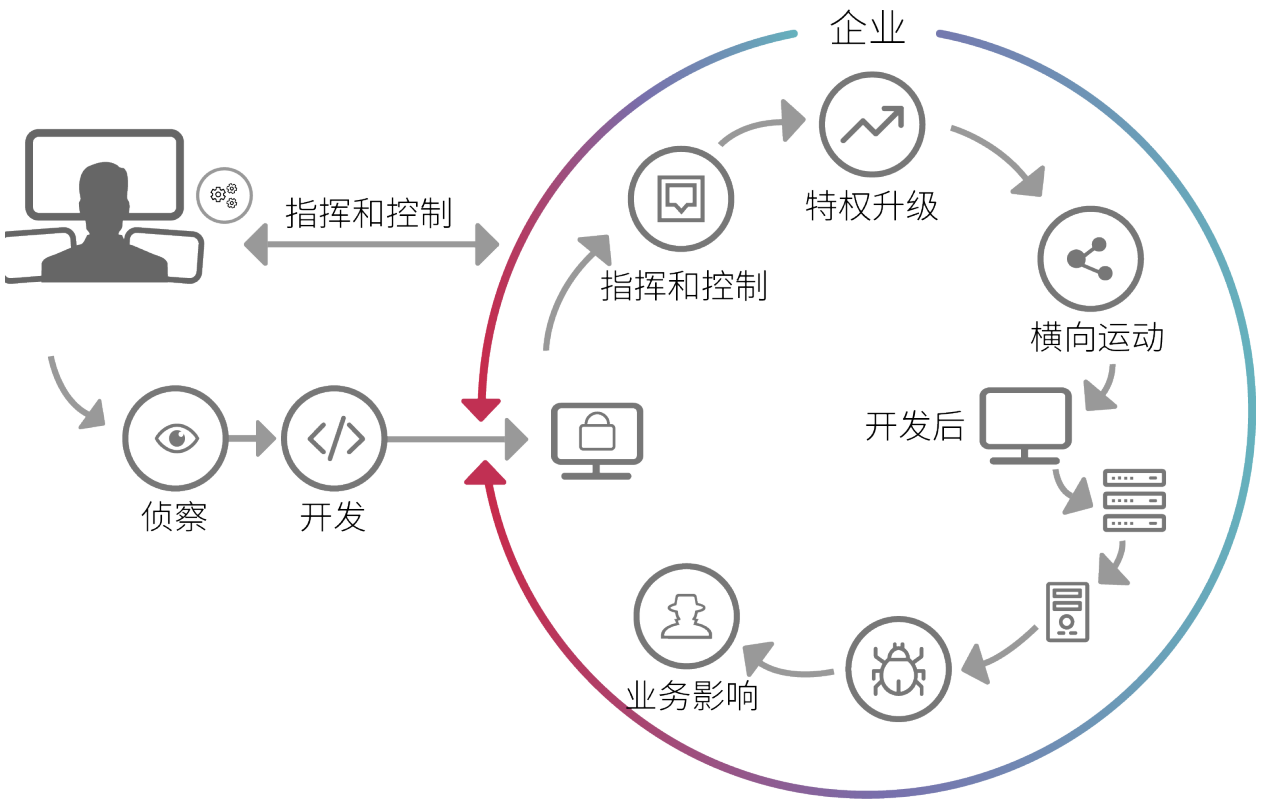

图 24 BAS 技术原理

BAS 通过模拟对端点的恶意软件攻击、数据泄露和复杂的 APT 攻击，测试组织的网络安全基础设施是否安全可靠，在执行结束时，系统将生成关于组织安全风险的详细报告，并提供相关解决方案。同时结合红队和蓝队的技术使其实现自动化和持续化，实时洞察组织的安全态势。

BAS 可以确定漏洞的覆盖范围并对检测出的漏洞提供补救意见，防止攻击者对漏洞加以利用。除了自动化和持续监控之外，BAS 还使安全团队改变了他们的防御方式，采取更为主动积极的策略，维护组织各个方面的安全。

4.4.2.2 WAF

早在 2007 年，国家计算机网络应急技术处理协调中心检测到中国大陆被篡改网站总数累积达 61228 个，比 2006 年增加了 1.5 倍。其中，中国大陆政府网站被篡改各月累计达 4234 个。为了更好的应对网络攻击，Web 应用防护系统也被称为网站应用级入侵防御系统（Web Application Firewall, WAF）应运而生，WAF 可以对来自 Web 应用程序客户端的各类请求进行内容检测和验证，确保其安全性和合法性，对非法的请求予以实时阻断，从而对各类网站站点进行有效的安全防护（如图 25 所示）。

图 25WAF 技术原理

WAF 通过增强输入验证，可以在运营阶段有效防止网页篡改、信息泄露、木马植入等恶意网络入侵行为，从而减小 Web 服务器被攻击的可能性。同时，WAF 还可以判断用户是否是第一次访问，将请求重定向到默认登录页面并且记录时间，通过检测用户的整个操作行为可以更容易识别攻击。

4.4.2.3 RASP

作为第一道防线，WAF 能够阻止基本攻击，但难以检测到 APT 等高级威胁，不仅如此，企业需要持续“调整”WAF 以适应不断变化的应用程序，这一过程消耗了安全管理人员大量的精力。此时，运行时应用程序自我保护技术（Runtime Application Self-protection，RASP）（如图 26 所示）作为新一代运行时保护技术被引入，RASP 可以提供更深入的保护能力，更广泛的覆盖范围，并且可以花费更少的时间。

RASP 将保护代码像一剂疫苗注入到应用程序中，与应用程序融为一体，使应用程序具备自我保护能力。RASP 结合应用的逻辑及上下文，对访问应用系统的每一段堆栈进行检测，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，无需人工干预，最终实现软件应用的自我保护，确保软件应用的安全运行。

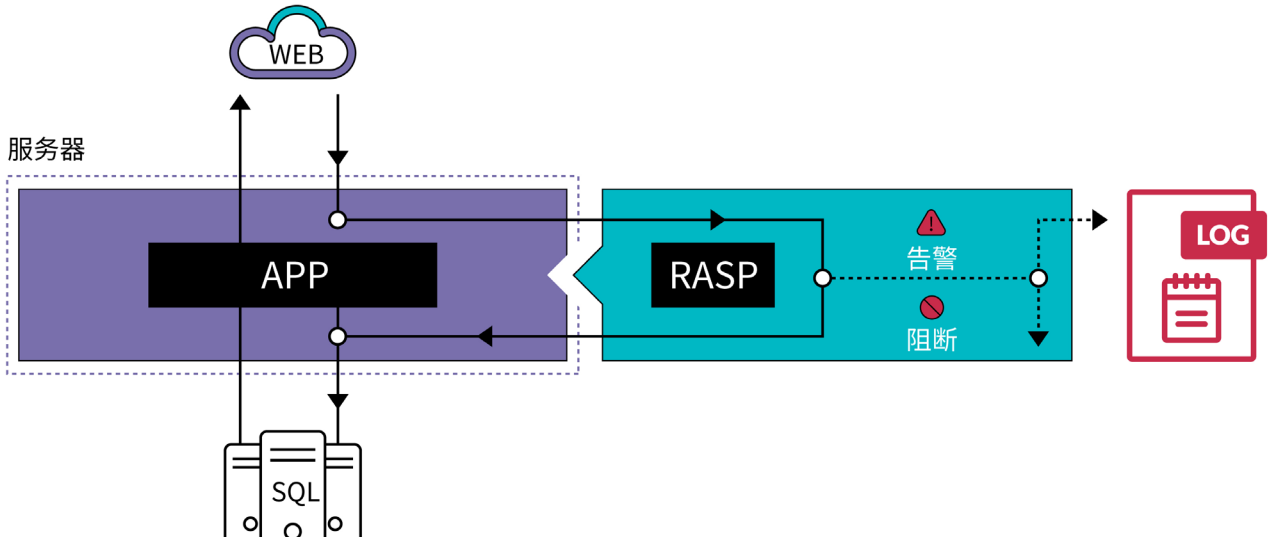

图 26 RASP 技术原理

RASP 在运营阶段可以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安全保护，可以精准识别应用运行时暴露出的各种安全漏洞，进行深度且更加有效的威胁分析，快速定位应用漏洞，大大提升修复效率，保障应用程序的安全性。

4.4.2.4 威胁情报平台

通过建立威胁情报平台（Threat Intelligence Platform，TIP）（如图 27 所示），可以帮助安全人员明确企业的在线资产和安全状况，根据企业自身资产的重要程度和影响面，进行相关的漏洞修补和风险管理；同时可以帮助安全人员了解企业自身正在遭受或未来面临的安全威胁，提供解决建议。

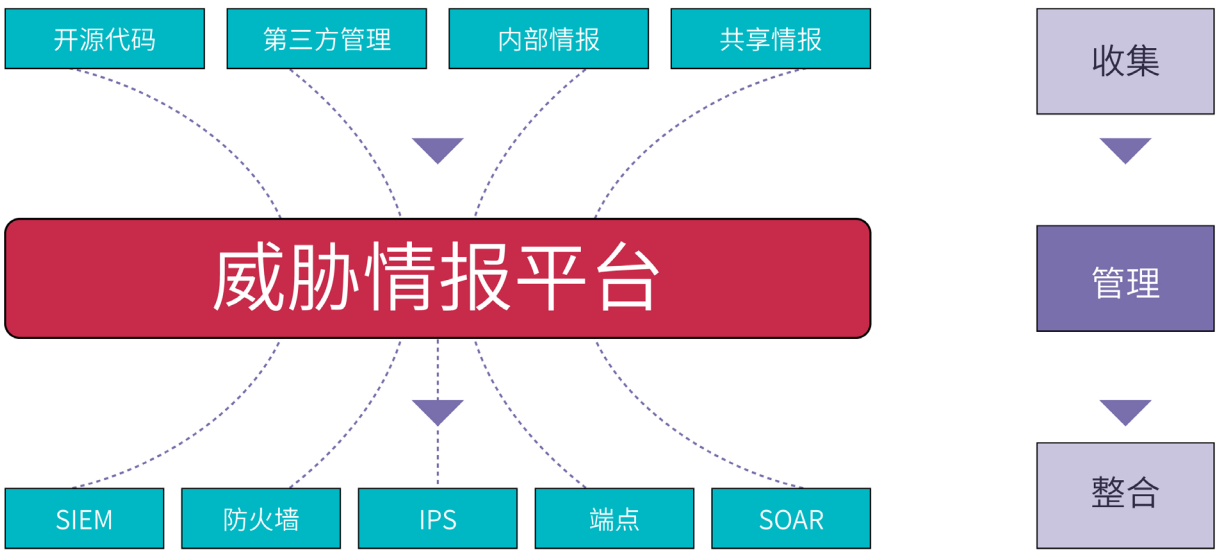

图 27 威胁情报平台原理

威胁情报平台与各类网络安全设备和软件系统协同工作，为威胁分析和防护决策提供数据支撑，通过对全球网络威胁态势进行长期监测，以大数据为基础发布威胁态势预警，实时洞悉风险信息，进而快速处置风险。

4.4.2.5 容器安全工具

在发布运营阶段，通过使用容器安全工具（Container Security）（如图 28 所示），可以自动化构建容器资产相关信息，提供容器环境中各类资产的状态监控，包括容器、镜像、镜像仓库和主机等基础资产信息，使资产拥有较强的可扩展能力；通过建立智能应用补丁扫描工具，为安全人员提供镜像管理、镜像检测以及自动化补丁修复建议。

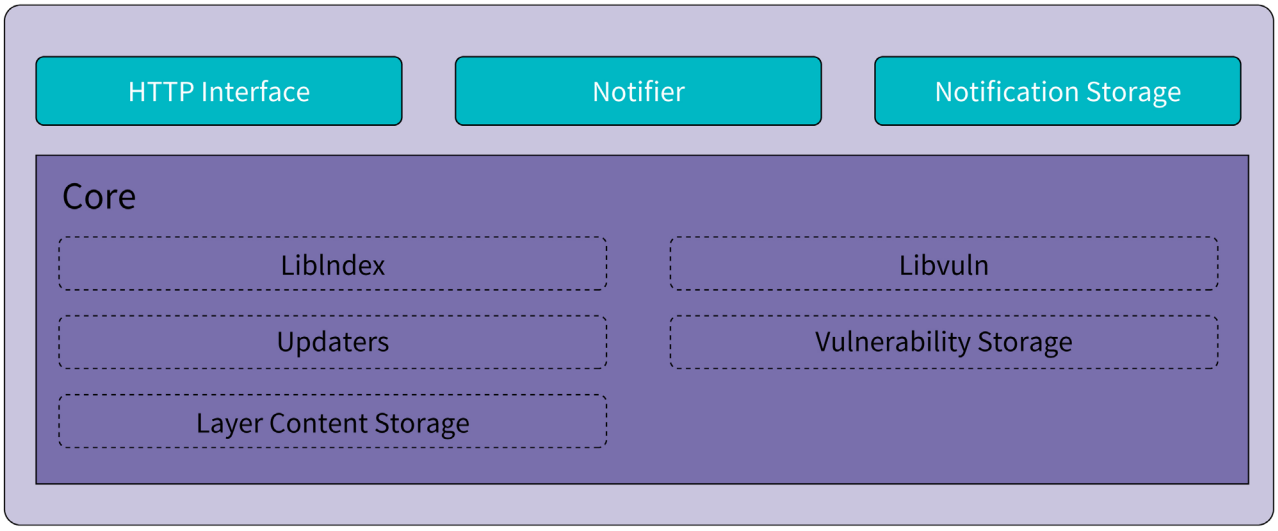

图 28 某容器安全工具架构

为了更好地应对未知和迅速变化的攻击，容器安全工具可以对数据进行持续监控和分析，通过结合系统规则、基线和行为建模等要素，自适应识别运行时容器环境中的安全威胁；建立一键自动化检测机制，给安全人员提供可视化基线检查结果，同时将企业现有的安全技术与持续运营的安全模型相结合，实现持续化的动态安全检测。

05 软件供应链安全应用实践

5.1 可信研发运营安全能力成熟度模型

中国信息通信研究院云计算与大数据研究所自 2019 年起，联合业界众多头部厂商专家制定《可信研发运营安全能力成熟度模型》标准，提出可信研发运营安全能力体系框架（如图 29 所示）。可信研发运营安全能力体系框架的构建继承 SDL 与 DevSecOps 的核心理念，安全前置，汲取 SDL 与 DevSecOps 体系的优点，优化具体安全实践要素，是一种贯穿研发运营全生命周期的安全理念。

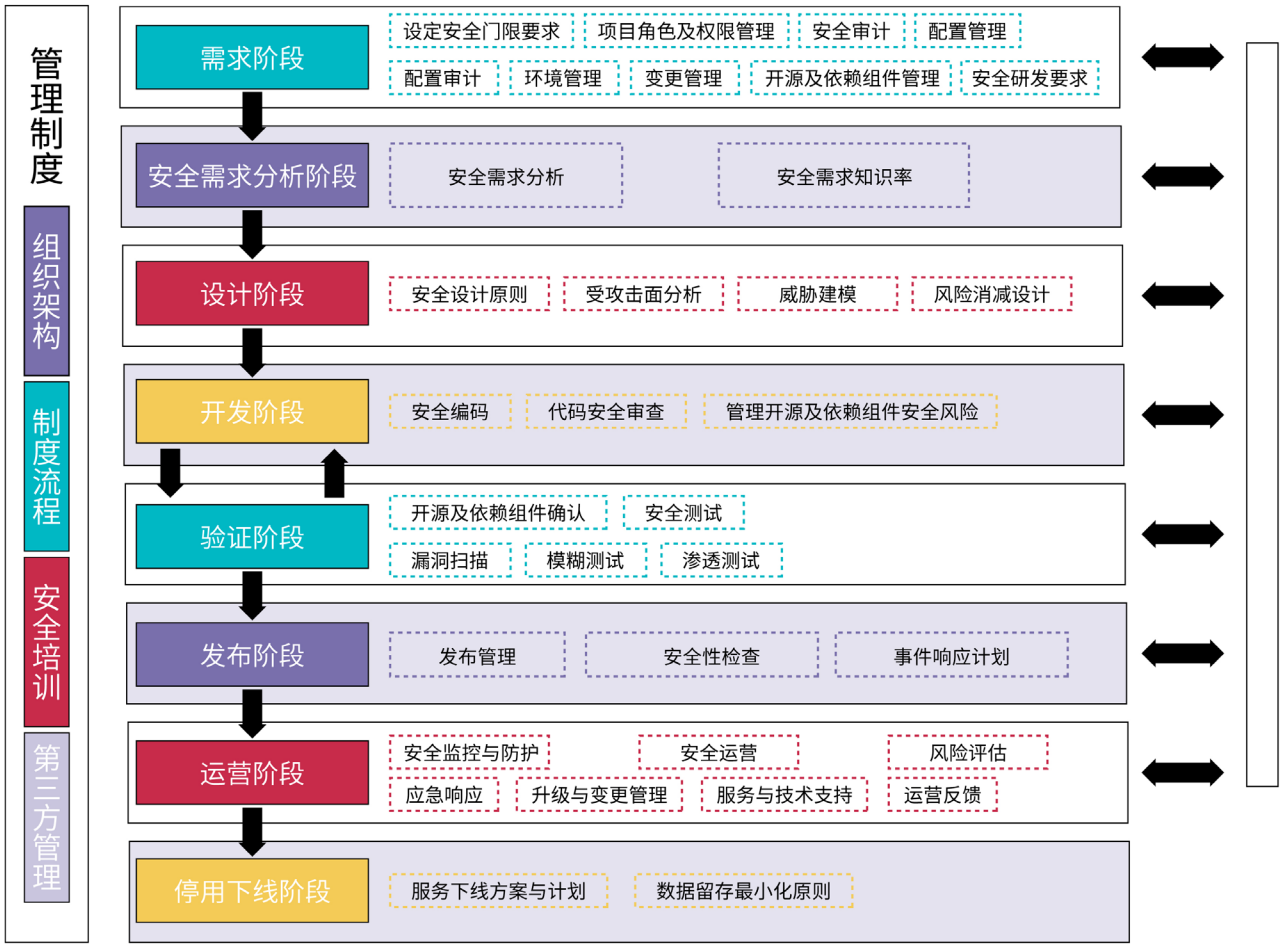

图 29 可信研发运营安全能力体系框架

可信研发运营安全能力体系框架强调安全左移，分为管理制度以及涉及软件应用服务全生命周期的需求阶段、安全需求分析阶段、设计阶段、开发阶段、验证阶段、发布阶段、运营阶段和停用下线阶段八大部分，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。根据各阶段安全要素达标情况，可信研发运营安全能力成熟度模型共分为 3 个级别，自低向高依次为基础级、增强级和先进级。

5.2 云安全共享责任模型

在云计算环境下，云服务提供商与云租户需要进行软件供应链安全共治，但云服务普遍存在安全责任划分不清晰与治理措施不明确等问题，为了解决这些问题，2019 年，微软在《Shared Responsibility in the Cloud》中提到云安全共享责任模型（如图 30 所示）。云安全共享责任模型指出，在基础设施即服务（IaaS）、平台即服务（PaaS）和软件即服务（SaaS）三种不同的云服务模式下，云服务提供商（Cloud Service Provider，CSP）和客户之间需要分担的安全责任不同。CSP 需要承担客户在使用云服务时保障物理安全的责任，客户需要负责确保其解决方案及其数据被安全地识别、标记和正确的分类，以满足任何合规义务的责任，其余的责任则由 CSP 和客户共同承担。

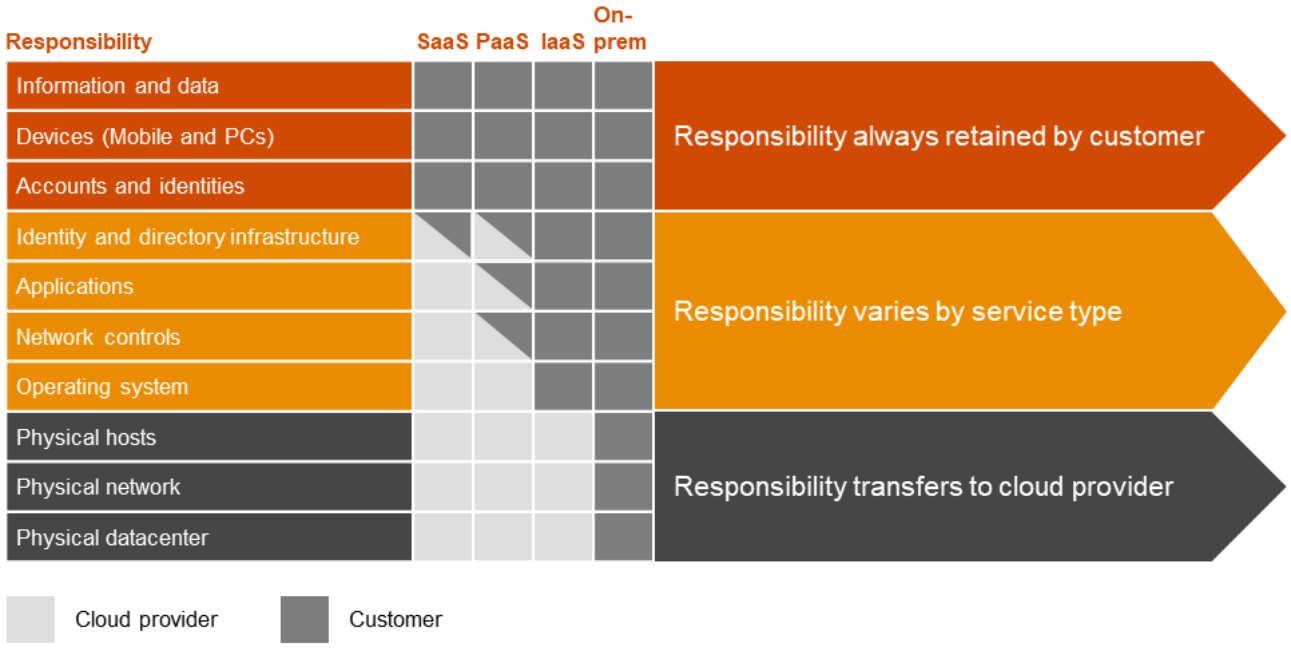

图 30 云安全共享责任模型

在构建以混合云作为运行环境的软件程序时，应仔细评估应用程序的依赖性和安全影响，通过使用成熟的 DevSecOps 模型可以帮助组织评估整个软件供应链，并确定需要严格控制的安全关键点。

为了提高可见性和支持混合云体系结构，许多云服务商显示或允许应用程序接口（API）与安全进程的交互。不成熟的 CSP 可能不知道如何以及在多大程度上向客户提供 API。例如，通过检索日志或权限控制的 API，云租户可以获得敏感性较高的信息。然而这些 API 可以帮助云租户检测到未经授权的访问行为，因此 API 的开放是必要的。

5.3 Grafeas 开源计划

Grafeas（希腊语中的“scribe”）是由 Google 发起，联合包括 Redhat、IBM 在内的多家企业共同发布的开源计划。Grafeas 是一个开源工件元数据 API（如图 31 所示），它提供了一种统一的方式来审计和管理软件供应链。Grafeas 定义了一个 API 规范，用于管理软件资源，例如容器镜像、虚拟机镜像、JAR 文件和脚本。组织可以使用 Grafeas 来定义和聚合有关项目组件的信息。Grafeas 为组织提供了一个中央事实来源，用于在不断增长的软件开发团队和管道中跟踪和执行策略。构建工具、审计工具和合规性工具都可以使用 Grafeas API 来存储和检索有关各种软件组件的综合数据。

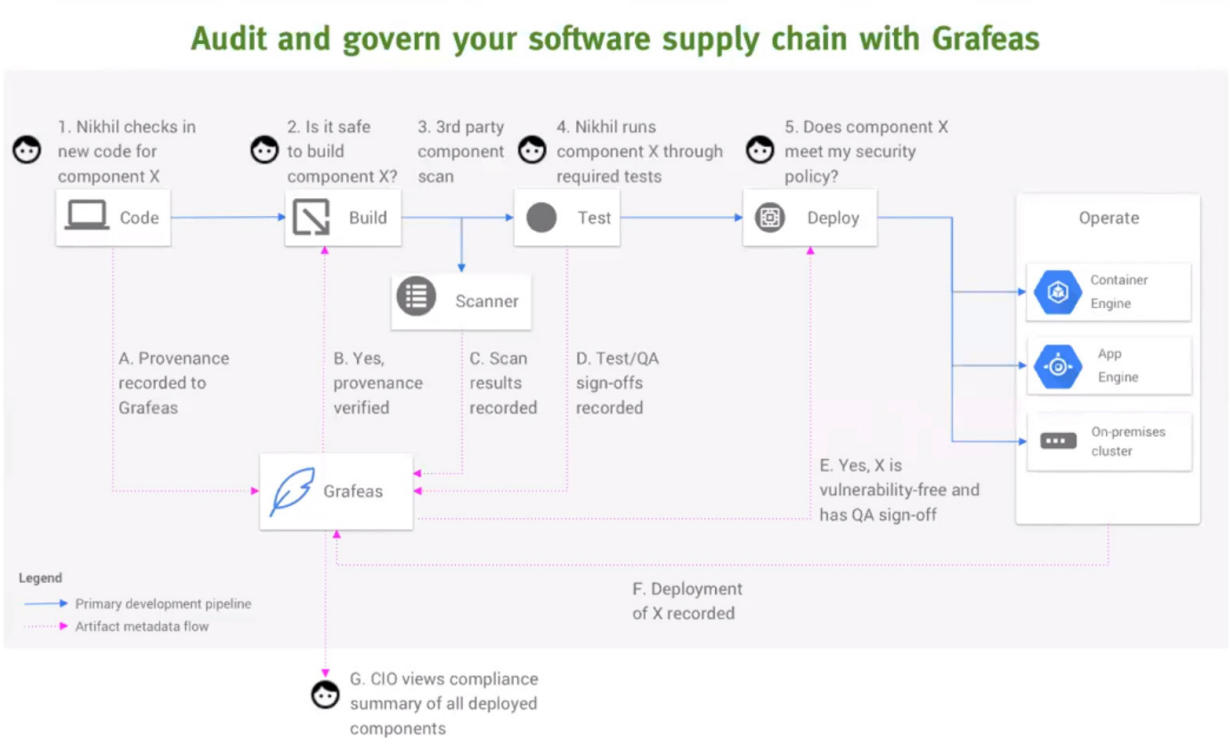

图 31 Grafeas 开源计划原理

同时，作为 Grafeas 的一部分，Google 推出了 Kritis，其是一种 Kubernetes 策略引擎，通过在部署前对容器镜像进行签名验证，可以确保只部署经过可信授权方进行过签名的容器镜像，降低在环境中运行意外或恶意代码的风险。

Grafeas 为组织成功管理其软件供应链所需的关键元数据提供了一个集中的、结构化的知识库。它反映了 Google 在数百万个版本和数十亿个容器中构建内部安全和治理解决方案所积累的最佳实践，包括：

- 使用不可变的基础设施来建立针对持续性高级威胁的预防性安全态势；
- 基于全面的组件元数据和安全证明，在软件供应链中建立安全控制，以保护生产部署；

• 保持系统的灵活性，并确保围绕通用规范和开源软件的开发人员工具的互操作性。

Grafeas 旨在帮助组织在现代软件开发环境中应用上述提到的最佳实践，实现以下功能和设计要点：

•全面覆盖：Grafeas 根据软件组件的唯一标识符存储结构化元数据，因此组织无需将其与组件的注册表放在一起，它可以存储来自不同储存库组件的元数据；

• 混合云适配性：组织可以使用 Grafeas API 作为中央通用元数据存储；

• 可插入：Grafeas 可轻松添加新的元数据；

• 结构化：针对常见的元数据类型的结构化元数据模式，让组织可以添加新的元数据类型，并且依赖于 Grafeas 工具可以管理这些新数据；

• 访问控制：Grafeas 允许组织控制多个元数据的访问；

• 查询能力：使用 Grafeas 的组织可以轻松查询所有组件的元数据，不必解析每个组件的单一报告。

在软件供应链的每个阶段，不同的工具会生成有关各种软件组件的元数据，示例包括开发人员的身份、代码、何时签入和构建、检测到哪些漏洞、哪些测试通过或失败等，然后 Grafeas 会捕获此元数据。

分析与展望

PROSPECT

数字经济时代，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，安全形势更加严峻。

然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。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，积极推动区块链等新技术在软件供应链安全领域的推广和应用，利用区块链的安全可信机制，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” （DSO 2021）的主题被定为“安全从供应链开始”的主要原因。此外，如何从技术创新的角度，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。

出品人 子芽

2021 年 8 月

引用参考

[1] Stephen Elliott. Introducing Grafeas: An open-source API to audit and govern your software supply chain[EB/OL]. 2017[2021].
https://cloud.google.com/blog/products/gcp/introducing-grafeas-open-source-api-，2017-10-12

[2] Sonatype. 2020 State of the Software Supply Chain[R]. USA:Sonatype, 2020.

[3] Forrester. The State Of Application Security,2021[R]. USA:Forrester., 2021.

[4] NCSC. Software Supply Chain Attacks[R]. USA:NCSC, 2021.

[5] Synopsys. 2021 Open Source Security And Risk Analysis Report[R]. USA:Synopsys, 2021.

[6] NTIA. Survey of Existing SBOM Formats and Standards[R]. USA:NTIA, 2019.

[7] NTIA. Roles and Benefits for SBOM Across the Supply Chain[R]. USA:NTIA, 2019.

[8] Vercode. Open Source Edition[R]. USA:Vercode, 2021.

[9] Herr T, Lee J, Loomis W, etal. BREAKING TRUST: Shades of Crisis Shades of Crisis Across an Insecure Across an Insecure Software Supply Chain[R]. USA:Atlantic Council, 2020.

[10] Veracode. State of Software Security[R]. USA:Veracode, 2020.

[11] Microsoft. Key considerations for software supply chain security in the cloud[R]. USA:PwC, 2019.

[12] Riel B V, Kuijpers S, Koning R D. Using the Software Bill of Materials for Enhancing Cybersecurity[R]. USA:Capgemini, 2021.

[13] Microsoft. SharedResponsibility for Cloud Computing[R]. USA:Microsoft, 2019.

[14] NIST. Best Practices in Cyber Supply Chain Risk Management[R]. USA:NIST, 2020.

[15] McKinsey & Company. In supply-chain risk management, organizations often don’t know where to start. We offer a practical approach[EB/OL]. 2019[2021]. https://www.mckinsey.com/business-functions/operations/our-insights/a-practical-approach-to-supply-chain-risk-management#.

[16] Smartsheet. Vendor Assessment and Evaluation Simplified[EB/OL]. 2020[2021].
https://www.smartsheet.com/content/vendor-assessment-evaluation.

[17] Disc. Discovery for Software Bill of Material (SBOM)[EB/OL]. 2021[2021]. https://blog.jdisc.com/2021/01/29/discovery-for-software-bill-of-material-sbom/.

[18] Imperva. Why Software Supply Chain Attacks Are Inevitable and What You Must Do to Protect Your Applications[EB/OL]. 2021[2021]. https://www.imperva.com/blog/software-supply-chain-risks-and-attack/.

[19] Debmalya Biswas. Enterprise Open Source Governance and Scanning Tool[EB/OL]. 2020[2021]. https://debmalyabiswas.medium.com/open-source-governance-and-scanning-tool-aa7cfd228b31.

[20] INSIDER PRO. Forget the users, the threat starts in the software supply chain[EB/OL]. 2021[2021]. https://www.idginsiderpro.com/article/3611543/forget-the-users-the-threat-starts-in-the-software-supply-chain.html.

[21] WhiteSource. Software Supply Chain Attacks[EB/OL]. 2021[2021]. https://www.whitesourcesoftware.com/resources/blog/software-supply-chain-attacks/.

[22] CSO. 6 most common types of software supply chain attacks explained[EB/OL]. 2021[2021]. https://www.csoonline.com/article/3619065/6-most-common-types-of-software-supply-chain-attacks-explained.html.

[23] ISS Group. Supply Risk Assessment[EB/OL]. 2021[2021]. https://issgroup.com/supply-risk-assessment/.

[24] GitHub. What is Clair[EB/OL]. 2021[2021]. https://github.com/quay/clair/blob/main/Documentation/whatis.md.

[25] NIST. Tips on Enhancing Supply Chain Security[EB/OL]. 2021[2021]. https://www.bankinfosecurity.com/tips-on-enhancing-supply-chain-security-a-16479.

[26] CSO. Supply chain attacks show why you should be wary of third-party providers[EB/OL]. 2021[2021]. https://www.csoonline.com/article/3191947/supply-chain-attacks-show-why-you-should-be-wary-of-third-party-providers.html.

[27] 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.

[28] 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.

[29] 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).

[30] 阿里云开发者社区 . 可信云原生软件供应链 (Software Supply Chain) - Kritis 项目调研 [EB/OL]. 2020[2021]. https://developer.aliyun.com/article/747637.

[31] 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.

[32] 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.

[33] 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.

[34] 张世琨 马森 高庆 孙永杰 . 北京大学专家：软件供应链安全的风险和成因分析 [J]. 中国信息安全 , 2018, 1(11).

[35] 阿里安全应急响应中心 . 软件供应链安全威胁：从“奥创纪元”到“无限战争” [EB/OL]. 2019[2021]. https://www.sohu.com/a/298892524_120055360.

[36] OSC 开源社区 . 开源和云原生带来的下一代软件供应链，正面临新的攻击 [EB/OL]. 2020[2021]. https://mp.weixin.qq.com/s/VTQ2C6eX7jYjGoqIn1axjQ.

[37] Linux 基金会 . 开源软件供应链安全报告 [R]. 美国 :Linux 基金会 , 2020.

[38] 汤志敏、汪圣平 . Kubernetes 时代的安全软件供应链 [EB/OL]. 2019[2021]. https://mp.weixin.qq.com/s/HBdSP3qT_W7a_RaHn_wyAQ.

[39] IMPERVA. 软件供应链攻击的 5 种方法，第 1 部分：攻陷供应商 [EB/OL]. 2021[2021]. https://mp.weixin.qq.com/s/VbObqpfCTYMNKTKl3fnJNQ.

[40] Imperva. 软件供应链攻击的 5 种方法，第 2 部分：第三方程序利用、开源库漏洞利用、程序依赖包混淆 [EB/OL]. 2021[2021]. https://mp.weixin.qq.com/s/kck64ZhNeloiHpaWLnez9w.

[41] 中国信息安全 . 供应链安全 | 典型软件供应链攻击安全事件列举介绍 [EB/OL]. 2019[2021]. https://mp.weixin.qq.com/s/x8UdyEQrYwZgRnkp6z7uOA.

[42] 关键基础设施安全应急响应中心 . 原创 | 关键信息基础设施供应链安全思考 [EB/OL]. 2021[2021]. https://mp.weixin.qq.com/s/QFelCwMwHFbsvvew0q6Q1g?.

[43] 中国信息安全 . 供应链安全 | 破阵：对国内外网络供应链攻击的思考 [EB/OL]. 2018[2021]. https://mp.weixin.qq.com/s/AleZLuX6ZGEsMWJAHfQq5g.

[44] 国家信息安全服务资质 . 基于软件安全开发生命周期实践的软件供应链安全保障方法 [EB/OL]. 2019[2021]. https://mp.weixin.qq.com/s/yY4zlFudKFREKaa5CU2mjQ.

[45] 中国信息安全 . 防范软件供应链安全风险 维护网络空间安全 [EB/OL]. 2018[2021]. http://www.yidianzixun.com/article/0Km24HTG.

[46] 中国开源软件（OSS）推进联盟 . 2021 中国开源发展蓝皮书 [R]. 北京 : 中国开源软件（OSS）推进联盟 , 2021.

[47] 云原生产业联盟 . 云原生架构安全白皮书 [R]. 北京 : 云原生产业联盟 , 2021.

[48] 中国信息通信研究院 . 开源生态白皮书 [R]. 北京 : 中国信息通信研究院 , 2020.

[49] 绿盟科技研究通讯 . 【云原生攻防研究】针对容器的渗透测试方法 [EB/OL]. 2020[2021]. https://mp.weixin.qq.com/s/o5fGivwgBMR-w60SXQbSYA.

[50] dbaplus 社群 . 一篇顶十篇！把复杂的云原生体系建设都理顺了 [EB/OL]. 2020[2021]. https://mp.weixin.qq.com/s/tCI8-MFseNzvoqoaTeM6OQ.

出品方