软件学报 ISSN 1000-9825, CODEN RUXUEW
2026,37(7):2766−2807 [doi: 10.13328/j.cnki.jos.007589] [CSTR: 32375.14.jos.007589]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

产业研发视角下的开源软件供应链攻击问题研究*

胡 帅[1,2], 王海军[1], 谢继刚[2], 裴鹏飞[2], 阿西伍合[1], 马辰达[1], 刘 烃[1]

[1](西安交通大学 网络空间安全学院, 陕西 西安 710049)

[2](联通西部创新研究院, 陕西 西安 710021)

通信作者: 王海军, E-mail: haijunwang@xjtu.edu.cn

摘 要: 开源软件深度嵌入企业的产品研发与交付流程, 缩短了研发周期、降低了研发成本并增强了系统兼容性; 与此同时, 针对开源软件供应链的攻击事件也呈现上升态势, 已成为软件行业最重大的安全威胁之一. 从产业研发视角分析研发效率与软件安全的内生矛盾, 发现产业组织以流程合规作为效率约束下, 被动应对开源软件供应链安全威胁的隐性共识. 结合实际案例论证了基于流程合规的安全体系无法应对开源软件供应链攻击; 基于产业视角提出了开源软件供应链攻击分类演化框架, 将开源软件供应链攻击分为 3 个阶段: 开源共建威胁产生、闭源研发威胁演化、成品使用攻击发作, 对不同阶段的攻击模式、技术手段等进行总结; 从面向开源生态的协同治理、面向产业研发的持续合规与攻击面收敛、面向成品使用的自适应防护这 3 个方面, 提出开源软件供应链的安全再平衡体系.

关键词: 软件安全; 开源软件供应链; 产业研发视角; 产业研发流程; 开源软件供应链攻击防护

中图法分类号: TP311

中文引用格式: 胡帅, 王海军, 谢继刚, 裴鹏飞, 阿西伍合, 马辰达, 刘烃. 产业研发视角下的开源软件供应链攻击问题研究. 软件学报, 2026, 37(7): 2766–2807. http://www.jos.org.cn/1000-9825/7589.htm

英文引用格式: Hu S, Wang HJ, Xie JG, Pei PF, Axi WH, Ma CD, Liu T. Research on Open-source Software Supply Chain Attacks from Industrial R&D Perspective. Ruan Jian Xue Bao/Journal of Software, 2026, 37(7): 2766–2807 (in Chinese). http://www.jos.org.cn/1000-9825/7589.htm

Research on Open-source Software Supply Chain Attacks from Industrial R&D Perspective

HU Shuai[1,2], WANG Hai-Jun[1], XIE Ji-Gang[2], PEI Peng-Fei[2], AXI Wu-He[1], MA Chen-Da[1], LIU Ting[1]

[1](School of Cyber Science and Engineering, Xi'an Jiaotong University, Xi'an 710049, China)

[2](China Unicom Western Innovation Institute, Xi'an 710021, China)

Abstract: Open-source software is deeply embedded in enterprise product research, development, and delivery processes, shortening development cycles, reducing costs, and enhancing system compatibility. Meanwhile, attacks targeting the open-source software supply chain continue to increase and have become one of the most critical security threats to the software industry. From an industrial research and development (R&D) perspective, this study analyzes the inherent tension between R&D efficiency and software security and identifies an implicit consensus in industrial practice: under efficiency constraints imposed by process compliance, organizations tend to respond passively to open-source software supply chain security threats. Through representative real-world cases, it is demonstrated that security systems primarily driven by process compliance are insufficient to address open-source software supply chain attacks. From an industrial perspective, this study further proposes an evolutionary classification framework of open-source software supply chain attacks, in which attacks are categorized into three stages: threat emergence during open-source co-development, threat evolution during closed-source

* 基金项目: 国家自然科学基金 (62232014, 62372367); 陕西省重点研发计划 (2024GX-ZDCYL-02-19); 陕西省高层次科技人才项目 (QCYRCXM-2022-345)
本文由"智能化基础软件可信构造和供应链安全"专题特约编辑王林章教授、司徒凌云研究员、钟浩研究员、 向剑文教授、张路教授推荐.
收稿时间: 2025-09-08; 修改时间: 2025-10-20; 采用时间: 2025-12-08; jos 在线出版时间: 2025-12-26
CNKI 网络首发时间: 2026-06-02

industrial R&D, and attack manifestation during product deployment and usage. For each stage, typical attack patterns and technical mechanisms are systematically summarized. Based on this analysis, a security rebalancing framework for the open-source software supply chain is proposed from three complementary dimensions: collaborative governance oriented toward the open-source ecosystem, continuous compliance, and attack-surface reduction oriented toward industrial R&D processes, and adaptive protection mechanisms oriented toward deployed products.

Key words: software security; open-source software supply chain; industrial R&D perspective; industrial R&D process; open-source software supply chain attack protection

1 引 言

自 20 世纪 80 年代理查德·斯托曼 (Richard M. Stallman) 发起 GNU 计划以来, 开源软件所倡导的开放共享研发模式逐步演化为信息技术领域的主导性范式. 该计划旨在构建完全自由的软件系统, 不仅开启了全球开源运动的先河, 也为开源理念的制度化演进与工程化实践奠定了理论基础和价值框架[1,2]. 随后, Linux 内核的发布显著推动了开源操作系统的发展进程, 标志着开源技术由学术研究向产业应用的关键转变, 并逐步形成以社区协同、版本共建和源代码开放为核心的技术生态体系. 随着 GitHub、GitLab 等代码托管与协同平台的广泛兴起, 围绕开源项目的开发、构建、测试、集成与发布的全生命周期流程不断演进, 催生出结构日益复杂、覆盖广泛的开源软件供应链系统, 现已成为现代软件工程体系中不可或缺的关键性基础设施[1–7].

根据统计数据, 截至 2024 年, 全球已有超过 96% 的软件开发组织在其研发流程和系统架构中广泛集成了开源项目, 开源组件已成为支撑现代软件工程不可或缺的核心资源[8]. 与此同时, 我国在开源领域的国际影响力持续增强, 已跃升为全球第 3 大开源贡献国. 以阿里巴巴、华为等头部科技企业为代表的本土力量, 不仅广泛采用开源技术, 还积极向 Linux Foundation、Apache Software Foundation 等国际开源社区贡献关键项目, 推动我国在开源领域的话语权与影响力持续提升[9,10].

然而, 开源软件供应链的快速发展也带来了前所未有的安全挑战. 开源生态开放性强、依赖链复杂、用户基础广泛, 为攻击者提供了隐蔽且破坏力强的攻击向量. 近年来, 围绕开源组件的软件供应链攻击事件频发, 已成为全球范围内高影响、高传播、高危害的系统性安全问题[11–13]. 典型事件如 XZ Utils 后门、 CrowdStrike 更新蓝屏等, 不仅造成大规模服务中断和经济损失, 更暴露出版本控制、依赖管理和审计机制上的系统性薄弱[14–16]. 典型的软件供应链攻击技术路径可追溯至 2003 年, Levy[17]率先指出, 通过篡改开发与分发链条, 可在无感知前提下将恶意代码注入用户的软件产品系统. 随着开源技术呈现全球化协作、产业化采纳的特征, 攻击者逐渐演化出“代码即武器”的攻击范式: 在合法开源项目中隐匿后门, 通过自动化工具链实现恶意代码的编译与分发; 在依赖树中埋藏钓鱼组件以诱导下游集成; 在数据管道中嵌入指挥与控制 (command and control, C2) 信道以实施远程控制与数据窃取[5,18]. SolarWinds 攻击事件即为典型案例, 其攻击链深入持续集成与持续交付 (continuous integration/continuous delivery, CI/CD) 流程核心节点, 通过污染构建系统成功劫持软件制品分发渠道, 最终波及大量政府与企业用户, 危害程度空前[3,19].

为应对日益严峻的软件供应链攻击威胁, 学术界与工程实践正从多维视角构建系统化防御机制. 其中, 软件成分分析 (software composition analysis, SCA)[20]作为核心技术, 广泛应用于开源组件的依赖溯源、许可证合规性审查与漏洞识别, 已成为供应链安全治理的重要基础. SCA 工具支持对依赖关系的全量扫描与动态监测, 能够有效识别潜在风险路径及其传播链路. 与此同时, 漏洞治理体系不断演进, 自动化检测与修复框架日趋完善[21–25], 结合语义补丁生成与版本感知机制, 显著提升了漏洞响应的及时性与有效性. 随着预训练大模型技术的快速发展, 其在软件供应链安全中的赋能效应持续显现. 依托对代码语义的深层理解, 大模型不仅可实现复杂依赖结构的自动建模与异常行为的智能识别[26,27], 亦可模拟攻击行为、评估系统防御能力, 并生成具针对性的防御策略[28–32], 为构建兼具高可解释性与高准确性的智能化供应链安全分析体系奠定坚实基础, 推动防御技术迈向更高发展阶段.

尽管已有研究从技术原理、攻击类型及分类体系等角度对软件供应链攻击进行了系统归纳与详细的事件分析[2,12,33–35], 但相关工作普遍对产业实践关注不足, 缺乏从实际研发流程出发的全景式系统研究. 现代产业软件研

发以 DevOps 为核心的高度集成模式为主导[36], 通过敏捷迭代、持续集成/持续部署与持续交付及全生命周期闭环实践, 实现开发与运维的深度协同, 显著提升研发效率、系统兼容性与用户响应速度. 同时, 高度自动化的流水线、紧密集成的工具链及对外部组件的广泛依赖, 也显著扩大了潜在攻击面: 关键环节如依赖引入、构建过程、工具链集成及部署阶段的正常研发流程均可能被利用成为攻击目标. 一旦攻击者在早期阶段 (如依赖获取或构建) 植入恶意逻辑, 即可利用自动化 CI/CD 流水线高效传播, 并在生产环境中隐蔽触发, 形成典型的软件供应链攻击路径. 这表明, 产业研发流程自身的高效率和高度自动化虽然“强化”了研发能力, 却同时天然衍生出“自我反噬”的系统性风险.

产业界以规模化研发组织、自动化研发流水线工具平台、敏捷化交付流程为典型特征来适应市场需求价值的快速交付. 研发效能提升和系统上线后的容器动态伸缩与扩缩容使得传统的研发阶段的“动”和交付运维阶段的“稳”在当下变得界限模糊. 当安全管理手段并没有产生与之匹配的变化时[10,37], 在研发阶段产业组织一般通过代码扫描、物料治理、系统渗透测试保障系统安全, 但在系统上线后不得不应对系统快速升级、频繁变动下层出不穷的新漏洞. 在价值交付优先的压力下, 研发组织不得不在有限时间内、既定安全流程和平台上进行既定的测试自证系统安全, 形成了以流程合规保障软件研发安全的产业研发管理隐性共识. 在此共识的制约下, 利用开源软件与自动化技术提升研发效能以及利用相同的研发链路进行攻击渗透并放大破坏性之间形成了一体两面无法解决的内生矛盾. 软件供应链攻击正是“看碟下菜”, 针对已有的研发交付模式与工具, 采用复杂的手段将代码、依赖包等带菌组件污染正常的研发交付流程, 从而造成破坏. 因此相关攻击事件层出不穷, 众多的新技术无法有效解决现有问题.

为系统地刻画产业研发视角下的开源软件供应链风险, 本文提出涵盖开源共建威胁产生、产业闭源研发攻击演化传播及成品使用发作的三阶段攻防演化框架. 在研发效能不断强化的背景下, 软件供应链暴露出“自我反噬”的安全威胁, 即攻击者采取复杂的手段针对已有的研发范式开展攻击. 开源共建生态阶段涉及社会工程学攻击、恶意项目伪装、代码后门植入及大模型数据投毒; 产业闭源研发阶段攻击者可以利用复杂的软件依赖引入漏洞或恶意包、污染构建系统、篡改工具链及分发渠道, 以及大模型驱动的持续集成攻击; 在下游成品使用终端阶段攻击者可能采取动态初始化攻击链、分阶段载荷释放及隐蔽 C2 通道控制等手段达到隐匿的攻击目的. 尽管当前产业界已在多个环节初步建立防护措施, 但仍无法有效解决软件供应链安全研发的内生矛盾, 本文提出安全再平衡的理念为新矛盾下软件供应链安全体系的重建提供思路. 开源生态共建阶段可通过开源项目安全治理体系、代码分析与检测技术及大模型数据安全治理实现源头防护; 产业内源研发阶段依托安全需求工程、DevSecOps 实践及内源研发机制实现构建安全内生化; 下游成品使用终端阶段则依靠运行时应用自我防护、安全漏洞修复及大模型驱动的供应链安全治理构建纵深防线.

本文以产业研发流程为分析切入点, 构建了“开源共建威胁产生-产业闭源研发攻击演化传播-成品使用攻击发作”三阶段软件供应链攻防分析范式, 将开源共建生态、产业闭源 CI/CD 流水线研发生态及成品终端运行环境纳入统一分析框架, 此框架场景丰富, 系统呈现各阶段典型攻防态势, 并深刻揭示了效能“强化”下, 软件供应链安全威胁“自我反噬”的内生矛盾. 在此内生矛盾的挤压下, 倒逼产业研发组织形成了以研发流程合规, 来被动适应层出不穷的安全威胁的隐性共识, 揭示复杂攻击演化规律及防御策略设计思路. 通过构建以产业研发视角为导向的分析范式, 本文不仅弥补了现有研究对产业研发实践关注不足的空白, 也为学术研究与工程实践建立紧密联系提供方法支撑, 为理解软件供应链全景攻防机制、优化安全策略及提升产业系统韧性提供理论与实践依据.

2 研究方法

本文采用“系统性文献调研-产业研讨与实践-问题分析与总结”三阶段研究方法, 如图 1 所示. 围绕产业研发视角下的开源软件供应链攻击问题展开研究. 在系统性文献调研阶段, 系统检索并筛选了相关学术文献, 同时广泛收集来自主流安全社区等渠道的白皮书、技术报告与新闻等产业资料, 并结合产业专家经验对产业文献进行二次筛选. 在产业研讨与实践阶段, 对前期调研内容开展专家研讨, 并对典型安全事件进行深入分析与复盘, 在研发组织

内进行了开源软件供应链攻击与防护的产业研讨与实践. 在第 3 阶段, 系统性地开展问题分析与总结, 从产业研发流程与开源软件供应链攻击的协同关系、产业研发视角下的攻击分类与演化路径、基于产业研发需求的供应链安全再平衡机制这 3 个维度, 对开源软件供应链攻击问题进行全面剖析与总结.

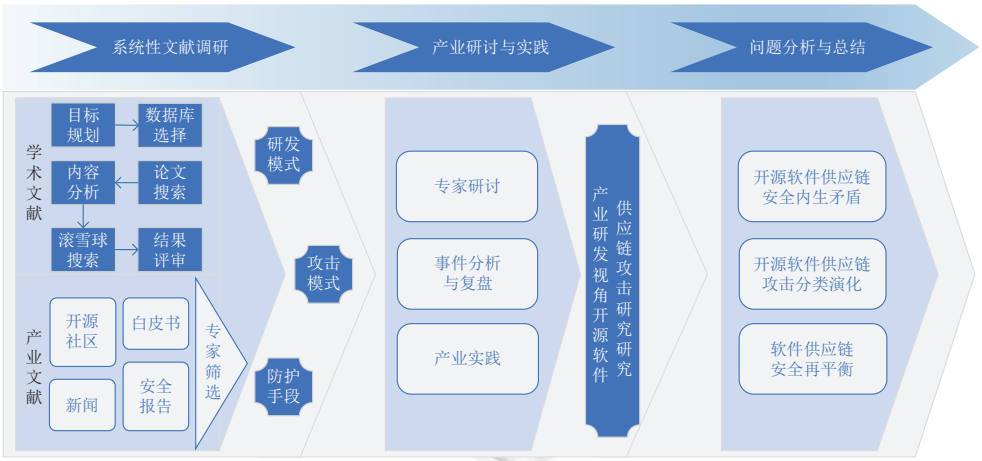

图 1 研究方法框架模型

2.1 系统性的文献调研

学术文献主要聚焦于对安全问题内在机理与防护模型进行深度理论剖析, 为本文奠定了坚实的理论基础; 而产业文献则更侧重于反映行业演进态势、解析典型安全事件及分享工程最佳实践. 为确保研究的全面性与前瞻性, 本文系统性地整合了如下多源文献资料.

● 学术数据库资源. 广泛检索了包括中国知网 (CNKI)、万方数据、IEEE Xplore、ACM Digital Library、ScienceDirect、SpringerLink 及 DBLP 在内的主流学术数据库. 这些平台汇集了经过严格同行评议的高质量学术论文与学位论文, 为本文提供了坚实的理论基础与研究前沿支撑.

● 学术搜索引擎与开放平台. 利用 Google Scholar、百度学术、Bing Academic、ResearchGate 及 arXiv 等在线平台, 高效追踪并获取了广泛的文献资源. 这些工具不仅提供了便捷的文献检索途径, 也涵盖了诸多领域的最新预印本与研究动态, 为课题提供了及时的信息更新.

● 专业书籍与专著. 从实体图书馆、在线书店及 Springer、Wiley、Cambridge University Press 等权威学术出版社的系统检索中, 筛选出与本课题高度相关的专业书籍. 这些著作为本文提供了相关领域系统性的理论框架与经典研究方法.

● 专业机构报告. 重点参考了国家相关部门、国际组织、行业协会及知名研究机构 (如中国信通院) 发布的权威报告、白皮书等公开文件. 此类资料具有显著的政策指导意义与行业共识价值, 为分析提供了关键的宏观背景与产业洞察.

● 其他补充资料. 此外, 本文还涵盖了 GitHub、悬镜安全中心、Ubuntu 等国内外主流开源社区的安全公告, 以及专业安全情报机构的分析报告与相关新闻资讯. 这些来源为还原真实安全事件的发展脉络与深入分析提供了宝贵的实践细节与情境信息.

通过上述多源信息的交叉验证与整合, 本文构建了一个全面、立体的开源软件供应链攻击研究资料体系. 在学术论文方面, 首先以“CI/CD 研发流程安全”“开源软件供应链安全”“开源软件供应链攻击”“软件供应链投毒攻击”等关键词进行初步检索, 筛选出 30 篇核心文献. 随后, 以此批文献为起点, 通过追踪其参考文献 (向前追溯) 及被引文献 (向后追溯) 进行迭代扩展, 共获取 398 篇相关文献. 经过严格的相关性评估与去重处理, 最终选定 144 篇高相关性文献. 在产业文献与工具的调研方面, 本文初步搜集了 206 篇产业文献. 随后, 依托作者所在超过 300 人的专业化研发组织, 对产业文献的正确性 (结论可靠、数据真实)、先进性 (代表当前技术趋势) 与可复现性

(方案具体、具备实践条件) 进行了严格研讨与评估, 最终采纳 67 份高质量产业公开资料 (含脚注所列内容). 具体统计情况如表 1 所示.

表 1 相关文献的数量统计

分类	学术论文	产业文献	小计
产业研发分析	60	20	80
攻击分类演化	31	37	68
安全再平衡	53	10	63
总计	144	67	211

2.2 产业研讨与实践

本文还设立了一个由 19 名具备 10 年以上产业经验的技术专家组成的技术委员会, 成员涵盖项目经理、架构师、高级开发人员及质量保证人员. 该组织每年成功完成上百个版本的迭代或项目交付, 在软件研发与系统集成方面具备扎实的产业基础. 此外, 本文还邀请了 9 位学界专家共同参与, 与产业专家一起组成本文的专家团队.

基于上述组织背景, 本文系统性地开展了产业研讨与实践. 技术委员会对前期系统调研所得的学术与产业文献组织了多轮研讨, 并对典型开源软件供应链攻击事件进行了深入分析与复盘. 在此基础上, 结合组织内部的实际研发流程, 研究首先识别出研发流程中存在的安全薄弱环节, 进而剖析产业研发视角下开源软件供应链安全的内生矛盾, 系统性地梳理了相关攻击模式与防护方法, 并对多种防护工具开展了系统化评测, 从而为开源软件供应链安全问题提供了基于真实场景的产业视角.

作为研讨成果的落地实践, 本文设计并实现了开源软件供应链治理平台. 该平台集成软件成分分析、漏洞扫描与修复、源码治理与架构一致性检查等核心能力, 并将其嵌入至组织的 CI/CD 流水线中. 在关键研发节点通过 Webhook 机制自动触发供应链安全扫描、策略干预与优化动作, 显著提升了研发过程的安全性与合规水平. 自 2021 年至今, 该治理平台及相关方法已为该集团内部 31 家分公司、2 200 余个应用系统提供软件成分分析超过 15 万次, 成功闭环修复研发阶段漏洞 3 万余个. 同时, 系统累计协助漏洞治理与攻防演练团队识别生产环境漏洞 60 万余个, 有效提升了组织整体的安全防御能力. 上述实践为开源软件供应链攻击问题的分析与总结, 提供了产业实践视角与产业实证基础.

2.3 问题分析与总结

基于系统的产业研讨与实践, 本文从产业研发视角对开源软件供应链攻击问题进行了深入分析与总结, 形成了理论探索与工程实践相结合的特色研究路径. 具体而言, 本文从产业研发流程与开源软件供应链的协同演化关系出发, 识别出研发过程中存在的流程合规保障研发安全的隐性共识, 并据此提炼出开源软件供应链安全的内生矛盾模型. 在这一共识约束下, 研发范式与工具为追求效能提升而持续“自我强化”, 而其依赖的开源软件供应链却同步呈现出“自我反噬”的安全风险. 在流程合规的约束下, 研发流程的“自我强化”与开源软件供应链安全威胁的“自我反噬”构成了核心内生矛盾, 攻击者通过系统性地利用这一矛盾, 实施开源软件供应链的攻击.

在此基础上, 本文构建了一个基于产业研发视角的开源软件供应链攻击与安全再平衡的分类与演化框架. 该框架遵循“引入-传播-发作”三阶段演进路径, 系统性地刻画了开源软件供应链攻击与安全再平衡在不同阶段的典型技术与模式. 在攻击层面, 在开源共建威胁产生阶段, 主要表现为开源生态中的社会工程学攻击、仿冒流行开源项目、代码投毒与后门植入、大模型训练数据污染等威胁形式; 在产业闭源研发攻击演化传播阶段, 主要涵盖产业闭源研发过程中的源码篡改、构建污染、CI/CD 流水线工具链攻击、分发渠道信任滥用、大模型辅助的持续集成攻击; 在成品使用攻击发作阶段, 攻击最终表现为终端攻击链动态初始化、多阶段载荷释放以规避检测、隐蔽 C2 通道构建等完整攻击链.

进一步地, 在防护层面, 本文提出基于产业研发流程的软件供应链安全再平衡机制, 围绕上述 3 个阶段分别构建相应的防护体系: 在开源共建阶段, 进行协同治理与代码异常检测; 在产业闭源研发阶段, 实施持续合规与攻击面收敛策略; 在成品使用阶段, 则采用动态自适应防护技术. 这一分析框架完整揭示了软件供应链攻击从开源社区

感染、经产业研发环节传播、到最终用户爆发的全链路关键问题, 为系统性地理解与应对此类威胁提供了整体性视角, 对本文的体系化展开起到了纲领性指导作用.

3 产业研发流程和开源软件供应链攻击

3.1 产业研发流程和开源软件供应链的协同演化

1990 年, IEEE 在《软件工程术语》标准中将软件工程定义为“运用系统的、规范的、可度量的方法来开发、运行和维护软件”, 明确突出了软件工程活动的科学性与工程化特征. 基于该定义所蕴含的核心理念, 学术界与工业界普遍将过程、方法、工具归纳为软件工程的 3 大基本构成要素. 过去 30 年间, 随着软件工程理论与实践的持续演进, 产业界在研发范式、工具平台及协同机制等方面经历了多次深刻转型. 从早期的瀑布模型, 到强调快速迭代的敏捷模型, 再到自动化交付的 CI/CD, 直至当下以平台化与云原生为特征的 DevOps 模式, 软件工程持续应对数字化时代下高频变更与大规模协同的挑战. 在此变革进程中, 开源软件作为关键推动力量, 在方法论与工具体系层面深度融合, 重塑了产业研发体系的演化路径. 本文将结合典型演化阶段, 剖析开源软件对产业研发范式演进与效率跃升的核心驱动作用.

现代软件工程实践的演进历程中, 敏捷开发 (Agile)、CI/CD 以及 DevOps 等关键范式, 共同构成了产业级研发效能提升的核心支柱, 呈现出一条从方法论创新到工程化实践的清晰演进路径. Agile 作为一种文化范式与项目管理方法论, 其核心在于倡导迭代式增量开发、紧密的跨职能协作与持续的用户反馈循环, 旨在通过“小步快跑”的方式显著提升团队响应需求变化的灵活性与软件交付速度. 随着敏捷理念的广泛采纳, 其对频繁、可靠集成与交付的内在需求, 直接催生了 CI 的工程实践. CI 通过引入高度自动化的构建、测试与验证流程, 倡导开发者频繁地提交代码变更、快速集成, 从而在早期发现集成错误, 保障主干代码的持续可用性与系统稳定性. 在 CI 成功建立代码集成信心的基础上, CD 进一步将自动化边界延伸至软件发布的最后阶段. CD 实践致力于确保任何通过验证的代码变更都能以高效、安全且可重复的方式, 随时准备或直接部署到预发布乃至生产环境, 显著缩短了从代码提交到价值交付的周期. 作为这一演进路径的集大成者以及更高层次的整合范式, DevOps 超越了单纯的技术实践范畴, 旨在系统性地打破传统开发与运维职能间的壁垒. 它通过深度融合敏捷协作理念、CI/CD 自动化流水线以及共享的工具链与平台化能力, 构建起端到端 (从代码提交到生产运维) 的一体化协同文化与技术生态, 最终实现软件交付生命周期的高度自动化、可观测性与闭环反馈优化.

开源软件供应链的蓬勃兴起, 不仅深刻影响了现代软件工程范式的演进轨迹, 更与敏捷开发、CI/CD 实践及 DevOps 文化形成了深度的双向塑造与协同进化关系, 共同推动了软件研发效能质的飞跃. 首先, 开源组件的大规模复用直接加速了敏捷开发的迭代效率. 开源社区提供的丰富模块化资源 (如前端框架 Vue.js、容器引擎 Docker) 使开发团队能够快速集成成熟功能模块, 避免重复开发[38]. 这种“按需取用”的特性与敏捷模型强调的“小步快跑、快速响应”高度契合, 推动开发周期从数月缩短至数周. 同时, 开源工具链 (如 Git、Jenkins) 为 CI 提供了技术基础. 开发者通过版本控制系统实现分布式协作, 而自动化构建、测试工具则保障了代码频繁集成的稳定性, 使敏捷迭代具备可落地的工程支撑. 例如, 开源社区推动的 CI/CD 实践 (如 GitHub 的 Pull Request 机制) 将代码审查、自动化测试嵌入开发流程, 显著提升了代码质量与交付速度. 其次, 开源生态的协作机制与安全需求进一步催化了 DevOps 的成熟演化. 一方面, 开源社区倡导的透明协作文化与 DevOps 的跨职能协同理念深度融合. 社区通过文档共享、在线论坛促进开发 (Dev)、测试 (test)、运维 (Ops) 角色的信息同步, 而 DevOps 工具链 (如 Kubernetes、 Prometheus) 则依托开源技术实现部署自动化和环境隔离, 缩短了从代码提交到生产上线的路径. 另一方面, 开源软件供应链的依赖复杂性与安全风险推动催生了“安全左移”理念在 DevSecOps 中的落地实践. 软件物料清单 (software bill of materials, SBOM)[39]和开源漏洞扫描工具被集成至 CI/CD 流水线, 实现开源组件的持续安全检测与合规管理.

总的来看, 开源软件供应链与产业研发范式之间呈现出“工具与文化供给”与“资源与需求引领”的双向赋能格局: 开源软件供应链为敏捷开发模型到 DevOps 的进阶提供了工具链和文化基础, 而产业研发通过参与开源社区建设、开源项目研发, 将自身的需求实现在开源项目中并推广使用, 从而推动开源产业持续发展. 二者协同推动软

件工程向标准、高效、自动化与可持续方向发展, 也为后续的软件供应链安全挑战与防御体系的研究提供了现实基础与工程语境.

3.2 产业研发中的隐性共识

产业研发视角本质上是从软件生产的角度出发, 将软件生产视为研发者共同参与、基于统一契约并有组织地持续迭代交付物的过程, 这与理论研究视角中侧重问题解构的路径存在差异. 为深入探究产业研发过程中的安全问题, 本文调研了近年来关于主流研发流程安全的相关研究, 涵盖 Agile[40–42]、CI/CD[43–45]及 DevOps[40–58], 从中分析与提取了这些研究中所指出的安全挑战. 尽管 Agile、CI/CD 与 DevOps 在具体侧重上有所不同, 但这些研发流程均可以归纳为 3 个核心要素: 研发参与者 (人)、研发管理 (事)、研发技术 (物). 本文针对 2021 年以来所积累的 50 余个典型案例, 经过 3 轮研讨与评审, 归纳出研发流程中 3 个要素所面临的代表性安全防护挑战. 表 2 系统化地凝练了产业研发实践中的核心安全脆弱环节, 并通过对比分析量化了 Agile、CI/CD 与 DevOps 范式在各核心要素上脆弱性的显现程度差异.

表 2 产业软件研发过程中识别出的安全防护挑战

领域	ID	特征	Agile	CI	CD	DevOps
人 (参与者)	STHD-01	开发人员缺乏安全知识	●	○	○	◐
	STHD-02	安全文化缺失	●	○	○	●
	STHD-03	开发人员对安全的抵触	●	○	○	◐
	STHD-04	交付压力下, 引入的代码安全缺陷	●	○	○	◐
	STHD-05	工作中断, 开发人员抵触安全测试	◐	◐	◐	◐
	STHD-06	安全与开发团队之间的紧张关系	◐	○	○	◐
	STHD-07	成员和工具众多, 身份和访问控制复杂	○	●	●	●
	STHD-08	系统复杂导致开发人员安全认知负荷过高	○	●	●	●
	STHD-09	自动化工具安全误报, 开发人员信任度下降研发	○	●	●	◐
	STHD-10	流程与安全治理模式的冲突被人为利用	●	◐	◐	◐
事 (管理)	PRCS-01	缺乏明确的安全定义和流程	●	◐	◐	●
	PRCS-02	安全活动和文档审查之间的冲突	●	●	●	◐
	PRCS-03	安全需求的来源混乱和管理标准不一致	●	○	○	●
	PRCS-04	规划和预测, 容忍度不足	●	◐	◐	◐
	PRCS-05	方法的灵活性与安全需求的冲突	●	◐	◐	●
	PRCS-06	大规模团队协调困难	●	○	○	◐
	PRCS-07	缺乏足够的时间和成本, 支撑安全治理	●	○	○	◐
	PRCS-08	快速交付导致测试不完备和技术债	●	●	●	●
	PRCS-09	整合安全测试困难	●	◐	◐	●
	PRCS-10	安全措施的碎片化	●	◐	◐	○
	PRCS-11	流程自动化不完备, 人为的安全干预导致错误	◐	◐	◐	◐
	PRCS-12	自动化部署与安全审查的冲突	○	○	●	●
	PRCS-13	过度一体化和资源共享带来的数据和信息泄露	○	●	●	●
	PRCS-14	安全评估滞后, 增加修复成本和时间	●	◐	◐	◐
物 (技术)	ATFCT-01	软件的依赖性修复一个漏洞也许会引入新的漏洞	●	◐	◐	●
	ATFCT-02	频繁的代码部署和基础设施变更导致攻击面被扩大	●	●	●	●
	ATFCT-03	数据安全和隐私风险, 数据的频繁流动和共享导致数据泄露风险	●	●	●	●
	ATFCT-04	传统安全工具不适合快速自动化构建过程	○	◐	◐	○
	ATFCT-05	传统安全工具未能覆盖现代软件架构 (如微服务、无服务器架构)的完整数据流	◐	◐	◐	◐
	ATFCT-06	云原生环境中容器和微服务的动态性和短暂性使得漏洞管理更加复杂	◐	◐	●	◐
	ATFCT-07	分布式系统和云原生架构的复杂性, 安全管理越来越具有挑战性	◐	◐	◐	●

注: ○为不相关; ◐为弱相关; ●为强相关

“人” (研发参与者) 维度从个体行为与组织互动的视角, 阐释了研发流程范式可能面临的安全威胁, 其挑战贯穿从个体开发者 (安全知识、意识) 到团队协作 (沟通、信任), 直至系统层面 (工具链复杂性、治理模式) 的递进结构 (详见表 2). 实践中, 开发人员普遍存在系统性安全知识匮乏 (STHD-01), 对常见漏洞、依赖管理及版本控制风险的认知不足尤为突出. 此缺陷显著削弱了其识别软件供应链攻击 (如恶意依赖注入) 的能力, 具体表现为无法准确评估第三方依赖的安全性, 增加了引入恶意包或利用已知漏洞组件的风险. 若组织层面安全文化薄弱, 将安全视为次要目标 (STHD-02), 则导致安全被边缘化、资源投入不足, 根本性弱化整体防护体系. 部分团队中, 开发人员对集成安全流程存在抵触情绪 (STHD-03), 视其为发布进度的阻碍, 这在高压项目中尤为突出. 高压情境易诱使开发人员有意忽略安全检查或仓促引入未经充分验证的代码/依赖, 从而直接引入安全缺陷 (STHD-04), 为恶意代码注入或篡改合法组件创造了机会. 未能与开发工作流无缝集成的安全测试 (STHD-05) 易破坏开发连续性, 加剧抵触心理. 安全与开发团队间的沟通障碍及信任缺失 (STHD-06) 则进一步制约协作效率并阻碍安全措施的有效实施. 大型研发环境中, 人员众多与工具链复杂化加剧了身份识别与访问控制挑战 (STHD-07). 权限管理不善极易导致开发账户或构建环境凭证泄露、滥用, 或被用于非法上传/篡改组件, 成为攻击者渗透供应链的关键跳板. 系统复杂性亦增加了开发人员的认知负担 (STHD-08), 其不断扩展的安全边界常超出既有知识体系与工具支持能力, 导致问题恶化. 自动化安全工具 (尤其是集成于 CI/CD 流水线中的工具) 的误报问题 (STHD-09) 会削弱开发人员信任, 干扰开发节奏, 降低工具实用效能. 此外, 研发流程 (尤其是强调快速响应与迭代的敏捷开发模式) 与传统集中审批式安全治理机制之间的冲突被人为利用 (STHD-10), 这一矛盾可能被蓄意利用, 从而引发测试逃逸等风险. 例如, 卡内基梅隆大学软件工程研究所 (Software Engineering Institute of Carnegie Mellon University, SEI) 发布的产业案例研究[59]指出, 敏捷开发过程中常预设测试覆盖率阈值, 例如 80% (中国 GB/T 25000.51-2025 标准中要求企业类型的项目, 核心模块覆盖率不得低于 90%, 普通模块覆盖率不低于 80%). 在实际交付项目中, 外包方为满足甲方的合格率要求, 可能策略性地将易于通过的测试纳入达标范围, 而将复杂模块滞留于未覆盖的 20% 中, 以此方法来通过测试要求. 随着待办清单的持续积累, 相关安全问题可能被长期搁置甚至埋没, 最终为项目交付遗留技术债与安全隐患. 若甲方能够借助 CI/CD 自动化工具尽早介入, 实时查看测试规划与具体内容, 则可在相当程度上缓解此类风险. 该问题是在产业实践中以流程合规保障软件研发安全的隐性共识制约下所衍生的典型安全治理困境, 具有重要的研究意义与实践代表性.

“事” (研发管理) 维度聚焦于软件研发流程中的安全治理挑战, 涵盖流程设计、测试整合、资源配置及组织协同等关键环节, 其挑战呈现从基础流程规范 (定义、需求、规划) 到执行与协作机制 (协调、资源、测试实施), 直至系统集成与治理 (自动化、环境、评估时机) 的递进结构 (详见表 2). 实践中, 缺乏明确的安全流程定义与标准 (PRCS-01) 导致角色间理解错位与执行混乱, 削弱了对恶意依赖识别等软件供应链攻击关键防护的协同基础. 安全活动与文档审查流程脱节 (PRCS-02) 在快速迭代中易引发审计与溯源风险, 阻碍了供应链篡改事件的及时根因分析. 安全需求来源混杂且管理标准不一 (PRCS-03) 显著增加统一治理难度, 致使依赖验证等 SCA 控制策略难以有效落地. 研发流程规划常与可接受风险容忍度失衡 (PRCS-04), 可能默许引入高风险依赖, 间接扩大 SCA 攻击面. 研发方法的灵活性与安全控制需求间的固有冲突 (PRCS-05)(如敏捷变更性与安全规范性) 进一步加剧管理矛盾. 团队规模扩大或引入外部供应商时, 安全协调与责任界定困难 (PRCS-06) 尤为凸显. 资源投入 (时间、经费) 不足 (PRCS-07), 尤其在小型团队, 根本性制约安全实践, 致其常被优先级挤压. 快速交付节奏易诱发测试缺失与技术债累积 (PRCS-08), 显著推高后期修复成本, 并因缺乏持续依赖扫描而增加未检出供应链漏洞的风险. 安全测试的整合复杂性 (PRCS-09)(尤其在 DevOps 多变环境下) 阻碍其流程化执行. 安全措施碎片化 (PRCS-10) 导致缺乏端到端管理. 流程自动化若不完备仍依赖人工干预 (PRCS-11), 易引入新风险. 自动化部署流程与安全审查机制未有效结合 (PRCS-12) 极易导致未经审查代码上线, 直接放大恶意代码注入或篡改组件在供应链中传播的风险. 系统过度一体化与资源共享 (PRCS-13) 虽提升效率, 却加剧权限模糊场景下的数据泄露威胁, 为攻击者横向移动提供便利. 最后, 安全评估过度集中于开发后期 (PRCS-14) 具有显著滞后性, 大幅增加修复成本.

“物” (研发技术) 维度聚焦于产业软件研发的交付物、技术基础设施与工具链所面临的安全挑战, 其挑战源于技术工具与快速演进的研发范式 (如自动化、分布式、云原生) 的适配性缺口, 并体现在依赖管理、攻击面控制、

数据保护、工具集成及架构适应性等关键方面 (详见表 2). 实践中, 现代软件对第三方组件的深度依赖导致依赖关系高度复杂化, 漏洞修复过程极易引入新漏洞 (ATFCT-01), 尤其在缺乏自动化兼容性验证体系时, 此类连锁风险难以根除. 频繁的代码部署与基础设施更迭动态扩展了系统攻击面 (ATFCT-02), 显著增加了攻击者可利用的初始渗透点. 开发过程中跨服务数据流的细粒度追踪不足 (ATFCT-03) 带来隐私泄露与滥用风险, 传统工具对此类动态交互监管能力有限. 传统安全工具与现代 DevOps 流水线兼容性不足 (ATFCT-04), 难以适应快速迭代与自动化构建流程, 常导致检测延迟或误报频发, 削弱防护时效性. 现有工具对现代架构 (如微服务、无服务器) 的完整数据流监控支持有限 (ATFCT-05), 其分散的数据路径使传统检测技术失效. 云原生环境下容器与微服务的动态性及短生命周期特性加剧了漏洞管理难度 (ATFCT-06), 尤其在 CI/CD 场景中, 组件频繁重建与销毁导致安全策略失效, 易遗留未修补漏洞或引入恶意镜像. 分布式系统与云原生架构的快速发展大幅提升了整体安全管理复杂度 (ATFCT-07), 其高度分布、灵活部署及动态依赖的特性对安全工具的实时性、可扩展性及跨平台兼容性构成严峻挑战, 致使统一配置管理及策略同步异常困难.

总之, 虽然研发管理过程和研发工具已经持续进化, 研发效能已经大幅提升. 但在市场价值交付优先的驱动下, 以及面对产业软件不断迭代下待解决的技术栈时, 研发团队往往面临巨大的版本交付压力. 在此背景下, 每一次的发版都是在时间、功能、安全性等问题之间的妥协. 功能测试与安全测试一般重点围绕新功能开展, 而原有功能通过回归测试完成, 只要达到企业设定的安全质量阈值就可以发布. 此外, 系统上线后的容器动态伸缩与扩缩容使得传统的研发阶段的“动”和交付运维阶段的“稳”在当下变得界限模糊, 测试团队无法完全模拟系统上线后的动态行为, 造成了诸多的测试逃逸. 虽然流程合规无可厚非, 但在当下软件供应链攻击频发的背景下, 产业界并没有在安全管理方面形成与时代相匹配的新共识和新手段, 研发组织不得不在有限时间内、既定安全流程和平台上进行有限的测试来自证系统安全, 形成了以流程合规保障软件研发安全的研发管理隐性共识. 造成了当下安全治理失效, 软件供应链攻击频发的窘境.

3.3 产业研发视角下开源软件供应链安全内生矛盾

图 2 上半部分刻画了当前产业软件研发的主流范式: 以 DevOps 为核心的高度自动化、集成化的研发模式. 该模式以实现开发 (Dev) 与运维 (Ops) 的深度融合为宗旨, 有机整合敏捷开发理念 (如冲刺迭代、产品备忘录与回顾) 与 CI/CD 工程实践, 构建了覆盖编码、构建、测试、部署及运维的全生命周期闭环.

产业实践以敏捷迭代机制 (“规划-开发-交付-迭代”) 驱动用户需求导向的快速响应与持续优化. 高度自动化的 CI/CD 流水线构成此体系的技术基石: 在开发阶段, 持续集成平台实现编码、构建并运行构建验证测试 (build verification test, BVT) 活动的高效联动; 在交付阶段, 持续交付流程 (涵盖预发环境部署、系统验证测试 (system verification test, SVT)、生产环境部署及冒烟测试) 保障构建产物的可靠交付与快速反馈. 同时, DevOps、云原生、微服务、 基础设施即代码 (infrastructure as code, IaC) 等范式把“构建-测试-部署-监控”的闭环压缩到分钟级, 开发者通过自动化流水线、共享仓库、自动化配置, 获得了“一次编写、全球分发”“一键回滚、秒级扩容”的研发超能力.

然而, 研发体系与平台的高度的自动化、开放性将“攻击载荷的构建-投递-持久化-扩散”的研发攻击链路压缩到同等量级, 一次成功的账户接管即可通过 CI/CD 令牌横向移动, 在数分钟内把恶意代码注入数千个 Pod 中. 依赖管理的自动化让恶意包一旦进入镜像中, 就被镜像仓库的全球 CDN 迅速复制. IaC 模板被篡改后, 所有新创建的基础设施天生带后门. 最终在生产环境中隐蔽触发, 形成典型的软件供应链攻击路径. 图 2 下半部分系统建模了此类攻击在 DevOps 流程中的威胁渗透、扩散与执行过程, 抽象为开源共建威胁产生、产业闭源研发攻击演化传播、成品使用攻击发作这 3 个阶段, 贯穿开源生态、产业生态至用户生态. 由此可见, 在现代产业软件研发范式下, 软件供应链安全已超越传统的组件漏洞范畴, 演变为贯穿整个研发运维协作流程的系统性风险.

因此, 在产业研发范式下, 软件供应链的多层异构架构 (涵盖开源共建生态、产业闭源 CI/CD 流水线及成品下游终端用户) 天然滋生出海量攻击向量. 攻击者可依托上游依赖污染 (如恶意代码注入)、中游流程劫持 (如构建环境篡改) 或下游交付链渗透 (如签名滥用) 等多重路径实施入侵, 导致开源软件供应链攻击在实际场景中呈现出显著的路径多样性与深度隐蔽性. 本文系统性地梳理并分析了近年来行业中具有代表性的供应链安全事件, 结合

相关文献 [60–73] 中的案例研究, 经过 3 轮研讨与评审, 在表 3 中总结归纳了攻击行为在引入、传播与发作这 3 个阶段中所呈现的共性特征与差异传导路径. 该分类方法建立在对典型攻击行为的技术手段、入侵目标、危害程度及攻击面进行系统性比对的基础上, 特别是以传递嵌套性说明该攻击是否可以被其他组件依赖, 从而呈现传递嵌套依赖危害的特征. 最终, 本文提炼出软件供应链攻击的核心特征模式, 为后续风险识别与防御策略的制定提供了结构化支撑依据.

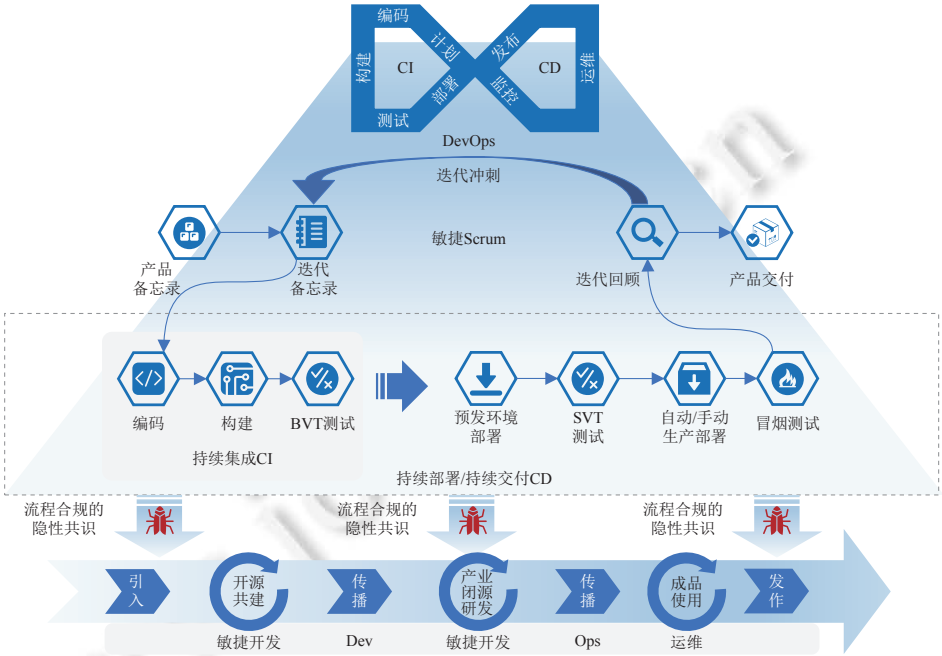

图 2 产业研发模式演化下的软件供应链攻击模型

表 3 主要软件供应链攻击类型分析

入侵阶段	攻击对象	入侵难度	危害程度	传递嵌套	直接危害目标	攻击面	典型事件
引入	IDE插件	低	中	√	开发者本地环境	恶意插件植入、代码注入	VSCode恶意扩展[74]
引入	编译工具污染	中	较高	√	开发者本地环境	编译器环境感染	XcodeGhost[60]
引入	源码依赖注入	中	较高	√	开发者与下游用户	源码仓库依赖篡改	event-stream[61–63]
引入	源码托管平台入侵	高	很高	√	下游软件用户	伪造commit植入后门	PHP Git仓库攻击[64]
传播	依赖解析与包下载机制	低	中	×	开发者内部构建环境	命名欺骗、包优先解析机制	依赖混淆或命名仿冒攻击[65]
传播	CI/CD平台与构建流程	高	很高	×	代码构建产物与用户	构建脚本篡改、系统构建劫持	SolarWinds Sunburst攻击[66]
传播	软件分发服务器	高	很高	×	企业或终端用户	替换更新包, 注入木马	CCleaner[67]、ASUSShadowHammer[68]
传播	第三方CI 服务/脚本仓库	中	高	×	公共CI用户群体	CI 配置滥用、Token 泄漏	Travis-CI Token泄露事件[69]
传播	签名证书与信任根	高	很高	×	信任签名文件的所有系统用户	被盗用或滥用的合法签名证书	NVIDIA签名证书泄露[70]
发作	项目部署包	高	很高	√	运维人员与部署流程	部署包被污染或注入恶意逻辑	Codecov Bash注入[71]
发作	目标终端用户系统	高	很高	√	最终客户、企业终端	C2回连、动态载荷释放、系统劫持	JumpCloud[72]、LNMP①

注: ① 案恒信息, https://mp.weixin.qq.com/s/OT7C1l5rjBNCawFXRIUJOQ

在开源共建威胁产生阶段, 攻击者通常以开源组件的源代码与构建流程为主要攻击对象, 依托开源生态开放性、参与门槛低、信任机制弱等结构性特征, 采用多种手段发起攻击, 包括但不限于伪造身份参与项目贡献、劫持维护者账号、污染上游依赖等方式, 以实现在源头层面对软件进行“供应链投毒”. 其中, 社会工程攻击是典型路径之一, 攻击者通过长期伪装为普通开发者, 逐步获取代码提交权限, 最终在关键构建流程中植入恶意逻辑, 形成隐蔽的源码污染与构建污染双重风险. 此类攻击具备高度隐蔽性和策略性, 难以通过传统的代码审查或静态分析检测. 以 XZ Utils 投毒事件[73]为例, 攻击者在多年参与项目后成功控制构建脚本, 将恶意代码嵌入预编译目标中. 该逻辑在特定系统环境下可劫持 OpenSSH 的认证流程, 实现在无需授权的前提下远程执行指令. 该事件揭示出开源组件引入过程中存在的身份信任失衡、贡献流程透明性不足以及审查机制脆弱等系统性问题, 展现了源码污染型攻击在开源供应链中“低频触发、高危害、强隐蔽”的典型特征, 成为近年来最具代表性的开源供应链投毒案例之一.

在产业闭源研发攻击演化传播阶段, 软件供应链面临的传播风险不仅来源于开发者无意引入的脆弱组件, 也包括攻击者主动投放的恶意依赖. 主要的攻击对象是 CI/CD 流水线, 攻击者通过篡改流水线产物, 构建脚本注入和签名伪造等方式实现软件供应链攻击. 2021 年, 代码覆盖率分析工具平台 Codecov 爆发供应链攻击事件, 成为 CI/CD 环境安全薄弱点被利用的典型案例. 攻击者在未经授权的情况下入侵了 Codecov 的 Docker 镜像构建流程, 并篡改了其 BashUploader 脚本——这是众多开发团队在 CI 流程中广泛使用的自动上传工具. 篡改后的脚本将环境变量(包括访问令牌、API 密钥、部署凭证等) 偷偷回传至攻击者控制的远程服务器, 持续隐蔽运行数月之久. 这些事件反映出当前依赖管理体系在命名机制、信任验证、版本控制与生态管理等方面的脆弱性, 表明企业亟需建立覆盖依赖溯源、SBOM 生成、包签名校验、漏洞检测与行为分析在内的全流程安全防护机制, 以有效阻断供应链威胁在 CI/CD 体系中的持续传播与放大.

在成品使用发作阶段, 攻击者的主要目标是激活此前在上游组件中预埋的恶意逻辑, 实现在终端系统中的数据窃取、远程控制或持久化后门植入等攻击行为. 该阶段的攻击通常具备高度隐蔽性与条件触发性, 依赖于运行环境的特定配置、权限状态、执行时机等上下文信息作为触发门槛, 常通过多阶段加载、动态解包、远程指令联动与反调试检测等手段规避安全检测. 由于大多数下游环境缺乏对安装脚本、部署流程与运行行为的强审计和隔离控制, 攻击逻辑往往能在目标系统中悄然完成执行链条. 以 LNMP 一键部署安装包事件为例[75], 攻击者在安装脚本中嵌入伪装为图片文件的压缩包, 借助多层 shell 脚本构造合法执行路径, 并在检测到特定 Linux 系统版本和非调试环境后, 动态下载远控脚本, 建立 C2 通道, 实现对目标服务器的持续控制. 该类攻击凸显出企业在终端部署过程中存在脚本信任模型薄弱、行为检测缺失与环境隔离不足等问题, 必须通过最小权限执行、运行时行为监控、脚本源验证与部署白名单等机制构建覆盖终端环节的供应链防护能力, 以有效阻断威胁在实际环境中的触发与扩散.

总之, 在以流程合规保障软件研发安全的隐性共识制约下, 研发范式与工具的每一次效能跃迁的“自我强化”, 虽然在表面上缩短了研发周期, 加快了需求到市场的交付时间. 这个背后既有管理方法的变革, 更是自动化流水线工具, 云平台使用所带来的红利. 但是, 开源软件的大量使用, 为产业软件带来更多的安全隐患. 虽然开源社区有大量开发者极力发现漏洞并发布更新. 但实际上, 每一次的漏洞修复对于代码庞大, 项目众多的产业组织来说都要付出极大的代价. 此外开源社区的修复, 并不能完全清除产业传播过程中的污染组件, 截至 2025 年 8 月, 事发 1 年后 Docker Hub 中仍含有数十个 XZ 后门的 Linux 镜像[76], 稍有不慎在问题修复过程中还会引入新的漏洞. 因此, 攻击者可以长期渗透开发者或伪装成合法贡献, 以“自我强化”相同的路径规避安全检测、注入恶意逻辑[77,78]. 通过污染广泛依赖的第三方组件 (含恶意包/有漏洞包) 或劫持核心 CI/CD 工具与服务, 污染自动化构建流水线; 受污染的产物在后续的工具链驱动的再分发过程中, 将感染指数级扩散至下游产业用户[79,80]. 可以看到效能跃迁都天然伴随攻击手段合规和破坏性放大的副作用.

本文以产业研发视角提出“自我强化”与“自我反噬”并生的软件供应链安全内生矛盾, 攻击者正是利用了这一内生矛盾促成复杂的软件供应链攻击. 产业研发视角下开源软件供应链安全的内生矛盾具备以下特点.

- 矛盾触发的全流程覆盖性. 攻击者将目标扩展至研发全生命周期的核心资产, 流程的赋能者同时可能是恶

意攻击者, 其目标涵盖源代码仓库、软件包管理器、构建产物、开发者账户凭证以及 CI/CD 流水线配置. 这表明攻击者策略性地选择能够深度嵌入并利用软件构建与分发渠道的节点, 极大地扩展了攻击的潜在影响面.

● 矛盾内部的不可调和性. 效能杠杆与安全风险均源于同一组技术要素, 从开源、产业再到用户的传播路径与攻击技术的基础篡改、产业演化、成品攻击呈现同源、同构性. 这些手段往往内生于研发流程本身 (如利用合法工具链功能), 规避传统检测, 并借助自动化构建与分发机制实现恶意载荷的快速、广泛传播, 对下游产业用户造成大规模影响. 这两者之间形成了一体两面的矛盾体, 无法自我调和.

● 矛盾产生的多维破坏性与信任瓦解. 开源软件被产业界的不断采纳, 源于产业对开源生态产品开放性, 安全性, 兼容性的信任. 软件供应链攻击不仅造成了严重的技术后果、重大经济损失, 更深层次地侵蚀了产业界与用户对软件供应链生态的信任基础. 这种信任危机对软件研发协作模式、开源生态可持续性及数字化经济的正常运转构成严峻挑战.

4 产业研发视角下的开源软件供应链攻击分类演化

如图 3 所示, 本节从产业研发流程视角审视, 基于一条从开源共建生态、产业闭源研发, 到成品终端用户的分析框架, 深刻揭示了软件供应链攻击利用“自我强化”与“自我反噬”内生矛盾的关键技术特征. 基于此框架, 产业组织的开源软件供应链攻击可概念化为开源共建威胁产生、产业闭源研发攻击演化传播及成品使用攻击发作这 3 个阶段. 攻击者通过渗透研发流水线的信任链与自动化机制, 在任意阶段植入恶意载体: 在开源共建威胁产生阶段, 利用结构性弱点实施身份欺骗、权限劫持或源码/依赖污染, 于源头隐秘植入恶意功能或制造可利用的安全缺陷; 在产业闭源研发攻击演化传播阶段, 代码提交和代码评审等编码行为是触发闭源研发 CI/CD 流水线的第 1 个动作. 代码提交后恶意包/有漏洞包经由依赖解析、构建流水线集成或镜像仓库分发传播, 攻击者可同步劫持 CI/CD 工具链、篡改构建脚本或注入恶意配置实现深度嵌入, 并随着 CI/CD 测试环境部署、测试执行、生产环境的准备和部署而将恶意软件自动化扩散到所有环境中; 并在正式版本发布后流入下游. 伴随着产业软件的开源或者专有分发渠道的传播, 在成品使用攻击发作阶段, 恶意负载经环境条件触发激活, 执行系统完整性破坏、敏感数据窃取或持久化访问建立.

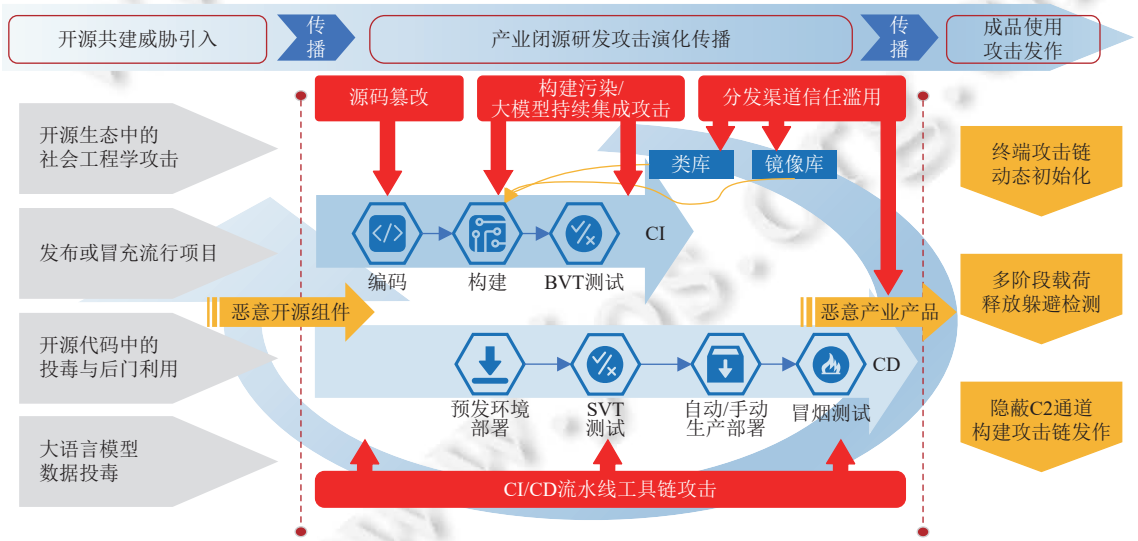

图 3 产业研发视角的开源软件供应链攻击演化分类

4.1 开源共建威胁产生阶段

开源社区作为软件供应链的核心源头, 其开放性与高度协作的特点使其成为攻击者重点瞄准的对象. 攻击者

通过多种手段渗透开源生态, 实施恶意操控, 严重威胁供应链的整体安全性. 为系统化揭示攻击者在开源生态中的渗透路径与攻击手法, 本文将重点探讨其如何利用利益相关者进行社会工程攻击、对源代码实施恶意篡改, 以及通过发布恶意组件或冒充知名项目等手段, 在开源生态中展开多层次的渗透与攻击活动.

4.1.1 开源生态中的社会工程学攻击

开源软件开发依赖大量分布式且多为志愿者的贡献者, 这种开放且协作密集但缺乏统一治理的生态环境, 成为供应链攻击者实施社会工程学攻击的重要切入点. 攻击者通过操控项目中的核心维护者、外部贡献者以及企业内部人员, 利用人员管理和信任体系中的弱点, 达到隐蔽、持久且影响深远的攻击目的. 针对这一现象, 本文将从职业疲劳与项目治理能力下降、社会操控与诱导攻击、贡献者伪装与账户接管这 3 个方面进行深入探讨.

● 职业疲劳与项目治理能力下降. 开源软件维护者多数为兼职志愿者, 他们基于兴趣或技术理想参与社区建设, 但长期的无偿投入和不断增加的维护压力导致“职业倦怠”现象普遍存在. SonarSource 2023 年的调研显示, 约 58% 的开源维护者感受到职业倦怠, 已有 22% 实际退出, 36% 正在考虑退出[81], 这直接影响了代码审查、安全加固及社区治理的质量[82], 为攻击者利用治理薄弱环节提供了机会. 在此背景下, 攻击者常借项目维护不足、代码审计薄弱之机, 将恶意代码注入源代码库或通过包管理平台传播被污染版本. 典型案例为 2018 年的 npm 库 event-stream 投毒事件. 该库作为 Node.js 重要的流处理组件, 被逾 1 600 个项目依赖. 攻击者利用原维护者长期缺席, 通过假借协助维护取得信任, 进而接管发布权限, 并引入含有条件后门的依赖包 flatmap-stream, 该后门专门窃取加密货币钱包私钥信息. 该恶意依赖在社区中隐蔽传播数月, 最终波及数百万下游用户[63].

此外, 商业机构与开源社区之间的不对等关系进一步加剧了治理风险. 企业大规模使用开源软件获取收益, 而维护者却难以获得相应支持, 导致社区对核心项目的维护动力下降. 以 2021 年“Log4Shell”漏洞为例, 维护团队在短时间内承受巨大修复压力, 多位维护者公开表达对企业“长期受益却未提供实质支持”的不满, 认为自己被迫充当“无偿安全外包团队”[83]. 这种结构性失衡加剧了核心项目弃管风险, 暴露出开源生态在人力资源和信任机制上的系统性脆弱, 为攻击者实施社会工程攻击和权限劫持提供了可乘之机.

● 社会操控与诱导攻击. 攻击者大量利用社会工程方法, 借助操纵人心和引导行为的方式渗透软件开发与运维流程, 实现非技术路径下的攻击. 他们通过社交网络和社区舆论等外部渠道, 针对开发者或组织内部成员在资源分配不均、工作压力大等方面的情绪波动进行影响, 甚至通过利益诱导和意识形态操控, 使相关人员无意中放松安全审查或执行潜在风险操作. 同时, 攻击者也可能基于政治立场或经济压力对关键维护者施加长期影响, 诱使其主动植入破坏性代码, 形成内部自毁式的供应链风险. 以 2022 年备受关注的 Colors.js 和 Faker.js 事件为例, 维护者 MarakSquires 因不满大型企业长期免费使用其成果, 在两个高依赖度的 npm 包中注入包含死循环和乱码输出的恶意版本 (如 colors@1.4.44-liberty-2 和 faker@6.6.6), 导致数千个依赖项目运行异常和构建失败. 同年, node-ipc 维护者基于政治抗议发布含文件擦除操作的恶意版本, 严重危及用户文件系统安全. 这些案例表明, 攻击者不仅是外部黑客, 受操控的内部开发者或被劫持的可信角色同样构成重大威胁[84,85]. 这种社会操控的攻击方式揭示了攻击者如何通过发布恶意更新、篡改依赖和操纵构建流程等灰色手段, 在供应链生态中隐蔽干预系统运行, 即使不直接修改核心业务代码, 也能对系统行为产生实质性影响.

● 贡献者伪装与账户接管攻击. 在开源协作环境中, 攻击者主要通过两种途径非法获取项目权限: 一是利用凭据泄露、访问令牌滥用或身份验证缺陷直接劫持合法开发者账户; 二是长期伪装为可信贡献者, 逐步渗透项目治理流程, 实现权限攫取与代码投毒. 后者依赖社区治理中的结构性弱点, 如身份审核松散、权限演化不透明以及维护者资源不足导致的信任放任. 随着大型项目贡献者数量增加和审查压力加大, 攻击者通过伪造身份和操控社交互动获得信任的成功率显著提升, 凸显了加强身份验证和审查自动化的必要性.

2024 年披露的 XZ Utils 后门事件 (CVE-2024-3094) 即为此类基于社会工程渗透策略的典型案例[73]. 攻击者利用 GitHub 账号 JiaT75, 于 2021 潜入项目, 初期以积极贡献者身份赢得社区和维护者 Lasse Collin 的信任. 在项目维护资源有限且 Collin 疏于管理的背景下, JiaT75 主动承担管理与发布工作, 2022 年获授权提交与发布权限. 随后, 他潜伏近两年, 直到 2024 年初在 xz-utils 5.6.0 和 5.6.1 版本中隐蔽植入后门. 该后门通过伪装为测试数据文件 (bad-3-corrupt_lzma2.xz 与 good-large_compressed.lzma), 并借助构建脚本中的宏展开机制, 在特定条件下导致

构建产物与源码不一致. 更为隐秘的是, 后门利用 GNU libc 的 IFUNC 机制在运行时 hook 关键函数 RSA_public_decrypt, 允许攻击者绕过 OpenSSH 的公钥验证, 实现远程身份验证绕过, 只要系统同时存在 liblzma 与 OpenSSH.

后门发布时机亦颇具策略性, 攻击者选择在 Ubuntu 等主流 Linux 发行版测试冻结前夕推动版本合并, 试图规避严格审查. 该后门最终不是因为代码审计工具而被发现, 而是因为 PostgreSQL 核心开发者 Andres Freund 在调试 Linux 发行版 sshd 启动延迟时注意到 liblzma 异常 CPU 使用率, 进一步反编译分析揭露了源码与构建产物不符及隐藏钩子函数. 此事件经 oss-security 邮件列表 (https://www.openwall.com/lists/oss-security/2024/03/29/4) 曝光后, 引发广泛关注并促使各大 Linux 发行版迅速下架受影响版本.

此事件揭示出, 传统依赖信任与人工审查的开源项目治理机制难以有效应对社会工程主导的深度渗透型攻击. 攻击者通过伪装贡献、操控信任路径, 完成对构建链的长期劫持, 其策略性与隐蔽性远超传统技术型入侵, 凸显了加强开源社区关键人物身份管理、混淆代码评审与构建产物测试验证等机制在当前开源安全治理中的重要地位.

4.1.2 发布或冒充流行开源项目

在软件供应链的上游环节, 攻击者日益利用开发者对主流开源平台 (如 GitHub) 及其生态机制所建立的高度信任, 突破传统依赖关系注入路径, 通过伪造仓库、克隆热门项目或发布带毒版本, 实现对下游用户的直接渗透与攻击. 这些策略的核心在于构建高度可信的伪装环境, 以规避安全防护机制和用户警觉, 进而实现恶意代码在供应链中的隐秘传播与持久存在.

2024 年, Gelb[86]的研究揭示了一种基于平台搜索机制操控的新型攻击技术. 攻击者通过自动化脚本大规模生成名称相似且界面结构高度一致的 GitHub 仓库, 辅以多种搜索引擎优化策略, 如在项目元数据中嵌入热门关键词、伪造 Star 与 Fork 数量, 从而人为提升这些仓库在搜索结果中的曝光度和排名. 这类仓库通常伪装成活跃且可信的开源项目, 诱使开发者访问和使用. 恶意代码多被隐藏在构建脚本或工程配置文件 (如 Visual Studio 项目文件) 中, 一旦被目标系统构建或运行, 即可能执行如加密货币钱包窃取等破坏性操作.

除搜索劫持策略外, 攻击者还大量滥用 GitHub 的 Fork 机制实施项目伪装与代码注入. Cao 等人[87]对 35 个与加密货币相关的主项目及其 68 879 个 Fork 实例进行了系统分析, 发现其中 26 个 Fork 仓库含有恶意代码或可执行文件, 攻击目标覆盖 Linux 容器、物联网设备和服务器等多种典型部署环境. 攻击者通常通过复制知名项目, 继承其结构与命名规范, 在此基础上隐蔽植入恶意逻辑. 由于 Fork 仓库在名称和结构上与原项目高度相似, 开发者极易误判其合法性, 导致中毒版本被无意中引入软件构建流程.

此外, 部分攻击者甚至在无依赖链污染的情况下, 构造名称、描述及元数据高度近似的独立软件包或代码仓库, 利用第三方镜像站点、开源推荐平台及开发者论坛等渠道进行推广, 诱导用户手动下载安装. 这种结合命名相似性误导与项目伪装的组合式攻击, 技术门槛相对较低, 社会工程效果显著, 已成为近年来开源软件供应链攻击的高发手段, 对主流开发生态系统构成了持续且严重的威胁.

4.1.3 开源代码中的投毒与后门威胁

在开源软件生态中, 攻击者一旦获取项目的访问或提交权限, 即可能在源代码层面实施投毒和后门注入行为, 将恶意逻辑深度嵌入版本控制体系与构建流程中, 从而实现后门植入. 这类供应链攻击不同于基于漏洞利用的即发型入侵, 其攻击链条呈现高度隐蔽性、条件触发性与信任链传染性. 攻击者通常将恶意逻辑伪装为正常功能改动, 以规避人工审查与静态分析检测, 并借助自动化构建与依赖继承机制, 在下游项目构建或运行阶段悄然触发, 实现跨项目、跨版本的隐形传播. 表 4 中系统性地总结了源码层面投毒与后门注入的典型案例, 并对比分析了主要的实现方式与攻击路径. 投毒和后门注入的实现路径多样, 既包括在核心业务逻辑中嵌入经过语义伪装的恶意语句, 也可能通过构建脚本、依赖配置、测试逻辑或格式化配置等低关注度模块引入恶意组件, 以增强攻击链的隐蔽性与传播性. 部分攻击者利用编程语言的宏特性、动态解析能力或结构混淆手段对恶意逻辑进行掩饰, 例如将 payload 编码为 Base64 字符串隐藏于资源文件或注释中, 并在运行时通过特定触发条件动态解码执行.

典型案例包括 XZ Utils 后门事件 (CVE-2024-3094)[73], 攻击者在长达数年的渗透后获得核心提交权限, 并在 liblzma 压缩库中植入条件触发的命令执行逻辑, 使得特定 SSH 连接能够绕过认证远程控制 Linux 服务器. 由于 XZUtils 是 OpenSSH 等系统核心组件的底层依赖, 一旦全面释放, 将对全球 Linux 生态造成灾难性冲击. 2021 年

PHP 源代码仓库后门事件中, 攻击者通过入侵 Git 服务器, 在 PHP8.1-dev 分支嵌入可由特定 HTTP 请求头触发的远程执行逻辑, 为后续恶意植入提供隐蔽入口[64]. 此外还有 Lodash 的原型污染事件, 攻击者可以利用语言的先天性漏洞, 埋入恶意代码从而达到攻击目的[88]. 这些攻击得以长期潜伏, 主要依赖 3 类隐匿机制协同作用: 其一, 利用逻辑重构与语义伪装, 如细微调整条件表达式、函数重载、变量命名混淆等方式降低改动可感知性; 其二, 恶意逻辑仅在特定环境条件下触发, 诸如特定操作系统版本、调试器状态、网络配置或构建参数, 从而规避 CI/CD 流水线中的沙箱测试与静态分析; 其三, 利用项目审查机制的缺陷, 策略性地将恶意改动隐藏于大规模格式化提交、样式调整或文档更新中, 并借助信任关系及审查疲劳推进合并.

表 4 源代码层面后门注入的典型机制与案例

技术机制	实现方式	典型案例	主要影响
恶意提交/内部投毒	正常开发提交中混入伪装后门逻辑	PHP Git仓库后门[64]	任意代码执行, 远程植入
构建工具篡改	通过IDE注入隐性逻辑	XcodeGhost[60]	App后门注入, 广泛传播
构建工具链注入	利用 IDE 构建流程注入后门 JAR	Octopus Scanner[89]	注入NetBeans项目构建流程, 开发者构建即感染
条件逻辑炸弹	环境感知触发恶意行为	XZ Utils[73]	静默触发、系统控制
脚本注入	构建/上传脚本窃取敏感信息	Codecov[71]	CI 环境变量泄露
编码与混淆	使用 Base64、宏、资源文件隐藏代码	ctx 恶意包[90]	静态检测规避
信任链嵌套	依赖项内置远控逻辑, 滥用SBOM信任	Ethcode 事件[91]	安全清单绕过, 远程控制

系统庞大的组件生态中可能含有尚未修复的缺陷, 但缺乏可触发其漏洞的源代码 (漏洞不可达), 因此攻击者可以通过编写触发其漏洞的代码从而达到攻击的目的. npm 生态中, 过期的维护者域名、长期无人维护的包以及安装脚本滥用, 都可能成为恶意代码注入的切入点[92]; Maven 生态的长期积累则导致已知漏洞在多个版本间持续存在并被广泛调用, 其中约 1/4 的漏洞函数可直接被下游项目访问, 为攻击者在构建阶段发起精准渗透提供条件. 以 Java 生态为例, 后端项目通过 Maven 将 Java 源码与 SpringBoot 等第三方依赖编译成 jar 或 war 包. 研究表明, 在超过 44450 个包含 CVE 信息的库中, 25.7% 的漏洞函数可被至少一个下游项目访问[93]; 另外针对 300 万个 Maven 包的分析显示, 高危漏洞数量逐年增加, 其中某些漏洞 (如 CVE-2020-36518 和 CVE-2022-24823) 可能影响超过 37 万个版本包, 最常见的漏洞类型为“非信任数据的反序列化”, 占比 37%[94].

威胁已延伸至开源社区代码托管平台, 实现对供应链“源头路径”的深度嵌套操控. 2017 年, Mortenson[95]通过一个 GitHub 仓库展示了如何将恶意代码偷偷加入 GitHub 的 Pull Request (PR) 中, 揭示了软件供应链攻击的一种潜在途径. Goyal 等人[96]研究了如何在 GitHub 上识别不寻常的提交. 他们指出透明环境和社交编码平台 (如 GitHub) 虽然有助于开发者及时了解项目变更, 但过多的通知消息可能导致信息过载使开发者难以区分重要变更. 为此他们开发了一种异常检测机制, 通过统计模型识别与同一仓库中其他提交或同一开发者提交相比异常的提交. Benedetti 等人[97]针对 GitHub Actions 工作流的安全性进行了自动评估研究, 分析了 GitHub 工作流在软件供应链中的安全风险, 并开发了工具 GHAST 来检测工作流中的安全漏洞和配置错误, 通过对 50 个开源项目进行实验, 发现了大量安全问题, 强调了进一步研究和改进 GitHub 工作流安全性的必要性.

综上, 开源生态中的投毒与后门威胁正从单点攻击演化为以“构建-触发-传导”为核心的链式攻击模式. 攻击者通过逻辑混淆、环境触发、审查绕过及工具链嵌套等多维策略, 构建兼具隐蔽性、持久性与传播性的攻击通路. 应对此类威胁, 需在开源治理、自动化审查、构建安全和版本发布管理等环节, 建立跨阶段、跨链路的信任验证与溯源体系, 从根源阻断隐蔽渗透, 提升供应链安全韧性.

4.1.4 大语言模型数据投毒

数据投毒是一类针对 AI 供应链源头的数据完整性破坏攻击, 其核心思路是在模型训练阶段向数据集中注入经过精心构造的恶意样本, 以诱导模型在推理阶段表现出错误预测或触发隐蔽的后门行为[98–100]. 不同于直接入侵模型或推理系统的即发型攻击, 数据投毒在训练阶段完成隐蔽植入, 具备高度持久性与跨阶段传播性. 在大规模预训练背景下, 基础模型普遍依赖海量、公开的互联网数据进行构建, 攻击者可通过污染上游公共数据源、向开源

数据集提交恶意样本、滥用数据抓取与标注平台等方式, 将少量高隐蔽性的“毒数据”混入数以亿计的训练样本中. 由于大模型的高容量与泛化能力, 这些恶意样本即便占比极低, 仍可能在特定触发条件下引导模型执行攻击者预设的行为[99]. 例如, 研究已证实仅需在训练集中嵌入少量特定水印式触发样本, 就可在推理时诱导模型生成虚假事实、泄露敏感信息[101], 或在代码生成任务中自动插入安全漏洞[102]. 进一步地, Xu 等人[103]提出了 Shadowcast 攻击, 其通过在视觉-语言模型 (vision-language model, VLM) 的训练数据中注入看似无害但语义扰动的样本, 实现了跨架构的误分类与叙事操控. 实验表明, 即便只需少量投毒样本, 该攻击仍可在不同模型之间迁移并保持有效, 凸显了在多模态大模型时代, 数据投毒的隐蔽性与跨域传导能力.

这种威胁在 AI 供应链中的扩散路径具有显著的层级传导效应: 上游被污染的开源数据集→预训练基础模型→下游企业微调模型→业务生产系统. 一旦后门条件被满足, 攻击者便可在无需直接入侵下游系统的情况下, 实现对企业应用的远程操控或数据破坏. 典型案例显示, 数据投毒可能以不同形态影响模型的可信性[104]. 2016 年, 微软推出的聊天机器人 Tay 在上线数小时内便因遭遇恶意用户的持续输入而迅速学习并模仿不当言论, 最终在不足 24 h 内被迫下线并公开致歉, 该事件凸显了外部数据输入对模型行为的即时性风险[105]. 2024 年, 媒体报道字节跳动商业化团队在大模型训练过程中曾发生“内部投毒”事件 (https://www.yicai.com/news/102386162.html). 据称, 一名实习生因对团队资源分配不满, 利用 Hugging Face 平台漏洞在共享模型中注入破坏性代码, 导致部分训练成果受损. 尽管该事件细节仍有待进一步验证, 但其反映出在企业级 AI 开发实践中, 除外部数据来源外, 内部人员行为与第三方平台漏洞同样可能成为 AI 供应链安全的关键隐患.

与传统软件供应链攻击相比, 数据投毒的隐蔽性更强、影响面更广, 其恶意逻辑与模型参数深度绑定, 往往难以通过后期模型检测或部署审计完全剔除. 这一特性使得数据投毒成为大模型安全中的系统性风险源, 亟需在数据采集、标注、清洗与训练各环节引入跨阶段、跨主体的信任验证与可追溯机制, 从源头遏制“中毒模型”的生成与传播.

4.2 产业闭源研发攻击演化传播阶段

在开源软件的全生命周期中, 上游项目在完成构建与版本发布后, 其产物如软件包、容器镜像等需通过分发与集成机制进入并部署于下游的产业环境, 这条从开源共建生态延伸至产业生态的交付与传播路径构成了软件供应链的关键环节. 随着开源软件不断参与企业应用研发, 以及企业应用的客户多数为政府、大型企业等付费用户, 一经渗透其破坏性更大. 攻击者的关注点正由普通上游项目逐步转向那些能被企业软件供应链依赖的、传播广泛的开源项目, 甚至直接渗透到企业 CI/CD 流程中.

与开源社区中松散的代码共享模式不同, 企业级的 CI/CD 系统往往依托固定的工具链和标准化的构建脚本, 运行过程高度依赖事先配置的规则和任务链. 这种高度自动化和信任机制使得一旦构建过程被篡改, 影响就会在很短时间内覆盖到所有使用相同流程的版本和环境. 例如, 部分攻击事件中, 攻击者通过修改构建脚本或利用配置中的缺陷, 使得带有隐蔽后门的产物在构建结束后直接进入企业的内部发布环节, 甚至跨越多个产品线传播. 由此可见, CI/CD 流程不仅是交付效率的保障, 也可能成为攻击在不同系统之间迁移和放大的通道.

4.2.1 源码篡改

在开源软件被引入产业研发环境的过程中, CI/CD 流程不仅承载着研发与生产的核心衔接功能, 同时也可能成为有漏洞或含恶意组件的开源依赖进入产业系统的关键途径. 它不仅负责将外部组件与内部代码进行集成与测试, 还将构建结果交付至生产环境, 从而直接连接研发与业务系统. 由于 CI/CD 流程高度自动化且环节众多, 一旦攻击者在输入端或中间环节植入漏洞、恶意软件或带有后门的组件, 这些安全隐患可能在缺乏人工复核的情况下被持续传递, 并迅速扩散到多个下游系统, 形成潜在的供应链安全威胁[106].

在产业研发环境中, 虽然已有很多工作可以通过组件复用完成, 但组件的组合以及项目特殊的需求仍需要通过开发人员提交的源码完成. 源码提交是持续集成的第 1 个动作, 在很多场景中源码提交可以驱动持续集成流水线开始运行. 相对于开源社区而言, 产业组织的源码管理平台常因缺乏及时维护, 更容易遭受攻击者的恶意利用. Torres-Arias 等人[107]的研究揭示了一类针对 Git 的新型攻击: 元数据操纵攻击, 该攻击通过篡改 Git 的元数据 (如

分支和标签引用), 导致开发者看到不一致的仓库状态, 最终将未测试代码合并到生产分支进行软件供应链攻击. IDE 工具也是产业研发组织遭受攻击的重灾区, 例如 XcodeGhost 事件中[60], 攻击者发布含恶意配置的非官方 XCode 版本, 导致使用该工具构建的 iOS 应用在编译阶段自动植入后门. 由于后门应用获得合法签名并正常分发至 App Store, 最终影响数百款应用和上亿终端用户. 此外, 现如今 AI Coding 已经深入企业研发实践中, 结合大模型数据投毒的相关案例来看, 自动化代码生成的 IDE 插件以及智能体工具本身也可能被利用进行源码恶意篡改.

总体来看, 产业组织的源码保护机制并非固若金汤, 一旦攻击者通过代码管理平台的恶意利用获取代码提交权限, 就可以篡改代码, 并在后续版本迭代中持续传播. 这也使其成为供应链攻击在产业 CI/CD 体系中实现注入并向下游系统扩散的关键节点. 通过结合实证研究和生态案例可以看出, 源码提交不仅是开发与交付的重要环节, 更是整个产业软件供应链中潜在安全威胁落地与扩散的高风险点, 亟需在代码审计、依赖管理、提交策略和 CI/CD 安全防护中引入更严格的监控与验证机制.

4.2.2 构建污染

CI/CD 代码构建通过构建流程将包含漏洞或恶意逻辑的开源组件直接引入产业项目, 从而对交付物造成污染. 除了这种直接引入的风险之外, 构建系统自身亦可能成为现代软件供应链攻击的关键载体, 这一风险被称为 CI/CD 流水线构建污染. 其核心特征在于: 攻击者无需修改源代码, 即可借助构建系统的高度自动化特性和默认信任机制, 隐蔽地植入漏洞或恶意逻辑, 使构建产物本身成为投毒媒介, 进而实现上游渗透与下游大规模传播. CI/CD 构建系统污染通常通过依赖解析机制、生命周期钩子脚本及构建缓存等路径形成复杂的联动攻击链, 这使其在攻击方式和影响范围上, 与传统 CI/CD 代码构建风险显著不同.

CI/CD 流水线构建污染的典型路径, 通常始于恶意依赖包的制作与发布. 在依赖解析环节, 攻击者常利用依赖混淆和命名仿冒等技术, 诱导构建工具拉取其控制的恶意组件, 进而引发软件供应链信任危机[108–110]. 例如, 攻击者可在公共仓库 (如 npm、PyPI) 注册与企业私有包同名但版本号更高的组件, 使构建系统在默认优先公共源或高版本策略下下载并执行该恶意依赖. Birsan[111]于 2021 年首次系统性地验证了此类攻击路径, 成功入侵多家头部科技企业, 引发行业广泛关注. 同时, 命名仿冒攻击在实证研究中被广泛证实: 攻击者注册拼写近似的组件名 (如 requests 伪装为 request), 诱导开发者手动误拼或脚本自动拉取. 据 Ohm 等人[112]研究, 超过 61% 的恶意包采用此类命名诱导策略. Truong[113]进一步指出, npm 与 PyPI 在此类攻击中呈现出显著高敏感性.

一旦恶意依赖成功注入交付物构建流程, 攻击者可利用生命周期钩子脚本 (如 preinstall、postinstall、prepare) 在构建阶段自动执行任意命令, 实现初始入侵、凭据窃取、远程通信建立或后门模块写入等操作. 例如, 在 Codecov 构建污染事件中[71], 攻击者通过篡改上传脚本, 在正常构建流程下持续窃取环境变量与认证令牌, 且该行为在数月内未被检测. 此外, 构建工具插件或脚本加载模块亦可用于注入恶意逻辑, 借助构建节点的默认信任路径, 在未显式声明依赖的前提下完成远程控制代码的加载与执行.

构建缓存机制进一步放大了污染的隐蔽性与持久性. 部分平台在拉取缓存时缺乏完整性校验与签名验证, 使得被篡改的构建缓存能够在后续构建中被自动还原并执行, 即便主项目已启用依赖版本锁定机制, 也难以彻底阻断攻击链条. 在针对 GitHub Actions 构建缓存的投毒攻击中, 研究发现攻击者可以通过缓存服务的运行时令牌或键碰撞机制, 将恶意逻辑长期残留于构建环境, 并在 CI/CD 的周期性任务中反复激活, 形成持久化的后门风险[114–116]. 为了提升攻击隐蔽性, 攻击者常辅以多种规避机制: 在载荷隐写层面, 通过脚本伪装、 Base64 编码或图片隐写隐藏恶意逻辑; 在触发逻辑层面, 通过构建环境识别或编译参数判断限制执行场景, 仅在特定平台或身份下激活载荷; 在信任链滥用层面, 通过将恶意组件嵌入构建工具默认插件或中间层依赖路径, 绕过签名验证与 SBOM 审计追踪.

综上所述, CI/CD 流水线构建污染已演化为现代软件供应链攻击中极具风险的环节, 其核心特点在于: 不依赖显式源码篡改, 而是利用构建系统默认信任与自动化执行, 实现隐蔽植入与远程操控, 从而影响整个制品交付链路. 针对该类威胁, 业界亟需建立集成依赖源白名单与私有源优先策略、SBOM 强制生成与验证、构建产物签名追溯、生命周期钩子审计及构建缓存隔离等多维度防御体系, 以保障软件供应链从源头到产物的全链条可信性与完整性.

4.2.3 CI/CD 流水线工具链攻击

在现代软件工程实践中, CI/CD 工具链作为连接源代码、依赖管理、制品构建与部署发布的核心基础设施,

已成为攻击者重点投毒与控制的目标. 与构建系统污染主要操控“组件拉取逻辑”不同, 工具链攻击聚焦于流水线基础设施的篡改与操控, 攻击者通过渗透 CI 环境本身获取构建系统控制权限, 从而实现对整个制品生成流程的隐蔽劫持与远程控制. 在这一过程中, 攻击者通常利用 CI/CD 系统中的高权限环境、配置不当的代理程序、第三方插件漏洞或密钥管理缺陷实施入侵. 一旦攻破工具链核心组件, 攻击者即可操作构建脚本、任务配置及环境变量等关键执行路径, 注入后门逻辑或篡改构建逻辑, 最终生成携带恶意代码的污染制品. 同时, CI 系统所存储的敏感凭据 (如访问令牌、部署密钥、私有仓库地址) 常被用于二次渗透企业内部系统, 进一步扩大攻击范围.

更为严重的是, 一旦攻击者掌握 CI/CD 工具链的控制权限, 即可将恶意逻辑写入企业内部制品仓库、容器镜像仓库或二进制分发渠道, 从而污染后续所有构建、测试与部署流程. 研究表明, 攻击者可在 CI 脚本中嵌入条件触发逻辑, 使后门代码仅在特定构建用户、操作系统或编译参数下激活, 从而提升其在工业环境中的生存能力[44]. 此类污染路径通常绕过常规审计, 尤其在缺乏构建产物签名验证机制时, 攻击者可低成本实现长期、持续且深度的控制. 典型案例包括 2021 年的 Codecov 攻击, 攻击者通过篡改 Bash Uploader 脚本, 在 CI 构建过程中持续获取用户环境变量与认证信息, 受影响企业涵盖 Twilio、HashiCorp 等[71]. 2023 年, 美国联邦调查局 (Federal Bureau of Investigation, FBI)、网络安全和基础设施安全局 (Cybersecurity and Infrastructure Security Agency, CISA) 与美国国家安全局 (National Security Agency, NSA) 联合发布报告指出, 俄罗斯对外情报局 (Foreign Intelligence Service of the Russian Federation, SVR) 利用 JetBrains TeamCity 持续集成平台中的远程代码执行漏洞 (CVE-2023-42793) 对全球数百台服务器实施渗透, 攻击者不仅远程控制构建流程, 还访问企业制品管理系统与源代码仓库[117].

值得注意的是, CI/CD 工具链攻击在开源共建生态与企业研发环境中呈现不同特征. 在开源社区中, 攻击者通常通过社会工程学手段劫持开发者身份或贡献权限; 而在企业内部环境中, 由于权限隔离与审计机制更为严格, 攻击者更多依赖工具链组件漏洞、配置失误或密钥泄露完成入侵. 实践表明, 即便在受控研发环境下, 当构建权限集中或插件生态复杂且未受审计时, CI/CD 工具链仍可能成为攻击链的关键突破口.

综上所述, CI/CD 工具链攻击代表了供应链安全体系中“以构建系统为中枢, 向下游制品发起多点扩散”的典型上游投毒路径. 其成功不仅影响构建产物的完整性与可信性, 更对软件系统生命周期安全构成系统性威胁. 针对该类风险, 亟需实施最小权限原则、强化插件完整性校验、构建脚本审计, 并在制品层引入签名验证与追溯机制, 从而实现从构建入口到交付终端的全流程可信保障.

4.2.4 分发渠道信任滥用

开源社区在完成软件发布后, 会依据软件包、镜像、研发框架等不同形式, 通过软件仓库、镜像仓库等分发媒介进行传播. 目前, 国内外不同编程语言和技术生态中, 主流分发渠道相对集中 (表 5), 多数由产业界主导运营, 少部分由产业基金会负责管理. 这种由权威组织主导的模式在提供高效流通路径的同时, 也为其托管的软件构建了一种基于平台、流程和推荐机制的隐性信用背书. 用户在使用这些平台时, 往往默认其分发的软件版本具备合法性与完整性, 而不再对来源与内容进行深度核验. 然而, 这种默认信任机制在便利软件传播的同时, 也可能为恶意代码的传播提供隐性支撑.

表 5 开源软件主流分发渠道

年份	名称	语言	工具	功能	产业组织
2002	Python Package Index (pypi.org)	Python	pip	包管理	Python软件基金会
2003	Maven Central (maven.apache.org)	Java	mvn	包管理	Apache基金会
2008	GitHub (github.com)	综合	Git	项目管理、代码管理、 CI/CD	微软公司
2009	npm (npmjs.com)	JavaScript	npm	包管理	GitHub、微软公司
2009	阿里云 (developer.aliyun.com/mirror/)	综合	综合	镜像管理	阿里云
2013	DockerHub (hub.docker.com)	Docker容器	docker	容器镜像管理	Docker公司
2013	Gitee (gitee.com)	综合	Git	项目管理、代码管理、 CI/CD	开源中国
2016	Hugging Face (huggingface.co)	大语言模型	大模型	大模型管理	Hugging Face公司
2022	魔搭社区 (modelscope.cn)	大语言模型	大模型	大模型管理	ModelScope社区

一旦攻击者在构建阶段成功植入恶意逻辑, 即可借助这些被广泛信任的分发渠道实现快速、大规模的下游传播. 典型手段包括滥用自动更新机制与盗用数字签名证书. 前者是通过劫持更新流程, 将原本来自官方服务器的请求引导至攻击者控制的服务器, 并下发包含恶意逻辑的更新包; 若系统仅依赖域名或服务器地址判断更新合法性, 而缺乏对完整性的严格校验, 便容易被利用. NotPetya 攻击事件中, 攻击者正是通过篡改财务软件的自动更新模块, 将恶意程序推送至用户终端, 造成了严重破坏[118]. 后者则通过窃取合法证书对恶意程序签名, 使其外观与正常软件无异. 例如, Adobe 在 2012 年遭遇攻击, 其签名服务器被入侵, 攻击者利用合法证书为恶意程序签名, 导致用户在毫无察觉的情况下安装了受感染的软件[119].

类似风险同样存在于容器镜像的分发中. 在云原生环境中, 企业普遍通过公共镜像仓库获取基础镜像并在其上构建应用. 一旦这些镜像中被植入恶意内容, 下游项目便会在不知情的情况下继承并扩散这些风险. 攻击者可通过修改构建脚本、嵌入恶意命令或篡改配置文件, 在镜像中加入后门功能, 并利用持续集成系统在构建过程中自动触发. 例如, Codecov 事件中, 攻击者利用上传脚本漏洞篡改了构建过程, 间接影响了众多依赖该工具的项目[71]. 此外, 有研究指出, Docker 镜像平台的整体安全评分偏低, 缺乏有效的内容审计与管控机制[120].

综上, 软件供应链分发环节不仅传递技术产物, 更传递信任. 一旦攻击者控制了分发渠道、更新机制或社区推荐路径, 便可在大范围内传播篡改过的软件, 显著扩大攻击影响. 防范此类风险需要在软件来源、传输与安装环节引入多层验证机制, 包括强制数字签名校验、增强平台访问控制、审计镜像构建流程, 以及完善社区治理与发布审核制度, 从而构建更稳健的分发信任体系.

4.2.5 大模型软件供应链持续集成攻击

传统软件供应链主要关注代码库、依赖项和构建过程中的漏洞, 而大语言模型 (large language model, LLM) 供应链则引入了更为抽象的风险层面, 涵盖数据集、模型权重以及各类工具链组件等要素, 其核心特征可概括为“重模型、轻代码”[100,121,122]. 大语言模型供应链不仅继承了 DevOps 体系的传统风险, 还因“模型即产物”的特性, 产生了新的攻击面, 涵盖模型训练、权重管理、推理部署及适配器集成等多个环节[123]. 其核心基础设施高度依赖 TensorFlow、PyTorch 等深度学习框架, 以及 MLflow 等模型实验管理与追踪工具, 这些基础工具的漏洞可直接传导至上层模型构建与交付流程, 成为攻击者入侵供应链的突破口. 例如, TensorFlow 的 Keras 模块曾曝出 Lambda 层代码注入漏洞 (CVE-2024-3660), 允许攻击者在模型定义阶段嵌入任意执行逻辑; MLflow 曾存在路径遍历漏洞 (CVE-2023-1177), 可被利用实现敏感配置文件的未授权访问[121]. 此类基础设施级漏洞一旦被利用, 将威胁整个模型构建链的安全完整性.

模型微调及部署阶段同样呈现复杂且多样化的潜在攻击面. 以低秩适配 (low-rank adaptation, LoRA) 为代表的参数高效微调技术 (parameter-efficient fine-tuning, PEFT) 因其轻量、模块化特性, 成为后门注入的理想媒介. 已有研究表明, 攻击者可训练带有隐蔽触发器的 LoRA 适配器, 并将其伪装成高质量开源模型发布至社区平台, 一旦被加载至生产系统, 该组件可在特定输入下触发敏感操作, 如下载恶意软件、调用外部工具或实施钓鱼攻击[123]. 此外, 大规模训练过程通常在分布式计算环境中进行, 这又引入了跨节点、跨集群的安全隐患. 例如, PyTorch RPC 通信系统曾曝出远程代码执行漏洞 (CVE-2024-5480), 攻击者可借助输入验证缺陷在训练集群中执行任意指令, 控制模型训练节点. 与此同时, CI/CD 系统作为模型构建与集成的核心骨干, 也可能被滥用. 攻击者可通过 GitHub Actions 中的令牌泄露、脚本注入等手段污染构建过程, 或利用云平台 (如 SAP AI Core) 中的租户隔离漏洞, 实现跨项目的模型篡改与数据窃取[122].

综上所述, 语言模型供应链的持续集成攻击不仅继承了传统 DevOps 风险, 还在训练、微调和部署等环节引入全新攻击面, 其攻击载体涵盖基础框架、实验管理工具、微调适配器以及 CI/CD 流程. 攻击方式高度自动化和隐蔽, 能够跨节点、跨项目扩散, 一旦成功, 攻击者可在不修改源代码的情况下劫持整个模型构建链, 破坏模型完整性和可信性, 对模型产出及下游应用形成系统性威胁.

4.3 成品使用攻击发作阶段

软件供应链攻击的最终目标是促使早期安全漏洞或者嵌入的恶意逻辑在下游成品使用环境中被激活并完成

执行, 从而在关键业务系统或敏感操作节点上实现数据窃取、远程操控、权限劫持等攻击者所预谋的战略行为. 典型攻击生命周期表明, 终端发作阶段通常遵循一条高度程序化的攻击路径: 攻击者往往在上游污染组件中预置延迟触发机制, 并利用运行时漏洞、加载器劫持或构建链篡改等方式, 实现恶意载荷的动态注入; 随后, 通过多阶段解包与行为解锁释放攻击代码, 联动远程资源下载与加密数据解密流程, 逐步建立与云端 C2 的持久通信通道, 以实现动态命令注入、远程控制与状态反馈. 整个攻击链条常伴随环境识别、反沙箱检测、自解密混淆等规避技术, 显著降低其在静态分析、入侵检测系统 (intrusion detection system, IDS) 及主流行为分析引擎中的可见性. 总体而言, 该阶段攻击具备显著的上下文依赖性、链式释放性与远程联动性, 成为软件供应链攻击中技术复杂度最高、检测难度最大、破坏潜力最强的关键环节.

4.3.1 终端攻击链动态初始化机制

软件供应链攻击的全生命周期中, 恶意载荷的动态执行链初始化构成攻击意图终端触发的核心技术枢纽. 攻击者系统性滥用开发者工具链的默认信任机制, 通过软件安装脚本、子组件配置、构建配置脚本乃至集成开发环境 (integrated development environment, IDE) 等自动化执行路径, 实现恶意代码的隐蔽加载与执行.

安装脚本劫持表现为攻击者污染依赖管理工具的安装钩子 (如 Python 的 setup.py 或 npm 的 preinstall/postinstall), 在包安装阶段触发恶意操作. 典型案例如 OneinStack 与 LNMP 事件, 攻击者利用包安装触发并释放了一系列恶意脚本. 在编译期注入恶意逻辑, 如 XZ Utils 供应链攻击即通过篡改 build-to-host.m4 脚本, 劫持 OpenSSH 认证流程实现权限提升[73]; 此类技术亦延伸至 CI/CD 领域, 如 Codecov 事件通过污染 Bash Uploader 脚本窃取 CI 环境变量.

构建配置劫持进一步渗透至部署环节, 攻击者篡改构建钩子 (如 MSBuild 编译脚本) 或环境初始化配置 (如 Dockerfile), 实现深度系统植入. SolarWinds 事件即通过污染 MSBuild 工程文件, 在编译阶段将 Sunburst 后门注入合法 DLL 并借数字签名绕过验证, 此类攻击因依赖动态触发机制 (如环境变量解析) 导致静态检测效能显著受限[124]. IDE 工程钩子攻击则将矛头指向开发环境本身, 攻击者通过分叉合法项目嵌入恶意构建脚本 (如 Visual Studio 的.sln 工程或 PreBuildEvent), 在本地构建时执行远程控制指令; 演进手法包括利用 IDE 扩展市场漏洞 (如 2025 年 Open VSX Registry CVE-2025-319[125]), 通过恶意更新感染开发者环境.

综上, 攻击技术呈现清晰的演进逻辑: 从显式脚本注入 (安装阶段) 向隐式信任链寄生 (构建流程与开发工具) 的范式迁移, 逐步渗透软件生命周期各环节, 凸显跨阶段协同防御的紧迫性.

4.3.2 多阶段载荷释放躲避检测

在近年来日益严峻的软件供应链安全背景下, 攻击者不断演化其攻击策略以绕过日益成熟的检测机制. 其中, 多阶段载荷释放逐渐成为一种典型且高度隐蔽的攻击规避技术. 该技术指恶意软件在运行过程中并非一次性加载全部恶意逻辑, 而是通过一系列精心设计的阶段性操作, 包括解码、解密、重组或远程获取等方式, 逐步释放最终的恶意载荷 (如后门组件或数据窃取模块). 需特别指出的是, 该技术与高级持续性威胁 (advanced persistent threat, APT) 等描述攻击生命周期的多阶段攻击在概念上存在根本差异, 前者关注攻击链内部的载荷构造与执行方式, 而非攻击流程的全局宏观分解.

随着软件供应链安全意识的普遍提升, 以及检测防御技术的持续演进 (涵盖静态代码分析、动态行为监控、软件成分分析与漏洞扫描等), 攻击者面临的植入与激活恶意逻辑的门槛显著提高. 这种背景下, 若要在开源共建生态中实现有效渗透, 恶意代码必须具备极高的隐蔽性与环境适应性, 以同时绕过研发构建阶段 (如代码审查、CI/CD 流程中的静态检测) 与生产运行阶段 (如沙箱执行、行为监控) 的多重防护机制.

在厘清多阶段载荷释放的技术内涵之后, 其之所以具备强大的规避能力, 根本原因在于 3 大核心机制的深度协同与有机融合. 首先, 规避性设计通过将攻击链解耦为多个离散阶段 (如 XZ Utils 事件中从脚本下载、伪装包解压、环境检测到 C2 通信的层层递进过程), 系统性地规避了从开发到运行的全链路检测手段 (如静态扫描、动态沙箱分析、内存哈希校验等), 显著降低了单点暴露的可能性. 其次, 动态触发逻辑确保恶意行为的激活严格依赖于对运行环境的实时验证, 例如在 OneinStack 事件中, 载荷仅在 Red Hat 系统且无调试进程的前提下才被激活, 从

而精准地避开非目标环境下的分析行为[126]. 再次, 信任链劫持机制通过将恶意逻辑深度嵌入被信任的构建脚本或依赖项中, 滥用已有的系统信任关系以绕过审计流程. 例如, Ethcode 事件即攻击者通过刻意模仿合法依赖项 keythereum, 篡改生成恶意依赖 keythereum-utils, 在扩展加载过程中植入远控模块, 从而规避了 SBOM 等软件物料清单审计手段[91]. 这 3 种机制之间形成递进协作关系, 极大地增强了对防御体系的穿透能力.

在上述 3 大核心机制协同构建的基础规避能力之上, 攻击者还进一步引入多维度的隐匿技术策略, 以持续强化攻击链的隐蔽性与生存能力 (详见表 6). 在载荷伪装与动态获取维度, 攻击者利用多样化手段绕过静态扫描工具的检测: 包括文件格式伪装 (如 OneinStack 中对脚本后缀名的篡改)、图像隐写术 (如 XZ/YoColor 事件中通过 LSB 在测试图像中嵌入代码) 及内存中执行加密载荷以规避杀毒软件的扫描. 此外, 环境感知与条件触发维度通过嵌套的环境验证机制提高攻击的准确性与专一性, 涵盖对目标环境参数的识别 (如 glibc 版本、sshd 路径)、反分析策略 (如调试器检测、CPU 核心数判断以识别沙箱环境) 及跨平台自适应执行 (如 Ethcode 根据 OS 类型选择载荷). 而在多层封装与混淆强化维度, 攻击者采用结构性嵌套和依赖混淆技术构建逆向防御屏障, 例如 XZ 事件通过 Shell 脚本→控制流扁平化 C 代码→ELF 二进制的三重封装结构, 或在 Ethcode 中将 Base64 模块植入合法依赖项以规避 SCA 分析, 从而进一步削弱安全分析工具的效果. 上述维度彼此协同, 从静态、动态、结构多个层面提升了整个攻击链的存活与规避能力.

表 6　多阶段利用隐匿技术演进对比

攻击事件	载荷伪装技术	环境感知维度	封装层级
g01pack[127]	多态编码	进程黑名单检测	双重 (Java→二进制)
OneinStack[126]	后缀篡改 (.jpg)	系统类型 (Red Hat)	双重 (Shell→ELF)
XZ Utils[73]	LSB隐写+数据包伪装	服务路径+调试器检测	三重 (Shell→C→ELF)
Ethcode[91]	依赖混淆+AI 生成代码	跨平台运行时检测	动态依赖加载

综上所述, 多阶段载荷释放技术正逐步演化为现代软件供应链攻击中的核心规避手段. 其通过规避性设计、动态触发、信任链劫持等核心机制构建起坚实的检测规避基础, 并辅以载荷伪装、环境感知、多层混淆等技术手段, 在多个技术维度实现攻击链条的深度隐匿. 诸如 XZ Utils、Ethcode、OneinStack 等事件的发生, 充分验证了该类技术的实用性与破坏力. 随着攻击者不断优化其规避技术路径, 防御体系亟需建立起贯穿开发与运行全生命周期的协同安全保障机制, 以有效应对新型载荷释放策略所带来的系统性挑战, 进而推动软件供应链安全范式的更新与重塑.

4.3.3　隐蔽 C2 通道构建与攻击链发作

在现代软件供应链攻击的发作阶段, 云服务环境逐渐演化为构建隐蔽命令与控制 C2 通道的核心基础设施. 这些隐蔽通道不仅承担多阶段载荷释放后的远程指令下发、数据窃取和持久化控制任务, 还深度依托云原生环境固有的信任链与业务逻辑嵌入, 实现跨平台与跨地域的长期潜伏与隐形渗透. 云平台的多租户架构、合规加密通信机制以及跨地域调度能力, 为攻击者提供了天然的流量合法化掩护, 使其能够有效规避传统网络边界的检测与行为分析的双重防护, 保障供应链攻击的隐蔽性与持久性.

如表 7 所示, 在 SaaS 服务滥用 (E1) 领域, 攻击者常利用弱认证或未启用多因素认证的 API 密钥入侵 SaaS 管理平台. 典型的 JumpCloud 攻击事件中, 入侵者通过窃取 API 密钥向超过 5 000 台终端设备下发伪装成合规性检查脚本的 C2 加载器. 该加载器将 Base85 编码的指令嵌入 JSON 状态报告的 metadata 字段, 并通过平台原生 TLS 1.3 加密信道传输至攻击者控制的域名, 实现了大规模、跨组织的隐蔽指令扩散[128]. 数据通道隐匿技术同样体现在云服务驱动的数据泄露通道 (E2) 中, 攻击者将敏感数据编码为看似正常的云存储对象元数据、日志字段或统计指标. 例如, 有攻击者利用 GitHub API 周期性获取仓库的 Star 数量, 将其数值转码为 C2 服务器 IP 地址 (如 star_count=3232235521 对应 192.168.1.1), 并通过 CI/CD Actions 构建日志输出 AES-GCM 加密指令 (如 echo “a2V5PXgweDFm...” » $GITHUB_OUTPUT), 由接收端实时解析 API 日志完成解密, 构筑了一个完全绕过传统网络层检测的隐蔽信息流[129].

云原生架构的事件驱动特性为多阶段攻击链 (E3) 及隐秘 API 滥用 (E4) 提供了天然的执行框架和隐蔽通信渠道. AWS Lambda 攻击链中, 攻击者首先利用 SQL 注入污染 DynamoDB 更新事件流, 随后创建权限过度的 Lambda 函数, 将 C2 指令封装为 SQS 消息队列中的数据库变更通知. 目标容器监听该队列时, 解析事件消息中特定字段 (如 event.Records[0].dynamodb.NewImage.command) 进行指令解密与执行, 使 C2 通信深度嵌入云服务信令体系, 绕过大多数动态监控与静态检测工具[130]. 此外, 攻击者在 Kubernetes 集群中滥用 etcd 组件的未授权 2379 端口访问漏洞, 注入恶意 Pod 并部署伪装 Fluentd 日志采集服务的 C2 代理. 该代理通过监听 514/UDP 端口, 将控制指令隐匿于正常日志流, 既规避了网络异常检测, 也利用云原生组件的可信通信范式, 成功实现对下游容器的远程控制与持久渗透[131,132].

表 7 近年开源软件供应链攻击模型

日期	开源组件	开源共建阶段				传播渠道	产业闭源研发阶段				成品使用阶段								
		A1	A2	A3	A4		S1	S2	S3	S4	I1	I2	I3	E1	E2	E3	E4	E5	E6
2021.8	pyfetchx[133]	—	—	—	√	PyPI	—	√	—	—	√	—	—	√	√	—	—	—	—
2022.1	IconBurst[134]	—	√	√	—	npmjs	—	—	—	—	√	—	—	—	√	—	—	—	—
2023.1	yandex-travel[135]	—	√	—	—	npmjs	—	—	—	—	√	—	—	—	√	—	—	—	—
2023.4	fsevent version[136]	—	—	—	—	npmjs	—	—	√	—	√	—	—	√	√	—	—	—	—
2023.9	LNMP[75]	—	√	√	—	官方网站	—	—	—	√	√	—	—	—	√	√	√	—	—
2023.10	OneinStack[137]	—	√	√	—	官方网站	—	—	—	√	—	√	—	—	√	√	√	—	—
2024.1	Discord-Token-Generator[138]	—	—	√	—	GitHub	√	—	—	—	√	—	—	—	√	√	—	—	—
2024.1	ember-cli-3@1.0.0[139]	—	√	—	—	npmjs	—	√	—	—	√	—	—	—	√	—	—	—	—
2024.1	fix-this@3.2.3[140]	—	—	√	√	npmjs	—	—	—	—	√	—	—	—	—	—	—	—	√
2024.3	build-benchmark@15.2.4[139]	—	—	√	√	npmjs	—	—	—	—	√	—	—	—	√	—	—	—	—
2024.3	yocolor[141]	—	—	√	—	PyPI	—	—	√	—	√	—	—	—	—	√	—	√	—
2024.3	XZ-Util@5.6.0[73]	√	—	√	—	Linux包管理	—	—	—	—	√	—	—	—	—	√	—	√	—
2024.4	rhermann[142]	—	—	√	√	PyPI	—	—	—	—	√	—	—	—	√	—	√	—	—
2024.4	multi-plerequests[143]	—	√	√	—	PyPI	—	—	—	—	√	—	—	—	√	—	√	—	—
2024.9	github.com/0xjiefeng①	—	—	√	√	GitHub	—	—	—	√	—	—	√	√	—	√	—	√	—
2025.1	SearchFilter.7z[144]	—	—	√	√	GitHub	—	—	—	√	—	—	√	—	—	√	—	√	—

注: ① 微步在线研究响应中心, https://mp.weixin.qq.com/s/ih36z93y6BazatjeoGjp1A

为确保远控的持续性与隐蔽性, 攻击者还在软件供应链产物中植入后门模块 (E5). 这些后门通常将加密的 C2 地址及认证密钥封装于第三方依赖库的更新补丁中, 借助正常的依赖下载和版本更新流程触发激活逻辑. 结合无文件化技术, 后门模块在内存中直接解密并建立 C2 会话, 极大地降低持久化痕迹和检测风险. 此外, 反向 Shell 技术 (E6) 作为云服务 C2 通道的关键执行环节, 攻击者通过触发云函数或 CI/CD 作业中的特定回调, 建立加密的反向连接. 该连接流量伪装成合法的云 API 调用或消息队列通信, 实现双向交互式隐蔽控制, 进一步提升操控的灵活性和隐蔽性.

综上所述, 云服务环境下的隐蔽 C2 通道构建已成为软件供应链攻击发作阶段的关键技术支撑. 攻击者通过滥用 SaaS 平台、利用数据泄露通道、多阶段攻击链、隐秘 API、后门植入及反向 Shell 等多维手段, 构建起隐蔽、持久且跨平台的远程控制体系, 实现载荷释放后的精准指令下发与持续渗透. 这种多层次协同的隐匿通信机制极大地提升了供应链攻击的隐蔽性与破坏力, 有效突破传统网络和行为检测防线, 对软件供应链安全防护提出了贯穿开发、构建与运行全生命周期、跨层级多维度的监测与防御新挑战. 深入理解云服务隐蔽 C2 通道的构建与运作机制, 对于提升软件供应链攻击的威胁感知与防御能力具有重要意义.

4.4 软件供应链攻击案例总结

近年来, 开源软件供应链攻击呈现出明显的阶段性、链式传播与系统性危害特征. 基于对各类开源软件供应

链攻击行为的系统分析, 并结合本文筛选的学术文献及 67 篇产业报告 (表 1 所示) 中的典型案例, 本文构建了开源软件供应链攻击模型 (表 7 所示). 本文通过 3 轮深入的研讨与评审, 最终确定了 16 起具有代表性的开源软件供应链攻击事件. 如表 7 所示, 攻击过程可划分为 3 个关键阶段: 开源共建威胁产生、产业闭源研发攻击演化传播和成品使用攻击发作. 该模型不仅清晰地展现了攻击的时序演进路径, 还系统揭示了各阶段采用的细分技术手段及其内在关联.

如表 7 所示, 在“开源共建阶段”, 攻击者主要通过社会工程 (A1)、伪装成流行项目 (A2)、代码混淆 (A3) 或直接发布恶意项目 (A4) 等手段, 向开源仓库 (如 PyPI、npmjs) 注入恶意代码. 例如, “XZ-Util”事件中, 攻击者结合 A1 与 A2 手段, 逐步获取维护权限并植入后门; “yandex-travel/ui@1.1.0”则通过伪装合法包 (A2) 诱骗开发者引入. 此类行为污染了开源生态的原始素材, 为后续传播奠定基础.

进入“产业闭源研发”阶段，恶意代码借助依赖攻击 (S1)、误拼写包名 (S2)、劫持官方资产 (S3, 包括开发者账户、DNS、代码库、容器等) 或篡改研发流水线产物及利用 CI/CD 工具链漏洞 (S4) 等方式，渗透至企业开发环境与构建流程. 例如, “IconBurst”通过误拼写包名 (S2) 渗透下游应用依赖链; “OneinStack”则通过劫持官方资产 (S3) 实现传播. 这一阶段的关键在于利用产业界对开源组件的高度信任和自动化集成机制, 实现恶意代码的横向扩散与深度嵌入.

最终, 在“成品使用”阶段, 恶意代码通过安装脚本 (I1)、子组件安装配置 (I2) 或 IDE 脚本 (I3) 等机制在用户环境中激活, 并执行 SaaS 服务攻击 (E1)、数据泄露 (E2)、多阶段攻击 (E3)、隐秘 API 调用 (E4)、后门驻留 (E5) 或反向 Shell (E6) 等恶意行为. 例如, “fsevent”包通过安装脚本 (I1) 触发恶意行为, 并进一步实施数据泄露 (E2); “Discord-Token-Generator”则通过 GitHub 传播, 实施多阶段攻击 (E3) 与反向 Shell (E6).

综上所述, 攻击者利用开源软件供应链“自我反噬”的安全缺陷, 已形成从源码污染到构建传播、再到终端执行的完整链条. 3 个阶段中采用的细分技术在研发效能提升的“自我强化”下相互衔接、逐级放大, 正是利用了现代产业研发中的以流程合规保障软件研发安全隐性共识, 隐匿的完成攻击. 例如, 在 XZ 事件中, 攻击者针对的目标是欠维护但具备广泛传播能力的 XZ 项目, 它被例如 Ubuntu、Debian 等多款企业级 Linux 系统所依赖, 攻击者在长达 3 年的攻击、传播实施中采用了社工攻击、代码混淆、依赖攻击、多阶段等多种复杂的软件供应链攻击方法, 最终被微软的研发人员所发现. 时至今日仍有大量被污染的 XZ 相关镜像存在于 DockerHub 中, 这凸显了开源软件虽然强化了产业软件的能力, 但同时攻击者可以沿着相同的路径, 对开源社区, 产业研发以及下游用户进行攻击, 最终反噬软件供应链与产业研发生态. 未来需构建覆盖全链路的动态安全防护机制, 尤其加强对开源组件来源验证、CI/CD 流程中组件变动的管控, 特别是针对系统的运行时行为监控来达到安全再平衡的目的.

5 基于产业研发视角的软件供应链安全再平衡

在研发效能的不断“强化”下, 非法的攻击者利用开放的社区成为合法的贡献者, 被污染的组件在产业软件依赖下被包装成为产业组织信用背书的流行企业软件, 自动化流水线的使用让研发态的威胁快速传入了成品阶段, 这形成了一体两面无法解决的内生矛盾, 只要能成功利用暴露面众多的软件供应链体系, “自我反噬”的安全威胁就在所难免. 在产业数字化进程频频受扰的当下, 软件供应链安全不再是一道可选项, 一体两面的内生矛盾是在研发组织流程合规的产业隐性共识下所产生的, 因此应该直面问题, 在开源共建、产业研发与成品使用的链路中, 建立主动暴露风险的安全机制. 邬江兴院士曾提出以拟态防御思维 (https://www.edu.cn/info/media/yjfz/qyjs/201612/t20161215_1476207.shtml) 应对网络空间中不可预知的安全威胁. 既然无法预判未知的未知, 无法完全阻止攻击, 那就让攻击的成本更大, 发现的时机更靠前. 通过化整为零、大模型智能化主动感知自动修复的方式伴随研发流程, 降低安全发现成本, 让安全研发不再是被动的流程妥协, 实现攻防之间的再平衡.

5.1 在开源共建阶段进行协同治理与代码异常检测

为应对开源生态在供应链早期阶段所面临的安全威胁, 现有技术路径主要体现于两个层面: 一方面是开源项目安全治理机制, 通过组织协作与制度建设提升整体生态的风险协同防控能力; 另一方面是开源代码分析与检测

技术, 依托自动化工具与方法增强对潜在漏洞和恶意代码的动态识别与管控.

5.1.1 开源项目安全协同治理体系

在软件供应链攻击日益频发的背景下, 开源社区因其高度开放性与参与主体的多样性, 逐渐成为攻击者重点渗透的对象. 为提升生态系统整体的安全韧性, 业界围绕开源项目的管理、构建与交付流程, 逐步建立起一套较为系统的安全协同治理机制与实践框架, 以期从源头遏制恶意代码的引入与传播.

其中, 开放源代码安全基金会 (Open Source Security Foundation, OpenSSF) 由 Linux 基金会牵头成立, 汇聚产业界与社区的协同力量, 专注于提升开源项目开发与使用过程中的安全水平. OpenSSF 推出了多项基础性建设项目以弥补开源项目的安全薄弱环节, 例如: 用于评估项目安全成熟度的评分体系 (scorecard)、开发者安全认证标准 (secure software development fundamental, SSDF)、自动化签名与验证服务 (sigstore) 等. 这些举措不仅有助于建立透明的安全基线, 也推动了在社区内部形成更为清晰的责任体系与信任模型. 与此同时, OpenSSF 强调对核心维护者账户与权限的保护, 推广更严格的身份认证与访问控制措施, 以降低因账号劫持、权限滥用而导致的供应链投毒风险. 此外, 研究表明, 采用代码审查和分支保护等实践的项目漏洞率显著降低, 其中代码审查被认为是缓解供应链风险的关键措施之一, 而分支保护机制则有效遏制未授权提交与流程绕过[145]. 基于此, OpenSSF 进一步推动对关键开源项目的第三方代码审计、漏洞赏金计划以及与政府和标准化机构的合作, 从而在更广范围内提升开源生态的安全保障能力. 此外, XZ 事件爆发后, 维护者选择加入 JazzBand (一个针对欠维护开源代码的协同治理社区), 来抵御日益增加的软件供应链投毒攻击.

总体来看, 开源安全治理正在经历从“事后响应”向“过程控制”与“前置预防”的战略转型. 通过组织协作、规则制定与工具支持的协同推进, 供应链攻击的潜在入侵路径不断被压缩. 然而, 面对日益隐蔽与多样化的威胁策略, 现有框架仍需进一步增强对异常行为的识别能力, 并提升从依赖关系解析到版本更新的全链路可见性与可验证性, 以构建更加稳固的开源信任基础.

5.1.2 开源代码异常检测

为遏制攻击者在开源软件源头注入恶意逻辑的行为, 源代码分析与检测技术逐渐成为保障软件供应链安全的核心防线. 该类方法主要聚焦于代码提交、审核与合并等关键环节, 通过静态、动态以及混淆识别与流程检测等多维度手段识别潜在威胁, 从而提升上游代码仓库的可审计性与可信度.

静态分析作为应用最为广泛的检测手段, 可在无需执行代码的前提下, 通过语法解析、结构检查与数据流建模识别注入类漏洞、特权调用、代码克隆与恶意植入等风险. 例如, SonarQube 和 CodeQL 等工具已被广泛用于企业与开源项目的早期安全审查[146]. 在恶意更新场景下, Froh 等人[147]提出基于 CodeQL 的差分静态分析方法, 利用版本间行为差异定位后门注入, 在误报率低于 1.4% 的情况下成功识别所有已知恶意版本. Wu 等人[93]进一步结合元数据信号 (如可疑安装脚本、过期邮箱域名等) 提出风险预测模型, 用于提前预警可能的投毒行为. 然而, 由于供应链攻击往往通过微小变更混入大型项目, 且攻击逻辑在视觉与功能层面高度隐蔽, 现有规则式检测面临显著挑战. Zheng 等人[148]指出, 这已形成一种攻防“军备竞赛”, 攻击者不断采用代码混淆等手段规避检测. 因此, 学术界提出通过识别非常规字符混淆、字符串编码与冗余逻辑等特征, 将混淆度作为可疑指标辅助人工审查. 例如, 通过代码熵值、抽象语法树 (abstract syntax tree, AST) 与控制流图建模, 结合机器学习方法, 有效识别如控制流平坦化、字符串加密等混淆手法[149–151].

动态分析则通过在沙箱等受控环境中运行待测程序, 捕捉运行时行为以发现静态检测难以覆盖的威胁, 尤其适用于识别条件触发后门、下载执行载荷及混淆逃避型逻辑. 尽管该方法存在覆盖率不足与性能开销大的问题, 近期研究在提高动态检测可扩展性方面取得进展. Duan 等人[152]提出的 MALOSS 框架首次实现了大规模生态下的动态分析, 结合元数据与行为特征成功识别出 npm 与 PyPI 平台上的数百个恶意包. 随后, Zheng 等人[148]提出的 OSCAR 系统通过结合沙箱执行、模糊测试与关键 API 行为监控, 实现了对更隐蔽恶意组件的高精度检测, 其在真实数据集上的 $F1$ 分数达 0.95, 误报率显著低于传统工具. OSCAR 还可自动生成测试输入并监控网络请求、文件写入及子进程生成等行为, 从而弥补静态方法在动态触发和环境感知方面的不足.

软件供应链攻击通常会通过混淆代码插入恶意代码, 因此混淆识别与流程绕过检测成为防御高度隐蔽攻击的

关键. 攻击者常利用字符串编码、控制流重构、动态代码生成等方式混淆恶意逻辑, 以规避规则检测. Fass 等人[153]提出的 JaSt 系统通过结合 JavaScript AST 与随机森林模型, 实现了对混淆代码典型结构的识别, 准确率接近 99.5%. 同时, 基于异常命名模式、冗长无意义字符串与嵌套 eval 调用等指标的语义异常检测方法也被提出, 用于提示高风险模块进入人工复核[154]. 在流程层面, 攻击者可能通过绕过审查机制或篡改标签发布进行投毒. 为此, Torres-Arias 等人[107]提出 RSL (reference state log) 机制, 利用加密签名日志保障仓库在提交与拉取过程中的一致性, 从根源上降低元数据篡改的风险. 此外社区应当加强针对异常行为的关键性检测, 例如加强针对在非正常时间进行代码变更, 含有大量混淆代码的变更, 任务名称与代码修改不一致的变更等异常行为的识别.

总体而言, 应当在以规则驱动的静态审查基础上, 针对代码变更、新依赖引入、配置更改等关键环节建立行为建模、模糊检测与流程验证的多层次防护体系. 未来研究需要在提升检测覆盖率与降低误报率之间取得平衡, 并进一步强化与社区审查机制的协同, 从而在恶意代码渗透的早期阶段暴露攻击行为.

5.1.3 大模型数据安全治理

相较于传统软件系统, 大语言模型对训练数据的分布特征与语义一致性具有更高敏感性, 攻击者可通过在训练集中注入少量恶意样本诱导模型行为发生偏移, 进而为模型植入后门或埋藏泄露敏感信息的风险. 因此, 构建覆盖训练数据源头治理与训练过程鲁棒优化的双层防御体系, 已成为提升大模型内在安全性与抗攻击能力的关键途径[155,156].

在软件物料安全数据训练与治理层面, 数据筛选与可信审计机制被广泛引入, 以防范低质量或恶意语料污染模型. 典型做法包括构建 CycloneDX、ML-BOM 等数据物料清单 (bill of materials, BOM) 体系, 系统记录数据来源、处理链路与版本信息, 并辅以供应商合规审查与多维度一致性验证[157]. 此外, 针对监督微调中潜在的知识泄露与后门植入风险, 知识净化方法通过对特定参数子结构 (如前馈网络层) 实施定向微调, 配合安全响应目标训练, 在控制通用能力损失的同时有效抑制敏感信息泄露. 例如, Ishibashi 等人[158]基于参数高效微调技术 (如 LoRA), 显著降低了多个开源模型在隐私泄露基准上的暴露风险. 尽管如此, 现有数据清洗与审计方法在面对超大规模异构语料时, 仍面临高误报与自动化处理能力不足的挑战.

在训练过程鲁棒性层面, 优化算法设计是抵御投毒与后门攻击的核心技术路径. 对抗训练通过引入对抗样本增强模型在污染环境下的泛化能力, 被广泛验证可有效缓解后门触发行为[159]. 在此基础上, Zhao 等人[160]提出 W2SDefense 框架, 借助清洁教师模型与蒸馏机制引导污染模型实现后门遗忘, 在多项基准中实现了高精度攻击清除与最小性能损失. 为进一步应对由投毒导致的模型行为偏差, 研究者提出上下文鲁棒架构[161]与自去噪学习机制[162], 通过增强输入敏感性建模与语义一致性约束, 提升模型输出的稳定性与可靠性.

总体而言, 大语言模型的数据投毒威胁其于训练阶段, 其防护策略核心应围绕训练数据质量控制与训练算法鲁棒性增强展开. 当前研究已从孤立的技术点防御, 逐渐发展为覆盖训练全流程的综合治理框架. 未来工作需致力于提升数据筛查的自动化水平与精度, 发展更高效的无害化训练方法, 并探索数据与模型协同分析的统一治理范式, 以构建更加健壮和可信的大模型安全基座.

5.2 在产业闭源研发阶段持续合规与攻击面收敛

在开源软件供应链攻击向产业环境扩散的过程中, 依赖管理平台、CI/CD 流水线与构建工具链常成为威胁的“放大器”. 尽管 GitHub、npm 等平台推动生态繁荣, 并已提供签名验证、漏洞告警、分支保护等安全能力, 但其保障体系整体呈现“可用而不均衡”: 机制并非缺失, 而是多为自愿接入或按项目配置, 覆盖与深度有限、跨项目一致性与自动化强度不足; 同时部分高阶能力 (如强制性供应链签名、策略化审批、企业级密钥管控) 往往绑定于高级订阅, 导致大量社区项目难以获得持续性强防护. 叠加开源维护模式的异质性与依赖链条的传递性, 攻击者得以借助 CI/CD 流水线实现快速且隐蔽的跨环节传播, 因而产业侧亟需在产业研发阶段进行系统化的持续合规并在源码采纳和成品交付的出入口构建灰度管控机制, 以“抓中间、掐两头”的方式做好产业研发的动态防护.

5.2.1 DevSecOps 持续合规实践

DevSecOps, 即 development (开发)、security (安全) 与 operations (运维) 的缩写, 是一种将安全实践系统性嵌

入 DevOps 方法论的软件开发模式, 强调开发、运维与安全团队在软件生命周期各阶段 (包括规划、编码、测试、部署及运维) 中保持紧密协作, 以确保安全性成为整个流程的核心约束[163–166]. 作为开发、安全与运维高度融合的理念, DevSecOps 正逐渐成为产业界防护开源软件供应链攻击的关键策略, 其本质在于打破传统开发流程中安全与业务隔离的格局, 通过从开发起点即深度嵌入安全控制, 并在流水线全周期中持续监控和校验, 实现安全与敏捷交付的协同优化. 例如, Kumar 等人[165]提出的基于云开源软件自动化开发安全运维 (automated DevSecOps using open-source software over cloud, ADOC) 概念模型通过自动化机制实现了 DevSecOps 的持续安全性与敏捷性, 而 Haverinen 等人[167]的 Cyberismo 方案采用开放信息模型与“安全即代码”范式, 实现了 DevSecOps 流程中的网络安全合规自动化管理. 这类方法不仅提升软件交付效率, 也显著增强在复杂供应链环境中对威胁传播阶段的响应能力, 从而为产业界提供系统化的全生命周期安全保障, 同时兼顾敏捷研发与持续交付的业务需求.

● 安全需求工程. 在快速迭代的敏捷研发环境中, 尽管小步快跑的节奏加速了产品从需求到市场的周期, 但安全实践与敏捷实践之间存在的潜在不一致性不容忽视[168]. 特别是在客户未明确提出安全需求时, 研发团队往往容易忽视系统的整体安全性, 进而在设计阶段埋下隐患. 因此, 产业界已普遍认识到, 在研发早期阶段加强安全需求工程的培训与实践至关重要, 通过“安全左移”策略将安全性前置, 已逐渐成为行业共识[169].

为系统性地应对这一挑战, 产业界逐渐重视安全需求工程 (security requirements engineering, SRE) 及其配套的安全策略. 基于 SRE, 可开展安全需求提取与分析 (security requirements elicitation and analysis, SREA) 等核心工作[2,169,170]. 在具体执行过程中, 可以借助 STORE 框架识别软件供应链中的攻击点、信任点、推测点与依赖点, 并在此基础上开展安全风险评估、安全需求建模与验证, 最终将抽象的安全需求转化为可操作的需求文档[170,171].

● 持续集成阶段 (CI) 防护. 在 DevSecOps 持续集成阶段, 构建与测试环节的安全防护至关重要. 在构建阶段, 产业界要求对软件包进行完整性和来源校验, 确保未被篡改或植入恶意代码, 例如通过数字签名验证保证软件包来源的可靠性[165]. 为应对开源组件 (OSS) 潜在的供应链风险, 软件组成分析工具可扫描项目中使用的依赖库, 检测已知漏洞和许可证合规性问题, 从而降低供应链攻击可能性. 同时, 通过建立 SBOM, 能够精确掌握细粒度依赖关系, 并将扫描分析融入自动化 CI 流程, 提高软件依赖的可视化、可追溯性与合规性[120,172]. 针对二进制依赖的安全威胁, 产业界重点关注恶意二进制插入和代码混淆问题. Barr-Smith 等人[173]提出基于版本差异检测的方法识别供应链中恶意二进制文件, 而 Greco 等人[174]与 Jiang 等人[175]则通过可解释人工智能 (XAI) 和函数级嵌入技术提升恶意代码检测的准确性. 此外, 为防御恶意源码攻击, 可重复构建 (reproducible build) 方法确保从源代码到最终发布的二进制包可独立验证, 任何未经授权的修改均可被发现[176]. 在容器镜像构建过程中, 通过 SBOM 对镜像内部软件包进行物料分析, 并结合多阶段安全检查与动态行为分析, 可防止已知漏洞镜像被发布或重用, 从而提升容器安全性[177].

在测试阶段, 动态应用安全测试 (dynamic application security testing, DAST) 与渗透测试工具用于模拟真实攻击场景, 发现运行时安全漏洞. 使用工具如 Burp Suite, 可对 Web 应用、API 与网络接口进行漏洞扫描. Rangnau 等人[178]通过案例研究展示了在 CI/CD 流水线中集成 DAST 工具的实践方法, 从而在快速迭代的开发周期中有效应对安全挑战. 此外, 自动化测试和持续监控机制可与构建阶段的 SCA、SBOM 和可重复构建策略结合, 实现从源码到生产环境的端到端安全保障.

总体而言, DevSecOps 持续集成阶段防护机制通过构建、依赖管理、二进制检测、容器安全、动态安全测试等多维手段形成闭环, 既保证了软件在流水线各环节的完整性与安全性, 也为产业界提供了一套可量化、可追溯且可持续的防护策略, 显著降低了供应链攻击在 CI/CD 流程中的传播风险[165,172–178].

● CD 防护. 在 CD 阶段, 软件从开发测试环境过渡至生产运行环境, 其安全性直接关系到系统的完整性与可信度. 由于部署高度依赖自动化脚本、配置文件、容器镜像及其他构建产物, 一旦流水线被篡改, 可能绕过前序安全检测, 将恶意代码、后门逻辑或配置污染注入正式系统. Bass 等人[179]指出, 通过将复杂、不可信的部署流水线拆解为多个小型、权限受限、功能独立的可信单元, 并严格限制网络访问与资源范围, 可显著降低攻击面并提升可验证性, 从而增强系统对渗透行为的可观测性与控制能力.

IaC 已成为现代 DevOps 的核心实践, 但大量实证研究表明 IaC 工具链和脚本本身存在安全短板. War 等人[180]对超过 1 600 个 IaC 仓库的分析显示, 源代码、依赖项和清单文件普遍存在漏洞, 覆盖 OWASP Top 10 几乎全部类别, 其中配置错误、访问控制失效及硬编码密钥尤为突出; 声明式配置文件 (如 YAML/JSON) 是攻击重点. 此外, 许多缺陷源于跨角色操作的误用, 例如开发者修改网络配置或运维人员调整认证逻辑, 若安全测试工具 (如 Snyk、Horusec) 未系统性集成于 DevOps 流程中, 漏洞可能在多个版本间长期存在. Zeini 等人[181]指出, 当前组织普遍缺乏系统性的 IaC 风险管理框架, 对配置失误、自动化脚本变更及权限滥用缺乏持续监控, 超过 60% 的云安全事件可归因于手动修复行为, 凸显了 IaC 生命周期中“非计划性”操作的高风险.

为系统化应对 CD 阶段的威胁, 产业界和学术界提出全链路纵深防御方案. 在部署前阶段, 可利用静态分析工具 (如 Checkov、Snyk IaC) 检测过度授权、未启用加密等配置风险, 并通过 OPA (open policy agent) 对 IaC 脚本进行策略合规性验证; 在产物管理层面, 结合 Sigstore 提供的 Cosign 与 Rekor 对容器镜像及模块实现签名溯源; 在运行时环境中, 通过 Kubernetes 准入控制机制 (如 ValidatingAdmissionWebhook 和 PodSecurity 标准) 对资源请求进行动态审查, 防止未签名或异常配置的资源进入生产环境; 在凭据管理上, 引入集中式密钥管理平台 (如 AWS Secrets Manager), 实现 API Token 和 SSH Key 的动态注入与最小授权[182]. 此外, 部署完成后需关注配置漂移问题, 通过 Drift Detection 工具与持续审计手段监控容器弹性伸缩、手动干预等引起的实际状态偏离, 确保系统与 IaC 描述保持一致, 形成可追溯、可验证的安全部署闭环[181].

综上所述, DevSecOps 在持续交付与部署阶段强调多层次、防御纵深的安全策略, 通过流水线拆解与权限隔离、IaC 脚本分析与配置合规性验证、产物签名溯源与运行时动态审查, 以及凭据管理和配置漂移监控, 形成全生命周期、可追溯、可验证的安全闭环. 这种系统化防护不仅有效降低了部署阶段的攻击面与风险暴露, 还确保了自动化、敏捷交付流程中安全目标的持续实现, 为产业环境中的软件供应链提供了坚实的安全保障.

5.2.2 产业闭源研发中的攻击面收敛

● 内源软件研发机制. 开源软件虽被广泛采用, 但在产业环境中其安全性与可靠性仍存在潜在风险[183]. 为系统性地应对这一挑战, 产业界不仅需建立严格的外部开源软件采纳评估机制, 还应构建内源软件研发机制 (inner source software development, ISSD), 将开源软件转化为组织内部可控的研发资产[184]. ISSD 通过建立内部研发社区, 使不同团队和部门能够共享知识、复用软件组件并协同开发, 从而提升组织创新能力及软件供应链的整体安全性.

在实施层面, 产业界可对关键开源软件进行内源化改造, 将其纳入内部研发体系, 通过二次开发与定制满足组织特定安全需求, 同时增强对软件生命周期的掌控. 例如, 在对开源 Web 框架进行内源化改造时, 可嵌入自定义安全模块, 以强化访问控制、输入验证和漏洞防护. 内源研发机制不仅提高了软件开发过程的可控性, 还强化了版本管理、安全策略执行及风险缓解能力, 使组织能够依据自身安全标准和业务需求对软件进行灵活调整, 从而降低因开源软件供应链风险引发的潜在安全隐患.

此外, 产业组织应建立安全情报仓库, 持续从内部代码仓库、漏洞报告及外部威胁情报中积累安全知识, 并通过构建安全知识图谱辅助开发人员识别和管理开源组件带来的安全威胁[185,186]. 这一机制增强了对内源化软件的安全监控能力, 并为 DevSecOps 流程中各阶段提供数据支持和决策依据, 从而在内源软件研发与持续集成/持续部署与持续交付过程中形成系统化、安全可控的防护闭环.

● 安全灰度撤销机制. 企业应当建立安全灰度发布机制. 在渠道方面, 建立可熔断的恢复发布渠道, 新版本先在灰度环境推送, 通过行为基线比对 (syscall 序列、网络连接资源占用), 让攻击者的行为暴露, 最终决定是否推送给全量客户. 此外建立构建以及部署阶段的组件签名验证机制, 一旦发现签名与官方的不一致, 即阻止流水线的运行或者系统的部署. 避免下游用户大规模的成品感染.

综上所述, 内源软件研发机制与安全灰度撤销机制管控了开源软件进入企业的入口以及产业组织交付成品的出口. 这不仅提升了软件开发过程的安全性和可控性, 还通过内部研发社区促进团队协作、知识共享与组件复用, 从而增强组织创新力和供应链韧性. 结合安全情报仓库和安全知识图谱, 产业组织能够在开发、集成和部署各阶

段主动管理潜在威胁, 构建系统化、可追溯、可验证的内源软件安全防护闭环, 为整个产业软件供应链的安全提供坚实支撑.

5.3 在成品使用阶段进行动态自适应防护

软件供应链攻击的最终目的在于使恶意逻辑在下游用户环境中被触发并完成执行. 然而, 无论攻击者采用何种复杂的技术手段来规避检测, 恶意代码在运行过程中都不可避免地在系统行为、资源消耗及交互模式等方面表现出与正常软件不同的异常特征. XZ Utils 后门事件即为典型案例: 高度隐蔽的后门程序最终因运行时导致 sshd 服务 CPU 占用异常而暴露, 印证了恶意代码的运行轨迹仍与正常软件存在可观测的差异. 因此, 下游防护的关键在于建立覆盖系统全链条的监测与审计机制, 从运行时行为、资源利用模式到用户交互特征进行多维度监控, 以实现对潜在威胁的及时识别与有效阻断.

5.3.1 运行时应用自我防护

运行时应用自我防护 (runtime application self-protection, RASP) 已成为软件供应链下游防御的重要技术路径, 其核心在于将安全逻辑深度嵌入应用运行时环境, 使应用能够在执行过程中实时感知、分析并主动阻断潜在攻击行为, 从而实现从外围防御向内生防御的转变[23,187,188]. 与依赖 Web 应用防火墙或入侵检测系统的传统模式不同, RASP 直接在攻击载荷执行点展开防御, 显著缩短响应链路, 并在应对零日漏洞和复杂攻击链时表现出更高的检测精度.

在实现机制上, RASP 通常通过运行时插桩与代理模块, 对函数调用、数据流和系统交互进行细粒度监控. 一旦检测到可疑输入或越权操作, 系统即可立即采取拒绝执行、修改返回值或中断会话等措施, 从而切断攻击链路. 这种运行时介入能力对于防御仅在特定条件下触发的隐蔽后门尤为关键.

在工程与学术实践方面, 百度开源的 OpenRASP 将监测逻辑深度嵌入 JVM, 重点审计文件读写、数据库查询与网络访问等敏感操作, 从而有效拦截命令注入和恶意上传等攻击[189]. He 等人[190]提出的 SecRASP 框架在检测准确性、零日漏洞防护和性能优化方面取得突破, 进一步的研究尝试将 RASP 与基于模型的异常检测相结合, 通过学习应用历史行为模式自动识别异常调用链并动态调整防御策略, 这对于金融交易、关键基础设施和 SaaS 平台尤为重要, 可显著降低运行时风险. 同时, Buildwatch 框架通过动态分析安装过程中的可疑工件与依赖行为揭示潜藏的后门释放路径[191], Wang 等人[192]基于信息流跟踪提出精细化检测方法, 可在数据包层面识别异常交互并揭示潜在攻击信息流, 而深度学习方法也被用于挖掘运行时行为特征以提升恶意软件检测准确性[193,194].

总体而言, RASP 及相关运行时检测技术正推动软件供应链安全从“外围防御”向“内生防御”转型, 但在大规模应用中仍面临性能开销、误报控制与可扩展性等挑战. 未来的发展方向应结合威胁建模、分级防御与智能化分析, 构建更具弹性和可持续性的下游运行时防护体系, 从而为关键业务系统提供更可靠的运行时安全保障.

5.3.2 安全漏洞修复

漏洞修复作为保障软件供应链安全的关键环节, 正加速从传统的人工驱动模式向智能化、自动化方向转型. 传统漏洞修复流程通常依赖安全专家进行漏洞分析[195]、开发者执行代码修改以及人工测试, 存在周期长、成本高、易引入兼容性问题及二次漏洞等局限. 为此, 近年来学术界与工业界提出了多种面向智能化和精准化的修复方案, 尤其以大语言模型驱动的自动程序修复 (automated program repair, APR) 技术为代表. 这类方法通过学习大量历史漏洞修复实例, 能够生成高质量的修复代码, 显著提升漏洞修复的自动化程度和效率.

在工程实践中, 为了确保修复的有效性和兼容性, 研究者提出了多种基于静态分析、依赖图与调用图的优化方法. 例如, CORAL 工具通过分析 Java 项目中的依赖关系和函数调用路径, 实现了在修复第三方库漏洞的同时保持项目功能完整性, 从而显著提高了修复成功率和编译通过率[196]. 在智能化修复方面, VulRepair 结合大规模预训练模型和 BPE 子词编码技术, 提升了漏洞代码生成的表示能力和词汇泛化能力, 能够在真实漏洞数据集上实现更高的修复准确率[197]. 此外, VulAdvisor 则通过自然语言提示和局部上下文注意力机制, 为开发者提供可解释且针对性强的修复建议, 有效辅助漏洞修复过程[198].

- 热修复. 在下游用户环境中, 快速遏制漏洞和潜在恶意逻辑的扩散成为关键挑战, 这使得热修复技术逐渐成为软件供应链安全防御体系中的重要组成部分. 与传统依赖完整软件升级或服务重启的修复方式不同, 热修复可

在程序运行期间, 通过动态替换函数或指令级代码, 快速缓解漏洞并屏蔽恶意逻辑, 从而显著降低补丁部署延迟. 这一能力对于高可用性系统、嵌入式设备及工业控制系统尤为重要, 能够在不中断系统运行的前提下应对紧急威胁. Linux 社区的内核热补丁机制, 如 Red Hat 的 kpatch、SUSE 的 kGraft 以及 Ubuntu 的 Livepatch 服务, 已能够动态加载安全补丁以替换内核函数, 实现实时修复[199].

近年来, 工业界和学术界涌现出多种高效热修复方案. 例如, AutoPatch 利用 LLVM 静态分析与逆向路径推导, 针对嵌入式设备自动生成功能等效的热补丁, 可在不重启设备或依赖专用硬件的前提下部署补丁, 平均修复延迟低于 12.7 μs, 对 90% 以上的真实 CVE 实现修复, 并且性能和内存开销均优于现有方案[200]. OSSPatch 则结合执行轨迹和符号信息, 实现对移动系统的热补丁动态注入, 在运行时覆盖漏洞函数或绕过攻击路径[201]. 为进一步提升热补丁的兼容性与适应性, Binary Patch Decomposition 方法将补丁拆解为最小依赖单元, 并基于上下文选择性应用, 有效缓解补丁对系统行为的干扰, 尤其适用于生产环境中对稳定性要求极高的系统[202].

● 镜像动态自愈. 在云原生系统层面, 应当为安全白名单回滚机制, 以滚动升级的方式在每日凌晨自动回滚到“最后已知良好版本”, 实现“日清日结”式的污染清除. 避免攻击者恶意镜像的持续驻留.

总体而言, 漏洞修复与热修复技术正推动软件供应链安全从传统人工修复向智能化、动态化方向演进. 智能化修复能够提高漏洞修复的准确性与效率, 而热修复则在下游运行环境中提供即时防护, 二者结合能够显著降低漏洞利用风险并提升系统整体安全性. 然而, 仍需持续优化补丁生成与部署策略, 以在保证系统可用性与兼容性的前提下, 实现对大规模软件生态的高效安全防护[203].

5.3.3 基于大模型的供应链安全治理

近年来, 大语言模型的快速发展为软件供应链安全提供了新的智能化防护手段. 在下游终端的攻击检测中, 大语言模型表现出显著优势, 包括对代码和执行上下文的深度理解、潜在漏洞的精确识别以及动态安全策略的生成. 然而, 大语言模型在应用中仍面临模型泛化能力、推理可靠性和可解释性等挑战. 通过微调 (fine-tuning) 及检索增强生成 (retrieval-augmented generation, RAG) 机制, 大语言模型能够在保持高精度检测的同时提供可追溯的推理依据, 从而提升系统的可解释性和用户信任度.

在漏洞检测方面, 大语言模型能够结合源代码与 CVE 描述信息, 精准定位潜在安全风险, 如缓冲区溢出、SQL 注入等. Wu 等人[32]提出的 VFFinder 方法, 通过定制上下文学习, 使大语言模型能够提取关键实体并与源代码进行优先级匹配, 提高了漏洞函数定位的准确性. 类似地, Li 等人[204]通过微调预训练模型开发的漏洞检测器, 在网络安全环境下实现了对异常行为与漏洞的高效识别. 此外, 大语言模型能够生成安全代码片段替换存在风险的原始代码, 简化漏洞修复流程、降低潜在攻击风险, 并提升代码健壮性和开发效率.

在运行时防护方面, 传统的 RASP 方法通常依赖规则匹配和浅层语法特征, 难以覆盖复杂的跨阶段攻击和提示注入行为. 为此, Meta 提出的系统级防护框架 LlamaFirewall 引入大模型语义推理能力, 为 LLM Agent 构建多层次运行时防护系统. 该框架整合 PromptGuard 2、AlignmentCheck 和 CodeShield 这 3 大核心组件: PromptGuard 2 基于 DeBERTa 的小型分类模型, 可高精度拦截通用越狱式提示注入攻击; AlignmentCheck 利用大型语言模型对 Agent 行为链条进行推理审计, 有效识别任务偏移与目标劫持; CodeShield 执行多语言语义检测, 覆盖 50 余类 CWE 漏洞, 并可在毫秒级响应中实现高准确率扫描[205]. 通过策略驱动的模块整合, LlamaFirewall 提供了系统级、可扩展且可审计的运行时安全防护方案.

进一步而言, 大语言模型可运行时动态识别异常行为、生成安全响应策略 (如替换漏洞代码、注入校验逻辑) 以即时防护、降低攻击扩散风险. 结合多种技术, 其在复杂攻击场景中提升检测精度与鲁棒性, 为终端提供智能动态防护能力, 增强软件供应链终端安全性, 为高可用系统动态防御提供可行路径[29].

5.4 产业界开源软件供应链防护工具总结

从产业实践视角来看, 防护工具的建设已成为提升软件供应链安全韧性的核心抓手. 表 8 系统性地总结了当前产业界在软件供应链安全领域的代表性工具, 覆盖了软件生命周期的多个阶段, 包括安全设计、代码审查、物料扫描、安全测试、渗透测试、运行时入侵检测以及缺陷修复等环节.

表 8 产业界软件供应链攻击防护工具

功能	生命周期	引入	传播	演化	发作	名称	目标	许可证
安全设计	设计	—	—	√	√	微软Threat Modeling Tool①	—	商业
	设计	—	—	√	√	OWASP Threat Dragon②	—	Apache 2.0
	设计	—	—	√	√	Pytm③	Python	MIT
	设计	—	—	√	√	Easponge	—	MPL-2.0
	设计	—	—	√	√	OpenThreatModel④	—	CC-BY-SA 4.0
	设计	—	—	√	√	Threat-composer	—	Apache 2.0
代码审查	编码	√	—	√	—	Crucible	—	商业
	编码	√	—	√	—	阿里 Coding Guidelines	Java	商业
	编码	√	—	√	—	Retire.js	Chrome, Firefox	Apache 2.0
	编码/构建	√	—	√	—	pmd	主流开发语言	Apache 2.0
	编码/构建	√	—	√	—	Semgrep	—	商业
	编码/构建	√	—	√	—	Codiga	主流开发语言	商业
	编码/构建	√	—	√	—	SonarQube	主流开发语言	LGPL-3.0
	编码/构建	√	—	√	—	Gosec	Go	Apache 2.0
	编码/构建	√	—	√	—	悬镜 SCA, Codesec	主流开发语言	商业
	编码/构建	√	—	√	—	中国联通数字化研发平台⑤	主流开发语言	商业
	编码/构建	√	—	√	—	Obfuscation_detection	Python	GPL-2.0
	编码/构建	√	—	√	—	Malicious-code-ruleset	主流开发语言	MIT
物料扫描	构建	√	—	√	√	opensca	应用, 容器, 代码包	Apache 2.0
	构建	√	—	√	√	Trivy	容器, 文件, Git 仓库, 虚拟机, Kubernetes	Apache 2.0
	构建	√	—	√	√	Dependency-track	应用, 容器, 代码包	Apache 2.0
	构建	√	—	√	√	中国联通数字化研发平台	应用, 容器, 代码包	商业
	构建	√	—	√	√	Sbom-tool⑥	应用, 容器, 代码包	MIT
安全测试	测试	√	√	√	√	Cloudgoat	云安全测试演练	BSD 3-Clause
	测试	√	√	√	√	Kubernetes-goat	云原生安全测试	MIT
渗透测试	测试/运维	—	—	√	√	Nmap	操作系统端口	GPL-2.0
	测试/运维	—	—	√	√	RustScan	操作系统端口	GPL-3.0
	测试/运维	—	—	√	√	Burp Suite套件⑦	Web系统渗透测试	社区版
	测试/运维	—	—	√	√	Sqlmap	SQL渗透测试	GPL-2.0
	测试/运维	—	—	√	√	OpenVAS Scanner	漏洞渗透测试	GPL-2.0
	测试/运维	—	—	√	√	ZAP⑧	漏洞渗透测试	Apache 2.0
运行时入侵检测	运维	—	—	√	√	Elkeid	Kubernetes平台应用	Apache2.0
	运维	—	—	√	√	Snort3	操作系统端口	GPL-2.0
	运维	—	—	√	√	Suricata	操作系统	GPL-2.0
	运维	—	—	√	√	Openedr	操作系统	CASL
	运维	—	—	√	√	OpenRASP	Web应用	Apache 2.0
缺陷修复	编码	√	—	√	—	CVEfixes	针对NVD漏洞进行修复	MIT
	编码	√	—	√	—	Snyk	漏洞修复	商业

注: ① Microsoft Threat Modeling Tool, https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool
② OWASP Threat Dragon, https://owasp.org/www-project-threat-dragon/
③ PYTM, https://pytm.readthedocs.io/
④ The Open Threat Model (OTM) Standard, https://www.iriusrisk.com/the-open-threat-model-standard
⑤ 中国联通数字化研发平台, https://www.yicai.com/news/102386162.html
⑥ Sbom-tool, https://github.com/microsoft/sbom-tool.git
⑦ Burp Suite, https://portswigger.net/burp
⑧ Zed Attack Proxy, https://www.zaproxy.org/

在前期阶段, 安全设计工具 (如微软 Threat Modeling Tool、OWASP Threat Dragon) 与代码审查工具 (如 SonarQube、Semgrep、pmd 等) 形成了较为完整的组合. 前者通过威胁建模和安全策略分析, 在设计阶段提前识别潜在攻击面; 后者则依托静态分析与模式匹配, 在源码层面发现漏洞与不安全实现. 与此同时, 物料扫描工具 (如 Trivy、Dependency-track、opensca) 对依赖包、构建产物及容器环境进行检测[206], 从而增强组件级安全可视性并降低供应链投毒风险.

在测试与运维环节, 安全测试与渗透测试工具为系统提供了多维度的安全评估. Cloudgoat、Kubernetes-goat 等工具通过模拟云环境与云原生场景的攻击, 帮助企业在真实运行环境中验证防护策略的有效性; 而 Nmap、Burp Suite、Sqlmap 等渗透测试工具则面向操作系统、Web 应用与数据库开展深度漏洞挖掘. 进一步地, 运行时防护工具 (如 Elkeid、Snort3、Suricata) 在云原生系统上线后提供实时入侵检测, 能够对异常流量与恶意行为进行监测与阻断. 这类工具强调高灵敏度与低误报率, 在供应链攻击多阶段演化与持久化入侵的背景下尤为关键. 同时, 缺陷修复工具 (如 Patch、Dependabot、Renovate) 通过自动化补丁生成与依赖更新, 显著缩短漏洞从发现到修复的时间窗口, 使整个防护体系能够保持动态演进与持续强化.

总体而言, 产业界的软件供应链防护工具已形成“设计-开发-测试-运维-修复”的全生命周期覆盖格局, 构建了多层次、可扩展的纵深防御体系. 这些工具在功能分布上与生命周期环节高度契合, 把攻击者依赖的流程合规性反向利用, 通过开源阶段的协同治理、产业闭源研发阶段持续合规与攻击面收敛、成品使用阶段的动态自适应防护这 3 层机制, 让被污染的软件在全生命周期传播中持续显形、快速撤销、自动恢复. 这使得攻击者精心潜伏的沉默成本无法在产业研发价值链路中有效兑现, 达到研发流程合规约束以及所引发的内生矛盾之间的再平衡.

6 研究挑战与未来展望

6.1 现有研究挑战

开源软件效能的“自我强化”与攻击手段隐蔽、传播链条复杂化的“自我反噬”之间, 已成一道无法回避的内生裂缝. 在此矛盾的不断挤压之下, 产业界的真实反应是以“把流程做合规”作为最低成本的止损动作: 把安全检查塞进 CI/CD、把 SBOM 填进审计表、把许可证扫描写进门禁脚本——看似动作繁多, 本质仍是被动合规. 这套“合规式防御”把风险转嫁给了流程本身, 把漏洞藏进了更多文档和签字, 却无法解决攻击入口不断翻新、潜伏周期不断拉长的根本问题.

攻击者早已看穿这套逻辑. 开源社区成为首选突破口: 攻击者只需成功完成一次精心伪造的代码提交, 就能把恶意逻辑写进主干分支; 社区若提高门槛, 便直接挫伤自由贡献的活力, 反而让项目失去自我净化能力. 企业端在资源与时间双重压力下, 只能对海量依赖做“抽样体检”. 污染仓库、仿冒镜像、人工智能模型等新攻击面沿着“开源共建-产业闭源研发-成品使用”的缝隙一路蔓延.

防御层面, 当前策略仍呈现出碎片化与被动性. 一方面, 供应链治理机制分散在开发者、仓库维护者与终端用户等不同角色之间, 缺乏统一标准与跨域协同, 导致漏洞或后门可能长期潜伏. 另一方面, 现有检测技术依赖既有模式库, 自动化程度有限, 对 CI 流程中的增量扫描不足, 对物联网固件与机器学习模型等新兴场景缺乏成熟手段. 整体来看, 现有防御往往聚焦于单点环节, 缺乏覆盖研发、分发与运维全链路的系统性设计, 难以形成主动、协同与前瞻的防御格局. 因此, 未来亟需在制度规范、跨域治理、自动化检测与智能化分析等方面构建综合性框架, 以提升供应链整体的安全可控性.

综上所述, 开源软件供应链攻击在攻击路径的多样性、潜伏性与隐蔽性, 以及防御体系的碎片化与滞后性之间形成了长期而难以调和的矛盾. 这种矛盾使得产业在保障研发效率与实现全面防护之间面临严峻挑战, 也凸显了建立跨社区、跨产业、跨领域协同防御框架的紧迫性.

6.2 未来展望

为系统性地应对开源软件供应链安全挑战, 未来的发展路径将围绕跨环节、自动化与智能化的综合治理体系展开, 涵盖治理框架、管控流程与赋能技术等多个层面. 为此, 防御阵线必须建立“更加靠前”“更加动态”“更加智

能”的防御机制与手段.

“更加靠前”是指实现从源头开始的内生安全与自动化管控. 这意味着在软件的需求分析与架构设计阶段, 就需通过威胁建模等方法系统性地识别和规避潜在风险, 将安全策略内化为架构决策.

“更加动态”是指在开发阶段建立伴随研发动作的安全监督动作, 通过“策略即代码”和“安全即代码”将合规要求自动化地嵌入开发者的日常工作环境与 CI/CD 流水线中, 实现对开源组件引入、代码提交及构建过程的强制性、无缝式安全管控, 降低研发者的安全疲劳, 避免产业界在市场交付驱动下做出安全折扣.

“更加智能”是指由先进的智能化检测与响应技术作为支撑. 利用深度学习等智能化方法对代码语义、社区行为及供应链元数据进行深度分析, 实现从基于特征的检测向基于风险的预测转变. 通过融合静态分析、动态监控与行为审计等多种技术手段, 构建具备上下文感知能力的多维度防护体系, 以自动化响应机制应对日益隐蔽和复杂的跨环节攻击手法.

总结而言, 软件供应链安全体系必将演进为一个有机整体, 未来的软件供应链攻击防护目标不是造一堵“绝对攻不破”的墙, 而是让攻击者每成功污染一次都必须付出更高的暴露成本, 而防御方只需维持一个低成本、可持续、可进化的动态安全平衡, 才能从根本上提升软件供应链的透明性、可信性与韧性, 应对日益严峻的安全挑战.

7 结 论

本文以产业研发流程为切入点, 构建了“开源共建威胁产生-产业闭源研发攻击演化传播-成品使用攻击发作”三阶段演化框架, 首次将开源生态、产业闭源研发与成品下游终端运行环境纳入统一分析范式, 深刻揭示了开源软件供应链效能提升的“自我强化”与其攻击手段合规和破坏性放大“自我反噬”的内生矛盾, 以及在此矛盾挤压下, 产业组织做出的流程合规的隐性共识. 在攻击层面, 研究提出了产业研发视角下的攻击演化模型, 将社会工程学、代码投毒、CI/CD 劫持及终端载荷释放等离散手段系统性地整合, 揭示了攻击路径的链式放大效应及信任链滥用机制, 并通过对 XZ Utils、Codecov、SolarWinds 等典型事件的分析, 深入剖析了多阶段载荷释放、隐蔽 C2 通道构建及云服务滥用等新型战术. 在防御层面, 研究总结了产业界在治理实践中采取的措施, 包括 DevSecOps、内源研发、SBOM、RASP 及大模型驱动的智能化管理, 通过建立主动暴露风险的安全机制, 实现隐性共识与内生矛盾之间的安全再平衡. 同时, 当前治理仍存在碎片化、检测滞后及跨域协同不足等核心挑战, 未来可进一步探索“更靠前”“更动态”“更智能”的安全体系. 总体而言, 本文不仅填补了产业研发视角下软件供应链攻击研究的学术空白, 也为构建可验证、可防控、可信赖的供应链安全体系奠定了理论与技术基础, 对学术研究与产业实践均具有重要价值.

References

[1] Abrahão S, Staron M, Serebrenik A, Penzenstadler B, Capilla R. Open source software: Communities and quality. IEEE Software, 2023, 40(4): 96–99. [doi: 10.1109/MS.2023.3270779]

[2] Ji SL, Wang QY, Chen AY, Zhao BB, Ye T, Zhang XH, Wu JZ, Li Y, Yin JW, Wu YJ. Survey on open-source software supply chain security. Ruan Jian Xue Bao/Journal of Software, 2023, 34(3): 1330–1364 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6717.htm [doi: 10.13328/j.cnki.jos.006717]

[3] Luo D, Wu RC. Current status and risk analysis of domestic open-source software. Communications World, 2022(15): 33–34 (in Chinese with English abstract). [doi: 10.13571/j.cnki.cww.2022.15.008]

[4] Linåker J, Robles G, Bryant D, Muto S. Open source software in the public sector: 25 Years and still in its infancy. IEEE Software, 2023, 40(4): 39–44. [doi: 10.1109/MS.2023.3266105]

[5] Jiang HZ, Shi L, Che MR, Zhang YX, Wang Q. Bringing open source communication and development together: A cross-platform study on Gitter and GitHub. IEEE Trans. on Software Engineering, 2024, 50(11): 2807–2826. [doi: 10.1109/TSE.2024.3410292]

[6] Sánchez AB, Parejo JA, Segura S, Durán A, Papadakis M. Mutation testing in practice: Insights from open-source software developers. IEEE Trans. on Software Engineering, 2024, 50(5): 1130–1143. [doi: 10.1109/TSE.2024.3377378]

[7] Gao K, He H, Xie B, Zhou MH. Survey on open source software supply chains. Ruan Jian Xue Bao/Journal of Software, 2024, 35(2): 581–603 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6975.htm [doi: 10.13328/j.cnki.jos.006975]

[8] Synopsys. 2024 open source security and risk analysis report. Synopsys, 2024 (in Chinese). https://www.blackduck.com/content/dam/black-duck/zh-cn/reports/rep-ossra-2024-ch.pdf

[9] Chen LQ, Li NH, Liang KT, Schneider S. Computer Security—ESORICS 2020. Cham: Springer, 2020. [doi: 10.1007/978-3-030-59013-0]

[10] Meng YX, Long YT, Wang W, Sun Q, Jing Q, Tan ZY, Tang XY, Ju DY. 2024 China open source development Status. COPU, 2024 (in Chinese). https://copu-oss.org.cn/download/showdownload.php?id=32

[11] Setitra MA, Fan MY, Benkhaddra I, Bensalem ZEA. DoS/DDoS attacks in software defined networks: Current situation, challenges and future directions. Computer Communications, 2024, 222: 77–96. [doi: 10.1016/j.comcom.2024.04.035]

[12] Ladisa P, Plate H, Martinez M, Barais O. SoK: Taxonomy of attacks on open-source software supply chains. In: Proc. of the 2023 IEEE Symp. on Security and Privacy. San Francisco: IEEE, 2023. 1509–1526. [doi: 10.1109/SP46215.2023.10179304]

[13] Betul G, Leonardo A, Basel H. Software supply chain: A taxonomy of attacks, mitigations and risk assessment strategies. Journal of Information Security and Applications, 2026, 97: 104324. [doi: 10.1016/j.jisa.2025.104324]

[14] Ladisa P, Ponta SE, Sabetta A, Martinez M, Barais O. Journey to the center of software supply chain attacks. IEEE Security & Privacy, 2023, 21(6): 34–49. [doi: 10.1109/MSEC.2023.3302066]

[15] Siadati H, Jafarikhah S, Sahin E, Hernandez TB, Tripp E, Khryashchev D, Kharraz A. Devphish: Exploring social engineering in software supply chain attacks on developers. In: Proc. of the 15th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conf. Yorktown Heights: IEEE, 2024. 517–523. [doi: 10.1109/UEMCON62879.2024.10754708]

[16] Kshetri N. Economics of supply chain cyberattacks. IT Professional, 2022, 24(3): 96–100. [doi: 10.1109/mitp.2022.3172877]

[17] Levy E. Poisoning the software supply chain. IEEE Security & Privacy, 2003, 1(3): 70–73. [doi: 10.1109/MSECP.2003.1203227]

[18] Alami A, Pardo R, Cohn ML, Wąsowski A. Pull request governance in open source communities. IEEE Trans. on Software Engineering, 2022, 48(12): 4838–4856. [doi: 10.1109/TSE.2021.3128356]

[19] ReversingLabs. The state of software supply chain security 2024. 2024. https://www.reversinglabs.com/sscs-report-2024

[20] Imtiaz N, Thorn S, Williams L. A comparative study of vulnerability reporting by software composition analysis tools. In: Proc. of the 15th ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement. Bari: ACM, 2021. 5. [doi: 10.1145/3475716.3475769]

[21] Yang BY, Cai ZJ, Liu FL, Le B, Zhang LM, Bissyandé TF, Liu Y, Tian HY. A survey of LLM-based automated program repair: Taxonomies, design paradigms, and applications. arXiv:2506.23749, 2025.

[22] Li Y, Shezan FH, Wei BM, Wang G, Tian Y. SoK: Towards effective automated vulnerability repair. In: Proc. of the 34th USENIX Security Symp. Seattle: USENIX Association, 2025. 4441–4462.

[23] Wang LM, Bu L, Ma LZ, Yu XF, Shen NG. Attack detection method based on indicator dependence model construction and monitoring. Ruan Jian Xue Bao/Journal of Software, 2023, 34(6): 2641–2668 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6847.htm [doi: 10.13328/j.cnki.jos.006847]

[24] Zhan Q, Pan SY, Hu X, Bao LF, Xia X. Survey on vulnerability awareness of open source software. Ruan Jian Xue Bao/Journal of Software, 2024, 35(1): 19–37 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6935.htm [doi: 10.13328/j.cnki.jos.006935]

[25] Wang Y, Wu YX, Gao T, Chen ZY, Xu C, Yu H, Zhang CZ. Survey on governance technology of open-source software library ecosystem: Twenty years of progress. Ruan Jian Xue Bao/Journal of Software, 2024, 35(2): 629–674 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6983.htm [doi: 10.13328/j.cnki.jos.006983]

[26] Wen XC, Gao CY, Ye JX, Li YC, Tian ZH, Jia Y, Wang X. Meta-path based attentional graph learning model for vulnerability detection. IEEE Trans. on Software Engineering, 2024, 50(3): 360–375. [doi: 10.1109/TSE.2023.3340267]

[27] Zhang CY, Liu B, Xin Y, Yao LW. CPVD: Cross project vulnerability detection based on graph attention network and domain adaptation. IEEE Trans. on Software Engineering, 2023, 49(8): 4152–4168. [doi: 10.1109/TSE.2023.3285910]

[28] Zhang WX, Kong XR, Dewitt C, Braunl T, Hong JB. A study on prompt injection attack against LLM-integrated mobile robotic systems. In: Proc. of the 35th IEEE Int'l Symp. on Software Reliability Engineering Workshops. Tsukuba: IEEE, 2024. 361–368. [doi: 10.1109/ISSREW63542.2024.00103]

[29] Pasini S, Kim J, Aiello T, Lozoya RC, Sabetta A, Tonella P. Evaluating and improving the robustness of security attack detectors generated by LLMs. arXiv:2411.18216, 2024.

[30] Bandara E, Shetty S, Mukkamala R, Rahman A, Foytik P, Liang XP, de Zoysa K, Keong NW. DevSec-GPT—Generative-AI (with custom-trained meta's Llama2 LLM), blockchain, NFT and PBOM enabled cloud native container vulnerability management and pipeline verification platform. In: Proc. of the 2024 IEEE Cloud Summit. Washington: IEEE, 2024. 28–35. [doi: 10.1109/Cloud-Summit61220.2024.00012]

[31] Traykov K. A framework for security testing of large language models. In: Proc. of the 12th IEEE Int'l Conf. on Intelligent Systems. Varna: IEEE, 2024. 1–7. [doi: 10.1109/IS61756.2024.10705238]

[32] Wu YL, Wen M, Yu ZL, Guo XC, Jin H. Effective vulnerable function identification based on CVE description empowered by large language models. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Automated Software Engineering. Sacramento: IEEE, 2024. 393–405.

[33] Shen ZD, Chen S. A survey of automatic software vulnerability detection, program repair, and defect prediction techniques. Security and Communication Networks, 2020, 2020: 8858010. [doi: 10.1155/2020/8858010]

[34] Chen WB, Li JY, Ma JQ, Conradi R, Ji JZ, Liu CN. A survey of software development with open source components in Chinese software industry. In: Proc. of the 2007 Int'l Conf. on Software Process. Minneapolis: Springer, 2007. 208–220. [doi: 10.1007/978-3-540-72426-1_18]

[35] Manan WNW, Kahar MNM, Ali NM. A survey on current malicious JavaScript behavior of infected Web content in detection of malicious Web pages. IOP Conf. Series: Materials Science and Engineering, 2020, 769: 012074. [doi 10.1088/1757-899X/769/1/012074]

[36] Jin ZF, Zhang YW, Ye WH, Zhang H, Shao D. Research on application of DevOps in documentation towards full value delivery. Ruan Jian Xue Bao/Journal of Software, 2019, 30(10): 3127–3147 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5792.htm [doi: 10.13328/j.cnki.jos.005792]

[37] He XX, Zhang YQ, Liu QX. Survey of software supply chain security. Journal of Cyber Security, 2020, 5(1): 57–73 (in Chinese with English abstract). [doi: 10.19363/J.cnki.cn10-1380/tn.2020.01.06]

[38] Zhang DG, Li B, He P, Zhou HY. Characteristic study of open-source community based on software ecosystem. Computer Engineering, 2015, 41(11): 106–113 (in Chinese with English abstract). [doi: 10.3969/j.issn.1000-3428.2015.11.019]

[39] Sun ZY, Wu JZ, Ling X, Wei YL, Luo TY, Wu YJ. Research on key technologies of SBOM in software supply chain. Ruan Jian Xue Bao/Journal of Software, 2025, 36(6): 2604–2642 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/7308.htm [doi: 10.13328/j.cnki.jos.007308]

[40] Afaneh S, Al-Mousa MR, Al-Hamid HS, AL-Awasa BS, Alia M, Almimi H, Alkhatib AA. Security challenges review in agile and DevOps practices. In: Proc. of the 2023 Int'l Conf. on Information Technology. Amman: IEEE, 2023. 102–107. [doi: 10.1109/ICIT58056.2023.10226018]

[41] Valdés-Rodríguez Y, Hochstetter-Diez J, Diéguez-Rebolledo M, Bustamante-Mora A, Cadena-Martínez R. Analysis of strategies for the integration of security practices in agile software development: A sustainable SME approach. IEEE Access, 2024, 12: 35204–35230. [doi: 10.1109/ACCESS.2024.3372385]

[42] Ascenção C, Teixeira H, Gonçalves J, Almeida F. Large-scale agile security practices in software engineering. Information and Computer Security, 2024, 33(3): 344–361. [doi: 10.1108/ICS-07-2023-0136]

[43] Karanam R. Securing ci/cd pipelines: Strategies for mitigating risks in modern software delivery. Int'l Journal of Engineering and Technology Research (IJETR), 2024, 9(2): 1–9.

[44] Pan ZY, Shen WB, Wang XK, Yang YT, Chang R, Liu Y, Liu CW, Liu Y, Ren K. Ambush from all sides: Understanding security threats in open-source software CI/CD pipelines. IEEE Trans. on Dependable and Secure Computing, 2024, 21(1): 403–418. [doi: 10.1109/tdsc.2023.3253572]

[45] Düllmann TF, Paule C, van Hoorn A. Exploiting DevOps practices for dependable and secure continuous delivery pipelines. In: Proc. of the 4th Int'l Workshop on Rapid Continuous Software Engineering. Gothenburg: ACM, 2018. 27–30. [doi: 10.1145/3194760.3194763]

[46] Czekster RM. Continuous risk assessment in secure DevOps. arXiv:2409.03405, 2024.

[47] Throner S, Abeck S, Petrovic P, Hütter H. A DevOps approach to the mitigation of security vulnerabilities in runtime environments. In: Proc. of the 2023 IEEE Int'l Conf. on Service-oriented System Engineering. Athens: IEEE, 2023. 106–113. [doi: 10.1109/SOSE58276.2023.00019]

[48] Sermpezis E, Karapiperis D, Tjortjis C. Integration of security in the DevOps methodology. In: Proc. of the 15th Int'l Conf. on Information, Intelligence, Systems & Applications. Chania Crete: IEEE, 2024. 1–6. [doi: 10.1109/IISA62523.2024.10786669]

[49] Tatineni S. Compliance and audit challenges in DevOps: A security perspective. Int'l Research Journal of Modernization in Engineering Technology and Science, 2023, 5(10): 1306–1316. [doi: 10.56726/IRJMETS45309]

[50] Merkow MS. Practical Security for Agile and DevOps. New York: Auerbach Publications, 2022. [doi: 10.1201/9781003265566]

[51] Enoiu EP, Truscan D, Sadovykh A, Mallouli W. VeriDevOps software methodology: Security verification and validation for DevOps practices. In: Proc. of the 18th Int'l Conf. on Availability, Reliability and Security. Benevento: ACM, 2023. 135. [doi: 10.1145/3600160.3605054]

[52] Zhou X, Mao RF, Zhang H, Dai QM, Huang H, Shen HF, Li JY, Rong GP. Revisit security in the era of DevOps: An evidence-based

inquiry into DevSecOps industry. IET Software, 2023, 17(4): 435–454. [doi: 10.1049/sfw2.12132]

[53] Rajapakse RN, Zahedi M, Babar MA. An empirical analysis of practitioners' perspectives on security tool integration into DevOps. In: Proc. of the 15th ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement. Bari: ACM, 2021. 6. [doi: 10.1145/3475716.3475776]

[54] Ur Rahman AA, Williams L. Software security in DevOps: Synthesizing practitioners' perceptions and practices. In: Proc. of the 2016 IEEE/ACM Int'l Workshop on Continuous Software Evolution and Delivery. Austin: IEEE, 2016. 70–76.

[55] Sadovykh A, Widforss G, Truscan D, Enoiu EP, Mallouli W, Iglesias R, Bagnto A, Hendel O. VeriDevOps: Automated protection and prevention to meet security requirements in DevOps. In: Proc. of the 2021 Design, Automation & Test in Europe Conf. & Exhibition. Grenoble: IEEE, 2021. 1330–1333. [doi: 10.23919/DATE51398.2021.9474185]

[56] Mohan V, Othmane LB. SecDevOps: Is it a marketing buzzword?—Mapping research on security in DevOps. In: Proc. of the 11th Int'l Conf. on Availability, Reliability and Security. Salzburg: IEEE, 2016. 542–547. [doi: 10.1109/ARES.2016.92]

[57] Nikolov L, Aleksieva-Petrova A. Framework for integrating threat modeling into a DevOps pipeline for enhanced software development. In: Proc. of the 2024 Int'l Conf. on Software, Telecommunications and Computer Networks. Split: IEEE, 2024. 1–5. [doi: 10.23919/SoftCOM62040.2024.10721871]

[58] Bolling T, Lennon RG. Viewing DevOps security processes through an applied cyberpsychology lens. In: Proc. of the 2023 Cyber Research Conf. Ireland. IEEE, 2023. 1–6. [doi: 10.1109/Cyber-RCI59474.2023.10671453]

[59] Scanlon T, Morales J. Revelations from an agile and DevSecOps transformation in a large organization: An experiential case study. In: Proc. of the 2022 Int'l Conf. on Software and System Processes and Int'l Conf. on Global Software Engineering. Pittsburgh: ACM, 2022. 77–81. [doi: 10.1145/3529320.3529329]

[60] Xiao C. Novel malware XcodeGhost modifies xcode, infects apple iOS Apps and hits App store. 2015. https://unit42.paloaltonetworks.com/novel-malware-xcodeghost-modifies-xcode-infects-apple-ios-apps-and-hits-app-store/

[61] Northwood C. Today's JavaScript trash fire and pile on. 2018. https://cnorthwood.medium.com/todays-javascript-trash-fire-and-pile-on-f3efcf8ac8c7

[62] Tarr D. Backdoored dependency? Flatmap-stream-0.1.1 and flatmap-stream-0.1.2. 2025. https://github.com/dominictarr/event-stream/issues/115

[63] Grander D. Malicious code found in npm package event-stream downloaded 8 million times in the past 2.5 months. 2018. https://snyk.io/blog/malicious-code-found-in-npm-package-event-stream/

[64] Sharma A. PHP's git server hacked to add backdoors to PHP source code. 2021. https://www.bleepingcomputer.com/news/security/phps-git-server-hacked-to-add-backdoors-to-php-source-code/

[65] Ding YW, Cui BJ. Security analysis for third-party component dependency confusion: A risk assessment framework based on source code tree matching. In: Proc. of the 6th Int'l Conf. on Frontier Technologies of Information and Computer. Qingdao: IEEE, 2024. 243–250. [doi: 10.1109/ICFTIC64248.2024.10912978]

[66] Martínez J, Durán JM. Software supply chain attacks, a threat to global cybersecurity: Solarwinds' case study. Int'l Journal of Safety and Security Engineering, 2021, 11(5): 537–545. [doi: 10.18280/ijsse.110505]

[67] CCleaner. Security notification for CCleaner v5.33.6162 and CCleaner cloud v1.07.3191 for 32-bit windows users. 2017. https://www.ccleaner.com/knowledge/security-notification-ccleaner-v5336162-ccleaner-cloud-v1073191?cc-noredirect=

[68] Global Research & Analysis Team, Anti-malware Research Team. Operation ShadowHammer. 2019. https://securelist.com/operation-shadowhammer/89992/

[69] Lakshmanan R. Travis CI flaw exposes secrets of thousands of open source projects. 2021. https://thehackernews.com/2021/09/travis-ci-flaw-exposes-secrets-of.html

[70] Lisa V. NVIDIA's stolen code-signing certs used to sign malware. 2022. https://threatpost.com/nvidias-stolen-code-signing-certs-sign-malware/178784/

[71] Jonathan L. Popular Codecov code coverage tool hacked to steal Dev credentials. 2021. https://www.privacy.com.sg/cybersecurity/popular-codecov-code-coverage-tool-hacked-to-steal-dev-credentials/

[72] Phan B. June 20 Incident Details and Remediation. 2023. https://jumpcloud.com/blog/security-update-june-20-incident-details-and-remediation

[73] Cybersecurity & Infrastructure Security Agency. Reported supply chain compromise affecting XZ utils data compression library, CVE-2024-3094. 2024. https://www.cisa.gov/news-events/alerts/2024/03/29/reported-supply-chain-compromise-affecting-xz-utils-data-compression-library-cve-2024-3094

[74] Lakshmanan R. Newly discovered bugs in VSCode extensions could lead to supply chain attacks. 2021. https://thehackernews.com/

2021/05/newly-discovered-bugs-in-vscode.html
[75] Sangfor Qianlimu Security Technology Center. The mastermind behind four supply chain poisoning incidents in a year. 2025 (in Chinese). https://www.freebuf.com/articles/network/388355.html
[76] Ernestas N. Dozens of DockerHub Linux images still contain a critical XZ Utils backdoor. 2025. https://cybernews.com/security/xz-utils-backdoor-lurking-on-dockerhub/
[77] Li GL, Wu J, Li SH, Yang W, Li CL. Multitentacle federated learning over software-defined industrial Internet of things against adaptive poisoning attacks. IEEE Trans. on Industrial Informatics, 2023, 19(2): 1260–1269. [doi: 10.1109/TII.2022.3173996]
[78] Mao YW, Data D, Diggavi S, Tabuada P. Decentralized optimization resilient against local data poisoning attacks. IEEE Trans. on Automatic Control, 2025, 70(1): 81–96. [doi: 10.1109/TAC.2024.3424693]
[79] OWASP Foundation. CI/CD security cheat sheet. 2024. https://cheatsheetseries.owasp.org/cheatsheets/CI_CD_Security_Cheat_Sheet.html
[80] Krivelevich D, Gil O. OWASP top 10 CI/CD security risks. 2024. https://owasp.org/www-project-top-10-ci-cd-security-risks/
[81] Sonar. Maintainer burnout is real. Almost 60% of maintainers have quit or considered quitting maintaining one of their projects. 2023. https://www.sonarsource.com/blog/maintainer-burnout-is-real
[82] Roth E. Open source developer corrupts widely-used libraries, affecting tons of projects. 2022. https://www.theverge.com/2022/1/9/22874949/developer-corrupts-open-source-libraries-projects-affected
[83] Amin Y. The human toll of log4j maintenance. 2021. https://dev.to/yawaramin/the-human-toll-of-log4j-maintenance-35ap
[84] Gokkaya B, Aniello L, Halak B. Software supply chain: Review of attacks, risk assessment strategies and security controls. arXiv: 2305.14157, 2023.
[85] Moore M, Kuppusamy TK, Cappos J. Artemis: Defanging software supply chain attacks in multi-repository update systems. In: Proc. of the 39th Annual Computer Security Applications Conf. Austin: ACM, 2023. 83–97. [doi: 10.1145/3627106.3627129]
[86] Gelb Y. New technique to trick developers detected in an open-source supply chain attack. 2024. https://checkmarx.com/blog/new-technique-to-trick-developers-detected-in-an-open-source-supply-chain-attack/
[87] Cao A, Dolan-Gavitt B. What the fork? Finding and analyzing malware in GitHub forks. In: Proc. of the 2022 Workshop on Measurements, Attacks, and Defenses for the Web. 2022. [doi: 10.14722/madweb.2022.23001]
[88] SnykVulnerabilityDatabase. Prototype Pollution. 2023. https://security.snyk.io/vuln/SNYK-JS-LODASH-6139239
[89] Munoz A. The octopus scanner malware: Attacking the open source supply chain. 2020. https://github.blog/security/vulnerability-research/the-octopus-scanner-malware-attacking-the-open-source-supply-chain/
[90] Lakshmanan R. Popular PyPI package 'ctx' and PHP library 'phpass' hijacked to steal AWS keys. 2022. https://thehackernews.com/2022/05/pypi-package-ctx-and-php-library-phpass.html
[91] Freebuf. VS Code ETHcode extension suffers supply chain attack: Two lines of malicious code injected via GitHub pull request. 2025 (in Chinese). https://www.freebuf.com/articles/development/438383.html
[92] Zahan N, Zimmermann T, Godefroid P, Murphy B, Maddila C, Williams L. What are weak links in the npm supply chain? In: Proc. of the 44th IEEE/ACM Int'l Conf. on Software Engineering: Software Engineering in Practice. Pittsburgh: IEEE, 2022. 331–340. [doi: 10.1145/3510457.3513044]
[93] Wu YL, Yu ZL, Wen M, Li Q, Zou DQ, Jin H. Understanding the threats of upstream vulnerabilities to downstream projects in the maven ecosystem. In: Proc. of the 45th IEEE/ACM Int'l Conf. on Software Engineering. Melbourne: IEEE, 2023. 1046–1058. [doi: 10.1109/ICSE48619.2023.00095]
[94] Mir AM, Keshani M, Proksch S. On the effect of transitivity and granularity on vulnerability propagation in the maven ecosystem. In: Proc. of the 2023 IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering. IEEE, 2023. 201–211. [doi: 10.1109/SANER56733.2023.00028]
[95] Mortenson S. A repository demonstrating how you can sneak malicious code into GitHub PRs. 2017. https://github.com/mortenson/pr-sneaking
[96] Goyal R, Ferreira G, Kästner C, Herbsleb J. Identifying unusual commits on GitHub. Journal of Software: Evolution and Process, 2018, 30(1): e1893. [doi: 10.1002/smr.1893]
[97] Benedetti G, VerderameL, Merlo A. Automatic security assessment of GitHub actions workflows. In: Proc. of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses. Los Angeles: ACM, 2022. 37–45.
[98] Pathmanathan P, Chakraborty S, Liu XY, Liang YY, Huang FR. Is poisoning a real threat to LLM alignment? Maybe more so than you think. arXiv:2406.12091, 2024.
[99] Fu TC, Sharma M, Torr P, Cohen SB, Krueger D, Barez F. PoisonBench: Assessing large language model vulnerability to data

poisoning. arXiv:2410.08811, 2024.

[100] Hu Q, Xie XF, Chen S, Quan LL, Ma L. Large language model supply chain: Open problems from the security perspective. In: Proc. of the 34th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Clarion Hotel Trondheim Trondheim: ACM, 2025. 169–173. [doi: 10.1145/3713081.3731747]

[101] Carlini N, Jagielski M, Choquette-Choo CA, Paleka D, Pearce W, Anderson H, Terzis A, Thomas K, Tramèr F. Poisoning Web-scale training datasets is practical. In: Proc. of the 2024 IEEE Symp. on Security and Privacy. San Francisco: IEEE, 2024. 407–425. [doi: 10.1109/SP54263.2024.00179]

[102] Cotroneo D, Improta C, Liguori P, Natella R. Vulnerabilities in AI code generators: Exploring targeted data poisoning attacks. In: Proc. of the 32nd IEEE/ACM Int'l Conf. on Program Comprehension. Lisbon: IEEE, 2024. 280–292.

[103] Xu YC, Yao JR, Shu ML, Sun YC, Wu ZC, Yu N, Goldstein T, Huang FR. Shadowcast: Stealthy data poisoning attacks against vision-language models. In: Proc. of the 38th Int'l Conf. on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2024. 1841.

[104] VassilevA, OpreaA, Fordyce A, Anderson H. Adversarial machine learning: A taxonomy and terminology of attacks and mitigations. 2024. https://www.nist.gov/publications/adversarial-machine-learning-taxonomy-and-terminology-attacks-and-mitigations

[105] Amini MH, Moore E. How poisoned data can trick AI, and how to stop it. 2025. https://theconversation.com/how-poisoned-data-can-trick-ai-and-how-to-stop-it-256423

[106] Liu H, Zhou GG, Fu PH, Mao GH. Research of transferring attack based on supply chain network. Computer Science, 2013, 40(7): 98–101 (in Chinese with English abstract). [doi: 10.3969/j.issn.1002-137X.2013.07.022]

[107] Torres-Arias S, Ammula AK, Curtmola R, Cappos J. On omitting commits and committing omissions: Preventing git metadata tampering that (re)introduces software vulnerabilities. In: Proc. of the 25th USENIX Conf. on Security Symp. Austin: USENIX Association, 2016. 379–395.

[108] Boughton L, Miller C, Acar Y, Wermke D, Käastner C. Decomposing and measuring trust in open-source software supply chains. In: Proc. of the 44th ACM/IEEE Int'l Conf. on Software Engineering: New Ideas and Emerging Results (ICSE-NIER 2024). Lisbon: ACM, 2024. 57–61. [doi: 10.1145/3639476.3639775]

[109] Federrath IH. Typosquatting in programming language package managers [MS. Thesis]. Hamburg: University of Hamburg, 2016.

[110] Meyers JS, Tozer B. Bewear! Python typosquatting is about more than typos. 2020. https://www.iqt.org/library/bewear-python-typosquatting-is-about-more-than-typos

[111] Birsan A. Dependency confusion: How I hacked into Apple, Microsoft and dozens of other companies. 2021. https://medium.com/@alex.birsan/dependency-confusion-how-i-hacked-into-apple-microsoft-and-dozens-of-other-companies-4a5d60fec610

[112] Ohm M, Plate H, Sykosch A, Meier M. Backstabber's knife collection: A review of open source software supply chain attacks. In: Proc. of the 17th Int'l Conf. Detection of Intrusions and Malware, and Vulnerability Assessment. Lisbon: Springer, 2020. 23–43. [doi: 10.1007/978-3-030-52683-2_2]

[113] Truong MT. Typosquatting attacks and mitigations [MS. Thesis]. Sankt Augustin: University of Applied Science, 2023.

[114] GitHub. Cache action. 2025. https://github.com/actions/cache

[115] Khan A. The monsters in your build cache—GitHub actions cache poisoning. 2025. https://adnanthekhan.com/2024/05/06/the-monsters-in-your-build-cache-github-actions-cache-poisoning

[116] Unit42. GitHub actions supply chain attack: A targeted attack on coinbase expanded to the widespread tj-actions/changed-files incident: Threat assessment. 2025. https://unit42.paloaltonetworks.com/github-actions-supply-chain-attack/

[117] CISA. Russian foreign intelligence service (SVR) exploiting JetBrains TeamCity CVE globally. 2023. https://www.cisa.gov/news-events/cybersecurity-advisories/aa23-347a

[118] Brian C, Fredrick DL, Jacob W. Attacking the build through cross-build injection. 2007. https://img2.helpnetsecurity.com/dl/articles/fortify_attacking_the_build.pdf

[119] Adobe Systems Incorporated. Security advisory: Revocation of adobe code signing certificate. 2012. https://www.adobe.com/support/security/advisories/apsa12-01.html

[120] Mounesan M, Siadati H, Jafarikhah S. Exploring the threat of software supply chain attacks on containerized applications. In: Proc. of the 16th Int'l Conf. on Security of Information and Networks. Jaipur: IEEE, 2023. 1–8. [doi: 10.1109/SIN60469.2023.10474901]

[121] Wang SN, Zhao YJ, Hou XY, Wang HY. Large language model supply chain: A research agenda. ACM Trans. on Software Engineering and Methodology, 2025, 34(5): 147. [doi: 10.1145/3708531]

[122] Huang KF, Chen BH, Lu Y, Wu SS, Wang DJ, Huang YH, Jiang HW, Zhou ZT, Cao JM, Peng X. Lifting the veil on the large language model supply chain: Composition, risks, and mitigations. arXiv:2410.21218v1, 2024.

[123] Dong T, Xue MH, Chen GX, Holland R, Meng Y, Li SF, Liu Z, Zhu HJ. The philosopher's stone: Trojaning plugins of large language models. In: Proc. of the 32nd Annual Network and Distributed System Security Symp. San Diego: The Internet Society, 2025.

[124] Cybersecurity and Infrastructure Security Agency. Advanced persistent threat compromise of government agencies, critical infrastructure, and private sector organizations. 2021. https://www.cisa.gov/news-events/cybersecurity-advisories/aa20-352a

[125] Lakshmanan R. Critical open VSX registry flaw exposes millions of developers to supply chain attacks. 2025. https://thehackernews.com/2025/06/critical-open-vsx-registry-flaw-exposes.html

[126] GitHub. Oneinstack issue #487. 2023 (in Chinese). https://github.com/oneinstack/oneinstack/issues/487

[127] ThreatLabz. Anatomy of an ongoing drive-by-download campaign. 2013. https://www.zscaler.com/blogs/security-research/anatomy-ongoing-drive-download-campaign

[128] Mandiant. North Korea leverages SaaS provider in a targeted supply chain attack. 2023. https://cloud.google.com/blog/topics/threat-intelligence/north-korea-supply-chain

[129] Zanki K. Malware leveraging public infrastructure like GitHub on the rise. 2023. https://www.reversinglabs.com/blog/malware-leveraging-public-infrastructure-like-github-on-the-rise

[130] Ranjan I, Agnihotri RB. Ambiguity in cloud security with malware-injection attack. In: Proc. of the 3rd Int'l Conf. on Electronics, Communication and Aerospace Technology. Coimbatore: IEEE, 2019. 306–310. [doi: 10.1109/ICECA.2019.8821844]

[131] Damjanović DZ. Malicious code in the cloud. Vojnotehnički Glasnik, 2022, 70(3): 734–755. [doi: 10.5937/vojtehg70-37168]

[132] Rao BVS, Sharma V, Rathore N, Prasad D, Anandaram H, Soni G. A secure framework to prevent three-tier cloud architecture from malicious malware injection attacks. Int'l Journal of Cloud Applications and Computing, 2023, 13(1): 1–22. [doi: 10.4018/ijcac.317220]

[133] Alibaba Cloud. PyPi supply chain poisoning series incidents: Discord assists hacker attacks. 2021 (in Chinese). https://developer.aliyun.com/article/787142/

[134] Zanki K. Update: IconBurst npm software supply chain attack grabs data from Apps and websites. 2025. https://www.reversinglabs.com/blog/iconburst-npm-software-supply-chain-attack-grabs-data-from-apps-websites

[135] Snyk Research Team. Malicious package in @yandex-travel/ui. 2025. https://security.snyk.io/vuln/SNYK-JS-YANDEXTRAVELUI-3319915

[136] National Institute of Standards and Technology. CVE-2023-45311 detail. 2025. https://nvd.nist.gov/vuln/detail/CVE-2023-45311

[137] OneinStack. Malicious code found in domestic mirrors of mirrors.oneinstack.com. 2025 (in Chinese). https://github.com/oneinstack/oneinstack/issues/511

[138] Antiy Labs. Analysis of espionage trojan propagation attacks via GitHub. 2024 (in Chinese). https://www.freebuf.com/articles/network/395369.html

[139] Qi'anxin CodeGuard. Software supply chain poisoning—Analysis of malicious NPM packages (Part 2). 2024 (in Chinese). https://www.freebuf.com/articles/database/395549.html

[140] Xmirror Security Intelligence Center. Malicious NPM Package (fix-this) launches reverse shell remote control poisoning. 2025 (in Chinese). https://www.xmirror.cn/particulars?id=3225&type=dt

[141] Fiedler M. Malware distribution and domain abuse. 2025. https://blog.pypi.org/posts/2024-04-10-domain-abuse/

[142] Xmirror Security Monthly Report. Release of open-source supply chain poisoning monthly report April 2024. 2024 (in Chinese). https://opensca.xmirror.cn/resources/particulars?id=143

[143] Xmirror Security Intelligence Center (Supply Chain Poisoning Warning). Malicious Python package disguised as HTTP component launches CStealer espionage backdoor attack. 2024 (in Chinese). https://www.freebuf.com/news/400487.html

[144] Huorong Security. GitHub open-source projects poisoned: Backdoor viruses spread along with development processes. 2025 (in Chinese). https://huorong.cn/document/tech/vir_report/1802/

[145] Sonatype. State of the software supply chain. 2025. https://www.sonatype.com/state-of-the-software-supply-chain/Introduction

[146] Prates L, Pereira R. DevSecOps practices and tools. Int'l Journal of Information Security, 2025, 24(1): 11. [doi: 10.1007/s10207-024-00914-z]

[147] Froh FN, Gobbi MF, Kinder J. Differential static analysis for detecting malicious updates to open source packages. In: Proc. of the 2023 Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses. Copenhagen: ACM, 2023. 41–49. [doi: 10.1145/3605770.3625211]

[148] Zheng XY, Wei C, Wang SN, Zhao YJ, Gao PM, Zhang YC, Wang KL, Wang HY. Towards robust detection of open source software supply chain poisoning attacks in industry environments. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Automated Software Engineering. Sacramento: ACM, 2024. 1990–2001. [doi: 10.1145/3691620.3695262]

[149] Shariffdeen R, Hassanshahi B, Mirchev M, El Husseini A, Roychoudhury A. Detecting Python malware in the software supply chain

with program analysis. In: Proc. of the 47th IEEE/ACM Int'l Conf. on Software Engineering: Software Engineering in Practice. Ottawa: IEEE, 2025. 203–214. [doi: 10.1109/ICSE-SEIP66354.2025.00024]

[150] Raitsis T, Elgazari Y, Toibin GE, Lurie Y, Mark S, Margalit O. Code obfuscation: A comprehensive approach to detection, classification, and ethical challenges. Algorithms, 2025, 18(2): 54. [doi: 10.3390/a18020054]

[151] Mao TY, Wang XY, Chang R, Shen WB, Ren K. Software supply chain analysis techniques for Java ecosystem. Ruan Jian Xue Bao/Journal of Software, 2023, 34(6): 2628–2640 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6852.htm [doi: 10.13328/j.cnki.jos.006852]

[152] Duan RA, Alrawi O, Kasturi RP, Elder R, Saltaformaggio B, Lee W. Towards measuring supply chain attacks on package managers for interpreted languages. In: Proc. of the 28th Annual Network and Distributed System Security Symp. The Internet Society, 2021.

[153] Fass A, Krawczyk RP, Backes M, Stock B. JaSt: Fully syntactic detection of malicious (obfuscated) JavaScript. In: Proc. of the 15th Int'l Conf. Detection of Intrusions and Malware, and Vulnerability Assessment. Saclay: Springer, 2018. 303–325. [doi: 10.1007/978-3-319-93411-2_14]

[154] He XC, Xu L, Cha CL. Malicious JavaScript code detection based on hybrid analysis. In: Proc. of the 25th Asia-Pacific Software Engineering Conf. Nara: IEEE, 2018. 365–374. [doi: 10.1109/APSEC.2018.00051]

[155] Alibaba Group AI Governance Initiative. White paper on generative ai governance: Robustness, interpretability, and misuse prevention. 2025 (in Chinese). https://s.alibaba.com/cn/aaigWhitePaperDetails/9x5JBSX2fyLEZWkRnKQ8f

[156] National Institute of Standards and Technology. AI risk management framework. 2025. https://www.nist.gov/itl/ai-risk-management-framework

[157] OWASP Foundation. LLM04: 2025 data and model poisoning. 2025. https://genai.owasp.org/llmrisk/llm042025-data-and-model-poisoning/

[158] Ishibashi Y, Shimodaira H. Knowledge sanitization of large language models. arXiv:2309.11852, 2023.

[159] Yu L, Do V, Hambardzumyan K, Cancedda N. Robust LLM safeguarding via refusal feature adversarial training. In: Proc. of the 13th Int'l Conf. on Learning Representations. 2025.

[160] Zhao S, Wu XB, Nguyen CD, Jia YH, Jia MHZ, Feng YC, Tuan LA. Unlearning backdoor attacks for LLMs with weak-to-strong knowledge distillation. In: Findings of the Association for Computational Linguistics: ACL 2025. Vienna: ACL, 2025. 4937–4952. [doi: 10.18653/v1/2025.findings-acl.255]

[161] Zeng SL, He PF, Guo K, Zheng TQ, Lu HQ, Xing Y, Liu H. Towards context-robust LLMs: A gated representation fine-tuning approach. In: Proc. of the 63rd Annual Meeting of the Association for Computational Linguistics. Vienna: ACL, 2025. 10262–10276. [doi: 10.18653/v1/2025.acl-long.506]

[162] Agrawal A, Alazraki L, Honarvar S, Rei M. Enhancing LLM robustness to perturbed instructions: An empirical study. arXiv:2504.02733, 2025.

[163] Dai QM, Mao RF, Huang H, Rong GP, Shen HF, Shao D. DevSecOps: Exploring practices of realizing continuous security in DevOps. Ruan Jian Xue Bao/Journal of Software, 2021, 32(10): 3014–3035 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6276.htm [doi: 10.13328/j.cnki.jos.006276]

[164] Carter K. Francois raynaud on DevSecOps. IEEE Software, 2017, 34(5): 93–96. [doi: 10.1109/MS.2017.3571578]

[165] Kumar R, Goyal R. Modeling continuous security: A conceptual model for automated DevSecOps using open-source software over cloud (ADOC). Computers & Security, 2020, 97: 101967. [doi: 10.1016/j.cose.2020.101967]

[166] Tomas N, Li JY, Huang H. An empirical study on culture, automation, measurement, and sharing of DevSeCops. In: Proc. of the 2019 Int'l Conf. on Cyber Security and Protection of Digital Services. Oxford: IEEE, 2019. 1–8. [doi: 10.1109/CyberSecPODS.2019.8884935]

[167] Haverinen H, Janhunen T, Päivärinta T, Lempinen S, Kaartinen S, Merilä S. Automating cybersecurity compliance in DevSecOps with open information model for security as code. In: Proc. of the 4th Eclipse Security, AI, Architecture and Modelling Conf. on Data Space. Mainz: ACM, 2024. 93–102. [doi: 10.1145/3685651.3686700]

[168] Rindell K, Ruohonen J, Holvitie J, Hyrynsalmi S, Leppänen V. Security in agile software development: A practitioner survey. Information and Software Technology, 2021, 131: 106488. [doi: 10.1016/j.infsof.2020.106488]

[169] Janisar AA, bin Kalid KS, Bt Sarlana A, Maiwada UD. Software development teams knowledge and awareness of security requirement engineering and security requirement elicitation and analysis. Procedia Computer Science, 2024, 234: 1348–1355. [doi: 10.1016/j.procs.2024.03.133]

[170] Muñante D, Chiprianov V, Gallon L, Aniorté P. A review of security requirements engineering methods with respect to risk analysis and model-driven engineering. In: Proc. of the 2014 IFIP WG 8.4, 8.9, TC 5 Int'l Cross-domain Conf. (CD-ARES 2014) and the 4th Int'l

Workshop on Security and Cognitive Informatics for Homeland Defense. Fribourg: Springer, 2014. 79–93. [doi: 10.1007/978-3-319-10975-6_6]

[171] Ansari TJ, Pandey D, Alenezi M. STORE: Security threat oriented requirements engineering methodology. Journal of King Saud University—Computer and Information Sciences, 2022, 34(2): 191–203. [doi: 10.1016/j.jksuci.2018.12.005]

[172] Han S, Kim M, Kang J, Kim K, Lee S, Lee S. Similarity-based source code vulnerability detection leveraging Transformer architecture: Harnessing cross-attention for hierarchical analysis. IEEE Access, 2024, 12: 150295–150307. [doi: 10.1109/ACCESS.2024.3474857]

[173] Barr-Smith F, Blazytko T, Baker R, Martinovic I. Exorcist: Automated differential analysis to detect compromises in closed-source software supply chains. In: Proc. of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses. Los Angeles; ACM, 2022. 1–11. [doi: 10.1145/3560835.3564550]

[174] Greco C, Ianni M, Guzzo A, Fortino G. Enabling obfuscation detection in binary software through explainable AI. IEEE Trans. on Emerging Topics in Computing, 2024, 12(4): 1121–1132. [doi: 10.1109/TETC.2024.3439884]

[175] Jiang L, An JW, Huang HH, Tang QY, Nie S, Wu S, Zhang YQ. BinaryAI: Binary software composition analysis via intelligent binary source code matching. In: Proc. of the 46th IEEE/ACM Int'l Conf. on Software Engineering. Lisbon: ACM, 2024. 224. [doi: 10.1145/3597503.3639100]

[176] Vu DL, Massacci F, Pashchenko I, Plate H, Sabetta A. LastPyMile: Identifying the discrepancy between sources and packages. In: Proc. of the 29th ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Athens: ACM, 2021. 780–792. [doi: 10.1145/3468264.3468592]

[177] Brady K, Moon S, Nguyen T, Coffman J. Docker container security in cloud computing. In: Proc. of the 10th Annual Computing and Communication Workshop and Conf. Las Vegas: IEEE, 2020. 975–980. [doi: 10.1109/CCWC47524.2020.9031195]

[178] Rangnau T, Buijtenen R, Fransen F, Turkmen F. Continuous security testing: A case study on integrating dynamic security testing tools in CI/CD pipelines. In: Proc. of the 24th IEEE Int'l Enterprise Distributed Object Computing Conf. Eindhoven: IEEE, 2020. 145–154. [doi: 10.1109/EDOC49727.2020.00026]

[179] Bass L, Holz R, Rimba P, Tran AB, Zhu LM. Securing a deployment pipeline. In: Proc. of the 3rd IEEE/ACM Int'l Workshop on Release Engineering. Florence: IEEE, 2015. 4–7. [doi: 10.1109/RELENG.2015.11]

[180] War A, Diallo A, Habib A, Klein J, Bissyandé TF. Vulnerabilities in infrastructure as code: What, how many, and who? Empirical Software Engineering, 2025, 30(5): 120. [doi: 10.1007/s10664-025-10672-8]

[181] Zeini A, Lennon RG, Lennon P. Securing infrastructure as code (IaC) through DevSecOps: A comprehensive risk management framework. In: Proc. of the 2023 Cyber Research Conf. Ireland. Letterkenny: IEEE, 2023. 1–11. [doi: 10.1109/Cyber-RCI59474.2023.10671452]

[182] OWASP Foundation. Infrastructure as code security cheatsheet. 2024. https://cheatsheetseries.owasp.org/cheatsheets/Infrastructure_as_Code_Security_Cheat_Sheet.html

[183] Vadalasetty SR. Security concerns in using open source software for enterprise requirements. 2003. https://www.sans.org/white-papers/1305

[184] Edison H, Carroll N, Morgan L, Conboy K. Inner source software development: Current thinking and an agenda for future research. Journal of Systems and Software, 2020, 163: 110520. [doi: 10.1016/j.jss.2020.110520]

[185] Neil L, Mittal S, Joshi A. Mining threat intelligence about open-source projects and libraries from code repository issues and bug reports. In: Proc. of the 2018 IEEE Int'l Conf. on Intelligence and Security Informatics. Miami: IEEE, 2018. 7–14. [doi: 10.1109/ISI.2018.8587375]

[186] Wang LM, Wu JZ, Wu YJ, Rui ZQ, Luo TY, Qu S, Yang MT. Intelligent perception for vulnerability threats in open-source software supply chain. Ruan Jian Xue Bao/Journal of Software, 2025, 36(2): 511–536 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/7163.htm [doi: 10.13328/j.cnki.jos.007163]

[187] GitHub Resources. What is runtime application self-protection (RASP)? 2024. https://github.com/resources/articles/what-is-rasp

[188] Wu B, Zou F, Huang M, *et al*. Enhancing runtime application self-protection with unsupervised deep learning. In: Proc. of the 2026 Int'l Conf. on Security and Privacy in Communication Systems. Cham: Springer, 2026. [doi: 10.1007/978-3-031-94445-1_11]

[189] Baidu. Openrasp. 2025. https://github.com/baidu/openrasp

[190] He CG, Ding CHQ. SecRASP: Next generation Web application security protection methodology and framework. Computers & Security, 2025, 154: 104445. [doi: 10.1016/j.cose.2025.104445]

[191] Ohm M, Sykosch A, Meier M. Towards detection of software supply chain attacks by forensic artifacts. In: Proc. of the 15th Int'l Conf. on Availability, Reliability and Security. ACM, 2020. 65. [doi: 10.1145/3407023.3409183]

[192] Wang XY. On the feasibility of detecting software supply chain attacks. In: Proc. of the 2021 IEEE Military Communications Conf. San

Diego: ACM, 2021. 458–463. [doi: 10.1109/MILCOM52596.2021.9652901]

[193] Jiang WX, Synovic N, Sethi R, Indarapu A, Hyatt M, Schorlemmer TR, Thiruvathukal GK, Davis JC. An empirical study of artifacts and security risks in the pre-trained model supply chain. In: Proc. of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses. Los Angeles: ACM, 2022. 105–114. [doi: 10.1145/3560835.3564547]

[194] Irungu J, Girma A. Analysis of baseline security standards and predictive analytics for cyber supply chain attacks and artificial neural network as a proposed solution. In: Proc. of the 2023 Int'l Conf. on Electrical, Computer and Energy Technologies. Cape Town: IEEE, 2023. 1–6. [doi: 10.1109/ICECET58911.2023.10389476]

[195] Ge LL, Shuai DX, Xie JY, Zhang YZ, Xue YC, Yang JY, Mi J, Lu Y. Review on exception analysis methods for software supply chain. Ruan Jian Xue Bao/Journal of Software, 2023, 34(6): 2606–2627 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6850.htm [doi: 10.13328/j.cnki.jos.006850]

[196] Zhang LY, Liu CW, Xu ZZ, Chen S, Fan LL, Zhao LD, Wu JH, Liu Y. Compatible remediation on vulnerabilities from third-party libraries for Java projects. In: Proc. of the 45th IEEE/ACM Int'l Conf. on Software Engineering. Melbourne: IEEE, 2023. 1695–1707. [doi: 10.1109/ICSE48619.2023.00212]

[197] Fu M, Tantithamthavorn C, Le T, Nguyen V, Phung D. VulRepair: A T5-based automated software vulnerability repair. In: Proc. of the 30th ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Singapore: ACM, 2022. 935–947. [doi: 10.1145/3540250.3549098]

[198] Zhang J, Wang C, Li AR, Wang WH, Li TL, Liu Y. VulAdvisor: Natural language suggestion generation for software vulnerability repair. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Automated Software Engineering. Sacramento: IEEE, 2024. 1932–1944.

[199] Canonical Ltd. Linux Kernel Livepatch Mitigate Linux kernel exploits with Livepatch. 2025. https://ubuntu.com/security/livepatch

[200] Salehi M, Pattabiraman K. AutoPatch: Automated generation of hotpatches for real-time embedded devices. In: Proc. of the 2024 ACM SIGSAC Conf. on Computer and Communications Security. Salt Lake City: ACM, 2024. 2370–2384. [doi: 10.1145/3658644.3690255]

[201] Duan RA, Bijlani A, Ji Y, Alrawi O, Xiong YY, Ike M, Saltaformaggio B, Lee W. Automating patching of vulnerable open-source software versions in application binaries. In: Proc. of the 26th Annual Network and Distributed System Security Symp. San Diego: The Internet Society, 2019.

[202] Hanna C, Petke J. Hot patching hot fixes: Reflection and perspectives. In: Proc. of the 38th IEEE/ACM Int'l Conf. on Automated Software Engineering. Echternach: IEEE, 2023. 1781–1786. [doi: 10.1109/ASE56229.2023.00021.]

[203] He RZ, Zhou MH. Countermeasures for software supply chain risks of the intelligent era. Computing Magazine of the CCF, 2025, 1(3): 59–66 (in Chinese with English abstract). [doi: 10.11991/cccf.202507009]

[204] Li HP, Shan L. LLM-based vulnerability detection. In: Proc. of the 2023 Int'l Conf. on Human-centered Cognitive Systems. Cardiff: IEEE, 2023. 1–4. [doi: 10.1109/HCCS59561.2023.10452613]

[205] Chennabasappa S, Nikolaidis C, Song D, Molnar D, Ding S, Wan SY, Whitman S, Deason L, Doucette N, Montilla A, Gampa A, de Paola B, Gabi D, Crnkovich J, Testud JC, He K, Chaturvedi R, Zhou W, Saxe J. LlamaFirewall: An open source guardrail system for building secure ai agents. arXiv:2505.03574, 2025.

[206] Yu H, Wang Y, Xu MQ, Yang B, Xu C, Zhu ZL. Measurement method for complexity of software library dependency graph and its potential applications. Ruan Jian Xue Bao/Journal of Software, 2023, 34(11): 5282–5311 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6746.htm [doi: 10.13328/j.cnki.jos.006746]

附中文参考文献

[2] 纪守领, 王琴应, 陈安莹, 赵彬彬, 叶童, 张旭鸿, 吴敬征, 李昀, 尹建伟, 武延军. 开源软件供应链安全研究综述. 软件学报, 2023, 34(3): 1330–1364. http://www.jos.org.cn/1000-9825/6717.htm [doi: 10.13328/j.cnki.jos.006717]

[3] 罗丹, 吴荣春. 国内开源软件的发展现状与风险分析. 通信世界, 2022(15): 33–34. [doi: 10.13571/j.cnki.cww.2022.15.008]

[7] 高恺, 何昊, 谢冰, 周明辉. 开源软件供应链研究综述. 软件学报, 2024, 35(2): 581–603. http://www.jos.org.cn/1000-9825/6975.htm [doi: 10.13328/j.cnki.jos.006975]

[8] Synopsys. 2024 年开源安全和风险分析报告. 2024. https://www.blackduck.com/content/dam/black-duck/zh-cn/reports/rep-ossra-2024-ch.pdf

[10] 孟迎霞, 隆云滔, 王伟, 孙启, 荆琦, 谭中意, 唐小引, 鞠东颖. 2024 中国开源发展现状. 2024. https://copu-oss.org.cn/download/showdownload.php?id=32

[23] 王立敏, 卜磊, 马乐之, 于笑丰, 沈宁国. 基于指标依赖模型构建与监控的攻击检测方法. 软件学报, 2023, 34(6): 2641–2668. http://www.jos.org.cn/1000-9825/6847.htm [doi: 10.13328/j.cnki.jos.006847]

[24] 詹奇, 潘圣益, 胡星, 鲍凌峰, 夏鑫. 开源软件漏洞感知技术综述. 软件学报, 2024, 35(1): 19–37. http://www.jos.org.cn/1000-9825/

6935.htm [doi: 10.13328/j.cnki.jos.006935]
[25] 王莹, 伍盈欣, 高天, 陈子莺, 许畅, 于海, 张成志. 开源软件库生态治理技术研究综述: 二十年进展. 软件学报, 2024, 35(2): 629–674. http://www.jos.org.cn/1000-9825/6983.htm [doi: 10.13328/j.cnki.jos.006983]
[36] 金泽锋, 张佑文, 叶文华, 张贺, 邵栋. 面向完整价值交付的文档 DevOps 应用研究. 软件学报, 2019, 30(10): 3127–3147. http://www.jos.org.cn/1000-9825/5792.htm [doi: 10.13328/j.cnki.jos.005792]
[37] 何熙巽, 张玉清, 刘奇旭. 软件供应链安全综述. 信息安全学报, 2020, 5(1): 57–73. [doi: 10.19363/J.cnki.cn10-1380/tn.2020.01.06]
[38] 张得光, 李兵, 何鹏, 周华昱. 基于软件生态系统的开源社区特性研究, 计算机工程, 2015, 41(11): 106–113. [doi: 10.3969/j.issn.1000-3428.2015.11.019]
[39] 孙泽雨, 吴敬征, 凌祥, 魏怡琳, 罗天悦, 武延军. 软件供应链 SBOM 关键技术研究. 软件学报, 2025, 36(6): 2604–2642. http://www.jos.org.cn/1000-9825/7308.htm [doi: 10.13328/j.cnki.jos.007308]
[75] 深信服千里目安全技术中心. 1 年四起供应链投毒事件的幕后黑手. 2025. https://www.freebuf.com/articles/network/388355.html
[91] Freebuf. VS Code ETHcode 扩展遭供应链攻击: 两行恶意代码通过 GitHub PR 植入. 2025. https://www.freebuf.com/articles/development/438383.html
[106] 柳虹, 周根贵, 傅培华, 毛国红. 基于供应链网络的传递攻击策略研究. 计算机科学, 2013, 40(7): 98–101. [doi: 10.3969/j.issn.1002-137X.2013.07.022]
[126] GitHub. 官网的安装包被挂马 #487. 2023. https://github.com/oneinstack/oneinstack/issues/487
[133] 阿里云. PyPi 供应链投毒系列事件, Discord 助力黑客攻击. 2021. https://developer.aliyun.com/article/787142/
[137] OneinStack. mirrors.oneinstack.com 国内完整包含恶意代码. 2025. https://github.com/oneinstack/oneinstack/issues/511
[138] 安天实验室. 通过 GitHub 传播窃密木马的攻击活动分析. 2024. https://www.freebuf.com/articles/network/395369.html
[139] 奇安信代码卫士. 软件供应链投毒——NPM 恶意组件分析 (二). 2024. https://www.freebuf.com/articles/database/395549.html
[140] 悬镜安全情报中心. 恶意 NPM 包 (fix-this) 开展反向 shell 远控投毒. 2025. https://www.xmirror.cn/particulars?id=3225&type=dt
[142] 悬镜安全月报. 开源供应链投毒 202404 月报发布. 2024. https://opensca.xmirror.cn/resources/particulars?id=143
[143] 悬镜安全情报中心. 供应链投毒预警: 恶意 Py 包伪装 HTTP 组件开展 CStealer 窃密后门攻击. 2024. https://www.freebuf.com/news/400487.html
[144] 火绒安全. GitHub 开源项目被投毒, 后门病毒跟随开发流程传播蔓延. 2025. https://huorong.cn/document/tech/vir_report/1802/
[151] 毛天宇, 王星宇, 常瑞, 申文博, 任奎. 面向 Java 语言生态的软件供应链安全分析技术. 软件学报, 2023, 34(6): 2628–2640. http://www.jos.org.cn/1000-9825/6852.htm [doi: 10.13328/j.cnki.jos.006852]
[155] 阿里巴巴人工智能治理与可持续发展研究中心. 生成式人工智能治理与实践白皮书. 2025. https://s.alibaba.com/cn/aaigWhitePaper-Details/9x5JBSX2fyLEZWkRnKQ8f
[163] 戴启铭, 毛润丰, 黄璜, 荣国平, 沈海峰, 邵栋. Devsecops: Devops 下实现持续安全的实践探索. 软件学报, 2021, 32(10): 3014–3035. http://www.jos.org.cn/1000-9825/6276.htm [doi: 10.13328/j.cnki.jos.006276]
[186] 王丽敏, 吴敬征, 武延军, 芮志清, 罗天悦, 屈晟, 杨牧天. 开源软件供应链漏洞威胁智能感知. 软件学报, 2025, 36(2): 511–536. http://www.jos.org.cn/1000-9825/7163.htm [doi: 10.13328/j.cnki.jos.007163]
[195] 葛丽丽, 帅东昕, 谢金言, 张迎周, 薛渝川, 杨嘉毅, 密杰, 卢跃. 面向软件供应链的异常分析方法综述. 软件学报, 2023, 34(6): 2606–2627. http://www.jos.org.cn/1000-9825/6850.htm [doi: 10.13328/j.cnki.jos.006850]
[203] 何润之, 周明辉. 智能时代的软件供应链风险及应对技术. 计算, 2025, 1(3): 59–66. [doi: 10.11991/cccf.202507009]
[206] 于海, 王莹, 徐美秋, 杨博, 许畅, 朱志良. 软件库依赖图谱的复杂性度量方法及其潜在应用. 软件学报, 2023, 34(11): 5282–5311. http://www.jos.org.cn/1000-9825/6746.htm [doi: 10.13328/j.cnki.jos.006746]

作者简介

胡帅, 博士生, 主要研究领域为软件供应链安全, 软件工程.
王海军, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为软件供应链安全, 区块链安全, 程序分析, 逆向工程, 模糊测试.
谢继刚, 中国联通云计算领域专家, 主要研究领域为云原生安全.
裴鹏飞, 博士, 主要研究领域为多模态大模型安全.
阿西伍合, 硕士生, CCF 学生会员, 主要研究领域为软件供应链安全.
马辰达, 硕士生, 主要研究领域为软件供应链安全.
刘烃, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为信息物理融合系统安全, AI 软件工程.