透视软件供应链，携手应对安全挑战

软件供应链安全技术白皮书

CCSIA 新安盟
NSFOCUS 绿盟科技

关于绿盟科技

绿盟科技集团股份有限公司（以下简称绿盟科技），成立于 2000 年 4 月，总部位于北京。公司于 2014 年 1 月 29 日在深圳证券交易所创业板上市，证券代码：300369。绿盟科技在国内设有 50 余个分支机构，为政府、金融、运营商、能源、交通、科教文卫等行业用户与各类型企业用户，提供全线网络安全产品、全方位安全解决方案和体系化安全运营服务。公司在美国硅谷、日本东京、英国伦敦、新加坡及巴西圣保罗设立海外子公司和办事处，深入开展全球业务，打造全球网络安全行业的中国品牌。

中国网络空间新兴技术安全创新论坛，（简称“新安盟”，“CCSIA”），在中国产学研合作促进会的指导下成立，由中国工程院方滨兴院士担任理事长，中国科学院信息工程研究所作为秘书长单位，主要任务是应对新兴技术及其应用给网络空间安全带来的机遇和挑战，汇聚行业内各方力量，推动核心技术突破、产品和服务创新。

版权声明

推荐序

软件供应链安全作为现代供应链安全的重点考虑因素之一，是保障我国新基建稳步建设和落实网络强国战略的重要领域。软件供应链安全需要重点关注以下两点。

一是要靶向聚焦、着力剖析供应链风险产生的内部动因和运行机理。近年来，供应链安全事件频发。随着 SDX 和低代码等技术在数字化转型浪潮的广泛应用，软件开源组件化将成为势在必行的技术演进方向，而由于其本身的软件生产机制特征，网络安全风险的攻击面及风险环节会呈现发散和无规则逻辑状态，因此供应链风险的内部动因和运行机理尤其是业务数字化的内涵值得网安领域重点关注。

二是勿管中窥豹、从广义供应链安全的观测视角提出软件供应链风险对策。以供应链所在行业为中心视角研究软件供应链安全问题是一种挑战，行业发展对象将愈发依赖于数字化转型，原有的安全业务和业务安全范式都将被重塑。因而，以数字业务化作为观测出发点，以广义供应链安全作为观测着力点，以软件供应链安全作为观测落脚点，各行各业对网络安全产业的保驾护航定位将会更加明确。

作为业界领先的网安代表性企业，在新安盟指导下，绿盟主持撰写的软件供应链安全白皮书将带来一个崭新的全景视角，具有积极的借鉴价值，为网安行业的政产学研用健康生态如何推动提供有意义的模式示范。

中国科学院大学教授　翟立东

NSFOCUS

CONTENTS

执行摘要

习近平总书记在二十国集团领导人第十六次峰会第一阶段会议发表的重要讲话中强调：“要维护产业链供应链安全稳定，畅通世界经济运行脉络。”

作为世界第二大经济体，我国在世界局势不断变化的今天依靠自身供应链韧性保持了强有力的经济增长与制造业产业转型升级持续稳步推进。信息和通信技术（Information and Communications Technology，简称 ICT）供应商、第三方供应商、集成服务企业和服务提供商共同组成的 ICT 产业链承担着我国产业从工业化向数字化转型升级的重要任务。2018 年以来，中兴、华为、大疆等一系列事件，暴露出我国 ICT 产业链上游核心技术受制于人的问题，缺乏核心技术使我国网络安全与信息化建设存在巨大的风险与阻碍。尤其在近期俄乌冲突期间，西方社会对俄罗斯进行了全方位的制裁，同时将 ICT 供应链制裁上升到了战略层面对俄罗斯进行打击，这一行为也为我国敲响了警钟。软件供应链作为 ICT 供应链的重要组成部分，是各类关键信息基础设施平稳运行的重要基础，其关键组件的设计、开发、部署、监控和持续运营等生命周期核心环节供给过程的安全可控成为网络安全的关键考量因素。

绿盟科技推出软件供应链安全技术白皮书，旨在从软件供应链安全威胁与国内外形势来梳理软件供应链中存在的安全问题，提炼出软件供应链安全治理的核心理念、技术框架、关键技术，并从供应链安全监管和控制方面给出解决方案和最佳实践，期望为读者带来全新的技术思考，助力我国软件产业发展。本技术白皮书的主要观点如下：

观点 1：软件供应链威胁存在于软件供应链的全生命周期中，攻击的手段和实施的途径相较其他攻击技术手段更加多元化，且难以防范。近几年软件供应链攻击安全事件激增，软件供应链攻击已经成为重要的突破手段，其中利用开源社区、公共开源存储仓库这类开源软件生态入侵事件尤为严重。开源软件供应链安全不可忽视，其重要性日渐凸显。

观点 2：近年来，政府在供应链安全领域发挥着越来越重要的影响，但除积极方面之外，以美国为首的部分发达国家政府在其中注入了过多地缘政治因素的考量，甚至将其制定相关法规的权力作为国际竞争的工具，使其风险从通常意义上的网络安全风险上升为供应链断供风险。另一方面，面对发达国家以法令出台、贸易制裁、技术出口等手段带来的供应链跨境安全管控风险，中国、俄罗斯等国家立足国情，集中优势资源开展核心技术攻关，提升自研自制能力，补齐短板、发展优势能力，加强对供应链产品和服务的安全审查，逐步建立和完善供应链安全风险管理体系。

观点 3：为应对软件供应链的威胁，上游企业需要构建自身产品的软件成分清单来梳理软件供应链信息，向下游企业和用户清晰、透明的提供管理软件供应链所需要的基础条件。软件成分清单依据识别成分的粒度，可以分为不透明、微透明、半透明和透明几个阶段。透明程度高的软件成分清单，能显著提升最终用户进行软件供应链安全评估的准确性。

观点 4：软件供应链安全包括整个软件的开发生命周期，在开发阶段漏洞的引入不止在代码编写阶段，还有所依赖的开源组件、开发和构建工具等，依照软件的开发和构建过程，企业需要建设开发过程安全评估能力。在软件交付阶段，作为供应商，除保证交付软件安全外，也应将软件成分清单一并交付给下游企业，促使整个软件供应链的上下游都具备依据安全通报、威胁情报监控等第三方信息能够分析、评估软件供应链安全的基本条件。供应链软件产品交付运行后，供应商应在产品的生命周期内提供安全保障服务，对产品漏洞及时修复，最终用户也应根据供应商所提供的软件成分清单纳入企业资产管理范围，定期对资产进行安全评估，结合漏洞预警，对受影响的产品进行加固和修复。

观点 5：软件供应链风险贯穿了产品的整个生命周期，结合近年来所发生的供应链攻击和入侵事件，我们需要从监管层面加强供应链产品安全认证管理，提供企业软件 SBOM 托管和可信认证服务，企业也需要完善供应链资产管理和安全检查，可借助 SBOM 知识图谱理清企业供应链依赖关系，从而在监测到预警时能够从容应对。

观点 6：软件供应链在不断的技术迭代与产业发展中逐渐形成了包含复杂技术体系、多元产品组件及各路开发者、供应者与消费者为一体的庞大产业生态，大量新技术的引入令软件风险防护范围从个体层面提升至供应链层面，安全视角发生了重大变化。整个软件供应链缺乏有效的统筹与管理，将安全建设寄托于技术人员的自我道德约束无疑是一种“赌博”。我们急需建立供应链安全管理机制，杜绝“自我约束”的无监管行为。通过政策引导与配套法律建设保证软件供应链安全能够牢牢的把握在中国人自己手中，形成可信的软件供应链体系，真正造福社会。

1

软件供应链安全威胁与趋势

1.1 软件供应链面临的安全威胁

经济全球化发展给企业发展带来了更多机遇和成长，基于价值链的实现，促进了供应链的全球化，多样化和复杂化，同时如何管理供应链的问题给企业带来了更大的挑战和威胁。软件供应链涉及环节复杂，流程和链条长，供应商众多，暴露给攻击者的攻击面越来越多，攻击者利用供应链环节的薄弱点作为攻击窗口，供应链的各个环节都有可能成为攻击者的攻击入口。既有传统意义上供应商到消费者之间供应链条中信息流的问题，也有系统和业务漏洞、非后门植入、软件预装，甚至是更高级的供应链预制问题。

从软件供应链全生命周期考虑，可以简单分为上游安全、开发安全、交付安全、使用安全和下游安全，安全威胁存在于软件供应链的全生命周期中。

一、软件供应链上游安全

为实现业务需求的独特性，很多公司需要根据业务需求开发定制化软件或者使用云服务产品，可能会使用软件供应商或者云服务提供商的产品，那么公司对于上游企业面临的风险是否有考虑到呢？比如采购上游企业的软件产品，软件供应商是否能保证对软件使用的核心组件可信且组成清晰，在爆发软件内部漏洞的事情下，具备快速的定位组件风险的能力。软件在面临外部攻击风险时，是否具备一定的抗攻击能力，至少不会漏洞百出。不管是内部漏洞还是外部攻击，软件供应商是否有能力快速定位到漏洞，及时评估漏洞、代码的风险，能提供持续的更新能力。上游企业安全评估能力强，那么将更好的预防风险发生，安全评估能力弱，随之而来的存在脆弱性风险的可能也将增加。尤其关键数据泄露是重中之重，关注上游软件评估中关键数据的权限范围、证书管理等能力，成为企业选择上游企业的关键因素。当然也应当保证软件供应链的持续性，不能因为软件产品中组件或者环境因素，影响业务，比如云服务商的 SaaS、PaaS 等服务的中断可能性。

在上游软件供应链安全中，另一大安全风险是开源安全。开源软件的使用，一定程度上推动了技术的发展，提高了创新的可能性，但是另一方面，国际环境的复杂性，开源软件又具备代码公开、易获取和可重用的特点，使开源软件面临更大的风险，比如开源组件、框架的漏洞、缺陷逐渐增加，且开源软件分布在各大社区，使用范围略广，然而漏洞信息却不能及时被官方收录，使得漏洞的跟踪能力和整改能力降低，上游企业是否具备管控开源软件的能力呢？除此之外，开源软件隐形依赖关系使不同开源软件之间存在合规性和兼容性风险，从而引发知识产权风险，且隐形依赖关系增加了漏洞发现、修复的难度，为保障开源组件、

框架的稳健性，将提高其维护、修复和应对能力。

如果没有提前考虑到采购前的风险点以及应对能力，那么很可能，在爆发软件攻击或者漏洞的情况下，不能快速定位软件风险点，及时遏制住风险的扩散，快速的更新软件问题，那么影响的将会是使用此软件的所有客户企业群体，带来的更加直观的损失将会是小到影响业务正常运行，大到财务损失，更甚者是客户企业的存亡问题。

二、软件供应链开发安全

在软件开发过程中，如果开发者为了提高工作效率，并没有在设计之初将安全考虑在内，将导致后期安全问题的出现，增加整改的成本和难度。从安全角度考虑软件开发阶段简单分为开发环境安全、开发过程安全和编译构建安全三个阶段。

在软件开发阶段，需要借助提前准备的工具进行编程实现业务功能，作为开发过程的一个重要环境，如何选择开发工具将尤为重要，一旦开发工具遭到污染，使用了不安全的开发工具，那么开发完成的软件很大可能性同样存在安全风险。此外，开发环境以及 CI/CD 集成环境遭到污染，都有可能导致代码存在被篡改、恶意植入等风险问题，甚至可能导致整个编程环境存在安全风险。即使开发工具和环境不存在污染，那么完成开发部分的源代码同样需要确保其托管平台和代码存储平台未受到污染，否则恶意人员很可能通过托管平台或者代码存储平台对代码进行篡改以及恶意植入。

在确保开发环境安全的情况下，开发过程也可能引入安全威胁，开发者由于工期紧、任务重等原因，更偏重于功能开发，不规范的安全开发引入不可预测的安全风险。开发人员在开发过程中，不可避免的会借用网上现成的代码，既是学习成长的过程，也是提高工作效率的一种方式，但是对于开发功底不深厚、经验不够丰富的学习者来说，如果更注重功能的实现，而不具备识别和判定代码安全性问题的能力，就会在代码来源的安全性，代码内部逻辑的安全性以及在版本修订、剪裁和迭代中引入更多安全问题，内部漏洞、缺陷也将增加，这种行为会在后续软件实现中埋下一个定时炸弹，同时在开发过程中也将涉及到开源组件引入风险问题。

完成开发后，进入编译构建阶段，编译工具的不安全将导致可能植入后门。各大互联网公司在企业内部自建包管理工具用于存放自研软件包，若员工安装自研软件包时没有制定仅企业内部下载，就可能遭到包抢注攻击，以及开发者在使用过程中，生产配置文件引用了官方源不存在的包等问题。

三、软件供应链交付安全

目前很多企业软件开发大多数采用第三方供应商软件再开发方式，购买第三方软件的基础架构，再根据企业情况进行定制化的开发调整。源代码共享的情况下，源代码安全成为了双方共同的安全责任。但是一般情况下，企业更加注重本身对于源代码的保管安全，考虑到源代码的访问安全、物理环境安全、网络安全等因素，并做好充分的保护措施。但企业很少会对第三方保管的源代码提出管理要求。而软件供应商相对注重代码功能的交付，对于源代码的保管安全意识较为薄弱。比如办公环境和源代码存在于同一区域，网络环境内缺乏安全防护设备，未严格限制源代码的访问控制等问题普遍存在。不管是监管机构还是企业客户，均未对第三方源代码安全提出管理要求，导致第三方源代码安全成为盲区。

交付过程中，除了源代码安全，也应扩大交付范围管控，如果交付的软件未经过编译或者编译信息泄露，那么也将导致源代码不安全性增加，使攻击者能更轻松的获取源代码，通过篡改、植入等技术手段破坏软件的安全性，达到获利的目的。软件的正常使用，还涉及到证书、私钥的管理泄露风险。如果不能确保基础环境的可信，证书、密钥的泄露将更加便利攻击者的入侵。

完成基础交付准备工作，需要将软件安全以及维护后续安全更新途径传递到企业手中。2022 年央视“3 • 15”晚会对软件捆绑的互联网乱象进行了曝光，发现软件捆绑下载情况相当严重，一旦用户在这些网站下载安装软件就会“中招”，用户下载一个软件，实际上明里暗里的却被安装了数个捆绑软件，导致电脑运行速度明显下降，电脑安全风险大大增加。另外除了捆绑下载问题，安全问题也频繁出现，需要对维护软件进行频繁的升级、更新。升级、更新途径可能存在劫持风险，导致更新包早已被篡改、恶意植入。

四、软件供应链使用安全

软件在使用过程中，不可避免的会发生突发信息安全事件问题，比如 0DAY 的出现，或者其他软件病毒类安全事件，如不能快速发现、应对和处置，将会给企业带来严重的威胁，甚至带来财务损失。软件供应商是否能及时配合企业完成信息安全事件的排查和修复过程至关重要。

软件使用过程中，依赖于网络基础设施安全或者云平台安全，基础设施面临恶意攻击、自然灾害破坏引发的连锁反应可能造成跨部门大面积基础设施受损，并引发服务中断，给业务正常运行带来威胁。企业应确保网络基础设施符合国家监管政策要求，履行网络基础设施建设和维护职责。

五、软件供应链下游安全

供应链下游主要为营销、服务、品牌等活动，供应链下游的信息将成为上游信息的重要输入信息，但是供应链的各阶段环节由多家供应商参与，企业之间需求信息相对保守，信息从下游向上游需求放大传递时，就出现了“牛鞭效应”。“牛鞭效应”使得信息被扭曲、放大，无法实时有效的传递，而企业的需求信息决策取决于相关反馈数据输入，供应链环节越多，供应链越长，那么传递信息的失真性越严重，供应链效率就越低。

在供应链下游也同样存在上游的问题，比如下游采用独家供应商，下游对接的供应商自身管理水平、企业文化、经营模式和财务状况等存在差异性。企业对上游和下游的管理同样重要。

1.2 软件供应链攻击的发展趋势

1.2.1 软件供应链攻击事件持续高发

相较于传统安全威胁，供应链威胁的影响力具有扩散性，上游产品的漏洞会影响下游所有角色，引起安全风险的连锁传递，导致受攻击面不断扩大。近年来发生了多起影响力较大的供应链攻击事件，涉及开源组件、公开代码存储库、云安全 CI/CD 平台等方面。

表 1.1 软件供应链攻击事件

时间	关键词	备注
2022.05	开源组件	以依赖混淆的攻击方式在一系列恶意 NPM 软件包插入后门代码，针对德国的重要媒体、物流和工业公司发起攻击。研究者认为本次攻击有很强的针对性，并且依赖难于取得的内幕信息。 一家名为 Code White 的德国渗透测试公司申请为此事件负责，并补充说这是“为专门的客户模仿现实的威胁行为者”的尝试。
2022.03	开源组件	继百万周下载量的 npm 包“ node-ipc ”以反战为名进行供应链投毒后，又一位开发者在代码中加入反战元素。3 月 17 日，俄罗斯开发人员 Viktor Mukhachev 在其流行的 npm 库“event-source- polyfill”中添加了一段反战代码。这段在 1.0.26 版本中引入的代码意味着：使用该库构建的应用程序，将在启动 15 秒后向俄罗斯用户显示反战消息。
2022.03	开源组件	攻击者在 npm 仓库中利用 typosquatting 的方式，注册一系列包含恶意代码的重名组件包，对 azure 开发者进行供应链攻击。
2022.03	开源组件、代码注入	node-ipc 是使用广泛的 npm 开源组件，其作者出于其个人政治立场在该项目的代码仓库中进行投毒，添加的恶意 js 文件会在用户桌面创建反战标语。根据进一步的深挖，该作者还曾加入将俄罗斯与白俄罗斯区域用户数据抹除的恶意代码。Unity Hub 、vue-cli 等应用广泛的第三方软件受到该事件影响。
2022.01	开源组件、代码注入	使用众多的 npm 软件包 faker.js 与 color.js 的主要开发者 Marak Squires 因生活窘迫，维护项目不得回报等原因，删除所有 github 代码，并向 npm 仓库推送包含恶搞功能的更新（打印乱码），包括 aws-cdk 在内的众多应用受到影响引发对开源生态，开源项目维护者角色回报的激烈讨论。

时间	关键词	备注
2021.12	开源组件、软件漏洞	Apache Log4j2 是一个开源基础日志库，是对 Log4j 组件的升级，被广泛用于开发、测试和生产。该开源项目支持属性查找，并能够将各种属性替换到日志中。用户可以通过 JNDI 检索变量，但是由于未对查询地址做好过滤，存在 JNDI 注入漏洞。Log4j2 应用极其广泛，影响范围极大，同时随着供应链环节增多、软件结构愈加复杂，上述漏洞也更加难以发现、修复（尤其是间接使用到该组件的项目）。目前常见攻击形式有勒索、挖矿、僵尸网络（以及 DDOS）。
2021.09	云安全	Microsoft 的 Azure 容器服务存在跨账户接管漏洞，原因是使用了过时的 RunC 工具 (v1.0.0-rc2)。攻击者可攻陷托管 ACI 的 k8s 集群，接管平台上其他客户的容器，执行代码并访问平台上的数据。
2021.09	公开代码存储库	攻击者提交恶意代码到 GitHub 私有库，更改公司拍卖网站的前端并将钱包地址替换为自己的钱包地址。原因在于仓库没有强制执行分支保护设置。
2021.08	开发工具	Realtek 旗下 WiFi 模块的开发包 (SDK) 中存在多个漏洞，包括命令注入、HTTP 的内存损坏、自定义网络服务等。攻击者可利用这些漏洞破坏目标设备并以最高权限执行任意代码。
2021.07	代码注入、IT 管理平台、下游影响	与 REvil 勒索软件团伙相关的犯罪团伙发起了针对多家管理服务提供商 (MSP) 的大型勒索软件攻击。经调查这些厂商均使用了来自 Kaseya 的 VSA 产品服务，该服务提供对客户终端的统一远程监控与管理。由于 Kaseya 被攻陷，导致产品中被植入恶意代码，超过 1500 多家下游用户企业被感染。
2021.06	开源组件	Linux 平台基于 Pling 的各个免费开源软件市场存在漏洞，浏览器没有为本地 WebSocket 服务器链接实现同源策略。攻击者可利用该漏洞，进行 XSS 蠕虫攻击或者远程代码执行攻击。
2021.05	信息泄露	Colonial Pipeline 是美国最大的燃油管道运营商之一。犯罪组织 Darkside 利用该公司泄露到暗网上的虚拟专用网络账户密码，成功入侵后窃取大量数据并安装了勒索软件，最终索取了约 440 万美金的赎金。
2021.04	CI/CD、基础镜像	由于 Docker 映像创建中的错误，代码测试公司 Codecov 产品中的 bash uploader 脚本被修改，导致产品会向攻击者的服务器发送客户在持续集成 (CI) 中的软件源代码、凭据、令牌等敏感、机密信息。
2021.03	信息泄露、下游影响	硅谷初创公司 Verkeda 的摄像头数据库被攻破，约 15000 个监控摄像头的实时画面被泄露。根源在于该公司的一个管理员账户的用户名和密码被公开。
2021.03	公开代码存储库、代码植入	PHP 的独立 git 基础设施 git.php.net 服务器被攻击，攻击者冒充两名维护人员向服务器上的存储库推送了恶意提交，并成功植入后门。该后门能获得网站系统的 HTTP 请求远程代码执行权限。
2021.02	开源组件、依赖混淆	安全研究人员 Alex Birsan 通过利用开源生态安全机制上的漏洞（即依赖混淆），成功侵入了微软、苹果、PayPal、特斯拉、优步等 35 家国际大型科技公司的内网。
2021.01	云安全、下游影响	Mimecast 是一家为电子邮件和企业信息提供云安全和风险管理服务的供应商。2021 年 1 月发现，有攻击者成功攻击 Mimecast 服务并获得了 Mimecast 为微软 365 用户颁发的证书，使其能够干涉链接并从服务器窃取信息。
2020.12	软件供应商、下游影响	SolarWinds 遭遇国家级 APT 团伙的供应链攻击，对美国各个行业的大量客户产生了严重影响。攻击流程为：首先入侵 SolarWinds 达成初始妥协；此后将篡改软件部署后门，再使用合法证书为植入了后门的组件签名；最后，监视技术环境，识别漏洞后进行提权、横向移动等一系列操作，实现对 SolarWinds 的渗透，并获取用户数据。可能应用到的攻击技术有：社会工程学攻击、暴力攻击以及零日漏洞利用。
2020.12	代码注入、下游影响	黑客攻击了越南政府证书颁发机构（VGCA），并在提供的客户端应用程序中植入后门。该后门可以接收插件，也可以检索受害者的代理配置，并使用它来联系 C&C 服务器并执行命令。它的旧版本 PhantomNet 曾在菲律宾被发现。
2020.11	软件漏洞、密保软件、数字证书	WIZVERA VeraPort 是一款韩国集成安装工具，帮助用户在访问政府、银行网站时管理所需安全软件。该软件在使用中仅验证下载的二进制文件数字签名是否有效，却不验证其所属来源。Lazarus 网络犯罪组织利用该漏洞，针对支持 VeraPort 的网站，利用盗取的数字证书对恶意软件进行签名并替换，使得用户下载到恶意软件。
2019.01	升级程序、硬件供应商、下游影响	黑客更改了旧版本的华硕 Live Update Utility 软件，在其中植入了后门程序，并通过更新的方式将该版本分发给全球的华硕设备。该软件使用合法的华硕证书签名，存储在官方服务器上。攻击者可以通过远程服务器控制目标计算机，安装额外的恶意软件。

1.2.2 软件供应链攻击技术复杂多样

区别于孤立产品的安全漏洞利用，现实环境中的供应链攻击手法更加多样，实施途径更加多元化。

由于开发成本的优势，在项目中广泛使用开源组件已成为主流的开发方式。低门槛的开放社区与有限的维护资源的矛盾，给攻击者提供了可乘之机。在开源代码中插入恶意代码是造成影响最为直接的攻击形式，但常规的恶意代码插入手段会受到人工代码审核等方式的遏制。攻击者开始另辟蹊径，寻求人工代码审查存在的脆弱性。剑桥大学的研究员提出了一种名为 Trojan Source[1] 的新型攻击技术，该技术利用 Unicode 中的不可见字符构造肉眼难以识别的恶意隐藏代码，达到逃逸人工审查的效果。除此之外，明尼苏达大学的研究者提出向开源项目提交隐藏漏洞的攻击方式，并实践于 Linux 内核项目 [2]。虽然这种做法因开源社区的强烈谴责而被叫停，但仍然为安全社区警示了该攻击手段的隐蔽性和潜在的破坏性。

公共代码存储仓库是开源软件代码的载体，企业在产品的开发构建过程从中拉取第三方依赖。但有效的安全控制与自动化的反制措施的缺失，会使公开代码仓库成为恶意代码的潜在传播平台。2021 年初，安全研究员 Alex Birsan 提出名为依赖混淆（Dependency Confusion）的新供应链攻击方式 [3]。攻击者在公开存储库中创建私有依赖项的更高版本的同名项目，使得恶意代码在构建的过程中被拉取。该技术巧妙的利用了主流包管理器存在的设计缺陷，利用包名的模糊性在众多产品实现 RCE，被 PortSwigger 列为 2021 年度 Web 攻击技术之首 [4]。

在 DevOps 理念深入人心、IT 云化服务登堂入室的今天，自动化构建、测试与部署等自动化流水工具成为现代产品软件开发的选择。CI/CD 平台作为 DevOps 理念的落地实践，起着保证持续集成和持续部署的关键作用，瞄准此类平台的攻击事件也逐渐涌现。在 2020 年末的 SolarWinds 供应链攻击事件中攻击者在 IT 管理软件的代码仓库中插入恶意代码，编译部署后分发至全球政府与跨国商业机构，影响重大。在 2021 年 4 月，由于 Docker 镜像创建过程中的问题，代码测试公司 Codecov 产品中的 Bash Uploader 被修改，允许攻击者获取软件源代码、凭据令牌等敏感信息 [5]。

[1] Boucher, N. and R. Anderson, ***Trojan Source: Invisible Vulnerabilities.*** arXiv preprint arXiv:2111.00169, 2021.
[2] Open Source Insecurity: The Silent Introduction of Weaknesses through the Hypocrite Commit
[3] https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610
[4] https://portswigger.net/research/top-10-web-hacking-techniques-of-2021
[5] https://blog.sonatype.com/what-you-need-to-know-about-the-codecov-incident-a-supply-chain-attack-gone-undetected-for-2-months

2

供应链安全国内外形势

2.1 国内外供应链安全政策与发展

2.1.1 美国

2002 年，美国布什政府提出强调关注 ICT 供应链安全问题的信息安全战略，美国将 ICT 供应链安全问题提到了国家战略的高度予以重视。

2008 年，美国布什政府发布的 54 号国家安全总统令（NSPD54）提出国家网络安全综合计划（CNCI），其部署的一项重要工作就是建立全方位的措施来实施全球供应链风险管理。为落实该计划对 ICT 供应链安全问题的部署， 2008 年美国国家标准和技术研究院（NIST）启动了 ICT 供应链风险管理项目（SCRM），在原《信息保障技术框架》（IATF）提出的“纵深防御”战略的基础上，提出了“广度防御”的战略，开发全生命周期的风险管理标准。NIST 认为，“纵深防御”战略侧重于通过分层的防御体系对网络和系统进行保护，其关注的是产品在运行中的安全，因而不能解决供应链安全问题，而“广度防御”战略的核心是在系统的完整生命周期内减少风险，这一认识的变化也奠定了当前 ICT 供应链安全风险管理方法的基础。

2009 年，美国奥巴马政府在《网络空间安全政策评估报告》中指出，应对供应链风险除了对国外产品服务供应商进行谴责外，更需要创建一套新的供应链风险管理方法。

2011 年，美国奥巴马政府发布的《网络空间国际战略》中将“与工业部门磋商，加强高科技供应链的安全性”作为保护美国网络空间安全的优先政策。

2014 年，美国国会提议了《网络供应链管理和透明度法案》，意在确保为美国政府开发或购买的使用第三方或开源组件以及用于其他目的的任何软件、固件或产品的完整性。该法案要求制作物料清单 (BOM) 由“为美国政府开发或购买的软件、固件或产品的所有供应商”提供的所有第三方和开源组件。法案中的措辞承认开源是一种关键资源，并清楚地表示其在政府 IT 运营中持续发挥着关键作用。该法案是建立透明度的积极的第一步，这是一个非常有价值和可以实现的目标。通过遵守新立法，联邦政府供应商将准确了解他们的代码中的内容，并能够在发现任何问题时主动解决。

2018 年 12 月，美国国会通过了《安全技术法案》，《联邦采购供应链安全法案 2018》作为该法案的第二部分一并签发。《联邦采购供应链安全法案 2018》创建了一个新的联邦采购供应链安全理事会并授予其广泛权利，为联邦供应链安全制定规则，以增强联邦采购和采购规则的网络安全弹性。

2019 年 5 月，美国特朗普政府签署了名为《确保信息和通信技术及服务供应链安全》的行政令，宣布美国进入受信息威胁的国家紧急状态，禁止美国个人和各类实体购买和使用被美国认定为可能给美国带来安全风险的外国设计制造的 ICT 技术设备和服务。

2021 年 1 月，美国商务部发布《确保信息和通信技术及服务供应链安全》的最新规则，旨在落实 2019 年 5 月 15 日特朗普政府的《确保信息和通信技术及服务供应链安全的总统令》的相关要求，建立审查外国对手的 ICT 服务的交易流程和程序，禁止任何受美国管辖的人获取、进口、转让、安装、买卖或使用可能对美国国家安全、外交政策和经济构成威胁的外国 ICT 技术与服务。

2.1.2 欧盟

2012 年，欧盟网络和信息安全局（ENISA）发布了《供应链完整性——ICT 供应链风险和挑战概览，以及未来的愿景》报告，并于 2015 年更新。除了提供可供 ICT 供应链相关参与者借鉴的实践做法外，还建议设立国际评估框架，以有效评估 ICT 供应链风险管理。

2016 年上半年，欧洲标准化委员会（CEN）、欧洲电工委员会（CENELEC）与欧洲电信标准协会（ETSI）对欧洲 ICT 产品和服务的政府采购所适用的可接入性提出了新的标准，即通信技术产品和服务政府采购所适用的可接入性规定（EN 301 549）。该标准是欧洲首次对通信技术产品和服务的政府采购所适用的可接入性标准，并以法规的形式加以强调。该标准指出，政府部门与其他公共机构在采购通信技术产品和服务的时候，要确保网站服务、软件、电子设备与其他产品具有更好的可接入性，即：上述产品与服务的采购要本着让更多人使用的理念出发，即体现“以人为本”的原则进行。

2017 年 9 月，欧洲委员会主席 Jean-Claude Juncker 在其盟情咨文中提出了《欧盟网络安全法案》。2019 年 6 月，新版《网络安全法》正式施行，取代了旧版网络安全法案，该法案表现出了欧盟对中国 IT 供应商在欧洲市场日益增长的影响力的担忧和关切。

2019 年 4 月，欧盟《外国直接投资审查条例》生效。该条例指出， 欧盟有权对参与 5G 网络等关键基础设施投资的外商进行审查和定期监控，以保障 5G 网络等关键基础设施的安全性，同时避免关键资产对外商的过度依赖。这也是欧盟保障 5G 供应链安全的有效工具。

2021 年 7 月，欧盟网络和信息安全局（ENISA）发布了《供应链攻击威胁全景图》，该报告旨在描绘并研究从 2020 年 1 月至 2021 年 7 月初发现的供应链攻击活动。该报告通过分类系统对供应链攻击进行分类，以系统化方式更好地进行分析，并说明了各类攻击的展现

方式。报告指出，组织机构更新网络安全方法时应重点关注供应链攻击，并将供应商纳入防护和安全验证机制中。

2.1.3 英国

2013 年，英国可信软件倡议（Trustworthy Software Initiative，TSI）通过解决软件可信性中的安全性、可靠性、可用性、弹性和安全性问题，提升软件应用的规范、实施和使用水平，在 ICT 供应链的软件领域建立起基于风险的全生命周期管理。由 TSI 发布的可信软件框架 (Trustworthy Software Framework，TSF) 为不同领域的特点术语、引用、方法以及数据共享技术提供互通的可能，为软件可信提供最佳实践及相关标准。

2014 年，在 TSI 及其他机构的努力下，“软件可信度 - 治理与管理 - 规范”（PAS754：2014）发布。该规范在英国软件工程上具有里程碑的意义，涵盖了技术、物理环境、行为管理等多个方面，并规定了申请流程，为采购、供应或使用可信赖软件提供帮助，提高业务水平，降低安全风险。

2019 年，《英国电信供应链回顾报告》发布，报告结合英国 5G 发展目标，以及 5G 在经济和社会发展中的作用，强调了安全在电信这一关键基础设施领域的重要意义，并为电信供应链管理展开综合评估。

2.1.4 日本

日本在 2018 年版的《网络安全战略》中，强调了“加强组织的能力”，呼吁供应链各相关方积极协作，通过组织间多样化的连接，创造有价值的供应链。其具体举措一是明确供应链中存在的威胁，制定并推广相关保护框架；二是充分调动中小企业的积极性，推动其创新。

日本在 2020 年提交一项关于网络技术安全的法案——《特定高度电信普及促进法》，旨在维护日本的网络信息安全，确保日本企业慎重应用新一代网络技术，5G 和无人机将是新法的首批适用对象。新法要求日本相关企业在采购高级科技产品及精密器材时，必须遵守三个安全准则：第一，确保系统的安全与可信度；第二，确保系统供货安全；第三，系统要能够与国际接轨。这套准则标明了日本企业应考虑使用的日本或欧美的相关产品。同时，日本政府采购也已将华为和中兴排除在外，并考虑在水电和交通等基础设施领域只运用本国数据库，并要彻底防止使用有中国科技产品的其他国家数据库。同时，政府将通过税制优惠措施在内的审批手段，引导企业重用日本本国研发的新一代通信器材。

2.1.5 俄罗斯

在《2025 年前电子工业发展战略》明确提出，俄罗斯首先要解决的是对进口产品的依赖，而为了解决这些问题当务之急是要“尽快实现创新成果的应用”。为此，近年来，俄罗斯加大了对进口信息设备使用的管理和控制。普京曾公开要求俄罗斯仅有的三家具有部分政府管理职能的国家公司之一的俄罗斯技术公司使用国产软件产品，并称如果该公司不使用国产软件将取消其承接国家订货任务的资格。同时，国防部等涉及国家安全的关键及重点部门、领域、行业和个人也被要求必须使用俄制产品。为了保证信息安全，俄加大了国产安全手机的配发，国产操作系统亦开始在俄罗斯军方作战指挥系统终端中广泛应用。2018 年国防部还启动了所有计算机全部换装国产操作系统的工作。不仅如此，为确保重要部门之间能够实现安全的互联和通信传输，俄罗斯着手建设基于国产技术、能够实现安全加密通信的“国家信息通信网”，并计划于 2018 年开始为总统办公厅等机构提供接入服务。

总的来说，俄罗斯密集出台了一系列旨在推进俄制产品应用的政策措施，以期从源头上杜绝因使用不安全、留有后门的信息产品所带来的安全隐患，希望通过加大本国产品应用拉升市场需求、促进国内生产，并通过实际应用，不断检验、丰富、完善和提高本国产品的功能、特性及竞争力，形成产业发展的良性互动。

此外，在跨国合作上，中、俄等国在 2011 年和 2015 年两度向联合国提交的《信息安全国际行为准则》中，也就确保 ICT 供应链安全提出了具体倡议，强调应“努力确保信息技术产品和服务供应链的安全，防止他国利用自身资源、关键设施、核心技术、信息通讯技术产品和服务、信息通讯网络及其他优势，削弱接受上述行为准则国家对信息通讯技术产品和服务的自主控制权，或威胁其政治、经济和社会安全”。

2.1.6 中国

2014 年 5 月 22 日，国家互联网信息办公室宣布我国即将推出网络安全审查制度，初步界定了网络安全审查的含义。

2015 年 7 月 1 日，《中华人民共和国国家安全法》第 59 条规定了网络安全审查制度由国家建立。

2016 年 7 月，《国家信息化发展战略纲要》明确我国要建立实施网络安全审查制度，对关键信息基础设施中使用的重要信息技术产品和服务开展安全审查。

2016 年 11 月 7 日，《中华人民共和国网络安全法》中明确关键信息基础设施运营者采购网络产品和服务，可能影响国家安全的应通过国家有关部门组织开展的网络安全审查。

2017 年 6 月，我国颁布《网络产品和服务安全审办法 (试行)》和《网络关键设备和网络安全专用产品目录 (第一批)》。

2020 年 4 月，我国多部门联合发布《网络安全审查办法》，进一步细化明确了网络安全审查的范围、机制、流程等相关要求。

2021 年 4 月，国务院常务会议正式通过《关键信息基础设施安全保护条例》，在该条例中明确了运营者采购产品和服务的安全管理措施。

2021 年 11 月，国家互联网信息办公室通过《网络安全审查办法》，该办法将网络平台运营者开展数据处理活动影响或者可能影响国家安全等情形纳入网络安全审查，并明确掌握超过 100 万用户个人信息的网络平台运营者赴国外上市必须向网络安全审查办公室申报网络安全审查。

2.1.7 总结

近年来，政府在供应链安全领域发挥着越来越重要的影响，但除积极方面之外，以美国为首的部分发达国家政府在其中注入了过多地缘政治因素的考量，甚至将其制定相关法规的权力作为国际竞争的工具，任由敌视、焦虑情绪主导，人为割裂了原本可以互联、互通、互利的全球供应链，使其风险从通常意义上的网络安全风险上升为供应链断供风险。另一方面，面对发达国家以法令出台、贸易制裁、技术出口等手段带来的供应链跨境安全管控风险，中国、俄罗斯等国家立足国情，集中优势资源开展核心技术攻关，提升自研自制能力，补齐短板、发展优势能力，加强对供应链产品和服务的安全审查，逐步建立和完善供应链安全风险管理体系。

2.2 国内外供应链安全标准与实践

2.2.1 国际标准

供应链安全的标准化工作经历了由传统供应链安全向 ICT 供应链安全的演变，主要标准如下：

2.2.1.1 ISO 28000 系列标准

“ISO 28000 系列标准”主要针对传统供应链安全，包含：《ISO 28000：2007 供应链安全管理体系 规划》、《ISO 28001：2007 供应链安全管理体系 实施供应链安全、评估和计划的最佳实践 - 要求和指南》、《ISO 28003：2007 供应链安全管理体系 供应链安全管理系统机构审计和认证要求》、《ISO 28004：2007 供应链安全管理体系 实施指南》。ISO28000

系列标准帮助组织建立、推进、维护并提高供应链的安全管理系统，明确组织需建立最低安全标准，并确保和规定安全管理政策的一致性；建议组织通过第三方认证机构对安全管理系统进行认证或登记，并对供应链中弹性管理系统提出更明确的要求。ISO28000 系列标准对于 ICT 供应链安全管理体系的建设具有一定的借鉴意义。

2.2.1.2 ISO 27036 系列标准

ISO/IEC 27036 信息技术 安全技术 供应链关系信息安全（1- 概述和大纲 2- 要求 3-ICT 供应链安全指南 4- 云服务安全指南）

ISO 27036 系列标准中，《ISO/IEC 27036-1 综述和概念》对处于多供应商关系环境下相对安全组织的信息和基础设施安全管理进行了概述；《ISO/IEC 27036-2 通用要求》为定义、实施、操作、监控、评审、保持和改进供应商关系管理规定了通用性的信息安全要求，这些要求覆盖了产品和服务的采购、供应的所有情况，例如制造业或装配业、业务过程采购、知识过程采购、建设经营转让和云计算服务，适用于所有类型、规模和性质的组织；《ISO/IEC 27036-3 ICT 供应链安全管理指南》专门针对 ICT 供应链安全提出提出查询和管理地理分散的 ICT 供应链安全风险控制要求，并将信息安全过程和实践整合到系统和软件的生命周期过程中，且专门考虑了与组织及技术方面相关的供应链安全风险；《ISO/IEC 27036-4 云服务安全指南》专门针对云计算服务安全提出要求，在云服务使用和安全风险管理过程中提供量化指标，同时针对云服务获取、提供过程中对组织产生的信息安全应用风险隐患，提供应对指南使其更加有效。

2.2.1.3 ISO/IEC 20243 开放可信技术供应商标准——减少被恶意污染和伪冒的产品

该标准提出保障商用现货 (COTS) 信息通信技术 (ICT) 产品在生命周期内完整性，安全性的最佳实践和技术要求，是针对 COTS ICT 软硬件产品供应商在技术研发及供应链过程安全的第一项国际标准。

2.2.2 美国标准

美国针对供应链安全，制定了 NIST SP 800-161《Supply Chain Risk Management Practices for Federal Information Systems and Organizations（联邦信息系统和组织的供应链风险管理实践）》（以下简称 NIST SP 800-161）和 NIST IR 7622《National Supply Chain Risk-Management Practices for Federal Information Systems（联邦信息系统供应链风险管理实践理论）》（以下简称 NIST IR 7622）这两个影响力较大的标准。

2.2.2.1 NIST SP 800-161

该标准规定了 ICT 供应链风险管理的基本流程，参考了 NIST SP 800-39《管理信息安全风险》中提出的多层次风险管理方法，分别从组织层、业务层以及系统层三个层面和构建风险管理框架、评估风险、应对风险以及监控风险 4 个步骤来解决风险。

2.2.2.2 NIST IR 7622

该标准给出了供应链风险管理的实施流程。对于高影响系统，ICT 供应链风险管理被明确嵌入到采购进程中来分析潜在的供应链风险，实施额外的安全控制以及供应链风险管理的实践；对中度影响的系统，授权机构应该做出是否实施 ICT 供应链风险管理的决策；低影响系统不需要实施大量的 ICT 供应链风险管理。

2.2.3 我国标准

2.2.3.1 GB/T 36637-2018 信息安全技术 ICT 供应链安全风险管理指南

2018 年 4 月，美国商务部发布公告称，美国政府在未来 7 年内禁止中兴通讯向美国企业购买敏感产品，引起社会广泛关注。该事件反映出我国在某些关键核心部件的研发、生产、采购等环节存在的供应链安全风险，同时凸显出加强我国 ICT 供应链安全研究、评估和监管的重要性。基于此事件，我国及时出台供应链安全管理国家标准 GB/T 36637-2018《信息安全技术 ICT 供应链安全风险管理指南》，采用风险评估的思路，从产品全生命周期的角度，针对设计、研发、采购、生产、仓储、运输 / 物流、销售、维护、销毁等各环节，开展风险分析及管理，以实现供应链的完整性、保密性、可用性和可控性安全目标。

2.2.3.2 GB/T 24420-2009 供应链风险管理指南

该标准主要针对传统供应链风险管理，在《GB/T 24353-2009 风险管理 原则与实施指南》的指导下，参考国际航空航天质量标准（IAQS）9134、美国机动车工程师协会标准 SAE ARP9134 和欧洲航天工业协会标准 AECMAEN 9134《供应链风险管理指南》等编制而成，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，适用于任何组织保护其在供应链上进行的任何产品的采购。

2.2.3.3 GB/T 32921-2016 信息技术产品供应方行为安全准则

为贯彻落实《全国人民代表大会常务理事会关于加强网络信息保护的决定》的精神，加强信息技术产品用户相关信息维护，该标准规定了信息技术产品供应方在相关业务活动中应

遵循的基本安全准则，主要包括收集和处理用户相关信息的安全准则、远程控制用户产品的安全准则等内容。

2.2.3.4 其他标准

其他 ICT 供应链安全管理相关国内标准包括：

- GB/T 22239-2019 信息安全技术 网络安全等级保护基本要求
- 银办发〔2021〕146 号 关于规范金融业开源技术应用与发展的意见
- 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
- GB/T 31168-2014《信息安全技术云计算服务安全能力要求》

3

软件供应链安全技术框架

3.1 软件供应链安全

软件供应链安全覆盖整个软件生命周期过程，单从软件产品的复杂性来说，除了保障软件项目自身的安全之外，还包括每个项目的依赖关系与可传递依赖关系，以及这些关系链所组成的软件生态系统的安全。尤其在开源安全问题上，由于间接依赖关系的传递带来隐性漏洞包含问题，使用免责条款替代合同约束带来的信任风险与法律风险，都更显著，也是近几年软件供应链攻击技术暴露较多的地方。

对企业来说，谈及软件供应链安全，需要面临很多具体问题，如：

表 3.1 企业软件供应链安全需求

企业客户对软件供应链的需求
能否获得清晰的软件成分信息，采购时支撑选型决策?
获得可选范围内软件成分或开源组件已知的安全问题（如已知漏洞），危险程度。
供应商是否使用了安全开发工具进行有效管理?
选择的供应商或开源项目能否迅速响应安全问题，协助提供解决方案?
评估法律风险（知识产权、许可证范围、免责条款）、商务风险。
…
如何有效的向最终用户传递软件供应链安全能力?
接收到安全应急通告后，提供的信息是否能支撑最终用户快速完成风险评估?
如何向下游企业、最终用户提供软件供应链安全证明，获得信任?
国际软件公司要求提供 SBOM[6]（国际标准格式，软件成分清单）。
…
如何维护企业自身的软件供应链安全?
如何及时检查、评估软件直接依赖、间接依赖的开源组件，第三方组件安全?
如何记录、跟踪软件的组件的详细信息与更新情况，发生问题的时候能进行溯源、定位?
向下游企业、最终用户传递 SBOM 时，如何保证不泄漏给第三方?
…

企业要实现软件供应链安全，需要有效的机制来清晰的传递上下游间所依赖的软件信息，需要通过技术手段和管理手段来保障企业自身的安全和软件产品的安全，需要行业内形成可

[6] Software Bill of Materials | CISA

以相互信任的机制，向最终用户提供软件供应链安全的体系化评估方案。下面我们将基于这三个理念来阐述企业进行软件供应链安全管理所需要达到的能力。

3.2 软件供应链安全理念

一、软件供应链成分透明程度

企业要管理软件供应链，首先要有一种统一的描述方法，能清晰的传递上下游间的软件信息，向最终用户展示软件供应链的组成成分的情况。这种软件供应链描述方法，与软件资产管理中定义的软件标识相比，要深入软件产品的组成结构，力求从安全视角能足够充分的刻画软件产品的组成成分与依赖关系。

较早，在美国 H.R.5793 法案中参考传统供应链管理的“物料清单”的概念，提出了软件“最小要素”及“软件物料清单”（SBOM）的要求规范。目前，美国的政府单位，如 NIST、CISA 都将 SBOM 确定为推动软件供应链风险管理的优先事项，并在多项政策、标准中将其作为方法基础。然而，在实际使用中，SBOM 也遇到了挑战，如“最小要素”在软件开发过程中是对应具体的代码文件，还是功能模块。当粒度过细的时候，很多企业也担心过度暴露内部代码结构，无助于最终用户识别漏洞，发现安全风险。

考虑企业可实际落地情况，我们将软件“最小要素”分为不同粒度的“软件成分”，并引入软件供应链成分透明程度的概念描述，透明程度越高的软件成分粒度越小。

透明度指标	不透明	微透明	半透明	透明
软件成分作为“最小要素”单位的颗粒度	软件整体作为一个软件成分。	直接依赖检测出的开源组件、第三方组件，组件基本信息完整，直接依赖关系正确。	通过组件指纹识别出的开源组件、第三方组件，组件基本信息完整，直接依赖关系正确，包含必要组件扩展信息完整（如：核心组件的开源知识产权信息、关联漏洞信息）。	通过代码片段识别出的开源组件、第三方组件，组件基本信息完整，直接依赖及间接依赖关系清晰，包含重要的组件扩展信息完整。

软件成分在不同透明程度下，都会构成一张软件成分信息集合组成的“软件成分清单”，包括组成软件的所有组件名称、组件的信息、组件之间的关系以及层级关系。每一个软件，都对应一个组成成分表，通过标准的数据格式存储、记录构成的基础信息，并根据这些存储的数据唯一地识别出这些软件中的组件，溯源组件的来源与维护状态。

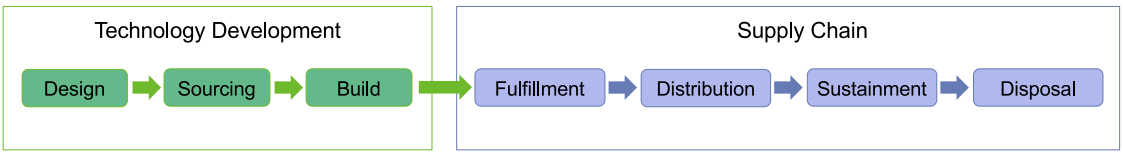
软件成分信息兼容 SBOM 标准，信息示例如下表：

表 3.2　软件成分信息

类型	组成项	描述
软件基本信息	软件名称	标识软件的实体名称
	软件作者名称	软件责任人或团体名称
	软件供应商名称	原始供应商名称
	软件版本	供应商用于标识软件修改的版本标识符
	软件列表、软件	包括开源许可证版权与开放标准、第三方授权信息等
	时间戳	记录软件基本信息生成的日期和时间
	软件信息签名（如校验哈希值）	保证软件信息真实性、完整性
	唯一标识	用于标识软件或在软件成分清单数据库中查找的唯一标识符
软件间的关系	包含关系	如源代码与编译后二进制的包含关系，发布容器镜像与二进制的包含关系等
	依赖关系	包括代码显示依赖、包依赖、编译依赖、运行时依赖等
	其他关系	其他关联关系
软件扩展信息	软件知识产权信息	包括开源许可证版权与开放标准、第三方授权信息等
	关联漏洞信息	漏洞信息，如对应 CVE、CNVD、CNNVD 等
	备注说明	

二、软件供应链组成可评估能力

为应对供应链的威胁，保障自身的 IT 基础设施安全，企业需要构建软件成分清单来梳理供应链产品，识别和管理关键软件供应商，在供应链的生命周期的各阶段通过安全评估控制安全风险，削减供应链攻击带来的威胁。



1. 软件开发过程安全可评估

在软件的开发生命周期中，漏洞的引入不止有开发人员编写的源代码，还有所依赖的开源组件、开发和构建工具等，依照软件的开发和构建过程，企业需要从如下方面建设开发过程安全评估能力。

- 开源组件引入安全管控

由于开发成本的优势，在项目中广泛使用开源组件已成为主流的开发方式，包括开发框架、

功能组件等，在系统开发过程中，应严格控制引入的开源组件风险，将已知漏洞摒除于软件交付运行之前。

开源组件安全评估能力由软件成分分析（SCA）提供，在代码构建时，通过 SCA 工具对项目的第三方组件依赖进行漏洞分析，由开发人员及时处理存在漏洞的组件。

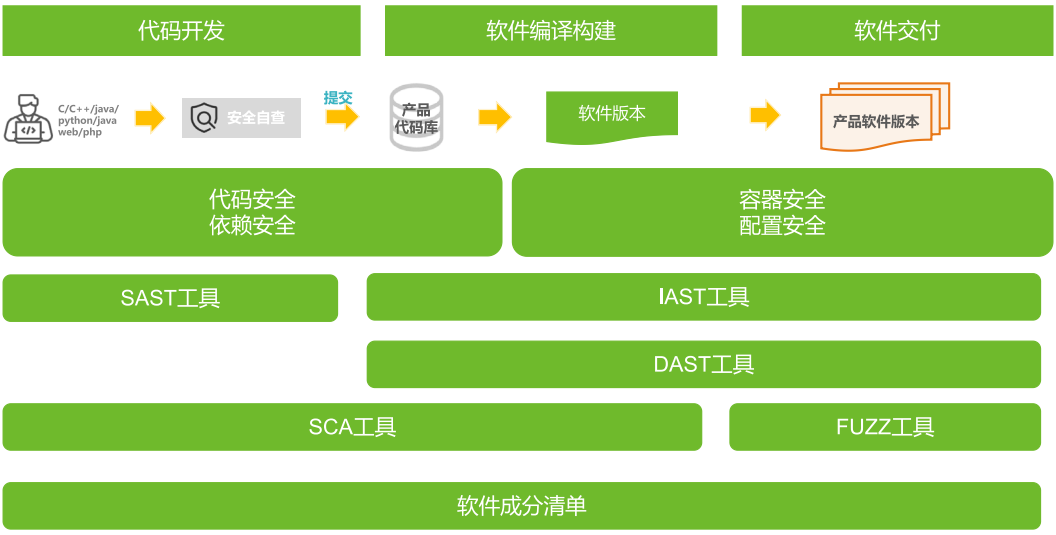
- 源代码漏洞管控

系统开发过程中，不规范的安全编码也会引入漏洞风险，如 SQL 语句拼接所导致的 SQL 注入漏洞。

为降低开发过程中源代码漏洞的产生，提升源代码安全质量，可使用 SAST（Static Application Security Testing，通常指静态源代码安全审计工具）在开发 IDE 工具和持续集成过程中，对源代码进行安全扫描，或者使用 IAST（Interactive Application Security Testing，通常指交互式安全测试工具）在系统运行时通过污点分析检出源代码漏洞。

2. 软件交付安全可评估

软件交付是开发商将成品完成封装，交付给下游用户，也是下游企业引入供应链产品的起始点，作为供应商，除保证交付软件安全外，也应将软件成分清单一并交付给下游企业；供应链下游企业在获得供应商的软件组件成分清单后，也可同步向其下游企业、最终用户提供自己维护的软件组件成分表，那么最终用户就已经具备了依据安全通报、威胁情报监控等第三方信息来分析、评估软件供应链安全的基本条件。



3. 软件运营安全可评估

供应链软件产品交付运行后，供应商应在产品的生命周期内提供安全保障服务，对产品漏洞及时修复；最终用户也应根据供应商所提供的 SBOM 将供应链产品纳入企业资产管理，定期对资产进行安全评估，结合漏洞预警，对受影响的产品进行加固和修复。

在实现技术上，对软件供应链组成的可评估能力与企业内部进行的安全开发治理有很多相同之处。需要指出的是，软件供应链安全更关注整个软件生命周期中每个迭代或每个构建涉及的不同组件、依赖关系的变化等特征，也同时关注整个系统的安全性评估状态。

三、可信任软件供应链

软件供应链安全的复杂性之一在于多级上下游安全问题的堆叠，很难依靠企业自身力量完成整个链条的安全评估与把控，需要从监管层面加强供应链产品安全认证管理，提供企业 SBOM 托管和可信认证服务，与企业共同构建一个可信的软件供应链生态体系。

1. 向软件行业提供开放的、可信任的软件供应链关键数据（如软件成分信息、组件情报信息）管理机制。

软件产品最终用户的软件供应链安全依赖与上下游企业与这个产品相关的组件信息的集合，透明程度高的软件成分信息对软件供应链治理至关重要，但是对软件企业来说增加了产品管理成本和安全风险。需要制定一套可信任的机制，既能鼓励软件企业参与软件供应链治理的积极性，又要保证软件供应链基础信息的高质量，向最终用户提供可信任的软件供应链评估核心数据。

2. 软件行业的供应链风险监控与管理方法。

企业需要监控软件产品及依赖的上游组件中是否存在高危组件，下游交付环节中使用软件公开基础平台（如云计算平台）或网络基础设施是否出现问题，同时要管控开源软件的使用，建立开源软件资产台账，持续监测和降低所使用开源软件的安全风险。但实际上，企业很难监控上下游的供应链风险监控与管理情况，需要从监管层面对高危组件、软件公开基础平台、网络基础设施、高风险开源软件进行监控，通过如黑白灰名单等机制提供给软件企业和软件用户进行软件供应链风险监控、管理。

3. 向社会提供可信任的厂商软件供应链安全度量标准与认证体系

可信任的软件供应商，应从可信需求分析与设计、可信开发、可信测试、可信交付、生命周期管理、开源及第三方管理、配置管理等不同维度进行评估。评估符合要求的企业应具

备良好的软件供应链安全评估与监控、验证能力，管理风险。

企业也需要完善供应链资产管理和安全检查，可借助知识图谱技术理清企业供应链依赖关系，从而在监测到预警时能够从容应对。

四、软件供应链安全理念与安全评估的关系

软件供应链透明程度、软件供应链安全可评估能力、可信软件供应链的三个理念，与最终用户能进行软件供应链安全检测与评估紧密相关，按：

1. 能对软件成分的安全性进行基础评估

上下游企业能向最终用户提供微透明程度的软件供应链成分信息查询能力。最终用户通过第三方威胁情报监控，或主管部门发布的安全通告，能获得受影响组件信息。通过查询、比对软件成分清单内的名称与版本信息，可以快速比对是否存在已知漏洞等安全问题。考虑到最终用户需要维护软件的便利性与未来自动化检测的需求，软件成分清单应为机读格式。

2. 能对软件成分的安全性进行完整评估

透明程度高的软件成分清单（半透明程度及以上），能显著提升最终用户进行软件供应链安全评估的准确性。对企业开展业务来说，知识产权等法律风险评估是必不可少。尤其对于开展国际业务的用户来说，项目使用的开源项目及第三方组件是否遵循许可证要求，合法地进行软件修改与重用，都存在着潜在法律风险。建立政府统一管理的情报库实现软件供应链安全预警平台。

3. 能评估软件产品开发过程安全性

完整的供应链覆盖了从开发设计到交付实施再到用户使用的各个环节，牵扯到最终用户和各级供应商，每个环节都有可能成为网络攻击的突破口，企业在引入供应链软件产品时需要充分对其进行安全评估，以降低引入的风险；

依赖安全评估：通过 SCA 工具，结合供应商提供的 SBOM，对其依赖的第三方组件进行风险评估，防范引入已知的漏洞和开源授权使用风险；

软件漏洞评估：根据供应商所提供的产品安全评估报告，包括但不限于 SAST、SCA、DAST 和 IAST 等工具的扫描报告，确认交付产品时不存在未修复的中高风险漏洞。

4. 能评估软件产品依赖开源软件的供应链安全性

企业管控开源安全，既包括直接依赖组件的漏洞，也包括间接依赖组件漏洞。随着软件

系统架构愈加复杂，所关联的开源组件之间的依赖关系也愈加复杂，一些多级依赖的组件漏洞、运行时依赖的攻击面都容易被忽略。

开源软件生态当前已经丰富繁杂，监管层应针对开源软件进行风险监测与监控。安全是一个长期持续且动态变化的过程，企业侧则需结合 SCA 建立自身开源资产台账进行开源软件资产安全管理。其曝出漏洞或者发布补丁时，一方面使用评估检查工具对资产进行漏洞影响排查，另一方面及时对受影响资产进行加固修复。

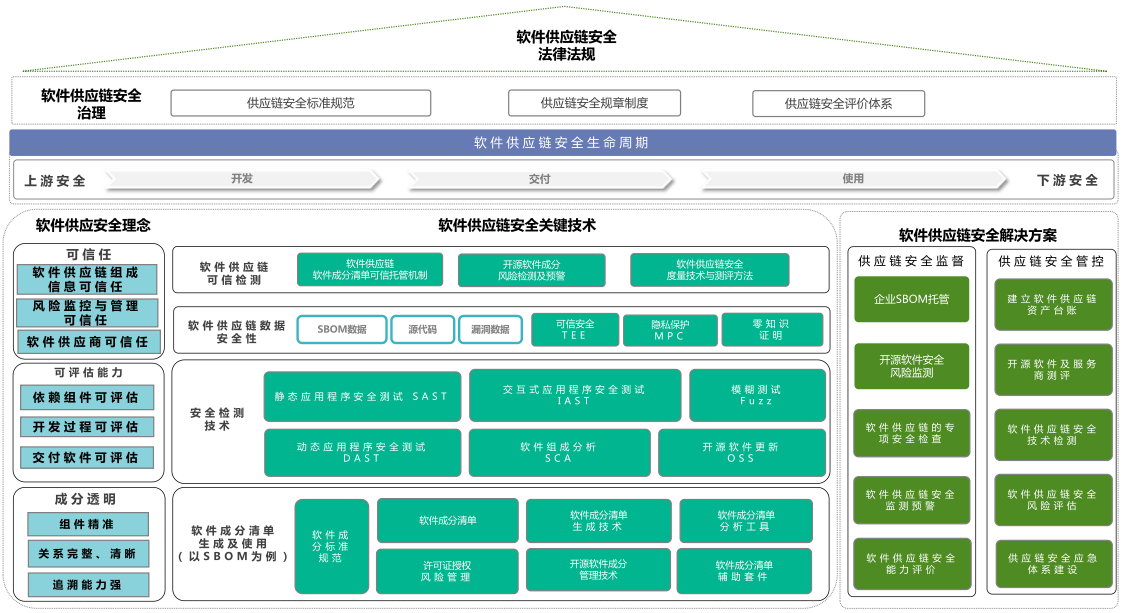
5. 可信任的软件供应链安全监管治理

被检测对象首先需依据检测机构提供的可信安全厂商获取安全检测服务及工具，保证工具及来源的安全合规。对自身软件资产进行审计，输出标准化清单：软件成分、代码审计、供应商资质等标准化清单。将清单提供给安全管理机构进行备案，以备审查。当安全主管机构审查时，使用标准化设备对软件资产，供应商资质，开发环境，应急方案等场景进行审计，对比清单内容并提出改进建设指导意见。保证全流程的安全可信。

6. 可信任的软件供应链生态

信创供应链安全是一个系统工程，也是企业重点关注的问题，随着网络攻击技术的发展，我们需要创新协同，积极引入可信计算、多方安全计算、区块链等技术，创新安全产品体系架构，不断提升产品自身安全性；以产品全生命周期安全保障为目标，从部件供应商、产品研发、产品测试、产品生产、储运销售、产品运维、召回等环节，运用可信计算、多方安全计算、区块链等技术建立信创产业可信软件、可信硬件供应链安全保障技术体系。

3.3 软件供应链安全技术框架



软件供应链安全管理依托现有的法律法规对供应链安全治理提出的指导意见与管理要求，多角度考量，将供应安全标准规范，供应链安全规章制度与供应链安全管理体系相融合，制定软件供应链安全治理顶层设计，管理范围覆盖软件供应链开发、交付到使用全生命周期及资产链条的上游研发至下游用户侧全方位覆盖。

软件供应链安全首先要树立正确的安全意识，目标是将系统打造成可信任、可评估、组成成分透明的可信实体。将安全检测结合安全可信白名单机制、风险预警、与情报收集机制保证内部环境安全。通过建立软件成分清单（如 SBOM），源代码管理、漏洞库管理等安全风险管控机制，保证软件供应链数据的安全可信。同时，配合安全检测技术、如代码审计、软件成分分析、动态安全检测等技术支持可评估能力建设。加快建立软件成分清单生成与使用规范，建立管理手段与工具方法库，标准化软件成分和软件成分可视化流程，保证组件透明理念的落实。

将安全理念与关键技术相融合，实现软件供应链安全技术保障，将具体方式方法抽象出可复制可执行的安全解决方案。针对具体场景实施供应链安全监督与安全管控，落实好监管部门所关注的供应链安全重点领域管理意见及建议，实现企业内部软件供应链安全防护体系化建设。

4

软件供应链安全关键技术

4.1 软件成分清单生成及使用技术

软件成分清单与软件物料清单（SBOM）之间的差别在于对软件“最小要素”的颗粒度的要求，在技术思路、实现步骤上并无本质差别。考虑到软件物料清单的生成工具与技术研发已相对成熟，在 4.1 章节我们将通过介绍 SBOM 的各项关键技术，来阐释软件成分清单的生成与使用过程。

先简要介绍 SBOM。国际上正在推广的 SBOM，可以看作软件成分清单的一种实现标准，根据 NTIA 提供的指导文档，要求软件企业提供的 SBOM 是一个正式的、机器可读的列表，以实现软件供应链的自动化识别与管理的需求。理想条件下，供应链中每一环节都要求该环节的上游环节提供 SBOM，同时该环节应提供 SBOM 给下游环节；SBOM 需要支持多层级的组件信息 (例如操作系统、安装器、包、文件等)；SBOM 还需要根据组件的改变而更改 (通过更新、补丁等达成)。当前 SBOM 有三种最为主流的格式，他们分别为：SPDX[7]、SWID[8] 以及 CycloneDX[9]。

由 LINUX 基金会主导的 The Software Package Data Exchange（SPDX）项目，旨在通过定义报告信息的标准来帮助减少软件的歧义。SPDX 通过为企业和社区提供共享重要数据的通用格式来减少冗余工作，SPDX 规范作为 ISO/IEC 5962：2021 被公认为安全性、许可证合规性和其他软件供应链工件的国际开放标准。SPDX 支持众多文件格式（.xls，.spdx，.rdf，.json，.yml 以及 .xml），特征是包括组件、许可证、版权以及开放标准。

SWID 由 NIST 推出，只支持 .xml 格式，由一组结构化的数据元素组成，标识产品、版本、产品生产分发中的组织和个人、组件信息、产品和其他描述性元数据之间的关系等信息。

CycloneDX 是一种轻量级 SBOM 标准，由 OWASP 推出，支持 .json 和 .xml。

不管选用哪种标准和格式，建立软件成分清单生成及使用机制都是十分必要的。这不仅有助于应对日益增多的软件供应链攻击事件，还能帮助供应链中下游环节理解上游环节的意图、解决内部环境的冲突，帮助企业更好的把握软件结构，并更好地管理软件风险。

4.1.1 软件成分清单生成技术

生成软件成分清单需要考虑两种情况：主动提供形式，即软件提供商主动提供软件成分清单，此时生成软件成分清单的工具可以包含在软件版本控制系统中，同时能够提供一定的

[7] International Open Standard (ISO/IEC 5962:2021) - Software Package Data Exchange (SPDX)
[8] NVD - SWID (nist.gov)
[9] OWASP CycloneDX Software Bill of Materials (SBOM) Standard

组成成分更新可追踪、历史版本可追溯能力；而另一种是被动分析形式，即下游环节需要自行分析软件成分。

1. 供应商主动提供组成成分

从 DevOps 角度来看，一个软件的生命周期包括规划、开发、构建、测试、发布、部署、运维运营以及监视阶段，如果是一个软件供应链中间环节的供应商，还可以从开发阶段中分出采购环节。

如果软件供应商是在软件研发后生成软件成分清单，只能依赖现有的设计文档、测试报告提取信息，借助人工的方式还原软件成分，毫无疑问，这样不仅效率低，也容易遗漏已更新的软件成分信息。甚至，可能演变为与被动分析模式相差无几的情况，增加了分析的难度，降低了追溯力。

那么供应商应该如何构建软件成分清单呢？如下图，NTIA 提供了一个“SBOM 生成流水线”，展示了软件开发商将 SBOM 的生成过程与整个 DevOps 流程相融合的图景， SBOM 跟随 DevOps 中的每一步，规范、高效且流程化地生成。

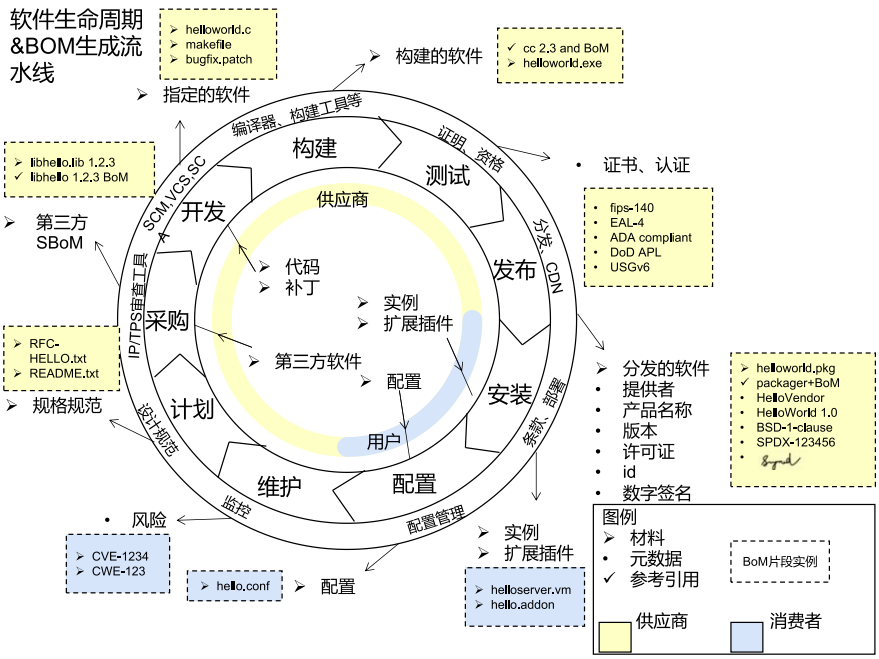


图 4.1　NTIA – 现有 SBOM 格式及标准调查 2021[10]

[10] NTIA SBOM formats and standards whitepaper.[Online.] Available: https://www.ntia.gov/files/ntia/publications/ntia_sbom_formats_and_standards_whitepaper_-_version_20191025.pdf

首先是规划阶段 (plan)。规划阶段，可以将软件的设计、规划阐明，加入 SBOM 文档。上文我们提到，可以将采购环节分离出来，作为一个独立的环节。此处采购的软件可以是开发中必须的第三方工具、插件、软件包、资源库等等，只要能够提供相应的第三方 SBOM 文档，将其声明引用在 SBOM 文档中即可。但是如果对方无法提供 SBOM 文档，要么寻找替代品，要么只能被动接受，此时就涉及到第二类 SBOM 生成，即“被动”生成的方法了，这里暂不论述，下文会进行说明。

来到开发阶段，此阶段包括初期编程开发以及后期编写软件补丁。在开发过程中，需要 SCM（软件配置管理管理）、VCS（版本控制系统）等管理系统，同时对于进阶的 DevSecOps 来说，这一阶段往往还需要应用 SCA（软件成分分析）、SAST（静态代码分析）等分析技术。此时，应当将源码、生成文件以及补丁的信息录入 SBOM。

进入构建阶段，构建完成时，应当将构建信息写入 SBOM 文档。至此，软件初步开发完成，进入测试环节。

测试环节对软件的性能、稳定性、可用性等衡量标准进行评测，DevSecOps 中还会进行黑盒、内外部渗透、交互式应用安全测试（IAST）等安全测试。通过测试后，对软件进行签名认证，并将所用标准、证书信息写入 SBOM。至此，软件生命周期中的开发周期暂时结束。

软件分发阶段，应完善 SBOM 信息，使其包括但不限于 NTIA 要求的最低标准信息（作者信息、提供商、产品名称、版本号、组件信息的哈希值以及 id）以及数字签名；开源软件还应在 SBOM 中声明许可证 license。

部署阶段，可以给 SBOM 附上条款、插件以及配置信息。

最后，在维护 / 监控阶段，最理想的情况是将已知安全漏洞信息插入文档，可以参见本章辅助套件小节对 VEX 的说明。

值得一提的是，在现有 SBOM 三大标准之中，除了 SWID 支持语言不多，SPDX 以及 CycloneX 在 java/python/javascript/golang/Maven 等语言中都有提供相应的许可证库以及 SBOM 文档生成插件。

2. 自行分析软件成分

如果不能获得软件提供商的一手软件成分清单或相关资料，那么必要时需要自行分析软件，生成软件成分清单。这种方法，也可用于验证供应商所提供的产品、软件成分信息。

- 对象是闭源程序时，软件企业或最终用户需要通过代码编译信息及配置文件、二进制分析工具、逆向工程等技术进行，通过人工智能算法进行相似性寻找同源、近源程序的组成分析成分表能提升分析能力。
- 对象是开源程序时，软件企业或最终用户无法直接获得软件成分清单与更新服务，需要自行维护。目前，企业用户多采用 SCA（Open Source Software Composition Analysis）这类专业开源软件成分分析与管理系统，管理所引入的开源软件的安全风险。自动化的 SCA 工具对应用程序的源代码，包括模块、框架、库、容器、注册表等工件进行自动化扫描，以识别和清点开源软件的所有组件构成和依赖关系，并识别已知的安全漏洞或者潜在的许可证授权问题。把这些风险排查在应用系统投产之前，也适用于应用系统运行中的诊断分析。除了提供开源软件成分可见性之外，一些 SCA 工具还基于漏洞风险等级进行优先级修复开源漏洞或提供相应解决办法。

在软件供应链中，无论是上游还是下游企业，都建议建立自己的软件成分清单，不论是通过数据库获取某一软件成分清单，比对软件成分清单信息判断是否被篡改；还是通过已有软件成分清单改进生成技术，建立完善的软件成分清单生成机制、存储机制，都能对软件安全起到积极的作用。

4.1.2 软件成分清单分析工具

根据 LINUX 基金会提供的分类 [11]，我们可以将 SBOM 工具分为 3 大类，分别是生产类、消费类以及转换类。生产方面，主要包括软件成分分析、SBOM 文档的自动构建以及人工编辑这三种功能；消费方面，分为浏览（人类可读）、SBOM 文件对比以及 SBOM 文件导入三种功能；转换方面，需要实现翻译、合并以及支持功能。当然，一个分析工具可以是这三类中的一类，也可以具备多类的功能。

表 4.1　SBOM 工具的分类

类别	类型	描述
生产	构建	文件在构建软件的过程中自动生成，并包含有关于这个 build 的信息
	分析	对源 / 二进制文件分析，通过检查软件还有相关资源的方式生成 SBOM
	编辑	支持人工访问、人工修改 SBOM 数据
消费	浏览	以人类可读 (图片 / 表格 / 图表 / 文本等) 形式提供信息，辅助决策、商业进程
	对比	能够对比多个 SBOM 之间的区别，以区分不同
	导入	将 SBOM 发现、取回并导入到系统或进一步的处理分析中
转换	翻译	保留相同信息的同时从一个文件类型转变为另一个文件类型
	合并	多个源的 SBOM 以及信息可以合并，以供分析、审计
	工具支持	支持通过 API、库、对象模型或其他引用资源的方式为其他工具所用

[11] https://linuxfoundation.org/wp-content/uploads/LF-Live-Generating-SBOMs.pdf

4.1.3 开源许可证授权风险管理

对于软件开发者来说，在软件供应链中使用开源软件，面对的风险不仅仅在于技术层面安全风险，也包括违反许可证授权的法律风险。

开源许可证，对使用开源项目中的软件代码、二进制等文件进行分发、修改和重用等行为进行合法定义、约束的条款，它规定了软件开发者和软件用户之间应当行使的权利义务。企业向（缺乏宾语）公开自己产品的部分源代码时，需要选择合适的开源许可证保留自身权益；违反开源许可证的修改、重用也会引入潜在法律风险。因此，分析软件产品所引用的开源软件许可证，是合法地使用开源软件的先决条件，特别是对于大型软件系统，许可证信息验证过程就显得尤为重要。

在开源领域内，著名的开源许可证认证国际组织包括 OSI 开源社区 [12] 和 FSF 自由软件基金会 [13]。OSI 目前已批注了 110 多份开源软件许可证，FSF 也列出了 90 多份开源软件许可证。其庞大的数量、繁多的种类，对开源许可证分析工作带来了很多困难，引发了软件产业对包含不同的开源许可证组件产生许可证兼容性冲突问题，商业软件如何选择许可证组合以减少法律问题的相关研究也屡见不鲜。

其中，比较著名的 FOSSology 许可证冲突检测软件与开源 Open Source License Checker (OSLC) 许可证冲突检测软件，可以对软件项目源代码进行开源许可证检查。

随着软件物料清单（SBOM）概念的提出，Georgia M 等人通过使用软件成分分析标准的软件包数据交换（SPDX）结构，来自动化检查许可证兼容性的方法 [14][15]。由此，可以看出，通过在软件成分清单中明确对开源许可证的要求，可以更简单、有效的解决终端用户评估许可证安全风险。为了更好地保障软件供应链的安全，软件许可证风险也应考虑在内，这就需要识别、管理许可证的能力。而管理许可证风险的大前提是不变的——明确软件都有哪些组件，以及它们的依赖关系。

4.1.4 开源软件成分管理

可传递依赖关系引入的安全风险在开源项目中尤为突出，一个代表就是 2021 年 12 月份出现的 Log4j2 漏洞（CVE-2021-44228）。 Log4j2 作为一个堪比标准库的基础日志库，受到

[12] https://opensource.org/
[13] GNU 操作系统和自由软件运动
[14] Kapitsaki G M, Kramer F. Open Source License Violation Check for SPDX Files[C]// International Conference on Software Reuse. Springer, Cham, 2015:90-105
[15] Kapitsaki G M, Kramer F, Tselikas N D. Automating the license compatibility process in open source software with SPDX[J]. Journal of Systems & Software, 2016.

无数开源 Java 组件直接或间接依赖。根据绿盟科技研究员分析 [32]，此次爆发的 Log4j2 漏洞影响范围广泛且危害严重，在各个场景领域皆应获得重点关注。分析这种庞大且复杂的关系链，可以使用知识图谱技术。

知识图谱技术近几年在工业界应用也越来越广泛。如网络安全、医疗、法律、金融垂直领域都已经有比较好的应用案例。知识图谱本质上是一种大型的语义网络，它旨在描述客观世界的概念实体事件及其之间的关系，以实体概念为节点，以关系为边，提供一种从关系的视角来看世界。

参考 CycloneDX 提供的开源组件成分及其关系规范化的格式描述，可以通过关系图描述开源组件的组成成分，如下图所示。

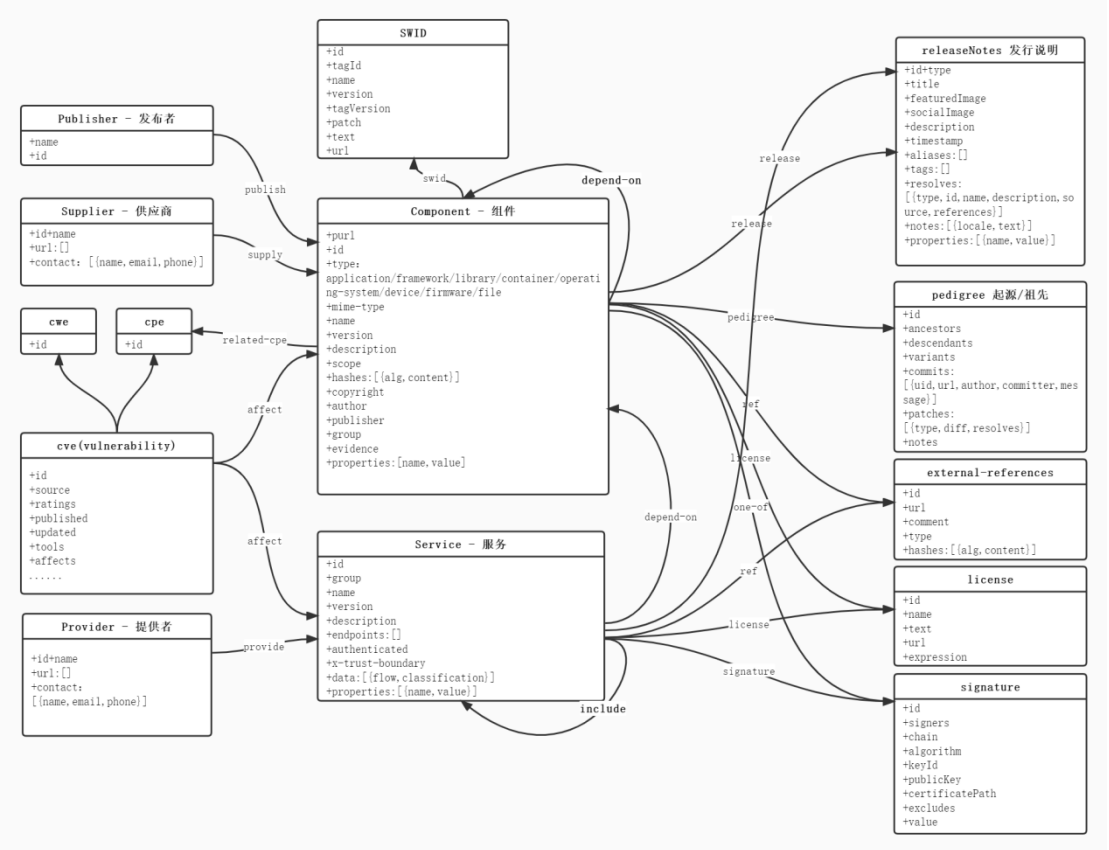


图 4.2　基于 CycloneDX BOM 的图关系模型

通过知识图谱可以很好的解决开源软件复杂的依赖关系表示不清晰问题，让开源软件成分语义化，可视化，变得机器可读可推理：

1．知识图谱的图表示方法构建开源软件的成分深层关系（如依赖、引用、许可等）；

2．使用开源软件相关的漏洞知识库和其建立关系，利用知识图谱的分层传播路径搜索算法有效分析开源软件的漏洞影响，以衡量项目漏洞的严重性；

3．应用图算法分析知识图谱（依赖关系图）获得最具风险的关键依赖关系，推荐有效的修复建议。

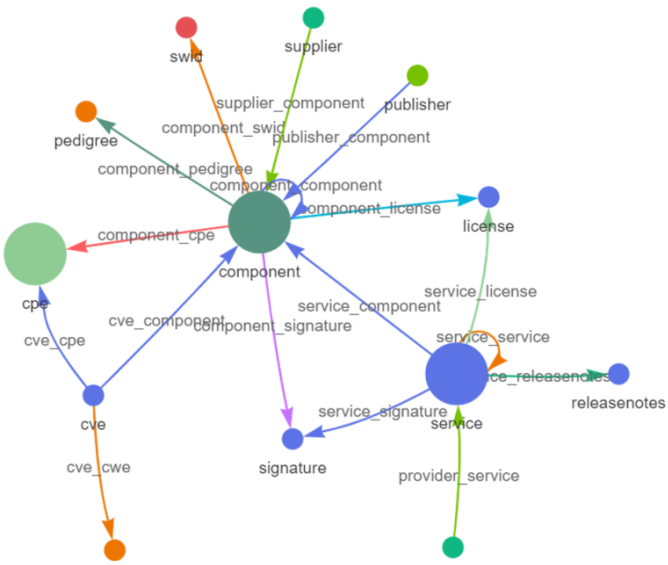


图 4.3　开源软件知识图谱本体模型

在安全分析研究方面，开源软件知识图谱可以帮助安全人员对开源软件进行全面的风险评估分析：

1. 基于开源组件属性特征（如源代码、包等文件的 hash 值）来判定组件是否存在被包装冒用或自身被篡改的风险；

2. 利用不同开源组件的依赖关系，结合组件版本信息，可以在高危漏洞应急响应中快速识别受影响的其他组件，通过图的聚合及子图拆分等图优化算法能够高效、持续的输出风险分析结果。

如下图，展示了 log4j-core@2.3 组件存在的多个高危漏洞关系，如果在软件中引用了存在高危漏洞的开源组件，无疑会给软件产品带来潜在的威胁，同样，软件供应商的软件产品也会面临同样的安全风险。

图 4.4　log4j 组件的依赖图谱

4.1.5　软件成分清单辅助套件

4.1.5.1　VEX

VEX（vulnerability exploitability exchange，漏洞可利用性交流 / 交换）概念和格式是美国国家通信和信息管理局 (NTIA) 开发的 [16]，它可以列出某一软件 / 组件某一版本中的漏洞，帮助用户获得这些漏洞的状态、信息，并借此评估这些漏洞的可利用性（辅助用户判断这些漏洞是否对软件有影响——不影响软件 / 已经修复 / 调查中 / 影响软件；以及如果受到影响，建议采取哪些补救措施）。部分情况下由于各种原因（例如，编译器未加载受影响的代码，或者软件中其他地方存在一些内联保护），导致上游组件中的漏洞不会被“利用”，此时可以对用户进行告知，节省用户调查的成本。

要生成一份 VEX 文档，需要三个步骤：组件标识、漏洞标识以及漏洞状态（见下图）。

[16] https://www.ntia.doc.gov/files/ntia/publications/draft_requirements_for_sharing_of_vulnerability_status_information_-_vex.pdf

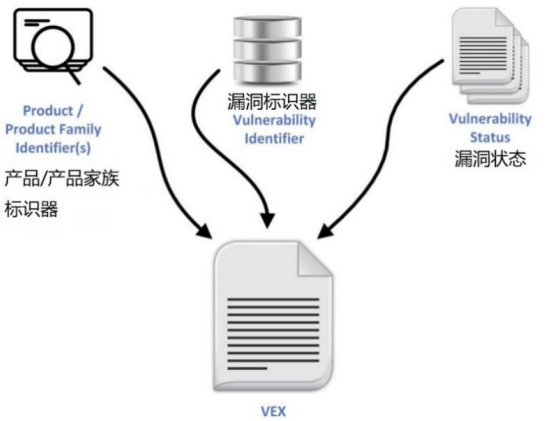


图 4.5　生成 VEX 所需过程 [17]

表 4.2　生成 VEX 所需过程

VEX 数据域		补充
VEX 目标	组件标识或者组件族标识	与 SBOM 相关联
VEX 元数据	VEX 文档的 id	-
	作者	-
	作者作用 / 角色	-
	时间戳	-
	完整性 / 签名 / …	-
	SBOM id（可选）	-
漏洞（每一个标识出的漏洞）	漏洞 id	-
	漏洞状态（机器可读）	-
	漏洞详细信息	-
	影响的组件（可选）	-

上表是 NTIA 给出的 VEX 数据结构初稿，为了更好地理解 VEX 文档的结构和内容，可以参考 Cyclone 在 github 上给出的示例文档（json 格式）[18]。

VEX 可以看作是 SBOM 的特殊需求，是 SBOM 所需（所要求）基本信息的延伸与扩展。但 VEX 并不必须与 SBOM 一起使用，也不要求整合到 SBOM 内（根据 NTIA 以及 CycloneDX 的描述，VEX 可以与 SBOM 合并，也可以作为独立的文档存在）[19][20] 这是因为 SBOM 对于给定的构建通常是静态的，但 VEX 在更改中可能是动态的（如漏洞当前状态可能随着调查

[17] https://www.ntia.doc.gov/files/ntia/publications/framing_2021-04-29_002.pdf
[18] https://github.com/CycloneDX/sbom-examples/blob/master/VEX/vex.json
[19] https://www.ntia.doc.gov/files/ntia/publications/vex_one-page_summary.pdf
[20] https://cyclonedx.org/capabilities/vex/

有所进展而变动）。这种差异导致如果将 VEX 与 SBOM 整合为一个文档，那么每次 CVE 内容发生变动的时候，就需要重新生成一个重复的 SBOM 文档，而这份多余的成本是可以通过 VEX 与 SBOM 的分割而避免的。因此，供应商应在实际应用中根据具体情况，选择是否将 VEX 与 SBOM 整合。下面是 NTIA 给出的一个流程图。

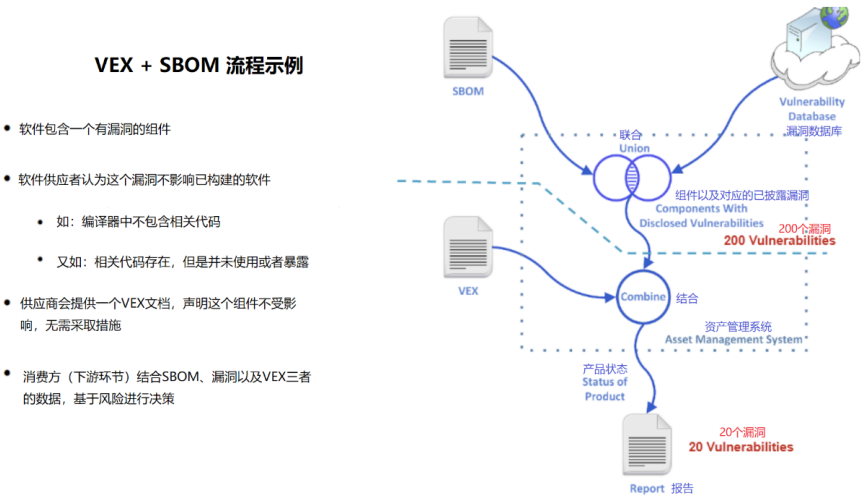


图 4.6　利用 VEX 和 SBOM 进行风险决策 [21]

VEX 已作为通用安全咨询框架 (CSAF) 中的配置文件实施。CSAF 是由 OASIS Open CSAF 技术委员会开发的机器可读安全建议标准。正如 CSAF 所定义的，VEX 还可以提供丰富的信息，例如供应商、系统集成商和运营商可以提供的修复、变通方法、所需的重启 / 停机时间、漏洞评分和风险。VEX 也可以在其他标准或框架中实现 [22]。

专家预计， VEX 概念“可能会在 2022 年迅速流行，因为它减少了供应商和最终用户的工作量”。供应商可以轻松地传达客户因漏洞而面临的风险（或不存在风险）。供应商可以单击一个按钮并为公司生产的所有产品的所有版本创建一个 VEX 文档，而不是生成一个长长的 PDF 列表 1000 多个产品及其潜在的漏洞暴露。这大大减少了在漏洞恐慌时安抚客户所需的努力。VEX 基于 JSON 格式，机器可读且易于构建工具。

4.1.5.2　SaaSBOM

SaaSBOM 是 CSOonline 提出的一个简单的框架，建议 SaaS 提供商仔细分析所售产品中为了实现数据的机密性、完整性、可用性所依赖的技术堆栈，表示方式则可以选择 SPDX 或

[21] https://www.ntia.doc.gov/files/ntia/publications/framing_2021-04-29_002.pdf
[22] https://www.secrss.com/articles/38485

SWID[23]。与 VEX 一样，SaaSBOM 可以看作对“传统”SBOM 的延伸与扩展——“传统”SBOM 只涵盖了传统软件供应链，而 SaaSBOM 则创新地将“软件即服务（SaaS）”的供应商也包含在内。

CycloneDX 则给出了更详细的信息：SaaSBOM 通过提供复杂系统的逻辑表示形式来补充 IaC（基础设施即代码），包括所有服务的清单、它们对其他服务的依赖、端点 URL、数据分类以及服务之间的数据定向流。可选地，SaaSBOM 还可能够包括构成每个服务的软件成分 [24]。SaaSBOM 的相关示例可以在 github 上查看 [25]。

此外，与 VEX 类似，由于 SaaSBOM 中部署信息更加动态，用户也可以视实际情况，选择将保持 SaaSBOM 的独立，或将其与 SBOM 整合。

4.2 软件供应链安全检测技术

软件供应链安全检测技术需要覆盖整个软件生命周期，主要涉及五类安全检测技术，分别是软件组成分析 (SCA)、静态安全分析 (SAST)、动态应用安全测试 (DAST)、交互式应用安全测试 (IAST) 和模糊测试 (FUZZ)。囊括了软件设计、开发、测试、运营各阶段的安全检测。不同的检测技术能够解决软件供应链特定阶段的安全问题，下表对五类技术作了简要介绍以及对比分析。

对比项	SCA	SAST	DAST	IAST	FUZZ
检测对象	源代码或二进制文件中依赖的开源和第三方组件	源代码	运行中程序的数据流	运行中程序的源代码、数据流	运行中程序
检测阶段	设计 / 开发	开发 / 测试 / 上线	测试 / 运营	测试 / 运营	开发 / 测试
误报率	低	高	低	极低（几乎为 0）	低
测试覆盖度	高	高	低	中	低
检测速度	与检测技术有关	随代码量呈指数增长	随测试用例数量稳定增加	实时检测	随测试用例数量稳定增加
漏洞检出率	高	高	中	较高	中
影响漏洞检出率因素	检测技术，组件知识库丰富程度	检测技术，缺陷规则	测试用例覆盖度	检测技术，缺陷规则	测试用例质量，覆盖度
使用成本	较低，漏洞基本无需人工验证	高，需要人工排除误报	低，基本没有误报	低，基本没有误报	低，基本没有误报
支持语言	区分语言	区分语言	不区分语言	区分语言	不区分语言
侵入性	低	低	较高，脏数据	低	低
CI/CD 集成	支持	支持	不支持	支持	不支持

[23] https://www.csoonline.com/article/3632149/the-case-for-a-saas-bill-of-material.html
[24] https://cyclonedx.org/capabilities/saasbom/
[25] https://github.com/CycloneDX/sbom-examples/blob/master/SaaSBOM/apigateway-microservices-datastores/bom.json

以下对这五种检测技术分别进行详细介绍。

4.2.1 软件组成分析 (SCA)

在软件供应链中，开源软件因其开放、共享、自由的特点，而被广泛应用。开源代码的使用大幅提高了软件研发的效率，降低了开发的成本。但是由于开源项目的质量参差不齐，很容易存在安全漏洞，或者受到攻击者的恶意篡改，项目维护者也可能未能对漏洞进行及时修复，造成极大的安全隐患。当今开源软件数量呈指数增长，开源软件之间存在复杂的依赖关系，导致无法单纯通过人力对漏洞进行筛查。为了自动和高效地解决开源应用威胁这一问题，软件组成分析技术（SCA）应运而生，这是目前对应用程序组成分析、漏洞检测最有效的办法之一。

SCA 是一种静态的，白盒的检测技术，通过自动化流程，对软件源代码、二进制文件进行分析，识别其物料清单 SBOM，检测其中存在的漏洞和许可违规风险，帮助用户及时修复。

SCA 从理论上来说是一种通用的分析方法，可以对任何开发语言对象进行分析，Java、C/C++、Golang、Python、JavaScript 等等，它关注的对象是构成软件的第三方工件，以及工件之间的依赖关系。SCA 从分析过程来看，可以概括为源代码 / 二进制分析、特征提取和识别、漏洞检测、SBOM 生成。

（1）源代码 / 二进制分析

从 SCA 分析的目标程序形式上分，既可以是源代码也可以是编译出来的各种类型的二进制文件，分析的数据对象对程序架构，编译方式都是不敏感的，比如：类名称、方法 / 函数名称、常量字符串等等，不管目标程序运行在 x86 平台还是 ARM 平台，不管是 windows 程序还是 Linux 程序，都是一样的，简而言之 SCA 是一种跨开发语言的应用程序分析技术。二进制分析格式如下：

二进制格式类型	详细
纯二进制格式	C/C++ 编译后的二进制文件 Java 编译后的二进制文件 .class .NET 编译后的二进制文件 Go 语言编译后的二进制文件
压缩包	Gzip（.gz）、bzip2（.bz2）、ZAP（.zaip,.jar,.apk 和其他衍形式）、7-Zip（.7z）、TAR（.tar）、RAR（.rar）、ARJ（.arj）、XZ（.xz）、LZMA（.lz）、LZ4（.lz4）、Compress（.Z）、Pack200（.jar）
安装包	Red Hat RPM（.rpm） Debian package（.deb） Mac instance（.dmg,.pkg） Windows installers（.exe,.msi,.cab）
固件格式	Intel HEX、SREC、U-Boot、Android sparse file system、Cisio firmware

二进制格式类型	详细
文件系统 / 电子盘	ISO 9660/UDF（.iso）、QEMU Copy-On-Write（.qcow2,.img）、VMware VMDK（.vmdk,.ova）、VirtualBox VDI（.vdi）、Windows Imaging、ext2/3/4、JFFS2、UBIFS、RomFS、FessBSD UFS
容器	Docker Image

（2）特征提取与识别

组件识别通常采用包管理解析、指纹识别两种技术。

- 包管理解析

通过分析源代码中包含的包管理配置文件，如 Python 项目中的 requirements.txt、NPM 项目中的 package.json 等，直接得到对应的软件组成。包管理解析的执行效率和准确率都很高，但是单纯的包管理解析，不适用于很多场景，例如： 对没有包管理的一些源码或者一些二进制文件等，就会产生漏报，如 C 语言本身是没有包管理器的，直接在代码中使用了开源组件的源码，因此也就无法判断出来使用了哪些组件。包管理文件也可能存在未定义组件版本的情况，此时无法定位实际使用的版本，也就无法确定该组件是否存在漏洞。商业软件中一般会使用包管理解析和指纹识别结合使用的方式，来提高组件识别的覆盖率。

- 指纹识别

将目标文件进行特征值计算，将计算得到的特征值与组件特征数据库进行比较，匹配组件信息。特征值计算常用的算法有多种，算法的不同会直接影响分析的准确性和分析效率。

1. 结构化的指纹识别，通过分析一个目录下的结构，来生成指纹做相似度对比，识别开源组件
2. 特征码的指纹识别，通过识别本地的文件的哈希值，文件的大小信息等，来识别开源组件
3. 代码片段的指纹识别，通过精细化将代码进行片段化切片，形成代码指纹，比较相似度，来识别组件
4. 二进制的反编译化的代码指纹，来进行相似度比较，识别组件
5. 字符串搜索和定制化指纹，用来识别非开源组件和第三方商业组件

（3）漏洞检测

将识别出的组件与组件漏洞库进行比对，发现组件中存在的漏洞、许可证授权风险，并提供解决方案，帮助用户修复问题。

（4）SBOM 生成

SBOM 包含软件的所有组件构成以及依赖关系，漏洞、许可证等关键信息，对于供应链过程中的透明程度至关重要。当前实现 SBOM 的格式有三种：

1、SPDIX：Linux 基金会的根本投入。该计划是沟通 SBOM 信息的开放标准，包括组件、许可证、版权和安全引用。

2、CycloneDX：专为安全上下文和供应链组件分析而构建。

3、SWID 标记：由记录关于软件组件唯一信息且协助存储管理计划的文件组成。

	CycloneDX	SPDX	SPDX Lite	SWID
定义（Definition）	一种轻量级 SBOM 标准，设计用于应用程序安全上下文和供应链组件分析	一种标准语言，用于以多种文件格式传达与软件组件相关的组件、许可证、版权和安全信息	是 SPDX 的轻量级子集，适用于不需要完整 SPDX 的情况。 它旨在让那些没有开源许可知识或经验的人易于使用，并成为“某些行业中 SPDX 标准和实际工作流程之间的平衡”。	SWID 标准定义了一个生命周期，其中 SWID 标签作为软件产品安装过程的一部分添加到端点，并在产品卸载过程中删除。
历史（History）	CycloneDX 最初旨在解决开源组件的漏洞识别、许可证书合规性和过时组件分析 核心工作组于 2017 年起源于 OWASP 社区，在内部广泛使用后，变成 OWASP 的开源项目 除了开源信息之外，CycloneDX 更关注漏洞和安全性。	Linux 基金会自 2010 年以来一直在开发和完善 SPDX。据 Linux 基金会称，“SPDX 的创建是为了提供一种通用的数据交换格式，以便可以收集和共享有关软件包和相关内容的信息。” SPDX 的核心重点一直是开源许可合规性。 2021 年 SPDX 规范成为了一个国际的开放标准： ISO/IEC 5962：2021	贡献主要由 OpenChain 日本工作组的行业参与者领导，包括日立、富士通、索尼、瑞萨和东芝。 SPDX Lite 包含在 SPDX 2.2 版本中	SWID 标签旨在通过在整个软件生命周期中更轻松地发现、识别和关联软件，帮助企业创建准确的软件清单。该标准是更广泛的 ISO IT 资产管理标准的一部分，并得到 NIST 的支持。 SWID 的第一个版本于 2009 年发布，然后在 2015 年进行了修订。
维护者（Working Group）	核心团队由来自 OWASP、Sonatype 和 ServiceNow 的人领导	由 Linux 基金会维护	由 Linux 基金会维护	由 NIST 维护
支持格式（Format）	XML，JSON，Protocol BUffers	RDFa，xlsx，spdx，xml，json，yaml	RDFa，xlsx，spdx，xml，json，yaml	xml
BOM 元数据（Metadata）	供应商，制作商，组件信息，证书信息，创建 BOM 的工具信息，外部 API 信息，依赖关系信息（依赖关系图）	SPDX 文档创建信息，组件信息，文件信息（可能包含在包信息里），文件片段信息，证书信息，SPDX 元素之间的关系，注释信息（例如：审查 SPDX 文件的信息）	文档创建信息：SPDX 版本、数据许可、SPDX 标识符、文档名称、SPDX 文档命名空间、创建者 组件信息：包名、包版本、包文件名、包下载位置、包主页、结束许可、声明许可、许可注释和版权文本	语料库标签：描述预安装阶段的软件（TAR、ZIP 文件、可执行文件） 主要标签：提供产品名称、标签的全球唯一标识符以及标识标签创建者的基本信息 补丁标签：标识并描述应用于产品的补丁 补充标签：增加主要或补丁标签的附加细节

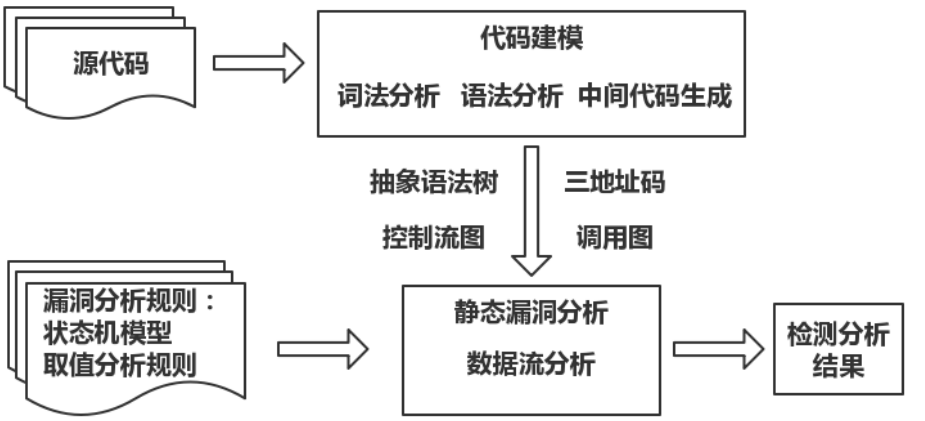
	CycloneDX	SPDX	SPDX Lite	SWID
组件唯一标识支持	软件坐标 Coordinates (group， name，version) PURL（Package URL） CPE （Common Platform Enumeration） SWID（ISO/IEC 19770-2：2015） Cryptographic hash functions (SHA-1，SHA-2， SHA-3，BLAKE2b， BLAKE3)	PURL（Package URL） Cryptographic hash functions SPDXID	PURL（Package URL） Cryptographic hash functions SPDXID	CPE Cryptographic hash functions SWID

近年来，由开源组件和第三方商业软件引发的供应链安全问题层出不穷，拿最近的 Log4j 和 SolarWinds 入侵事件来举例，因其应用广泛，可利用性强，造成极大影响。SCA 技术不需要运行程序，通过静态的方式分析第三方软件的组成，得到应用程序的组件知识图谱，能够有效提升软件供应链的透明度，进而关联出存在的已知漏洞清单，帮助用户及时修复问题。

4.2.2 静态安全分析 (SAST)

软件供应链从软件生命周期角度可划分为开发、交付、使用三大环节，每个环节都可能会引入供应链安全风险，上游环节的安全问题会传递到下游环节。开发环节作为软件供应链的上游环节，从该环节入手，及早发现和修复安全问题非常必要。早在 2005 年，美国总统信息技术咨询委员会关于信息安全的年度报告中就曾指出：美国重要部门使用的软件产品必须加强安全检测，尤其是应该进行软件代码层面的安全检测。而在美国国土安全部（DHS）和美国国家安全局（NSA）的共同资助下，MITRE 公司展开了对软件源代码缺陷的研究工作，并建立了软件源代码缺陷分类库 CWE（Common Weakness Enumeration），以统一分类和标识软件源代码缺陷。美国 CERT、SANS、OWASP 等第三方研究机构也在软件源代码安全检测领域开展了许多工作，包括：CERT 发布了一系列安全编程（C/C++、Java 等）标准，SANS 和 OWASP 发布了严重代码缺陷 TOP25 和 TOP10，用于指导开发人员进行安全的编码，尽量避免源代码中的安全缺陷。

静态安全分析正是这样一种针对开发过程中源代码进行安全检测的技术，内置多种缺陷检测规则，将源代码转换为易于扫描的中间数据格式，使用缺陷检测技术对其进行分析，匹配缺陷规则，从而发现源代码中存在的缺陷，并提供修复建议，帮助用户及早修复，从而降低后期缺陷修补的成本，增强软件的安全性。静态安全分析原理如下图所示：



其中常见的静态漏洞分析技术有以下几种：

（1）语法分析技术

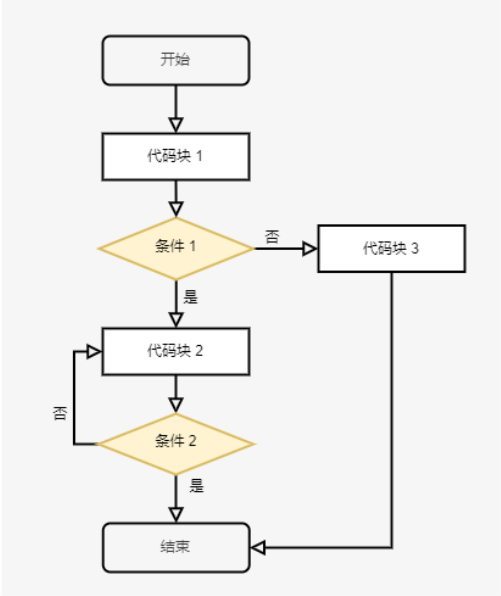
语法分析指按具体编程语言的语法规则处理词法，分析程序产生的结果并生成语法分析树的过程。这个过程可以判断程序在结构上是否与预先定义的 BNF 范式相一致，即程序中是否存在语法错误。程序的 BNF 范式一般由上下文无关文法描述。支持语法分析的主要技术包括算符优先分析法（自底向上）、递归下降分析法（自顶向下）和 LR 分析法（自左至右、自底向上）等。语法分析是编译过程中的重要步骤，也是其他分析的基础。

（2）类型分析技术

类型分析主要指类型检查。类型检查的目的是分析程序中是否存在类型错误。类型错误通常指违反类型约束的操作，如让两个字符串相乘、数组的越界访问等。类型检查通常是静态进行的，但也可以动态进行。编译时进行的类型检查是静态检查。对于一种编程语言，如果它的所有表达式的类型可以通过静态分析确定下来，进而消除类型错误，那么这个语言是静态类型语言（也是强类型语言）。利用静态类型语言开发出的程序可以在运行程序之前消除许多错误，因此程序质量的保障相对容易（但表达的灵活性相对弱）。

（3）控制流分析技术

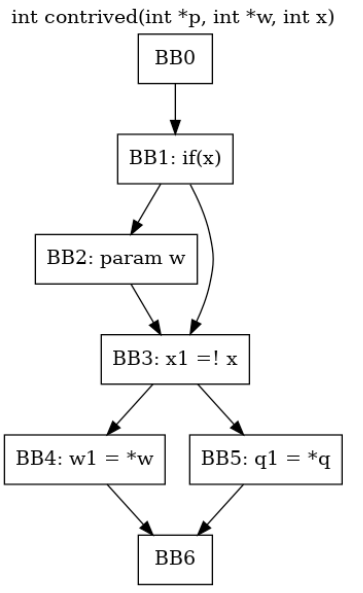
控制流分析的输出是控制流图，通过控制流图可以得到关于程序结构的一些描述，包括条件、循环等信息。控制流图是一个有向图，如下图所示，图中的每个节点对应一个基本块，而边通常对应分支方向。



（4）数据流分析技术

数据流分析用于获取有关数据如何在程序的执行路径上流动的信息。在程序的控制流图上，计算出每个节点前、后的数据流信息，通过数据流分析可以生成数据流图。

```
int contrived(int *p, int *w, int x) {
    int *q;
    if (x) {
        kfree(w); // w free
        q = p;
    }
    [...]
    if (!x)
        return *w;
    return *q;  // p use after free
}
```



数据流分析代码示例图如上所示，如参数 w 是存在漏洞的数据，则数据流路径 BB0->BB1->BB2->BB3->BB4->BB6 是不安全的。

数据流分析广泛应用于静态分析中，可对变量状态（如未使用变量、死代码等）进行分析。同时，污染传播分析也是数据流分析技术的一种应用，在漏洞分析中，使用污点分析技术将所感兴趣的数据（通常来自程序的外部输入）标记为污点数据，然后通过跟踪和污点数据相关的信息的流向，可以知道它们是否会影响某些关键的程序操作，进而挖掘程序漏洞。

缺陷类别一般分为以下三种：

（1）输入验证类 [26]

输入验证类缺陷指程序没有对输入数据进行有效验证所导致的缺陷。常见的输入验证类缺陷包括 SQL 注入、XML 外部实体注入、命令注入、XSS（跨站脚本）等。

针对输入验证类缺陷，需要对输入进行验证，验证的内容包括数据是否包含超出预期的字符、数据范围、数据长度、数据类型等，若包含危险字符，如＜、＞、"、'、%、（、）、&、+、\、\'、\"、. 等，还需对危险字符进行转义。也可以通过验证行为净化所有不可信的输出，如针对解释器的查询（SQL、XML 和 LDAP 查询）和操作系统命令，从而有效减少部分安全问题。

（2）资源管理类

资源管理类缺陷指因程序对内存、文件、流、密码等资源的管理或使用不当而导致的缺陷。常见的资源管理类缺陷包括缓冲区上溢 / 下溢、资源未释放、内存泄漏、硬编码密码等。

对于此类缺陷，需要内存分配时，检查缓存大小，确保不会出现超出分配空间大小的危险。在内存、文件、流等资源使用完毕后应正确释放资源。

（3）代码质量类

代码质量类缺陷指由于代码编写不当所导致的缺陷，低劣的代码质量会导致不可预测行为的发生。常见的代码质量类缺陷包括整数问题、空指针解引用、初始化问题、不当的循环终止等。

对于代码质量类缺陷，需要针对性地进行解决，如使用整数时，要避免操作结果超出整数的取值范围；使用指针时，要判断其是否为空，养成良好的编程习惯。

[26] 奇安信代码安全实验室 软件供应链安全：源代码缺陷实例剖析 电子工业出版社 2021 年 8 月

静态安全分析技术不需要运行程序，能够覆盖 100% 的代码库，但检查结果可能存在漏洞或误报的情况，一般需要不断地优化检测技术和检测规则，以降低误报和漏报。

4.2.3 动态应用安全测试 (DAST)

在实际攻击场景中，通常无法获得源代码，基于白盒的模式来分析软件中存在的漏洞。

黑客大多针对运行中的系统或业务进行黑盒扫描，寻找可能存在的薄弱点进行攻击。动态应用安全测试（DAST）模拟黑客的攻击行为，通过一种由外向内的检测技术，对运行中的系统和业务进行检测，发现可能存在的漏洞，便于及时修复。

4.2.3.1 系统漏洞扫描

对给定主机进行扫描，收集目标主机信息，如操作系统类型、主机名、开放的端口、端口上的服务、运行的进程、路由表信息、开放的共享信息、MAC 地址等。然后对主机系统或服务进行漏洞检测。

系统漏洞扫描流程可以概括为四大部分：

图 4.7 系统漏洞扫描工作流程

（1）主机发现

在进行系统漏洞扫描时，首先会探测目标系统是否存活——开机并正常联网。在默认配置下，对没有正常存活的系统，会自动跳过对目标系统的扫描，转而启动对下一个目标的扫描。主机存活性探测技术包括 ICMP PING、UDP PING 和 TCP PING 三种。

（2）端口发现

对已经存活的主机，还要探测主机上开启了什么端口。对于 1-1024 端口，可以知道占用该端口的默认服务是什么。对于端口的探测，主要采用 TCP 连接的方式进行探测。包括：完整的 TCP 连接、TCP 半连接（TCP SYN）。

（3）系统和服务识别

在进行操作系统版本识别时，会根据各个操作系统在 TCP/IP 协议栈实现上的不同特点，采用黑盒测试方法，通过研究其对各种探测的响应形成识别指纹，进而识别目标主机运行的操作系统。其使用的技术主要包括：

被动识别：通过抓包，对数据包的不同特征（TCP Window-size、IP TTL、IP TOS、DF 位等参数）进行分析，来识别操作系统。依赖于网络拓扑

主动识别：通过发特殊构造的包，分析目标主机的应答方式来识别操作系统。例如：如果送了一个 FIN/PSH/URG 报到一个关闭的 TCP 端口。大多数操作系统会设置 ACK 为接收报文的初始序列数，而 Windows 会设置 ACK 为接收报文的初始序列数加 1. 基于这一特点，在发现操作系统反馈 ACK 为初始序列数加 1 时，可以识别目标操作系统为 Windows 操作系统。

对应用服务版本进行识别主要依靠应用系统反馈的系统 Banner 信息。

一般情况下，对于 1-1024 端口，都对应着默认的服务，如邮件服务器对应 25 端口，Web 服务器对应 80 端口，域名服务器对应 53 端口。但是不能保证所有的端口都对应着开发默认的服务器。例如：对外网提供服务的服务器，需要远程管理，开放了 SSH 服务，但是为了防止 SSH 服务直接暴露在外网，管理员 SSH 的连接端口挂在了 9821 上。按照正常来看，这是个非标准服务端口。此时可以利用非标准服务识别技术对此端口提供的服务进行精确的探测和识别。非标准服务识别技术对端口使用标准服务进行探测，根据服务端返回的信息，匹配服务识别指纹库，判定端口对应的服务。

（4）漏洞检测

完成主机存活发现、端口发现、系统和服务识别后，会根据识别的系统与服务信息，对目标系统进行口令猜测，口令猜测成功后将启动本地扫描，通过和漏洞库对比分析，发现目标主机是否存在漏洞。或者通过远程扫描的方式，直接与目标系统或服务进行网络通讯，发送特定的数据包，接收目标系统的反馈信息，根据反馈信息判断系统是否存在特定漏洞。系统漏洞扫描因为本地扫描需要登录目标主机，应用技术难度较大，所以一般会采用远程扫描的方式。远程扫描技术按照扫描探测原理不同，可以分为版本扫描、原理扫描、模糊扫描三类，区别如下：版本扫描是远程漏洞扫描技术的一种，漏洞扫描系统通过 Banner 等信息识别系统或服务的版本信息，和漏洞库信息对比分析，判断目标是否存在漏洞。版本判断扫描不会对目标造成影响，安全性高。但是版本识别过程有可能误报，造成漏洞误报。目前大多数产品的远程扫描采用此技术。

原理扫描是远程漏洞扫描技术的一种，漏洞扫描系统通过构造特殊包，发送到目标主机，收集目标主机反馈的信息，判断系统是否安装特定的补丁程序，进而判断系统漏洞是否存在。原理扫描误报率极低，但是开发扫描插件有难点，大多数产品没有能力采用此技术。

模糊扫描介于版本扫描和原理扫描之间，通过构造特殊包，识别目标主机的特征，分析漏洞是否存在，准确性较高。但模糊扫描插件数量不多且存在误报与漏报的情况。目前仅有少量产品有能力采用此技术。

4.2.3.2 Web 漏洞扫描

对运行中的 Web 程序进行漏洞扫描，从大的方面可以分为页面爬取、探测点发现和漏洞检测三个阶段。第一个阶段由爬虫完成，后两个阶段依赖于第一个阶段的结果。页面爬取和漏洞检测之间可以同时进行，也可以等爬虫将站点爬完之后，再统一交给漏洞检测引擎处理。

（1）页面爬取

页面爬取重点在于快而全地获取整个站点的站点树。这个过程分为两步，网络访问和链接抽取。网络访问需要支持设置 cookie，自定义请求头，设置代理（http，https，sock4，sock5），支持各种认证方式（basic，ntml，digest），客户端证书等。拿到响应之后，需要自动识别响应的编码方式，并将其转换为统一的 UTF-8 编码，供后续抽取链接等操作使用。目前支持从 HTML，HTML 注释， Flash，WSDL 等静态内容中抽取链接之外，还用 webkit 实现了从 DOM 树，JS，Ajax 等重抽取静态和动态的链接。

除了使用前文提到的各种爬取设置和智能技术之外，还需要对站点做存活性判断、主动识别页面类型（图片，外部链接，二进制文件，其它纯静态文件等）、尝试猜测一些无法从其他页面解析出来的但可能存在的目录并做好标记。存活性判断主要是为了迅速给出站点是否可达（可能跟用户的输入，配置的代理、认证信息，站点本身都有关系）的一个结论，避免做一些无用功；页面类型主要为了帮助插件区分哪些页面可能存在漏洞需要被扫，哪些页面可以直接跳过；根据一定的字典猜测可能存在的链接，一方面是为了尽可能多地发现页面，另一方面是为了方便插件直接根据猜测的标记报告敏感文件的漏洞。

通过爬取的时候获取并标记尽可能多的信息，可以极大地减少逻辑冗余，提高扫描引擎的性能。

（2）探测点发现

不同的插件有针对性地在请求中寻找不同的探测点，可能的探测点有 URL 路径，GET 方

法URL中的参数，POST方法请求体中的参数，请求头中的字段，cookie中的键值，响应体等等。一般而言，插件会尝试对待扫描的 URL 进行解析，分解出各种可能存在漏洞的探测点，供后续进行相关的漏洞检测。

（3）漏洞检测

每个具体的漏洞都有相应的一个插件来进行具体的检测。插件根据得到的探测点，有针对性地构造特殊的网络请求，使用远程网站漏洞扫描检测技术进行漏洞检测，判断是否存在相应的漏洞。除了使用到的漏洞检测技术之外，为了缓解网络访问带来的性能问题，在需要发送多种探测请求的插件中，将网络请求并发而将网络响应的处理串行起来提高扫描速度；为了避免在短时间内发送重复的网络请求（某些插件不需要重新构造请求体，使用的是和爬虫一样的网络请求），使用了页面缓存技术，旨在降低网络访问对扫描速度的影响；引擎在扫描的过程中，能够根据系统当时的负载，自动调节处理 URL 的并发进程数（不超过任务配置的进程数的前提下），从而获得一个最佳的系统吞吐量。

对于漏洞检测，分为两大类的漏洞进行检测：

1. 针对 URL 的漏洞扫描：

例如 XSS：对将要扫描的 URL 进行拆分，然后针对每个参数进行检测，首先会在原有参数值后面添加一个正常的字符串，从响应页面内容中查找输入的字符串是否存在，并且分析上下文，根据分析的结果，再次重新输入特定的字符串，继续通过分析上下文，判断所输入的特定的字符串是否能被执行，如果不行或者是输入的某些字符串被过滤，则会重新输入其他的特定字符串进行验证。

2. 针对开源 CMS 的特定漏洞扫描

例如 Wordpress：在爬虫爬取的时候，会通过网站的一些特征进行识别，如果识别出当前被扫描站点使用了 wordpress，则会调用 wordpress 相关的所有漏洞检测插件，通过这些检测插件，发现存在于 wordpress 的特定漏洞。

DAST 技术不需要了解或查看软件的内部架构和源代码，可用于验证交付软件的质量，或在运营阶段定期扫描，便于及时发现并修补漏洞。DAST 技术对发现安全问题很有帮助，但也存在一些不足之处需要注意，一是 DAST 依靠安全专家来创建正确的测试程序，很难为每个应用程序创建全面的测试。DAST 工具也可能会创建假阳性测试结果，将应用程序的有效元素识别为威胁。二是 DAST 关注的是请求和响应，不能像 SAST 技术一样，准确识别出风

险代码位置。另外 DAST 通常检测速度较慢，需要几天或几周的时间才能完成测试，而且由于它发生在 SDLC 的后期，发现的问题会给开发团队带来很多任务，增加了相应的成本。

4.2.4 交互式应用安全测试 (IAST)

IAST 交互式应用安全测试技术是最近几年比较火热的应用安全测试新技术，被 Gartner 咨询公司列为网络安全领域的 Top 10 技术之一。IAST 工作在程序运行时，从应用内部持续监控和收集应用程序运行时流量或代码，并传递给安全分析引擎，识别出代码执行中的漏洞特征。针对收集到的数据流量和代码，可以分别执行类似 DAST 和 SAST 的检测，漏洞检出率极高、误报率极低，同时可以定位到 API 接口和代码片段。

常用有效的 IAST 技术是通过插桩来实现的，通过在代码运行的中间件上插入探针，通过探针来识别判断安全风险，直接从运行中的代码发现问题，以实现自动化识别和诊断在应用和 API 中的软件漏洞。

因为部署在服务的中间件里面，IAST 探针可以从中获取很多的信息，从某种角度来说，它可以说是 SAST 和 DAST 的结合体，包括但不限于代码、流量的分析，主要有以下几个方面:

（1）代码

IAST 能访问所有和应用一起部署的源代码和二进制代码，代码探头对应用中每一行代码做二进制的静态分析，包括库和框架；

（2）HTTP 流量 [27]

HTTP 流量采集，指采集应用程序测试过程中的 HTTP/HTTPS 请求流量，采集可以通过代理层或者服务端 Agent。采集到的流量是测试人员提交的带有授权信息的有效数据，能够最大程度避免传统扫描中因为测试目标权限问题、多步骤问题导致的扫描无效；同时，流量采集可以省去爬虫功能，避免测试目标爬虫无法爬取到导致的扫描漏报问题；

（3）库和框架

IAST 能看到部署的每一个库和框架，分析应用和 API 如何使用它们。不仅 IAST 能够根据已知漏洞（CVE）来评估库，也能识别部分或整体隐藏在库里面的未知漏洞。重要的是，因为 IAST 能精确知道库里面的哪一部分被应用真正调用，它能够过滤掉从未被调用的和库相关的漏洞；

[27] 软件供应链安全及防护工具研究

（4）应用程序状态

IAST 能够检查程序执行时的应用状态。例如，IAST 能看到调用安全方法时使用的参数来发现弱点，如传递“DES”参数给加密密码构造函数；

（5）数据流

从开始进入应用的时候开始，IAST 就追踪不信任的输入数据。这是识别最重要的漏洞种类—注入类漏洞的关键。这个技术，有时称之为“污点追踪”，跟踪真实数据在应用中的流动，非常精确；

（6）控制流

IAST 了解应用的控制流，能够识别出应用行为中漏洞的特征。例如如果一个应用要求在每个 web 服务中，采用 service（）方法检查访问控制，这个特征将很容易被 IAST 发现；

（7）后端连接

IAST 能够检查围绕着后端连接的所有细节，如数据库、操作系统调用、目录、队列、套接字、电子邮件和其他系统。IAST 使用这个信息识别出架构缺陷和安全缺陷；

（8）配置

IAST 能访问应用、框架、应用服务器、和平台的配置，保证正确的安全配置。IAST 甚至能查询应用组件的运行时配置，如 XML 解析器。注意某些平台，如 .NET，重度依赖配置来实现安全；

基于 IAST 技术的特点，可以在以下场景使用：

（1）开发测试阶段及早发现漏洞

基于交互式应用安全测试的特点，可以在软件开发测试过程中，使用该技术及时发现源代码或 API 中存在的漏洞，在开发过程早期就进行修复，其检测速度、精确度、流程上都比传统的静态安全分析技术和动态应用安全测试技术有优势。

（2）运营阶段持续检测和阻止漏洞利用

由于其持续检测的特性，可以识别出之前未发现的漏洞，如 0 Day 漏洞、新发布的漏洞等；基于其在应用程序内部插桩的实现方式，当检测到威胁时，可以自定义应急响应策略，如终止用户会话、关闭应用程序、警告安全人员等方式，来阻止攻击。

当前某些交互式应用安全测试产品还具有软件组成分析的能力。基于其在应用程序内部插桩的实现方式，采集应用程序运行过程中动态加载的第三方组件及依赖，自动化构建软件物料清单（SBOM），传递给组件漏洞分析引擎，检测依赖组件存在的安全风险，从而避免供应链上游软件被不安全的底层依赖所污染。

4.2.5 模糊测试 (FUZZ)

对于软件产品而言，健壮性是一个很重要的指标。尤其是在一些关键基础设施领域，如医疗保健、电力、通信、金融等，软件产品的故障可能导致灾难性的事件或大规模的数据泄露，甚至造成人身伤害或环境的破坏。故障本身的不可预见性，也很可能导致一些故障如系统信息被泄露，从而被攻击者进一步利用。模糊测试是一种评估软件健壮性和安全性的强大技术，其核心原理是将非预期的数据输入目标系统，查看目标系统是否发生故障，如崩溃、无限循环、资源泄漏或短缺、意外行为等，并提供故障原因及修复建议，以便用户进行修复。

模糊测试一般包括测试用例生成、测试用例输入、测试结果验证三个阶段 [28]。

（1）测试用例生成

每一个非预期的数据都是一个测试用例，理论上非预期数据是无限的，测试用例生成的目标是如何设计出最有可能在目标系统中触发故障的测试用例，从本质上讲，测试用例应该接近目标的期望输入，测试用例的质量对模糊测试的有效性至关重要。常用的测试用例生成技术包括随机生成和基于模板生成两种：

- 随机生成

最简单也是效率最低的生成技术。简单地使用随机数据作为测试用例。随机模糊测试通常是无效的，因为测试用例与有效输入完全不同。目标系统检查并迅速拒绝测试用例。在大多数情况下，测试用例无法渗透到目标代码中。

- 基于模板生成

以有效数据作为模板，进行变异以创建测试用例。一般来说，模板测试用例比随机测试用例更有效，因为它们大多是正确的。目标软件系统将处理测试用例，可以用来检验目标系统处理非预期输入的能力。

[28] what is fuzz

（2）测试用例输入

根据目标系统或测试类型的不同，可以分为针对网络协议、文件、API、用户界面的模糊测试等，不同的模糊测试类型，测试用例输入方式不同。如针对网络协议的，需要和网络服务端或客户端建立端对端的连接，将测试用例传递给接收者；针对用户界面的，需要模拟鼠标或键盘点击用户界面的操作等。大多数现代操作系统和编程环境都提供了一种编程方式来传递输入，以便通过软件自动化地传递测试数据。

（3）测试结果验证

失败类型一般包括崩溃、无限循环、资源泄漏或短缺、意外行为几种，通常采用人工观察或模糊测试工具自动判定两种方式。

模糊测试可用于在软件供应链的开发和交付阶段，检测未知的漏洞。在开发阶段，可以对源代码进行模糊测试，在早期快速地定位和解决漏洞；在软件交付运行阶段，可用来验证软件的健壮性和可靠性。

近年来，由于开源组件、软件代码恶意注入引起的供应链安全事件层出不穷，供应链安全检测技术以软件开发生命周期为主线，致力于解决各阶段存在的安全问题。

4.3 软件供应链数据安全技术

Garnter 公司在 2021 年供应链安全风险报告 [29] 中提到，机密或敏感信息的泄露是导致软件供应链风险的另外一个主要问题。黑客通过窃取源代码、构建日志、基础设施等中存在的硬编码凭证，如 API 密钥、加密密钥、令牌、密码等，或者通过泄露的 SBOM（软件物料清单）、源代码，寻找其中存在的漏洞，对目标系统发起攻击，极大增加了安全风险。因此，这些敏感信息在应用和管理过程中进行数据安全保护，针对不同场景可采用安全多方计算（MPC）和零知识证明（ZKP）和可信执行环境（TEE）等安全技术。

软件供应链全生命周期内离不开上下游企业的数据协作，如 SBOM（软件物料清单）的生成及维护阶段需要软件开发企业提供数据支持（软件成分清单及源代码）、软件供应链安全检测阶段需要第三方安全企业 / 组织提供最新的漏洞数据支持等。由于此类数据属于各上下游企业核心资产，并事关最终用户软件安全，一旦泄露将极大增加最终用户遭受黑客攻击的风险，具有一定的敏感性。因此，上下游企业之间很难直接对接这些敏感数据。此时，需要相应的数据安全技术，以便管理和保障这些敏感数据的对接和使用。针对不同场景，可采

[29] How Software Engineering Leaders Can Mitigate Software Supply Chain Security Risks

用安全多方计算（MPC）、零知识证明（ZKP）或可信执行环境（TEE）等数据安全技术予以解决。

(1) 安全多方计算

安全多方计算（Secure Multi-party Computation，MPC），是一种基于密码学技术实现的无需可信第三方的分布式计算协议与机制。具体地说，即在一个分布式的环境中，各参与方在互不信任的情况下通过密码学协议进行协同计算，输出计算结果，并保证任何一方均无法得到除应得的计算结果之外的其他任何信息（包括输入和计算过程的状态等信息）。MPC 模型如图 5-1 所示，它解决了不信任环境下多个参与方联合计算一个函数的问题。

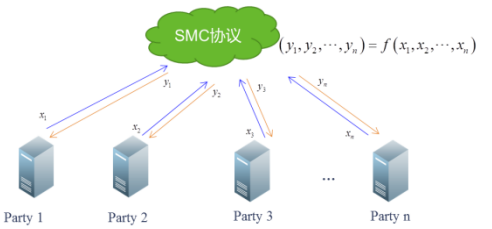


图 4.12　安全多方计算模型

实现多方安全计算协议主要有基于混淆电路（Garbled Circuit，GC）、秘密分享（Secret Sharing，SS）和同态加密（Homomorphic Encryption，HE）等密码学的实现方式。根据支持的计算任务 MPC 可分为通用 MPC 和专用 MPC：前者理论上可支持任何计算任务，具有完备性；而后者特定计算任务需设计特定的 MPC 协议，如安全比较协议支持参与方无法获知对方准确输入情况最后得到比较结果，隐私求交协议（Private Set Intersection，PSI）支持参与双方仅能获取双方数据集的交集信息，而其他信息无法获取，隐私信息检索协议（Private information retrieval，PIR）支持检索方匿名匹配获取被检索方相关信息，而被检索方无法得知检索方的查找条件和返回结果等。在实际应用中，由于通用 MPC 协议实现计算复杂度高，而专用 MPC 协议效率较快，因此通常采用专用 MPC 协议实现。

基于 MPC 的隐私保护特性，它可以应用于软件供应链漏洞信息查询场景中。在传统的漏洞查询场景，查询方向漏洞管理方提交软件名称和版本号，漏洞管理方将其与漏洞库进行匹配和关联，从而返回漏洞信息；在软件供应链场景中，客户（查询方）希望了解自身的软件产品以及相关的开源组件、引用库等中是否包含漏洞信息，客户不希望向漏洞管理方暴露这些敏感信息，即可以看成 SBOM 信息，但希望自身能获取该产品是否包含漏洞，以及包含哪些漏洞等信息。基于 PIR 协议，在客户端可对软件名称进行加密，将此加密软件名称提交给

服务端，它将其先对漏洞库进行加密处理，然后进行匿名匹配与检索，另外通过安全比较协议，类似的原理可对软件版本号进行漏洞库进行比较，最终实现软件供应链产品的漏洞查询以及数据的隐私保护。

(2) 零知识证明

零知识证明 (Zero-Knowledge Proof， ZKP)，也是一种基于密码学的安全技术，可应用在身份认证、数据验证场景中。它指的是证明者（Prover， P）能够在不向验证者（Verifier，V）提供任何有效信息情形下，使得验证者相信他们拥有某一个信息 X，但在此过程中没有泄露任何关于 X 的其他信息。实质上，ZKP 是一种涉及两方或多方的安全协议，即两方或多方完成一项任务所需采取的一系列步骤。

根据零知识证明的定义，可归纳出 ZKP 有以下性质 [30]：(1) 正确性。P 无法欺骗 V。换言之，若 P 不知道一个定理的证明方法，则 P 使 V 相信他会证明定理的概率很低。(2) 完备性。V 无法欺骗 P。若 P 知道一个定理的证明方法，则 P 使 V 以绝对优势的概率相信他能证明。(3) 零知识性。V 无法获取其他任何额外的知识。

零知识证明可以在区块链与数字货币系统实现身份认证或交易验证过程中，确保交易双方的身份匿名化和隐私保护。同理，结合 ZKP 技术，可确保 SBOM、漏洞数据等敏感信息在使用过程的数据安全。例如，在软件供应链安全监管场景中，监管方希望查看安装的软件的 SBOM 数据是否包含通报修复的软件版本。那么，通过结合零知识证明技术，监管方既得到正确的结果（是或者否），同时也完全保护了软件安装厂商的敏感信息。

(3) 可信执行环境

可信执行环境（Trusted Execution Environment，TEE）是基于硬件隔离机制构建的一个安全隔离区域，可保证在安全区域内部加载的代码和数据在机密性和完整性方面得到保护。它将系统的硬件和软件资源划分为两个执行环境，分别是可信执行环境和普通执行环境（Rich Execution Environment， REE），两者是安全隔离的，分别有独立的计算和存储空间。REE 环境的应用程序无法访问 TEE，即使在 TEE 内部，各个应用程序的运行也是相互独立的，不能未经授权的情况下相互访问。当外部的应用程序想要访问 TEE 时，需要对该应用程序或用户身份进行验证，只有通过验证的应用程序才能进入，从而为 TEE 内部运行的代码和数据提供了机密性和完整性保护。

[30] 曹天杰，张永平，汪楚娇．安全协议：北京邮电大学出版社，2009 年 08 月

当前业界主流的 TEE 技术路线包括 Intel SGX、AMD SEV 和 ARM TrustZone，以及国产芯片鲲鹏处理器。其中，Intel SGX 适用于 PC 机和服务器，它仅支持最大构造 128M 的安全区域；而 AMD SEV 仅适用于服务器，但它可以覆盖整个虚拟机的内存，即安全区域足够大；ARM TrustZone 多用于适用于移动设备中。

与基于密码学的 MPC 和 ZKP 相比，TEE 的安全性来源于硬件隔离的安全能力，避免基于密码算法及协议大量复杂计算和交互过程，从而避免了额外的计算开销和通信开销。因此，在需要保护大量数据或性能要求高的计算场景中，可使用 TEE 技术进行安全防护。比如在软件供应链的源代码和数据保护中，为了防止运行过程中被篡改和非法访问，可将源代码和数据迁移至 TEE 中，实现安全等级高的数据安全防护。

5

软件供应链安全解决方案

5.1 供应链安全监督

在企业业务发展及能力生态的建设中，由于涉及到众多的供应链主体，为避免供应链产品的恶意污染以及供应链攻击所带来的损失，开放可信技术供应商标准（Open Trusted Technology Provider ™ Standard ，O-TTPS）明确定义了 COTS 和 ICT 产品的完整性和全球供应链安全性要求，缓解整个供应链产品生命周期的威胁，通过应用良好定义、可监控和有效验证的供应链流程及最佳技术实践来管理其供应链。依照风险治理思路，在供应链安全治理方面，需要做到如下四个方面：

1、外防输入，做好软件供应商及供应链产品的专项安全检查

严进宽用，按照当前引入时间为起始点，摒除带有漏洞的软件版本，严格控制外部引入风险，认定后可认为引入的软件是安全的，可供内部使用，直至曝出漏洞，对软件进行标记风险状态，进入下一个循环。

2、内控扩散，将安全管控融入软件 SBOM 管理

结合供应链管理建立灰白黑名单机制，防范带有漏洞的软件版本引入企业；针对曝出漏洞但未完成治理的软件或系统进行标记，严格安全管控。

3、存量治理，结合企业软件 SBOM 理清依赖，有序治理

因存量系统较多，按照风险有限的原则，统筹考虑系统间的依赖关系，制定基础平台优先治理、互联网应用重点治理等差异化的治理策略，分批次有序开展治理。

4、持续监测，建立风险预警机制，保障供应链安全可控。

漏洞会长期存在，需要持续监测，做好应急流程，有序治理软件供应链相关安全风险。

供应链风险贯穿了产品的整个生命周期，结合近年来所发生的供应链攻击和入侵事件，我们需要从监管层面加强供应链产品安全认证管理，提供软件 SBOM 托管和可信认证服务，企业也需要完善供应链资产管理和安全检查，可借助 SBOM 知识图谱理清企业供应链依赖关系，从而在监测到预警时能够从容应对。

5.1.1 软件安全审查备案

现阶段在软件安全审查备案层面主要以合规检测为主，结合各领域各行业自身特色进行针对性的安全检查，推出适合自身领域的行业规范以引导业内厂商安全意识，提升风险防护能力。

对于软件供应链安全而言，审查备案的主要目的是通过对软件供应商等处在软件供应链头部组织机构进行相关资质认定保证源头的合规，进而保证软件供应链下游使用方能够减少安全检测压力，分散管控风险，软件供应链各个节点也可在安全审查备案信息的基础上灵活制定自身安全防御策略。

5.1.1.1 供应商领域资质

供应商主要对自身研发、管理、安全保障等能力进行备案审查，向采购方证明自身基础安全资质。采购方也可基于国家颁布现有规范标准对供应商进行基本资质提出要求，在双方都重视安全资质的大风气下，行业安全意识与能力将会得到协同提升。

结合目前已经对供应商资质提出明确要求的政策文件，外加其他与企业安全能力挂钩的国家认证评测体系，经过整理后生成如下列表：

企业资质：

- 系统等级保护评测——等级保护（1-5 级）；
- 信息系统保障评测——备案证明 + 保评测证书等
- 信息安全人员评测——CWASP CSSD/CSSP、CISSP、CISP 等评测标准；
- 工程服务能力评测——ISO/IEC 21827 SSE-CMM（1-5 级）；
- 信息产品安全评测——ISO/ICE 15408（EAL1-7 级）；
- 领域认证：行业协会专业认证，专业研究机构认证，国家级认证等；
- 所获荣誉：各级别具有广泛公信力的奖项，科研评比奖项，专业领域评比奖项。
- 技术服务收入：利用企业的人力、物力和数据系统等为用户提供技术服务所获得的收入。
- 知识产权数量：在软件著作权、专利等知识产权方面的认证数量。

当前政策法规对供应商领域资质没有明确的硬性要求，但从上述列表不难发现其中涉及到的资质内容是依托于国家现有的标准认证体系结合行业具备公信力的评测体系所产生的交集。

验证供应商资质的目的是从早期合规层面对供应商进行能力甄别，软件供应链涉及到的领域多样，技术标准复杂，行业发展迅速且发展日新月异，单纯的依靠国家层面的评测难以

适应快速变化的市场环境和技术发展速度。在当前技术条件与行业发展背景下，首先对对现有行业专业标准加以强调，形成产业导向，在未来技术发展与典型安全事件发生后，对现有安全体系进行升级，并针对性的设立对应的供应商资质认证，为未来的规范化打下基础。

5.1.1.2 组件情报

组件情报是软件的“成分表”，了解其具体成分能够充分分析其安全性并在漏洞产生时做出针对性的应对。组件情报审查针对的是供应商所提供的软件产品进行审查并匹配安全情报的过程。由于软件产品形态多样，可能会以组件、程序、代码集等多种形式存在，且其中也会存在多种形态的引用与嵌套，复杂的内部结构给采购相关产品的用户所引入的安全风险也不容小觑。

组件审查需要多角度立体化的评估制品可能产生的风险与保障措施，组建成分审查利用专业代码审计工具对可能的潜在漏洞进行梳理，排除当前存在的安全风险，让用户知晓软件成分，并可以向监管机构提供软件物料清单方便管理与审查。对已完成封装的软件可进行等保 / 分保评测，满足国家法定安全标准即可，配合安全情报分析与保障内容对组件安全风险做到初步管控。

组件审查的主要目的有两个：一、 将安全风险和监管要求向供应商进行传导；二、通过在采购环节进行组件审查督促供应商做好安全管控并强化业务连续性。 供应商需要对自身产品负责，从出场满足基本功能上升到组件清单的透明化。当前 APT 攻击越发成为供应链攻击的主要形式，不存在绝对的安全，而组件审查与梳理的作用是让采购方知晓当前组件应用情况，配合专业的组件审查工具了解软件产品中的弱点，并针对其可能发生的安全问题制定针对性的防范策略。

组件审查完毕后结合生成组件清单配合安全情报库中的数据进行特征匹配，查看其版本是否存在漏洞与风险，一旦发现确定下一步安全策略，要求供应商进行漏洞修补、推送软件版本更新或其他形式的改进。

对组件保障运营团队也需要进行相对应的资质审查，保证团队的真实性、专业性与服务的时效性。确定保障服务内容能够被落实，防止出现安全责任盲区。

建立立体化的组件情报审查机制是落实安全责任的重要方式。当前情报审查工作绝大部分是采购方在招标过程中落实，或用户在梳理自身系统组件过程中利用专业设备或工具进行内部审计，并未建立起制度化专业化的标准。软件开发理念越发突出安全左移，对于组件情

报审查也需要进行相关的“左移”，提升至产品采购的前期，建立规范化制度化的审查机制，为未来可能出现的专项审查打好基础，企业也应开始对现有存量软件进行相关审查，对自身进行摸底，了解存在的安全隐患与风险，防止未来国家一旦落实相关审查标准与制度后，企业因为存在无法解释的依赖关系与复杂嵌套无法梳理的组件被剔除出采购名录的现象发生，此事直接关系到企业利益，需要提起足够的重视。

5.1.1.3 开发环境

现有政策规范如银保监会发出的《关于供应链安全风险提示的函》中提到，开发环境需要严格遵守软件安全开发生命周期管理制度和流程，加强漏洞监测、定位和修复，要重点管控开源软件的使用，建立开源软件资产台账，持续监测和降低所使用开源软件的安全风险。

开发环境安全可分为以下三个主要方面：开发工具安全、开源社区安全、应用开发平台安全。

其中开发工具安全涉及到具体的开发工具，如编程语言工具、编译工具、管理工具、代码安全审核 / 审计工具、安全测试工具等。需要重点关注其中的自主可控能力与工具内可能隐藏的威胁及脆弱性等相关信息。自主可控能力决定了软件开发的透明度与管控能力，是软件可控的核心指标。通过外部信息了解到相关研发工具的安全水平，维护团队规模、升级频率、过往安全背景等信息有助于从侧面了解平台的按群全运行管理水平及可靠性与脆弱性。

开源社区安全取决于代码托管平台的访问控制风险、公共仓库访问控制风险、源代码缺陷及后门和开源软件漏洞及恶意代码植入等风险问题。既有案例表明，美国对伊朗实施开源社区访问控制，导致开发人员无法登录代码托管平台，直接影响行业运转与百姓生产生活。开源软件漏洞业曾经让大量组件引用者面对安全风险如 log4j2 安全事件。并且越来越多的 APT 组织开始利用源代码缺陷后门为组件置入恶意代码实现加密勒索、挖矿等非法获利行为。此类风险绝大部分来自于开源社区内部，社区的管理水平与安全管理能力严重影响着成员们的安全。

5.1.1.4 政策展望

国家对开源社区的自主可控提出了建设方向与意见建议，我们现如今需要建立起自主可控的意识，提前做好安全备份与准备，加入行业机构主导的开源组织与社区支持国产化开源平台的建设。毕竟当平台都掌握在外部，无论本地安全规则如何完备，都会存在根本上的安全风险。

通过对安全资质，开发环境，软件成分等公开信息有了充分的掌握，产业链本身都能够保证其合规性与完整性。用户通过安全资质及产品信息清单，帮助其了解自身产品的安全合规情况，保证其判断系统是否满足国家基本安全防护要求，并在此基础上进行针对性的强化补充。采购方也可依据安全标准保证自身安全底线，并在此基础上做出更适合自己的安全保障措施。

在软件供应链安全概念的早期，我们需要先依靠对现有标准的组合复用实现安全体系建设，解决领域内安全体系的“有无”问题。未来要在不断地实践探索中深刻理解软件供应链安全特点，并针对性的完善安全审查机制，提升行业整体安全建设水平是未来确定的发展方向。

5.1.2 软件供应链的专项安全检查

软件供应链的风险是动态的。软件供应链安全专项安全检查目的是检查和验证监督的组织的风险在可接受范围内。通过设置软件供应链安全专项检查计划，监视风险管理活动，定期评审控制措施。

整体实施流程：

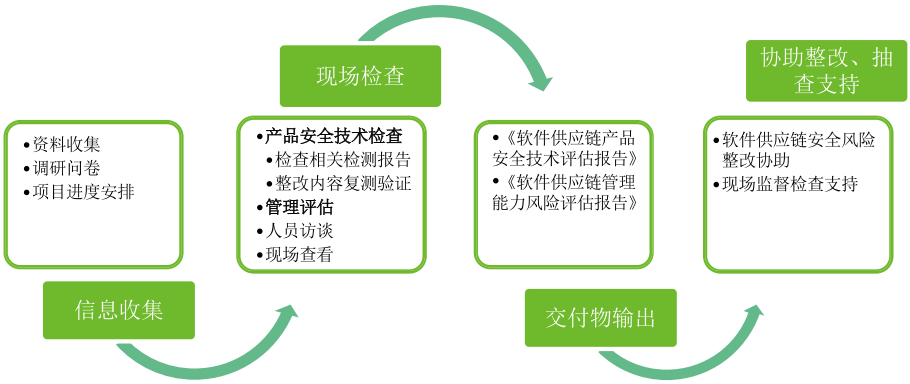


图 5.1 软件供应链专项安全检查实施流程

供应链安全专项检查主要从四个步骤开展进行：

（一）资料收集

针对识别、收集的关键软件供应链产品，与软件各供应商收集梳理此类产品已具有的销售许可证及近期相关检测报告（第三方安全检测报告、代码审计报告、漏洞扫描报告、渗透测试报告、等级测评报告、软件成分分析报告）。

（二）现场检查

- 产品安全技术检查

按照产品类型从开源组件、市场采购及定制开发分析产品的销售许可证、安全性进行检测，重点检查软件供应链产品是否有销售许可证，是否有相关检测报告（第三方安全检测报告、代码审计报告、漏洞扫描报告、渗透测试报告、等级测评报告、软件成分分析报告），检查是否存在已知但未修复的漏洞、开源代码、第三方组件安全隐患等。

- 管理评估

根据相关政策、标准及企业的供应链管理相关制度，制定软件供应链管理评估表，对软件供应链企业清单中的各供应商进行管理评估

检查方式：

1）人员访谈，通过与组织供应商管理人员及对应供应商进行交流、询问等活动，调研组织在设计、开发、测试、交付及生命周期管理等方面的信息安全管理状况；

2）现场查看：现场查看组织安全现状（如管理制度的制定和落实情况，技术措施的使用情况等），判断企业是否符合相关要求。

检查内容及结果：

主要内容主要包括：8 个模块、37 个细则。但不限于以下内容，可结合组织业务情况、侧重点进行增加。

关键信息基础设施供应链评估	组织管理
	流程与政策
	供应商管理
	人员管理
	产品漏洞管理
	源代码安全管理
	产品安全开发管理
	统一安全管理

（三）交付物输出

根据检查内容，识别软件供应链产品安全技术与组织安全管理的风险，提出安全问题提出风险处置建议，出具软件供应链产品安全技术评估报告、软件供应软件链理能力风险评估报告。

（四）协调整改、抽查支持

针对检查出存在已知但未修复的漏洞、开源代码、第三方组件安全隐患等，积极协助完成修复整改。配合与支持软件供应链专项安全检查后的相关抽查工作。

5.1.3 开源软件安全风险监测

在开源社区的驱动下，开源力量的不断发展，开源软件被广泛复用在实际工程项目中，其数量也在不断攀升，如 Maven、NPM、GitHub、Gitee 等开源社区或仓库都已到达千万以上级别；开源软件版本迭代相对频繁，其内部成分及关系也会发生复杂改变，如增加、更新依赖组件库或版本、复用其他组件源码、调用其他组件类库等。如今企业为了提高软件开发效率，大量引用开源组件实现业务功能，然而却忽略了开源组件的安全性带来的风险。从企业安全管理出发，理清企业内部应用产品或项目的开源软件引用依赖关系，开源软件存在的安全漏洞等成为了一个亟待解决的安全难题。

在监管侧，通过自动化的收集、融合、处理开源组件仓库中托管的开源项目信息，构建开源软件知识图谱。在此基础上，结合漏洞知识库和缓解措施，建设开源软件安全风险监测知识图谱，并向企业提供开源项目软件成分及依赖关系的语义查询、风险监测，安全开源项目推荐等数据服务。可以有效识别、降低企业引入开源软件的安全风险，解决企业引入开源软件资产的安全和合规问题。

构建过程如下图所示，在数据处理技术方面应用自然语言处理技术对软件实体识别，关系抽取以及开源软件的特征工程（指纹、属性、签名等）。上层可视化提供信息查询和基于图算法分析的能力，提供软件漏洞的级联拓展，可进一步挖掘背后的关系知识。

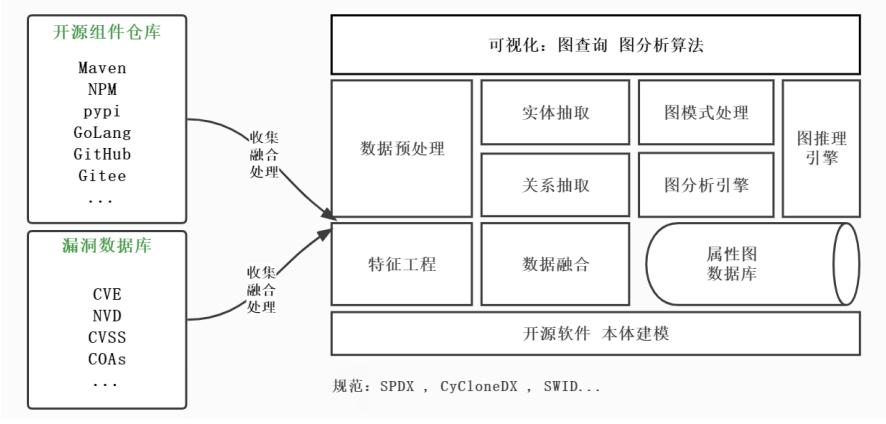


图 5.2 开源软件知识图谱基本框架

5.1.4 企业 SBOM“安全屋”托管服务

在软件供应链安全中，透明程度高的软件成分清单（如 SPDX 规范）有利于软件供应链安全管理，对软件供应链生态安全治理提供更有效信息输入和判定依据，但其如被泄露也会大大增加提供方的安全风险。因此，由上下游企业之间直接对接这种敏感信息存在很大难度，需要具备公信力的公立机构，可信的技术机制和方案，才能让企业间放心的提供详细的软件成分信息，并让最终用户可以查询是否存在风险。

结合软件供应链数据安全技术与可信计算环境，可以为 SBOM 数据提供一个安全可信的底座。整个过程中，第三方机构向企业提供软件产品的 SBOM 安全托管平台。企业在平台上租赁自己的 SBOM 安全屋，在安全屋中以密文的形式存放符合规范的 SBOM 数据。最终用户可以在托管平台中查询软件产品 SBOM 的一些重要信息，如是否包含某个组件，或某个具体组件是否存在高危风险，平台提供查询结果及证明。

在整个过程中，提供 SBOM 安全屋托管平台与服务的机构，需要具备以下能力：

1. 托管平台的 SBOM 安全屋中， SBOM 数据以密文存放、且在机密计算环境（TEE）中访问。平台无需持有企业的 SBOM 数据解密密钥，平台亦无法窃取企业的 SBOM 数据，意即：平台具备“自证可信”的安全功能。
2. 在安全屋中的 SBOM 数据具有“可搜、不可见”的特点，上游企业能够确知用户的查询次数，但无法获得用户的查询内容，意即：用户享有“隐匿查询”的权力。
3. 平台方确保用户查询的 SBOM 与之前公示的 SBOM 具有一致性，意即：用户享有“不被数据歧视”的权力。
4. 上游企业可撤销托管平台上的 SBOM，且可确保“撤销之后，托管平台无法继续使用该 SBOM”，意即：上游企业享有“SBOM 被遗忘”的权力。

5.1.5 软件供应链安全监测预警

软件供应链安全监测预警是通过针对软件供应链关键节点的情报信息采集，实现快速反应。是对当前软件供应链安全严峻形势充分认识的体现。软件供应链安全检测预警，能充分落实网络安全责任，有助于持续提高信息科技治理能力。软件供应链产业安全，配套产业及基础材料安全等领域的研究，从宏观角度对丰富自身安全防护手段进行政策指导，对现有软件供应链系统进行宏观梳理查找漏洞，建立风险预警机制，保障供应链安全可控。

5.1.5.1 风险环境判断

在软件供应链整个链路流程中，每个阶段都在面临不同的安全风险，为了更好的应对软件供应链的各类风险状况，需要对突发事件的场景由充分的应对策略，通过管理手段与技术手段相结合的方式避免突发事件的危害扩大，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。

软件项目都涉及众多的供应商，包括咨询设计方、软件开发方、信息化产品供应商、系统集成商、运维服务提供商、安全服务提供商等。供应商自身的网络安全建设缺失，也会带来运维权限盗用、源代码泄露等风险，众多供应商提供的安全信息与其所产品所涉及组件的安全信息同样是重要安全情报搜集对象。

从上述典型软件供应链的风险环境可大致判断软件供应链安全检测预警工作需要关注两个层面：一类是软件供应商自身所存在的安全风险，例如软件供应商网络安全缺陷、端口暴露、弱密码、弱口令等；另一类是软件产品本身的安全风险，例如软件漏洞等。

5.1.5.2 建立常态化机制

采购方需要针对软件供应链安全检测常态化，配合自身的安全检查工具通过深度检查对自身系统进行摸底与建模，整理出自身组件清单，结合安全情报服务，后期不断维持情报更新实现动态监测，对关键位置与关键组件进行针对性的安全加固与防范。

结合国家官方的漏洞管理机构仿照海外模式建立政府统一管理的情报库实现软件供应链安全预警平台建设也是完全可以预期的，供应链涉及面广，成分复杂，单一企业或某个组织一己之力难以保证安全情报来源的广泛性与时效性。企业做好自身安全检测预警机制，提前做好信息管控架构与情报引用，为未来安全监测情报打通提前做好准备。

5.2 供应链安全管控

5.2.1 建立软件供应链资产台账

一个组织的产品服务的类型多样、构成复杂。通过建立软件供应链资产台账，能够清晰组织供应链关系。组织建立软件供应链台账需要全面梳理供应链终涉及到的供应商、软件、工具、服务、上下游交付环节，保证供应链流程不会存在遗漏。

5.2.1.1 供应商的梳理

梳理范围：主要基于对组织的直接供应商进行梳理。没有对供应链间接供应商梳理也是考虑到如果对供应链的多个层次进行管理，可能造成管理成本的上升和流程复杂度的增加，而从终端客户角度来看，组织的直接供应商承担着主要责任。

梳理方式：问卷调查、访谈。

梳理内容及结果：供应商通常分为产品供应商、系统集成商、服务提供商，包括设计方、开发方、承建方、网络安全产品提供方、信息化产品提供方、运维方、安全服务提供方、信息安全测评方、其他参与方等等。对供应商梳理形成供应链企业清单，内容包括企业名称、所属省市、具体地址等信息，掌握组织的所属供应商。

5.2.1.2 供应链产品服务梳理

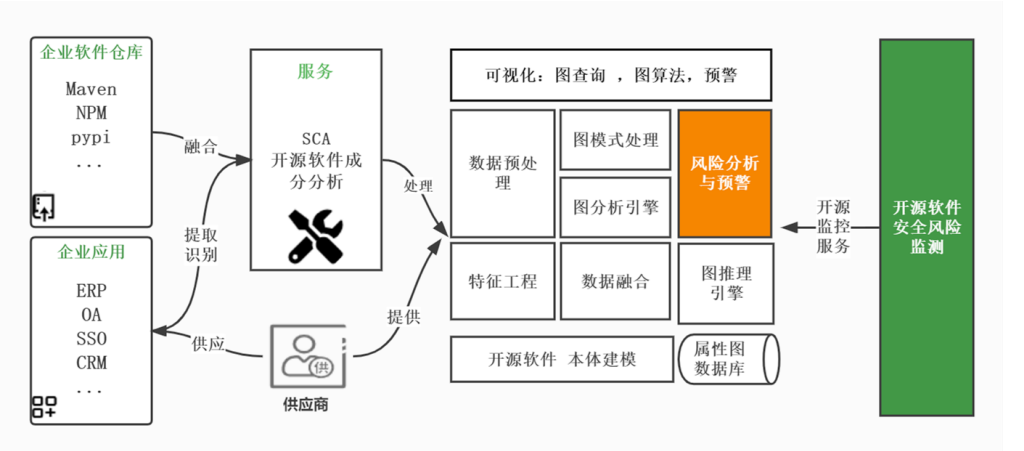
梳理范围：主要对软硬件产品服务进行梳理。

梳理方式：问卷调查、访谈。

梳理内容及结果：包括 OA 办公平台、电子邮件系统、网络监控软件、云盘或 FTP 等文件，文件共享平台、源代码仓库、VPN 产品、典型业务终端软件、视频会议软件、财务软件行业专用软件等软件及网络设备、安全设备、服务器、手持设备等硬件，查清重要供应链产品的版本、型号、生产厂商、开发类型、涉及操作系统、是否有信息回传厂商及回传信息的主要内容等基本要素，形成供应链产品清单。

5.2.1.3 供应链资产台账管理及可视化

已完成梳理形成的供应链企业清单、供应链产品清单，需要建立供应链资产台账平台进行管理，并通过建立供应链资产管理机制，如管理流程或自动化技术（如 SCA 产品）持续更新这些清单信息。建立的资产台账，可以结合开源软件安全风险检测等安全监督机制及时识别企业存在的软件供应链风险。对于软件管理能力较高的企业，可以结合已具备的漏洞预警能力、企业资产风险管理能力，建立企业资产台账风险管理平台，多维度评估风险和可视化展示能力，协助软件企业制定符合自己的漏洞预警与处置方案。同时，结合知识图谱的推荐服务还可以提供修复方案或解决办法供企业安全人员参考，并制定内部安全处置方案。



面向企业的资产台账管理平台

5.2.2 开源软件及服务商测评

在软件供应链中，出于成本的考虑，项目中使用开源软件或者使用开源组件进行二次开发，已被广泛应用。加之服务商众多，经验及服务质量不尽相同。所以针对开源软件及其服务商的测评是保障供应链各环节安全的重要方式。

1. 开源软件测评

测评范围：适用于对开源软件以及同类开源技术、软件和产品进行评测，为开源软件质量和成熟度提供一定参考。同时帮助进行技术路线选择和开源软件选型。

测评内容：评测模型具有一级评估属性、二级评估属性、评测指标三个层次，共有 12 项一级评估属性、42 项二级评估属性和 129 项评测指标。包括开源许可证、行业认可度、产品活力、服务交付、安全性、兼容性、可维护性、可扩展性、功能性、可靠性、易用性、性能效率。包括但不限于以下内容，可结合业务情况、侧重点增加测评项。

开源许可证	开源许可证类别
	开源许可证权力和限制
	开源许可证冲突
行业认可度	商业版本
	商业化实践或应用案例
	第三方评估结果
	用户满意度

根据对检查的评估属性和评估指标进行分级定义和分级运算每一级有相应的权重体系，自下而上计算得出开源软件的量化评分值。

2. 服务商测评

测评范围：对开源软件的直接服务商，包括整体组织机构或某个独立的具体开源技术领域的系统或部门。

测评内容：内容依据评测模型具有一级评估属性、二级评估属性、评测指标三个层次，共有 8 项一级评估属性、20 项二级评估属性、118 项评测指标。包括：行业经验、技术能力、服务质量、开源经验、企业资质、服务交付、运维支持、安全保障。包括但不限于以下内容：

行业经验	项目实施经验
	近三年同业服务能力
技术能力	技术人员
	核心技术
	技术方案和验证

软件服务商评测模型涉及定性评价和定量评价两类评测指标，使用专家评分法对服务商进行评测：首先根据服务商的评测指标制订评分标准，然后聘请若干代表性专家凭借自己的经验按此评分标准给出各指标的分值，最后汇总计算得到指标得分和服务商综合得分。

5.2.3 商业软件安全技术检测

5.2.3.1 渗透测试

渗透测试（Penetration Testing）是由具备高技能和高素质的安全服务人员在不同的位置（比如从内网、从外网等位置）利用各种手段对某个特定网络发起、并模拟常见黑客所使用的攻击手段对目标系统进行模拟入侵进行测试，以期发现产品中存在的安全隐患。渗透测试服务的目的在于充分挖掘和暴露系统的弱点，从而让管理人员了解其系统所面临的威胁。

渗透测试是脆弱性评估的一种很好的补充。同时，由于主持渗透测试的测试人员一般都具备丰富的安全经验和技能，所以其针对性比常见的脆弱性评估会更强、粒度也会更为细致。另外，渗透测试的攻击路径及手段不同于常见的安全产品，所以它往往能暴露出一条甚至多条被人们所忽视的威胁路径，从而暴露整个系统或网络的威胁所在。最重要的是，渗透测试最终的成功一般不是因为某一个系统的某个单一问题所直接引起的，而是由于一系列看似没有关联而且又不严重的缺陷组合而导致的。日常工作中，无论是进行怎么样的传统安全检查

工作，对于没有相关经验和技能的管理人员都无法将这些缺陷进行如此的排列组合从而引发问题，但绿盟科技的渗透测试人员却可以靠其丰富的经验和技能将它们进行串联并展示出来。

5.2.3.2 组件漏洞分析

软件成分，即软件物料清单，通常称为 BOM。高德纳公司也建议企业“持续构建详细的软件物料清单，以提供对组件的完全可见性”[31]。通过软件成分分析 SCA 工具，可以生成 BOM 清单，提供详尽的组件版本和许可信息等。维护准确的软件 BOM 可以帮助企业快速查明易受攻击的组件。

组件的典型风险包括：1. 组件中包括已知的漏洞；2. 组件中包含高危漏洞；3. 组件过期，多年未运维或已经不维护，可能存在未知风险。

针对开源软件成分分析，可以采取以下步骤：1. 使用 SCA 工具和技术，形成开源软件的 BOM 清单；2. 关注 BOM 中的组件，涉及到中高危漏洞的组件，进行升级或规避修复；3. 定期更新组件和漏洞知识库，对新发现和暴露的漏洞，及时进行修复。

5.2.3.3 代码审计

代码安全审计，这里指采用静态分析方法的一种白盒安全测试，也称之为静态应用程序安全测试 SAST。主要用于识别非运行（静态）代码中的编码缺陷。静态测试包括对代码缺陷检测和代码质量检测。

代码中静态风险包括：1. 开发人员在上线前做 AST 测试交付压力较大，如何在 IDE 编码阶段处理代码风险；2. 在编码阶段，可能会引入的安全性或代码质量问题；3. 一些编译后的字节或二进制代码，需要进行分析 [32]。

针对开源代码静态审计，步骤如下：1. 使用 SAST 工具和技术，对开源代码进行审计；2. 针对不同开发语言，配置相应的检测规则或采用默认检测规则；3. 对检查出代码中的漏洞，参照修复建议进行修复。

5.2.4 供应商安全风险评估

供应商风险评估是以对标国外先进水平、促进供应商服务持续改进; 保障供应链安全稳定; 保障敏感数据为核心。包括协助供应商分级分类、分别针对不同类别的供应商建立供应商风险度量指标：机房、非驻场开发、鉴证核查、数据处理共 4 类供应商风险度量指标。定期风

[31] Garter, 2019, Technology Insight for Software Composition Analysis
[32] Gartner, 2020, How to Deploy and Perform Application Security Testing.

险评估，发现安全风险并要求征整改。重点聚焦威胁防御，数据安全，业务连续性。最终实现发现并安全隐患、供应商评级参考、保障供应链业务安全。

5.2.4.1 供应商分类分级

供应商分级分类是对供应商进行分级分类管理，对重要供应商和一般供应商采取差异化管控措施。通过分类分级差异化管控，重点关注重要供应商的高风险等级评估项，对其进行优先整改，实现精准高效供应链安全管理。属于重要供应商的包括但不限于：核心业务软件系统开发测试和运行维护的供应商、软件涉及集中存储重要数据和客户个人敏感信息的供应商、直接影响实时业务和影响账务等重要信息准确性的供应商；其它对组织业务运营具有重要影响的供应商。

5.2.4.2 建立指标

各级别风险度量指标是对供应商建立服务效能和质量监控指标，并进行相应指标监控。依据与业务场景相结合的指标，能够准确分析供应商需承担的安全责任和所需的安全能力，特别是数据保护能力。针对不同类型的软件供应商可定制不同评估项，实现标准化。常见评估领域包括但不限于：基本情况、制度管理、人员安全、业务连续性管理、服务水平连续管理、机房管理要求、机房物理要求、办公环境、终端安全、系统安全、数据安全等。

5.2.4.3 风险评估及整改

现状调研：主要以文档调阅、人员访谈、现场勘查的方式进行。文档调阅主要是调阅供应商管理体系相关文档，包括供应商运维管理文档、信息安全管理策略、业务连续性管理文档、信息科技外包管理文档、系统设计文档等，以充分了解供应商管理现状。人员访谈包括对相关部门相关人员的访谈，人员访谈目的为验证管理制度和流程的制定 / 执行 / 监督 / 维护情况，以及人员对于本岗位相关的管理制度和流程的理解水平。现场勘查主要是对物理机房的现场走查，实地查看物理环境是否能够承载各信息系统稳定运行。

差距分析：以国家标准为主要依据，结合行业监管机构的相关监管规范的内容，根据实际情况，形成衡量供应商管理能力检查表、供应商风险评估表。覆盖了机房、非驻场开发、鉴证核查、数据处理 4 类供应商风险度量指标，涵盖了基本情况、制度管理、人员安全、业务连续性管理、服务水平连续管理、机房管理要求、机房物理要求、办公环境、终端安全、系统安全、数据安全等评估项。以完成供应商及供应商管理与国家、行业监管要求之间的差距合规性分析。

5.2.5 供应链安全应急体系建设

5.2.5.1 应急预案编制

编制一个适用于企业自身的应急预案应当包含以下步骤：

1. 现状调研

为了使编制的应急预案贴合企业实际情况，在正式编制应急预案之前，应当对企业现状进行调研，调研内容包括但不限于涉及软件成分信息，软件使用情况，涉及业务的重要程度，以及软件的部署架构及方式；运行维护涉及到的部门、供应商情况；软件系统的风险评估结果等。

2. 预案编制

收集完上述内容后，就可以成立相应的应急预案编制组进行应急预案的编制。同时还要确定应急预案的场景。供应链安全事件与现有安全事件分类标准存在一定的差异。

微软将供应链安全事件主要分为四类：1、损坏的软件生成工具或更新的基础结构；2、被盗的代码签名证书或使用开发人员公司的标识签名的恶意应用；3、硬件或固件组件中附带的已泄漏的专用代码；4、在相机、USB、手机等设备上预安装的恶意软件。微软的这个分类方法主要是从攻击手法的角度来对供应链安全事件进行一个分类。

欧盟在 2021 年 7 月发布的《ENISA THREAT LANDSCAPE FOR SUPPLY CHAIN ATTACKS》中指出欧盟网络安全事件分类学用于联盟一级的事件响应协调活动和信息共享。由于分类法在概念上不同，不允许对供应链事件进行详细分析，因此提供了一种对供应链攻击事件分类的方法，从 4 个要素进行考虑，分别为供应商、供应商资产、客户以及客户资产，这种分类方法可覆盖供应链攻击的整个过程。我们建议同时使用这两种分类法。

NIST 则在 2021 年 4 月发布的《Defending Against Software Supply Chain Attacks》中将常见攻击链安全事件分了三类：1、劫持更新：大多数现代软件都会收到常规更新，以解决错误和安全问题。软件供应商通常将更新从集中服务器分发给客户，作为产品维护的常规部分。威胁参与者可以通过渗透供应商的网络并将恶意软件插入传出的更新或更改更新以授予威胁参与者对软件正常功能的控制权来劫持更新。例如，NotPetya 袭击发生在 2017 年，当时针对乌克兰的俄罗斯黑客通过乌克兰流行的税务会计软件传播恶意软件。后来被称为 NotPetya 的恶意软件蔓延到乌克兰以外，并在国际航运、金融服务和医疗保健等关键行业造

成了重大全球破坏。2、破坏共同设计：共同设计用于验证代码作者的身份和代码的完整性。攻击者通过自签名证书、破坏签名系统或利用配置错误的帐户访问控制来破坏共同设计。通过破坏共同设计，威胁参与者可以通过冒充受信任的供应商并将恶意代码插入更新来成功劫持软件更新。例如，总部位于中国的威胁参与者 APT 41 在对美国和其他国家进行复杂的软件供应链妥协时经常破坏共同设计。3、破坏开源代码：当威胁参与者将恶意代码插入可公开访问的代码库时，就会发生开源代码妥协，毫无戒心的开发人员——寻找免费的代码块来执行特定功能——然后添加到他们自己的第三方代码中。例如，2018 年，研究人员发现了 12 个恶意 Python 库上传到官方 Python 软件包索引（PyPI）。攻击者使用拼写策略，创建了名为“diango”、“djago”、“dajngo”等的库，以吸引寻求流行“django”Python 库的开发人员。恶意库包含与模拟库相同的代码和功能；但它们还包含其他功能，包括在远程工作站上获得引导持久性和打开反向外壳的能力。开源代码妥协也可能影响私有软件，因为专有代码开发人员通常在其产品中利用开源代码块。

在编制的过程中可以参考相关法律法规、行业规范、技术规范等，结合企业自身情况和安全能力，制订相应的应急处置流程和操作规范。

3. 预案评审、发布及培训

编写完成的应急预案，应当邀请相关人员，如内外部专家，预案涉及部门等进行评审，评审通过后通过企业的文件发布流程对应急预案进行发布，召集所有涉及的人员进行相关培训。

5.2.5.2 专项应急演练

5.2.5.2.1 应急演练目标

开展供应链安全应急演练的目标主要是为了检验以下能力：

1. 情报收集 / 研判能力

情报收集 / 研判对于供应链安全事件应急处置至关重要。传统的网络攻击事件我们可以依靠各种安全设备的规则匹配，或者攻击行为进行建模判断，但是供应链安全事件往往由 0day 漏洞或 APT 攻击引起，具有较强的隐蔽性。依靠传统的监测手段往往很难发现。一般各种消息都是先出现在小密圈，技术论坛等渠道，同时各种信息混杂，需要进行一定的研判

工作，所以要快速的发现供应链安全事件并进行相应，需要具备一定的情报收集 / 研判能力。这个情报收集可以是通过官方的渠道和安全公司获取。一方面官方的渠道是指供应商相关信息推送或上级主管单位横向报送，另外一方面就是安全公司了，包括专职的威胁情报公司和一些具备较强专业性的安全公司。

2. 软件成分分析能力

供应链安全事件的严重性就表现在目前大量的软件中都可能使用相同的代码包、框架或者开源代码。所以面对供应链安全事件的时候客户或供应商都应具备软件成分分析能力，软件成分分析或记录越完整，在发生供应链安全事件时响应速度就越快，可以第一时间明确受影响的资产信息，根据资产的重要程度制订相应的处置优先级。如 2021 年 12 月 9 日披露的 Apache Log4j2 远程代码执行漏洞（CVE-2021-44228），根据 MVN Repository 显示有接近 7000 的项目引用了 Log4j2。如果企业没有软件成分分析能力或软件成分管理，企业内部排查受影响软件时，将耗费大量的时间，导致错过最佳的修复窗口期。也无法保证将所有受影响软件都进行了修复。

3. 完善的阻断能力

当我们发现供应链安全事件时，我们往往需要采取临时措施根据情报的 IOC 进行封禁，争取修复相关缺陷的时间。如果企业的阻断能力不完善，无法进行相关攻击或连接的阻断，那么在事件处置的过程中将大大增加难度，往往有可能出现修修补补反复工作的情况发生。

4. 良好的沟通机制

良好的沟通机制不管在平常的网络安全事件处置过程中还是在供应链安全事件处置过程中，都极其重要。这里要特别提出来时因为平常的网络安全事件的处置，大多数的沟通主要局限在企业内部跨团队之间的沟通。但是供应链相对比较复杂，不仅涉及企业内部不同部门之间的沟通，可能涉及到多个供应商之间的沟通协调。所以要提前建立良好的沟通机制，明确沟通流程和沟通渠道。

5.2.5.2.2 应急演练流程

应急演练工作的基本组织流程可如图所示。

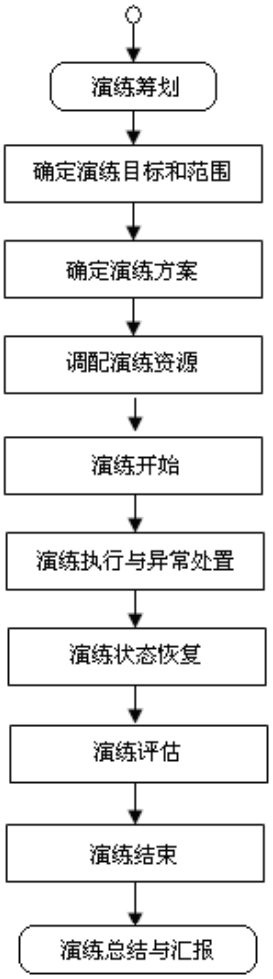


图 5.4　应急演练工作基本流程图

a）演练筹划：根据应急响应工作的实际需要，统筹提出应急演练的需求，制订应急演练工作计划；

b）确定演练目标和范围：由应急演练工作领导小组确定应急演练的目标、范围，以及参与的机构和人员，并授权各参与机构和人员按各自职责参加应急演练；

c）确定演练方案：由应急演练策划组根据应急响应预案，组织编制应急演练方案，设置演练场景，专家顾问组成员同期参与指导，应急演练工作领导小组确定与批准；

d）调配演练资源：由应急演练保障组根据应急演练方案调配演练所需的各项资源；

e）演练开始：由应急演练工作领导小组授权的演练总指挥宣布应急演练开始；

f）演练执行与异常处置：应急演练策划组启动应急演练条件，各组各行其责，按照预定方案实施应急演练，或在演练总指挥的指挥下处置演练过程中出现的非预设事件；

g）演练状态恢复：应急演练结束后，按照预案将演练所涉及到的环境、资源和信息系统等恢复到演练前的状态；

h）演练结束：由演练总指挥宣布应急演练工作结束；

i）演练总结与汇报：由演练总指挥组织相关人员，对本次应急演练工作进行评估和总结，并向应急演练工作领导小组报告应急演练工作情况；

j）针对演练中反映出的不足或问题，由相关发起部门进行改进或调整。

5.2.5.2.3 演练方式

5.2.5.2.3.1 桌面演练

桌面演练也称沙盘演练、沙盘推演，指参演人员利用地图、沙盘、流程图、计算机模拟、视频会议等辅助手段，依据应急预案对事先假定的演练情景进行交互式讨论和推演应急决策及现场处置的过程，一般可作为实战演练的基础。可分为讨论式桌面演练及推演式桌面演练，通常在会议室举行。

桌面演练的目的包括：明确相互协作和职责划分，锻炼演练人员解决问题的能力；发现和解决预案和程序中的问题，取得一些有建设性的讨论结果；

（一）讨论式桌面演练

讨论式桌面演练主要是围绕对所提出问题进行讨论。由总指挥根据演练方案以口头或书面形式，部署引入一个或若干个问题。参演人员根据应急预案及有关规定，讨论应采取的行动，按照应急预案和标准行动程序，对所讨论内容形成书面总结和改进建议。

（二）推演式桌面演练

在推演式桌面演练中，由总指挥按照演练方案发出控制消息，参演人员接收到控制信息后，通过角色扮演或模拟操作，完成应急程序。推演式桌面演练的准备工作至少需要完成演练执行程序控制流程图及演练脚本。

突出优势：规模较小，不需要协调应急响应工作组以外人员、资源的模拟演练，对系统无实际影响。

5.2.5.2.3.2 实战演练

实战演练可分为模拟操作演练和真实操作演练。

模拟操作演练一般在测试环境下进行。演练环境要求尽量逼近实际生产环境，演练过程要求尽量真实。模拟操作演练要求注重技术操作的验证、各方资源的协调和配合、各类问题的解决和风险的应对。

真实操作演练一般需要进行预演练（可采用推演式桌面演练、模拟操作演练等方式）。真实操作演练要求在真实环境下进行，注重技术操作和业务处理。演练过程应调用全部相关人员和资源，以检验相互协调配合的整体应急能力。

突出优势：规模较大，需要协调应急响应工作组及各个部门做好突发事件准备，对系统或网络可能产生影响，但效果真实。

5.2.5.3 事件响应处置

为最大限度科学、合理、有序地处置信息安全事件，采纳了业内通常使用的 PDCERF 方法学（最早由 1987 年美国宾夕法尼亚匹兹堡软件工程研究所在关于应急响应的邀请工作会议上提出），将应急响应分成准备（Preparation）、检测（Detection）、抑制（Containment）、根除（Eradication）、恢复（Recovery）、跟踪（Follow-up）6 个阶段的工作，并根据网络安全应急响应总体策略对每个阶段定义适当的目的，明确响应顺序和过程。应急响应 PDCERF 模型如图所示：

1. 准备阶段主要包括在事件处理过程中可能有用的可用工具和资源的示例，建立安全保障措施，制定安全事件应急预案，进行应急演练，对系统进行安装和配置加固等内容。

2. 检测阶段是应急响应全过程中最重要的阶段，在这个阶段需要系统维护人员使用检测技术或设备进行检测，确定系统是否出现异常，在发现异常情况后，形成安全事件报告，并由安全技术人员介入进行高级检测来查找安全事件的真正原因，明确安全事件的特征影响范围和标识安全事件对受影响的系统所带来的改变。

3. 抑制阶段是对攻击所影响的范围、程度进行扼制，通过采取各种方法，控制、阻断、转移安全攻击。针对上一检测阶段发现的攻击特征，比如攻击利用的端口，服务，攻击源，攻击利用系统漏洞等，采取有针对性的安全补救工作，以防止攻击进一步加深和扩大。

4. 根除阶段是在抑制的基础上，对引起该类安全问题的最终技术原因在技术上进行完全的杜绝，并对这类安全问题所造成的后果进行弥补和消除。

5. 恢复阶段主要内容是将系统恢复到正常的任务状态。在系统遭到入侵后，攻击者一定会对入侵的系统进行更改。同时，攻击者还会想尽各种办法使这种修改不被系统维护人员发现，从而达到隐藏自己的目的。

6. 跟踪阶段主要内容是对抑制或根除的效果进行跟踪和审计，确认系统或终端没有被再次入侵，还需对事件处理情况进行总结，吸取经验教训，对已有安全防护措施和安全事件应急响应预案进行改进。

但是在供应链安全事件中存在细微差别，因为供应链安全事件涉及到两类角色，供应商和客户。各自的职责和目的不一样，导致处置流程存在区别。

从供应商的角度来看，供应链安全事件，实则是提供的产品出现问题，那么相关的应急处置流程则和企业内部的变更流程类似。定位问题、制订解决办法、实施变更、变更测试、新版本发布、通知客户。所以从供应商的角度来看，更多的是对相关产品进行变更。

客户的角度来看，供应链安全事件的处置流程就基本和 PDCERF 方法类似。但是供应链安全事件涉及的供应商服务中断 / 更换，引发的安全问题则不能完全适用，供应商服务中断 / 更换的应急处置流程实则是一个备份恢复的过程。

6

行业、企业最佳实践

6.1 银行业金融机构信息科技外包安全实践

面对日益加剧的竞争及技术变革的加快，IT 外包以精专业、高效率、低成本等优势，成为了国内各大金融机构提高竞争力的重要手段之一。但 IT 外包强势进驻的同时，给各金融机构带来了巨大的安全隐患，近年来外包风险事件越来越多，也引起了监管单位的重视，2008 年初，银监会发布《银行业金融机构外包风险管理指引》（征求意见稿），并于 2010 年 6 月正式发布《银行业金融机构外包风险管理指引》（银监发 [2010]44 号），且先后下发了《银行业金融机构信息科技外包风险监管指引》（银监发 [2013]5 号）、《关于加强银行业金融机构信息科技非驻场集中式外包风险管理的通知》（银监办发 [2014]187 号）及《中国银监会办公厅关于开展银行业金融机构信息科技非驻场集中式外包监管评估工作的通知》(银监办发 [2014]272 号)，要求各地各级银行业金融机构遵照执行。

绿盟科技依据监管机构的相关要求，针对银行业金融机构 IT 外包管理现状，从组织架构、战略建设及风险管理、生命周期管理、集中度风险管理、非驻场外包管理、重点外包服务机构管理共六个方面为银行提供评估咨询建设服务方案，协助金融机构建立一套外包前、中、后期的全流程风险管理体系，形成有效外包风险内控机制，同时建立完善外包风险管理持续改进机制，并促使金融用户满足行业监管合规要求。

外包管理组织架构：包括但不限于外包管理制度与流程；外包管理职责及权限；外包管理报告路径；外包管理效果评价等。

外包战略建设和风险管理：包括但不限于外包战略制定；外包战略合理性评估；外包风险管理情况等。

外包服务生命周期管理：包括但不限于外包风险评估及准入；服务商尽职调查；外包服务合同及要求规范性；**外包服务安全管理；**外包服务监控与评价；外包服务的安全与应急等。

外包集中度风险管理：外包服务提供商机构集中度风险识别；外包服务提供商机构集中度风险防范等。

非驻场外包管理：包括但不限于非驻场外包风险管理；非驻场集中式外包风险管理；非驻场集中式外包监管评估等。

重点外包服务机构管理：包括但不限于重点外包服务机构准入；重点外包服务应急管理；重点外包服务机构风险管理及审计要求执行等。

其他方面：包括但不限于关联外包管理；内部人员风险管理；监管报送等。

6.1.1 信息科技外包评估内容

根据《银行业金融机构信息科技外包风险管理指引》、《关于加强银行业金融机构信息科技非驻场集中式外包风险管理的通知》等相关监管制度，以科技风险为导向，应用行业典型风险评估方法，对银行信息科技外包，从组织架构、科技外包战略与外包制度体系建设、外包风险管理、外包风险评估及准入、服务商尽职调查、外包服务合同及要求、安全管理、外包服务监控与评价、外包集中度管理、非驻场集中式外包风险管理等方面全面、科学、深入、客观地进行风险分析和评估，明确每一风险点等级，出具外包风险评估报告，并提出相应整改措施和建议。

6.1.2 实施步骤

1. 实施思路

2. 解读行业规范→②收集经验材料→③调研客户现状→④建立外包评估标准→⑤对标改进完善→⑥输出评估报告→⑦协助优化整理

3. 实施步骤

任务规划阶段：详细解读行业外包管理的规范要求和银监会监管要求，归纳提取出银行机构外包业务关键环节的标准要求，详查公司的组织过程资产。

任务实施阶段：进行现场调研，了解银行信息科技部门外包业务的详细现状，包括外包管理组织架构、外包业务的内容范围、外包业务的各项流程、外包风险的管控措施、外包相关的监督评价机制、已入围的外包服务商等等，梳理客户外包业务现状；梳理并建立可落地的信息科技风险外包评估标准。

任务收尾阶段：评估客户根据模板制定的外包管理办法和银监会监管要求情况，从专家的角度给出修订意见，出具评估报告及解决方案建议。

6.2 交通运输企业供应链安全监督检查实践

6.2.1 项目概述

某交通运输行业单位积极组织落实上级监管单位监管要求，对本单位供应链安全风险进行了自查，出具了自查报告，并对发现的安全风险进行整改落实。自查分为技术自查和管理风险自查两部分。

6.2.2 技术自查

（一）资料收集、梳理

针对识别、收集的关键供应链产品，与各供应商收集梳理此类产品已具有的销售许可证及近期相关检测报告（第三方安全检测报告、代码审计报告、漏洞扫描报告、渗透测试报告、等级测评报告）。

（二）环境准备

对关键供应链产品准备技术检测涉及的环境，包括协调供应商准备测试环境，协调测试时间窗口等。

（三）技术检测

技术检测主要通过代码审计、漏洞扫描、软件安全分析进行验证产品的安全性。具体检测方式如下：

- 代码审计：

人工源代码审计（由具备丰富的安全编码及应用安全开发经验的人员，根据一定的编码规范和标准，针对应用程序源代码，从结构、脆弱性以及缺陷等方面进行审查）和工具源代码审计（SDA 代码审查工具）

- 漏洞扫描：

通过评估工具对某交通运输行业单位供应链产品相关的脆弱性进行安全检查，以发现目标可能存在的安全隐患，有效发现产品涉及的操作系统和应用软件在用户账号、口令、安全漏洞、服务配置等方面存在的安全风险、漏洞和威胁，为进一步通过技术手段降低或解决发现的问题提供了参考依据和方法。

- 软件安全分析：

对开源组件及软件进行漏洞分析、后门分析，识别存在的漏洞、被植入的后门木马、访问的风险域名等问题。

（四）自查分析

整体分析上一步骤对供应链产品技术检测数据，形成自查报告。

6.2.3 管理风险自查

第一阶段：自查启动阶段。此阶段主要确定评估范围、评估方法和评估工作计划安排，确保某交通运输行业单位供应链管理能力自查活动能够顺利开展。

第二阶段：问卷调查阶段。以电子邮件的方式向供应商发放供应商调查内容清单，收集供应商基本情况，了解和掌握供应商信息安全管理状况，识别其在开展项目活动重存在的风险点，同时为后续各阶段的工作提供基础数据与资料。

第三阶段：文档审查阶段。通过对某交通运输行业单位进行文档审查，了解供应链管理落实情况，对于标准中的要求，是否在制度流程上得以控制，特殊的业务要求是否明确说明原因等。

第四阶段：业务参与风险梳理阶段。根据供应商参与业务系统工作流程需求，整理出相应风险点列表，为人员访谈和技术检查做好准备。

第五阶段：人员访谈阶段。在某交通运输行业单位供应链能力自查工作中，自查人员通过对某交通运输行业单位相关人员进行现场访谈，询问供应商工作中检查点内容是如何实现的，实现的具体要求及相关制度规定落实情况等。

第六阶段：技术检查阶段。在某交通运输行业单位供应链管理能力自查工作中，针对部分关键检查点，通过对其进行制度文审、人员访谈外，还需要进行现场检查，确认该检查点是否与制度规定、人员访谈结果一致。对于现场检查工作，采取漏洞扫描、代码审计、手工检查、渗透测试、抓包分析等手段，评估自查内容中的控制措施的有效性，做出是否符合自查要求的最终判断。

第七阶段：分析与报告阶段。根据调研及检查现状，梳理总结风险问题，形成风险问题清单。综合以上工作，建立风险评估报告。

第八阶段: 风险处置阶段。针对自查中存在的安全问题，提出改进意见，制定风险处理建议，并在后续工作中落实问题整改，跟踪整改情况。

7

典型供应链攻击案例复盘

7.1 IT 管理供应商 SolarWinds 供应链攻击事件

SolarWinds 是一家国际 IT 管理软件供应商，Orion 是该公司的网络管理系统（NMS）产品。该供应链攻击事件由安全公司 FireEye 发现，并在 2020 年 12 月公布，将幕后的国家级 APT 团伙称为 UNC2452。进一步的调查证实攻击者通过 Orion 软件更新包中植入的后门向下游植入恶意后门代码。

美国多家全球 500 强企业、重要政府机构都是 SolarWinds 的客户，使得该事件在该国造成了较大的社会影响。12 月 14 日，华尔街日报发表文章表示，美国财政部和商务部等数个联邦机构网络系统遭到入侵，政府部门的通信可能已经遭到攻击势力监控。12 月 15 日，路透社报道，SolarWinds 入侵也被用于渗透到美国国土安全部（DHS）的计算机网络。12 月 16 日，据美国“政客”新闻网报道，美国国家安全事务助理罗伯特·奥布莱恩当地时间 15 日缩短了他的出访行程，返回华盛顿，协调处理“美国政府机构遭遇网络攻击”事件。部分人怀疑白宫也受到了本次攻击的影响。SolarWinds 事件的高影响度，除了受攻击影响的目标重要性高，还因为它攻陷了由众多安全企业与国家安全部门共同构建的安全防线，令整个安全体系的安全性与可靠性遭到了怀疑，足以撼动其根基。

7.1.1 攻击流程分析

表 7.1 SolarWinds 事件供应链攻击技术与脆弱性

生命周期	安全风险发生阶段	攻击技术
上游安全	上游企业安全问题	外部攻击 关键数据泄露
开发安全	开发环境安全	开发环境污染 CI/CD 集成环境污染 托管平台代码、存储平台污染
	开发过程安全	遭受外部攻击 恶意代码植入

攻击者利用漏洞、密码爆破或社会工程学攻击的手段，获取了 SolarWinds 研发环境的控制权限。长时间的潜伏与刺探之后，在 Orion 平台软件的核心服务组件代码中植入了恶意后门，以在线升级包的形式被下游用户静默安装。包含恶意代码的升级程序利用来自厂商的合法数字证书绕过验证，在两周的潜伏阶段结束后通过隐蔽的自定义 DNS 通讯，实现隐蔽的命令控制与数据回传。攻击者对于产品代码与工作流程都十分熟悉，在后门代码中甚至调用了产品中的合法组件，侧面显示了攻击者对 SolarWinds 研发环境长时间的实际控制。

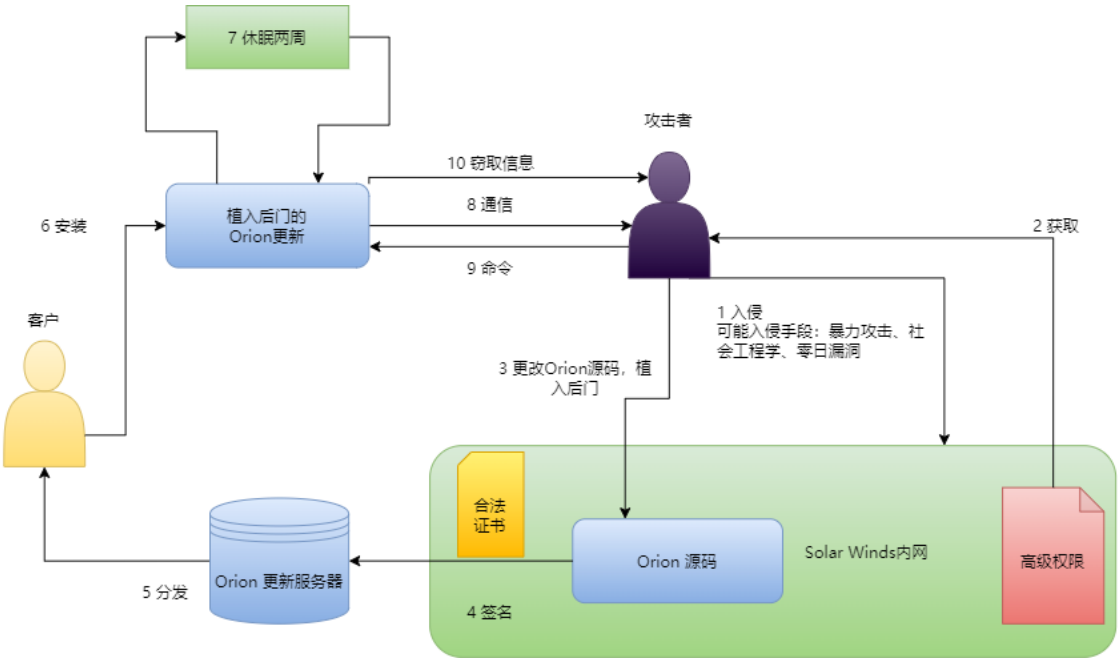


图 7.1　SolarWinds 事件攻击流程图

7.1.2　问题分析

此次攻击事件是一起典型的软件供应链安全事件，攻击者利用了用户对上游企业的信任关系（ATT&CK T1199），通过在恶意植入上游企业的产品及利用上游企业的数字签名，绕过了安全防御机制，造成大量下游企业及用户遭受攻击的严重影响。

实际上，通过攻破上游企业获取目标源代码，或在其中植入恶意代码等的攻击行为早有发生。近年来随着越来越多的企业将基础设施迁移至云端，供应链安全威胁出现了从本地到云端蔓延的趋势。当本地 ADFS 服务器中的凭据被窃取，攻击者可通过签名伪造的 SAML 令牌，伪造任意用户以任意权限访问企业的云端资产（包括 Azure、Vsphere）。这种攻击技术被称为 Golden SAML，在本次事件中首次披露在野利用 [33]。这些云端资产的沦陷扩大了本次供应链攻击事件的影响范围，对代码、数据等重要资产的修改权限为下一轮供应链攻击创造了条件。

[33] https://www.splunk.com/en_us/blog/security/a-golden-saml-journey-solarwinds-continued.html

7.1.3 应对措施

7.1.3.1 上游厂商应加强研发过程安全及软件成分监控管理

供应商有责任做好产品的安全管控、关键信息（如密钥、账号）的保护，建立完善的漏洞和安全事件的披露溯源机制。

对于供应商，假设在代码被感染，打包分发路径中引入 SBOM 自动生成机制，每次更新时都生成该软件最新构建的 SBOM（类比于 github 的版本控制，将 SBOM 看作另一种版本控制机制，自动化、强制性地指出每一处改动），可以辅助排查恶意代码。在构建环节结束后，将 SBOM 与上一版本软件的 SBOM 比对，进行二次筛查，检查是否有计划外的异常改动（比如计划中没有用到的软件包、第三方库，或者计划外文件内容出现更改等等），如果有异常改动，立刻对源码重新筛查；如果没有计划外文件的改动，则在测试环节重点对新增 / 改动的文件进行安全测试（IAST、SAST、SCA 等），筛查其异常行为。通过这些有效措施，代码 / 软件植入的风险能够得到一定的控制。

7.1.3.2 通过传递软件成分清单应对上游风险

应对供应链安全，下游企业和最终用户不应将自身安全寄托在供应商负责程度上，应当以零信任的态度谨慎供应商提供的产品，建立好企业资产管理和供应链产品物料清单管理，结合应急预案，争取在安全风险曝光的第一时间排查影响，压缩攻击入侵时间窗口，降低影响。

7.2 开源软件安全风险 Log4j2 漏洞事件

7.2.1 事件背景及影响

Apache Log4j2 是一个开源基础日志库，作为 Log4j 组件的升级版本广泛应用于软件项目的开发、测试和生产，在 Maven Repository 中被接近 7000 个项目引用。Log4j2 具有“Property Support”特性，该特性在打印日志时支持从配置文件、系统变量、环境变量、线程 Context 以及事件中存在的数据中引用所需的变量到日志中。而在支持该特性的 org.apache.logging.log4j.core.lookup 包中提供了一系列的插件，允许用户从自定义渠道获取属性。而问题就在于包中的 JndiLookup 插件允许用户通过 JNDI 进行变量的检索，但是未对查询地址做好过滤，导致产生 JNDI 注入漏洞，从而造成代码执行、命令执行等风险。

对于开源组件，自身漏洞对整个软件供应链的影响最为直接、隐秘且长久。Log4j2 作为一个堪比标准库的基础日志库，受众极其广泛，也就导致漏洞的影响范围极大，再加上此次

曝出的漏洞利用难度低（默认配置即可利用），在曝光伊始就吸引了整个行业的注意。

7.2.2 攻击流程分析

表 7.2 Log4j2 事件供应链攻击技术与脆弱性

生命周期	安全风险发生阶段	攻击技术
开源安全	开源代码安全	直接或间接包含的漏洞、缺陷代码
	开源组件、框架稳健性	活跃度、维护能力、安全修复能力与应对意识
开发安全	开发环境安全	开发工具污染 开发环境污染 托管平台、代码存储平台污染
	开发过程安全	使用开源组件引入漏洞风险
	编译构建安全	包管理工具安全

Log4j2 漏洞（CVE-2021-44228/CVE-2021-45046）目前常见攻击形式有勒索、挖矿、僵尸网络（以及 DDOS）。现有的在野攻击案例通常攻击链短、攻击手段粗暴直接，但考虑到漏洞曝出时日尚短，不排除以该漏洞为切入点的大型供应链攻击出现的可能。

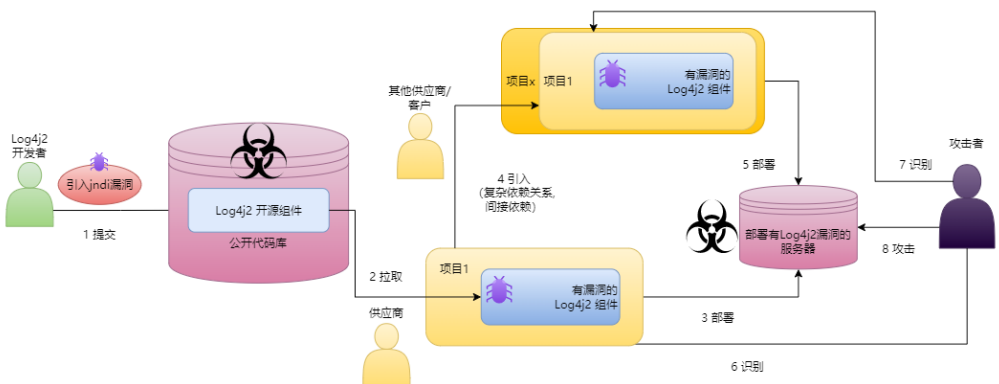


图 7.2 Log4j2 事件潜在攻击流程图

7.2.3 问题分析

Log4j2 漏洞是来自于开源组件的软件供应链威胁。与其他案例相比，Log4j2 漏洞相关攻击事件的目标分布广泛且攻击链极短，究其原因在于该组件极广的应用范围与较低的漏洞利用难度。

企业开源安全管控，既应涵盖直接依赖组件的漏洞，也应关注间接依赖组件漏洞。直接依赖主要发生于组件的集成构建阶段，Log4j2 作为基础组件集成到一些核心业务组件中，核心业务进而被感染，增加了攻击面。此阶段的依赖关系相对清晰，一旦发现漏洞，相对来说

容易排查。

而间接依赖的组件的情况则更为复杂，随着软件系统架构愈加庞大，组件之间依赖关系也更难理清，因而漏洞曝出时系统自身的复杂性就会掩盖影响，导致来自间接依赖的攻击面被忽略。此阶段往往需要进行大规模的分析排查，以抽丝剥茧的态度将各种依赖关系理清。站在攻击、防御的角度也同样如此，通过 hook、fuzz 等方式测试组件的调用深度，定位被隐藏的触发点。

由于次级供应商在单个上下游环节中可以视为消费者，在组件依赖关系不明确的情况下只能将复杂系统看作黑盒，对其包含的组件风险一无所知，因而在处理漏洞时更为被动。此时只能靠有责任心的软件提供商提供运维支持服务。若软件提供商响应不及时或漏洞应急流程不完善，只能依靠社区建议及旁路的安全设备来进行临时缓解。

7.2.4 利用开源软件成分清单识别间接组件漏洞

Log4j2 漏洞之所以影响严重，是因为它在许多项目中以基础设施的身份存在。随着项目结构体积逐渐庞大，引入组件的来源增多，项目结构、依赖关系也越加复杂，也就越难发现隐藏较深的间接组件。相比针对开源机制的攻击手段，比如开发工具污染、依赖混淆等，依靠企业现有技术手段很难规避这类间接组件漏洞风险，重点应放在应急响应上。在漏洞曝出的第一时间，基于开源软件成分清单识别间接组件漏洞，对于快速研判影响、制定决策至关重要。

企业应建立开源组件安全评估与风险响应能力，在代码构建时，通过 SCA 工具对项目的第三方组件依赖进行漏洞分析，规避已知的漏洞风险；同时应留存更新的软件成分清单，当出现安全事件时只需花少量时间、算力，就可以定位漏洞（问题组件）所在，加快排查速度，辅助决策，减少损失。

监管层面，应建立开源组件安全监测管理，涵盖四个方面：外防输入、存量治理、内控扩散、持续监测。外防输入即摒除带有漏洞的软件版本，严格控制外部引入风险；存量治理，即制定差异化的策略，分批有序开展治理；内控扩散，建立组件的灰白黑名单机制，按需管控；持续监测即持续监测漏洞情报，做好应急流程。概括起来就是要做到严格管理软件风险、因地制宜制定策略、居安思危防范于未然这三点。在 Log4j2 事件中，考虑到该组件的基础地位，以及应用范围之广，本应重视其安全监测管理，可惜业界没有保有足够的警惕，既没有制定合适的治理策略，也未安排好应急流程，拖慢了应急响应的步伐，只得自行承担后果。

8

软件供应链安全总结与展望

美国引领了信息化时代的发展，长期主导全球信息技术发展方向，其软件供需模式与供应链管理发展动向对全球软件产业发展具有重要影响。目前，美国正在政府和私营部门合作伙伴之间建立完善软件供应链安全方法论和最佳实践。这将对我国软件产业发展有着巨大的参考价值，结合此次西方阵营针对俄罗斯发动的大规模“断供”，其行为也为我国提供了充足的研究案例，对建立健全我国软件供应链安全治理体系具有重要的参考意义。

附表：

软件供应链安全风险表

软件供应链安全生命周期	安全风险发生阶段	攻击技术
上游安全	上游企业安全问题	内部漏洞
		外部攻击
		关键数据泄露（如证书）
	软件供应链中断	SaaS、PaaS 等服务中断
		软件更新、维护服务中断
开源安全	开源代码安全	直接或间接包含的漏洞、缺陷代码，恶意代码注入，等
	开源使用安全	恶意抢注攻击等
	开源项目维护稳健性	活跃度、维护能力、安全修复能力与应对意识
	知识产权	许可证授权风险
开发安全	开发环境安全	开发工具污染
		开发环境污染
		CI/CD 集成环境污染
		托管平台、代码存储平台污染
	开发过程安全	内部漏洞、缺陷
		编码过程不符合规范
		遭受外部攻击
		使用开源组件引入风险
		恶意代码植入
	编译构建安全	依赖库路径抢注
		编译工具植入后门
		包管理工具安全
交付使用安全	下载过程安全	升级、更新劫持
		捆绑下载
	交付范围扩大	代码、编译信息泄漏
		证书、私钥泄漏
	使用安全	依赖网络基础设施安全
		依赖云平台安全

天元实验室

专注于新型实战化攻防对抗技术研究。

研究目标包括：漏洞利用技术、防御绕过技术、攻击隐匿技术、攻击持久化技术等蓝军技术，以及攻击技战术、攻击框架的研究。涵盖 Web 安全、终端安全、AD 安全、云安全等多个技术领域的攻击技术研究，以及工业互联网、车联网等业务场景的攻击技术研究。通过研究攻击对抗技术，从攻击视角提供识别风险的方法和手段，为威胁对抗提供决策支撑。

天枢实验室

天枢实验室立足数据智能安全前沿研究，一方面运用大数据与人工智能技术提升攻击检测和防护能力，另一方面致力于解决大数据和人工智能发展过程中的安全问题，提升以攻防实战为核心的智能安全能力。

NSFOCUS绿盟科技

扫描绿盟科技官微二维码
可在手机端直接观看报告电子书