深入软件供应链攻击的核心

Piergiorgio Ladisa, *SAP SecurityResearch*，雷恩第一大学Serena Elisa Ponta, *SAP SecurityResearch*Antonino Sabetta, *SAP SecurityResearch*Matias Martinez, 加泰罗尼亚理工大学-巴塞罗那*Tech*Olivier Barais, 雷恩第一大学、*Inria*、*IRISA*

arXiv:2304.05200v1 [cs.CR] 11 Apr 2023

摘要——本文探讨开源软件供应链攻击，并提出一个用于描述攻击者实施此类攻击方式的通用分类体系。随后，我们列出了一系列防护措施，以缓解此类攻击。我们介绍了工具“软件供应链风险探索器”，用于探索相关信息，并讨论了该工具在工业领域的应用案例。

开源软件（OSS）在现代应用程序中无处不在。它可能占商业应用程序代码的 90% 以上，并广泛应用于技术栈以及开发和运维生命周期的各个环节。现代软件供应链错综复杂，攻击者有多种机会将恶意代码注入开源组件并感染下游用户。

近年来，我们观察到针对开源软件的攻击在数量和类型上均出现了异常增长 [12]。例如，近期发生的一起事件影响了 PyTorch 的一个夜间构建版本。PyTorch 是一个广受欢迎的机器学习模型构建框架：攻击者滥用 `pip` [13]的依赖解析机制，通过某个依赖项偷偷植入恶意代码。其他软件供应链攻击（例如，SolarWind 的 Orion 平台遭到感染 [10]）影响了政府机构和关键基础设施的供应商。因此，多家国家安全机构将软件供应链攻击列为首要威胁之一 [2], [3]，而各方也开展了多项工作（包括公共和私营领域），以提高软件供应链的安全性（例如 SLSA[1]）。

毫无疑问，软件供应链正日益频繁地遭受攻击，这一问题也受到业界和学术界的广泛关注。不过，我们也发现，现有文献较为零散，部分原因在于缺乏一种通用且独立于具体技术的描述，来说明攻击者如何将恶意代码注入开源软件项目。

在 [5] 中，我们提出了一套针对开源软件供应链攻击向量的分类体系，这些攻击向量独立于特定的编程语言和技术。此外，我们还提出了一系列针对已识别攻击向量的通用防护措施。为了便于直观呈现和探索该分类体系，并支持开源社区和安全社区对其进行扩展，我们开发了一款名为软件供应链风险探索器 [6],的可视化工具。该工具可在线访问，并以开源形式发布。

在本文中，我们将带领您踏上探索开源软件供应链攻击核心的旅程。

我们首先介绍开源软件开发模式的攻击面、相关风险以及攻击者模型，为后续内容做好铺垫。接着，我们提出一个分类体系，涵盖与开源软件供应链相关的**117 种独特攻击向量**，以及**33 项防护措施**。该分类体系是对 [5],中所述版本的更新，我们通过考察**370 份资料**构建了这一体系，这些资料涵盖现实世界攻击以及科学文献和灰色文献。我们进一步分析这些资料所涵盖的现实世界攻击中的攻击向量，并对比每种攻击向量的普遍程度及其受到研究界关注的程度。

接下来，我们将评述软件供应链风险探索器 [6]以及四个工业应用案例。

随后，我们将我们的分类体系与现有的软件供应链安全框架进行比较。

旅程的终点揭示了恶意软件包的内容，以及软件产业在保护开源软件供应链安全方面当前面临的挑战。

XXXX-XXX © IEEE 数字对象标识符
10.1109/XXX.0000.0000000

[1]https://slsa.dev/

准备下潜

为了理解现有的攻击向量，我们首先描述开源软件供应链的攻击面，即在开源软件制品创建过程中涉及哪些系统和利益相关者。然后，我们概述开源软件供应链的风险。最后，我们介绍我们所处情境中攻击者的特征。

攻击面：开源软件开发模型

开源软件供应链是指参与开源软件制品开发、构建以及向下游用户分发的所有系统和利益相关者。图 1概述了常见的开源软件开发模型。

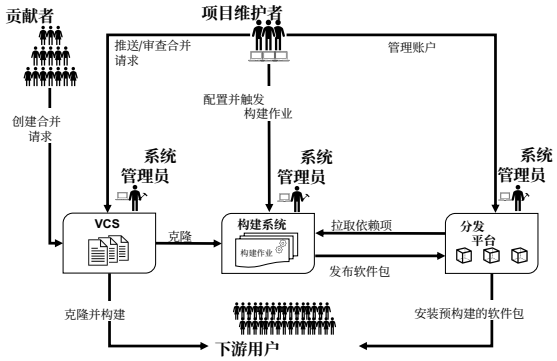


图1：与开源软件制品的开发、构建和分发相关的利益相关者、系统和数据流 [5]。

所考虑的系统包括版本控制系统（版本控制系统）、构建系统和分发平台（例如软件包仓库）。

版本控制系统托管开源软件项目的源代码，以及元数据、配置文件和其他资源。它们会在整个开发过程中跟踪和管理代码库的所有变更。Git 等纯粹的版本控制系统无需用户身份验证，但配套工具（例如 GitHub）则提供额外功能（例如问题跟踪器）或安全控制措施（例如身份验证、代码审查工作流）。

构建系统使用项目代码生成二进制制品，例如可执行文件或压缩包，并将其分发给下游用户。构建过程通常会用到所谓的依赖项管理器或软件包管理器（例如用于 Python 的 pip），它们会确定并下载成功完成构建所需的所有依赖项。持续集成 (CI)/持续交付 (CD) 流水线通常用于自动执行项目制品的构建、测试和部署。

发行平台将预先构建的开源软件制品分发给下游用户，例如在执行软件包管理器时分发，或供用户手动下载。这些平台不仅包括 PyPI 或 Maven Central 等知名公共软件包仓库，还包括内部和外部镜像、内容分发网络（CDN）或代理服务器。

开源软件维护者和管理员的工作站。开源软件项目维护者以及上述系统的管理员拥有访问敏感资源的特权，例如代码库、构建系统的 Web 界面或软件包仓库的数据库。因此，他们的工作站也属于攻击场景的范围。

关于参与开源软件供应链的利益相关者，包括贡献者、项目维护者、系统和服务管理员以及下游用户。

贡献者 为开源软件项目做贡献时，对项目资源的访问权限有限。常见的向版本控制系统贡献代码的方式包括提交合并请求。

项目维护者 拥有对项目资源的特权访问权限。例如，他们负责审查并整合贡献者的合并请求、配置构建系统并触发构建作业，或将现成的制品部署到软件包仓库中。

系统和服务管理员负责配置、维护和运行上述任何系统。这些利益相关方可能包括：第三方版本控制系统托管服务提供商的员工、运营项目私有构建系统的开源软件基金会成员，或运营软件包仓库（例如 npm 或私有镜像）的公司的员工。

下游用户 使用开源软件项目的制品。他们可以选择从版本控制系统获取源代码（然后自行构建），也可以像更常见的做法那样，从发行平台下载预构建版本。在下游开发项目中，下载通常由 pip 或 npm 等包管理器自动完成，它们可以帮助自动识别并获取某个项目的直接依赖项和传递依赖项。

系统和利益相关者都必须被视为角色，而一个主机、个人或第三方服务可以同时承担多个角色。例如，开源软件项目的维护者通常会使用其他项目的制品

开源软件供应链的风险

上述系统本质上是分布式的，而利益相关者中有一部分身份不明或匿名。项目协作者（包括贡献者和维护者）的真实身份未必为人所知。账户（包括匿名账户）通过持续贡献高质量内容来赢得信任（精英政治）。

每个使用的开源组件都有其自身的系统和利益相关方，因此扩大了兼具技术与社会层面的攻击面。与其他对抗性情境一样，攻击者只需找到一个弱点，而防御者则必须覆盖整个攻击面；在这种情况下，攻击面延伸至整个供应链。

即使是使用广泛的开源项目，获得的资金和贡献也很少，这使维护者难以安全地运营项目，或及时应对安全事件。此外，用户群体（直接和间接）越大，开源项目对攻击者就越有吸引力。

下游使用者无法控制特定项目的安全实践，且对此了解有限。依赖项数量庞大，使得单个使用者难以进行严格审查，因此只能信任社区及时发现漏洞和攻击。

攻击者模型

为识别与开源软件供应链相关的攻击向量，我们对攻击者作出以下假设。

攻击者的首要目标是将恶意代码**插入**开源制品中，使其在下游项目的环境中执行，例如在开发期间或运行时。目标资产可能属于下游软件项目的开发人员，也可能属于其最终用户，具体取决于攻击者的意图。事实上，分类体系关注的不是恶意代码做什么，而是攻击者如何将其植入上游项目。

> 分类体系的重点不是恶意代码做了什么，而是攻击者如何将其植入上游项目。

起初，攻击者只能获取公开可用的信息和可公开访问的资源，并可按照开源情报（OSINT）的方法收集和分析这些内容。当然，由于开源项目的性质使得许多项目细节都可以免费获取，例如项目依赖项、构建信息，以及提交记录和合并请求历史。攻击者可以与图 1中展示的任何利益相关者和资源互动，例如通过合并请求或问题跟踪器与维护者沟通，或创建虚假账户和项目。

攻击者可以针对任何类型的项目（例如库、文字处理器）。下游使用者可能会直接或间接受到影响，此外，攻击者的目标范围既可以是覆盖大量使用者，也可以是针对特定群体。通过根据恶意代码的生命周期阶段（例如安装、测试等）、应用程序状态、操作系统，或其所集成到的下游组件的属性来决定是否执行恶意代码，这一点是可以实现的 [9]。

我们的起点：攻击分类体系

本节介绍了由 117 种独特攻击向量构成的分类体系，这些攻击向量通过审阅学术文献和灰色文献收集而来（详情见 [5]）。此类分类体系采用 攻击树 的形式，以系统地表示攻击者的目标和技术。在攻击树中，根节点表示攻击者的最高层级目标，并通过其子节点逐步细化为子目标。

为了创建图中所示的分类体系2 我们对科学文献和灰色文献进行了系统性文献综述（SLR）。我们已尽最大努力收集尽可能多的资源，但不声称已经做到完整无遗漏。通过开展用户调查，该分类体系得到了软件供应链安全领域的17位专家和134名软件开发人员的验证与评估。

发起开源供应链攻击这是攻击者的最高层目标，具体方式是将恶意代码注入开源软件项目，使其被下游使用者下载，并在安装时或运行时执行。

下方围绕这一高层次目标展开的完整分类体系如图 2 所示，并在下文中进行了总结。树的第一级子节点反映了对现有软件包的不同程度的干扰。

从零开发并推广独特的恶意软件包

此节点涵盖创建全新的开源软件项目，意图从一开始或稍后的某个时间点利用它传播恶意代码。除了创建项目之外，还要求攻击者

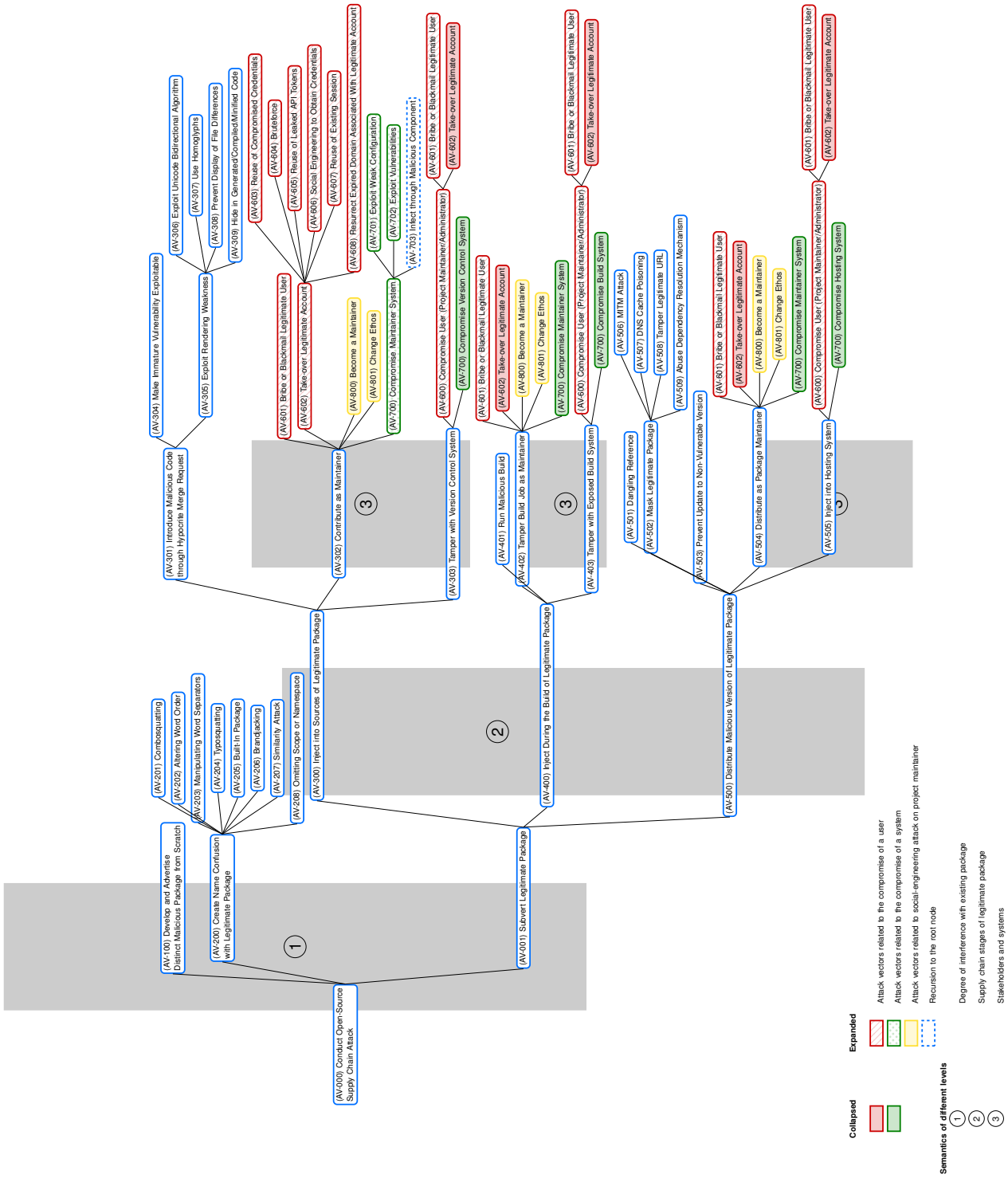


图2：开源软件供应链攻击分类体系的改进版本 [5]。

以宣传该项目并吸引受害者。现实案例涉及 PyPI、npm、DockerHub 或 NuGet。

通过与合法软件包混淆来创建名称此节点涵盖创建项目或制品名称的攻击，这些名称会模仿合法名称、暗示作者值得信赖，或利用常见的命名模式。一旦找到合适的名称，攻击者便会部署恶意制品，例如发布到源代码或软件包仓库，希望下游用户使用它。由于这种部署不会干扰启发该名称的项目资源（例如合法代码仓库、维护者账户），因此这种攻击成本相对较低。

此攻击向量的子节点涉及对合法项目名称进行不同修改的子技术：组合仿冒添加前缀或后缀，例如表示项目成熟度（dev 或 rc）或平台兼容性（i386）。调整词序重新排列词语顺序（test-vision-client 与 client-vision-test）。操纵词语分隔符更改或添加连字符等词语分隔符（setup-tools 与 setuptools）。拼写仿冒利用拼写错误（dajngo 与 django）。内置软件包仿冒其他场景中的知名名称，例如编程语言的内置软件包或模块（Python 的 subprocess）。品牌劫持使用热门品牌或组织的名称（例如 twilio、aws），暗示软件包来自值得信赖的作者（twilio-npm）。相似性攻击以不同于前述类别的方式创建误导性名称（request 与 requests）。

劫持合法软件包

此节点涵盖所有旨在破坏现有合法项目的攻击，这需要攻陷其众多资源中的一个或多个，如图 1所示。因此，与前面的子树相比，该子树要大得多，尤其是因为与用户和系统攻陷相关的子树会在不同的供应链阶段多次出现。

本节剩余部分专门介绍这一第一级节点的子技术。

注入合法软件包的源代码与将恶意代码注入项目代码库有关。从攻击者的角度来看，这种方式的优势在于会影响所有下游用户，无论他们使用的是源代码还是预构建的二进制制品（恶意代码作为代码库的一部分，会在项目构建和二进制制品分发过程中被包含进去）。此攻击向量包含若干子技术。攻击者可以伪装成贡献者，利用虚伪的合并请求来将不成熟的漏洞-能力转变为可利用的能力，或利用 IDE 渲染弱点来隐藏恶意代码，例如通过使用Unicode 同形异义字符和控制字符，或隐藏和抑制代码差异。要以维护者身份做出贡献，需要获得修改合法项目代码库所必需的权限，这可以通过不同方式实现：对合法项目维护者使用社会工程（SE）技术、改变其理念（例如抗议软件的情况）、接管合法账户 （例如重复使用已泄露的凭据，或入侵维护者系统 （例如利用漏洞）。后者也可以通过恶意的（开源软件）组件实现，例如IDE 插件，这通过对根节点的递归引用体现出来。合法项目的代码库也可以通过篡改其版本控制系统来修改，从而绕过项目既定的贡献工作流。例如，通过入侵系统用户账户，或利用配置/软件漏洞，攻击者可能以不安全的方式访问代码库。

在合法软件包构建过程中注入由于 Maven 或 Gradle 等针对特定语言的软件包管理器（例如用于 Java 的管理器）的大力推动，相较于从版本控制系统获取开源软件项目的源代码，从软件包仓库下载预构建组件已变得十分普遍。因此，恶意代码可能会在此类组件发表之前的构建过程中被注入。尽管与注入源代码相比，这种传播方式的范围受到限制，但对攻击者而言，其优势在于，通常更难检测预构建软件包中的恶意代码，尤其是对于编译型编程语言而言。一种子技术是运行恶意构建作业，篡改多个项目的构建任务之间共享的系统资源（例如，感染 NetBeans 项目中的Java 归档文件）。攻击者还可以以维护者身份篡改构建作业，例如接管合法的维护者账户、成为维护者，或攻陷其系统（例如 XCodeGhost恶意软件）。同样，攻击者也可能攻陷构建系统，尤其是可在线访问的构建系统，例如通过攻陷管理员账户或利用漏洞。

分发合法软件包的恶意版本预构建组件通常托管在 PyPI 或 npm 等知名软件包仓库中，也可能托管在范围较窄、知名度较低的仓库中。此外，这些组件可以远程或本地镜像，通过内容分发网络（例如 JavaScript 库的情况）提供，或缓存在代理服务器中。此攻击向量及其子技术涵盖攻击者篡改预构建软件包的托管、分发和下载过程中涉及的机制与系统的所有情况。

软件包。悬空引用会重新使用已弃用项目的资源标识符，例如名称或 URL。伪装合法软件包针对软件包名称或 URL 解析机制以及下载连接。其目标是通过破坏合法项目之外的资源来下载恶意软件包。这包括中间人（MITM）攻击、DNS 缓存投毒，或直接在客户端篡改合法 URL。具体而言，软件包管理器会遵循一套（可配置的）解析策略，以决定要下载哪个版本的软件包、从何处下载，以及联系多个仓库时的优先顺序。攻击者可以滥用此类解析机制及其配置。攻击者还可以通过操纵软件包元数据来阻止更新到无漏洞版本，例如，声称合法软件包的较新版本依赖于无法满足的依赖项。最后，系统和用户参与软件包分发，也会产生与前述攻击向量类似的攻击向量。攻击者可以冒充合法维护者，从而以维护者身份分发软件包，例如接管软件包维护者账户（如 eslint）；这是仅次于拼写仿冒的第二常见攻击向量。他们还可以入侵维护者的系统，或直接向托管系统注入内容，例如通过入侵管理员账户或利用漏洞。

备注

虽然所呈现的分类体系在很大程度上与具体对象无关，但某些攻击向量仅针对特定的生态系统。滥用依赖项解析机制攻击取决于相应包管理器用于从内部和外部存储库解析并下载所声明依赖项的方法和策略。例如，Maven、NPM、pip、NuGet 或 Composer 受到依赖项混淆攻击的影响，而 Go 和 Cargo 则未受到影响 [4]。以下几种攻击利用渲染弱点取决于用户界面以及编译器/解释器对（Unicode）字符的解释和可视化方式 [1]。此外，名称混淆攻击还需要考虑生态系统的特性，尤其是内置软件包。

供应链攻击史上的片段

自 1984 年 Ken Thompson 发表《信任信任的反思》以来，围绕软件供应链安全的讨论不断增多，近年来增长尤为显著。就攻击而言，据我们所知，已知最早的开源软件供应链攻击可追溯至 2010 年（参见图 3b）。第一起攻击是在 ProFTPD1.3.3c 的源代码中植入后门 [8]，第二起则是著名的极光行动 [7]。截至 2016 年，攻击主要是在源代码中或构建过程中注入恶意代码，或在托管系统中部署合法软件的恶意版本。直到 2017 年，我们才发现首起利用通过合法软件包创建名称混淆这一攻击向量的攻击；如今，这很可能是遭利用最多的攻击向量。具体来说，这起攻击通过 PyPI 上传软件包，冒用 urllib3 等热门项目的名称（采用了不同的子技术）。在这起攻击发生前，只有一篇未经同行评审的作品讨论了通过合法软件包创建名称混淆这一攻击向量，即 Nikolai Tschacher 于 [14] 2016 年发表的硕士论文，比这类攻击首次出现早了一年。

快速恢复：防护措施

本节概述了针对开源软件供应链攻击的防护措施。这些防护措施通过文献综述确定，并经过归纳总结，不依赖于特定的编程语言或生态系统。

完整的 33 项防护措施列表见表格1，其中包括按控制类型分类，并根据 17 位领域专家评估的效用成本比排序 [5]。所有防护措施都映射到其能够（部分或完全）缓解的攻击向量（如上所述）；有些措施因能够应对所有攻击向量而映射到顶层目标（例如，建立审查流程），其他措施则映射到更具体的子目标。有些防护措施可由一个或多个利益相关方实施，而其他措施则需要多方参与才能发挥作用（例如，创建和验证签名）。

这些防护措施的实施和使用都可能产生不可忽视的成本，具体还取决于所使用的编程语言和生态系统。因此，对安全措施的选择、组合和实施需要进行周密的规划与设计，以平衡所需的安全级别与成本。

在我们的工作中，[5] 我们通过对领域专家和软件开发人员开展调查，对每项防护措施的效用和成本进行了定性评估。

防护措施		控制类型				相关利益方			已应对的攻击向量
保护生产分支	2.10		✓	✓		•			AV-301，AV-302
移除未使用的依赖项	2.05		✓					•	AV-001
版本锁定	1.68		✓					•	AV-001
依赖项解析规则	1.58		✓					•	AV-501、AV-508、AV-509
用户账户管理	1.50		✓		✓	•	•		AV-302,AV-402,AV-504,AV-600
安全认证 (例如, 多重因素	1.48		✓			•	•		AV-*00 → AV-602
身份验证（MFA）, 密码循环使用, 会话超时、令牌保护)									
安全、质量和健康指标的使用	1.35	✓	✓			•	•	•	AV-000
拼写错误防护/拼写错误检测	1.34		✓	✓			•	•	AV-200
在发布作业中仅使用最少的可信构建依赖发布作业中的策略	1.32		✓			•			AV-400
通过密码学哈希对依赖项进行完整性检查	1.32			✓				•	AV-400、AV-500
维护详细的软件物料清单（SBOM），并执行软件成分分析（SCA）	1.24		✓	✓		•	•	•	AV-000
临时构建环境	1.24		✓			•			AV-400
防止脚本执行	1.23		✓					•	AV-000
拉取/合并请求审查	1.21		✓			•			AV-301, AV-302
限制对系统资源的访问，防止...	1.21		✓			•			AV-400
在每个构建步骤中执行的代码代码签名	1.19			✓		•	•	•	AV-200、AV-500
将开源漏洞扫描器集成到 CI/CD 流水线中	1.15			✓				•	AV-000
使用专用构建服务	1.09		✓			•			AV-400 → AV-700
预防性抢注已发布的软件包	1.07		✓			•	•		AV-200
审计	1.05	✓		✓		•	•		AV-000
安全评估	1.05			✓		•	•		AV-000
漏洞评估	1.05			✓		•	•		AV-000
渗透测试	1.05			✓		•	•		AV-000
可复现构建	1.02			✓		•	•	•	AV-400，AV-500
构建步骤隔离	1.00		✓			•			AV-400
作用域包	1.00		✓			•	•		AV-509
建立内部仓库镜像并参照-仅引用一个私有源，而不是多个	0.97		✓					•	AV-501,AV-502, AV-504, AV-505
应用安全测试	0.95			✓			•	•	AV-000
为开源项目建立审查流程	0.95		✓				•	•	AV-000
托管在内部/公共代码库中的组件-库									
代码隔离和沙箱隔离	0.93				✓			•	AV-000
运行时 应用程序 自我保护 (RASP)	0.88			✓	✓			•	AV-000
手动源代码审查	0.85			✓			•	•	AV-300
从源代码构建依赖项	0.73		✓				•	•	AV-400、AV-500

表 1：针对开源软件供应链攻击的防护措施，按 17 位专家评估的效用成本比均值排序 [5]。每项防护措施还按控制类型、参与实施的利益相关方，以及与可缓解攻击向量的对应关系进行描述（参见图2以解读其标识符） [5]。

通用防护措施 包括 4 项需要所有利益相关者积极参与的对策，即项目维护者、开源软件使用者和管理员（服务提供商）。例如，项目维护者必须编制并维护详细的软件物料清单，最好使用自动化软件成分分析工具。随后，软件包仓库必须安全地托管和分发该软件物料清单，下游用户则必须根据自身的安全、质量和许可证要求对其进行仔细检查。

项目维护者和管理员的防护措施 包含 8 项防护措施。 安全身份验证-*tion*，例如，建议服务提供商提供 MFA 或强制执行强密码策略，而项目维护者应遵循身份验证最佳实践，例如在可用时使用 MFA、避免重复使用密码，或保护敏感令牌。

项目维护者防护措施包括 7 项对策。通常，开源软件项目使用托管的、公开可访问的版本控制系统。维护者应采取措施，例如认真进行合并请求审查，或为敏感的项目分支启用分支保护规则，以避免恶意代码贡献。由于项目构建仍可能在维护者的工作站上进行，因此他们

应使用专用构建服务，尤其是临时环境。此外，他们可以隔离构建步骤，使其无法篡改其他构建步骤的输出。

面向管理员和使用者的防护措施包含 5 项对策。例如，软件包仓库管理员和使用者都可以选择直接从源代码构建软件包，而不是接受预构建制品。如果软件包仓库采用这种做法，就能降低项目构建被篡改的风险。如果使用者采用这种做法，就能消除所有与第三方构建服务和软件包仓库遭到入侵相关的风险，因为这些环节都不再参与其中。

面向消费者的防护措施 包括9 项应对措-施，下游用户可以采用这些措施。消费开源软件软件包时，可以通过隔离代码和/或进行沙箱隔离 来降低执行恶意代码的影响。另一个例子是建立内部仓库镜像 来存放经过审查的组件。

备注
所提出的一些防护措施专用于特定的软件包管理器，即作用域包（Node.js）和防止脚本执行 （Python 和 Node.js）。不过，其他防护措施与生态系统无关，只是控制措施的实现方式和技术选择有所不同，例如应用安全测试。

层层深入：攻击向量的流行度。

截至撰写本文时（2023 年 2 月），我们共收集了 369 篇参考文献，其中 81 篇讨论现实世界攻击，126 篇为同行评审论文。图 3展示了所有参考文献（图 3a）、与现实世界攻击相关的参考文献（图 3b）以及同行评审参考文献（图 3c）按年份统计的发表数量。

按攻击向量对同行评审和现实世界攻击相关参考文献进行分组后，我们观察到以下情况。

在我们收集的同行评审参考文献中，按流行度排序如下：
1）35 篇资料讨论实施开源软件供应链攻击（*AV-000*）这一普遍问题；2）33 篇资料讨论发行平台（*AV-500*）的安全方面及相关问题；3）24 篇资料讨论构建系统（*AV-400*）的安全方面及相关问题；4）19 篇资料讨论*VCS*（*AV-300*）的安全方面及相关问题；5）8 篇资料讨论软件包通过创建名称混淆来冒充合法软件包（*AV-200*）的问题；6）3 篇资料讨论从头开发并推广恶意软件包（*AV-100*）的问题。

关于现实世界攻击，按流行度排序如下：

1) 26 起攻击利用了通过创建名称混淆来冒充合法软件包（*AV-200*）；2) 25 起攻击利用了发行平台（*AV-500*）相关攻击向量；3) 20 起攻击利用了版本控制系统（*AV-300*）相关攻击向量；4) 14 起攻击利用了构建系统（*AV-400*）相关攻击向量；5) 7 起攻击涉及从头开发并宣传恶意软件包（*AV-100*）。

不难发现，在现实世界攻击中最常被利用的攻击向量（即通过与合法软件包名称混淆来制造名称混淆）却是学术论文中讨论最少的攻击向量。对于其他攻击向量，同行评审论文讨论的攻击向量与攻击的流行程度几乎相符。防止软件包名称抢注十分复杂，而且确实存在合理的使用场景，组织会上传名称相似的软件包，例如，为其开发的所有软件包使用相同的前缀，以便于识别。学术界已开始提出检测软件包仓库中恶意代码的技术。虽然这些技术并非专门针对名称抢注，但同行评审论文中讨论最多的防护措施之一是应用安全测试（*AST*）（包括静态和动态测试）；它既能防范此类攻击，又具有通用性（无论恶意软件包是否通过名称抢注）。然而，审核整个软件包仓库会带来沉重的计算负担，而且误报率必须足够低，才能切实可行地由分析人员进行人工审查。

另一个值得强调的方面是，学术研究讨论了开源软件供应链攻击这一普遍问题，而在 AST之后，讨论最多的对策分别是软件组成分析/软件物料清单和漏洞评估。

在现实世界攻击中最常被利用的攻击途径（即创建名称

与合法软件包的混淆）是学术论文中讨论最少的。

救援工具：SAP 的软件供应链风险探索器

鉴于分类体系的规模以及与其相关的信息量，我们开发了一个配套工具，以便于访问和使用该分 类体系。该工具是开源的[2] 旨在促进社区的创建，使社区能够从分类体系中受益并为其做出贡献。当拉取请求合并到主分支后，该工具的新版本会自动部署。

用户可以折叠和展开攻击树的不同节点，以探索基于开源的软件开发的攻击面。相应攻击向量的描述、参考文献以及相关防护措施显示在树状图下方（参见图4）。这种探索式可视化模式为用户提供了管理视觉复杂性并按需访问更易于消化的信息片段的优势。

分享按钮会生成指向各个攻击向量的深层链接，可供第三方网站引用，例如培训材料或安全公告，以引用特定攻击中使用的攻击向量，并对其进行更清晰的说明。右上角的搜索框可按名称搜索攻击向量。选中后，从根节点到所选攻击向量的相应路径会展开并以红色突出显示。下方的第二个搜索框用于搜索防护措施。选中后，所有由相应防护措施缓解的节点都会显示为绿色（参见图 4）。所有攻击向量、防护措施和书目参考文献也可以表格形式显示。攻击向量的表格视图还支持在模态窗口中显示相关防护措施的信息。参考文献可按标题、出版年份和受影响的生态系统排序。

工业应用场景

Risk Explorer 工具可支持以下工业活动。

培训与认知。 Risk Explorer 工具可交互式呈现分类体系，攻击向量与防护措施的描述，以及在知识系统化过程中审阅的资源数据集。因此，它有助于更好地理解软件供应链中的安全风险，并提升开发人员和安全从业者的认知。

例如，假设我们听说了针对 *event-stream* 软件包的攻击。为了确定所使用的攻击向量，我们可以通过参考文献的表格视图（参见图 5）在 Risk Explorer 中搜索该软件包。在 相关攻击向量 列中，我们可以找到分配给它的攻击向量。然后，我们可以点击深层链接，在分类体系中定位该攻击。此时，Risk Explorer 会提供用于感染 *event-stream* 软件包的特定攻击向量的描述、采用相同策略的其他攻击，以及可防护此类攻击的防护措施。查看从特定攻击向量到根节点的路径，我们可以了解导致供应链攻击的目标链。将分类体系停留在某一深度，我们可以了解攻击者可能采用的其他可能性，以及能够防护更广泛攻击的通用防护措施。

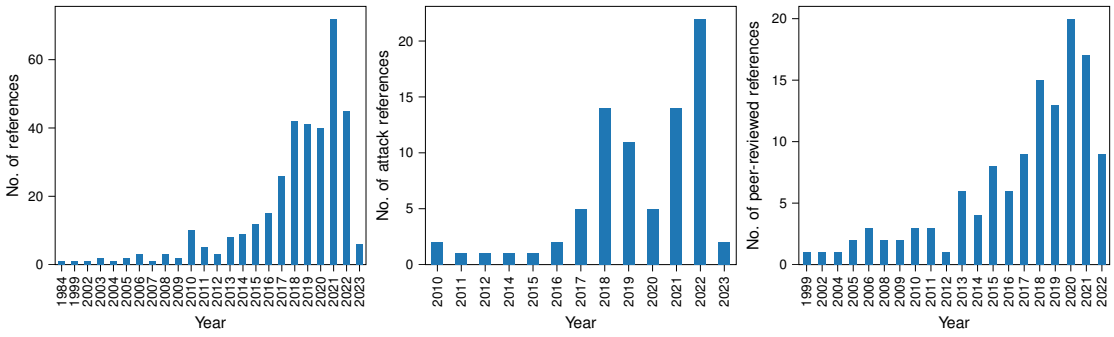
SAP 的一位安全专家在为整个组织提供供应链安全指导时使用了该工具。她表示，通过建议的防护措施和链接的参考文献，该工具“帮助人们更深入地了解预防性和检测性措施”。内容，尤其是该工具提供的功能组合，例如“可视化和映射，*[以及]* 查看交叉引用的能力”，被认为是其优势。这位专家总结说，她“在改进开发安全实践的过程中，会继续参考这项研究”。

> “*[风险探索器]* 帮助人们更深入地了解预防性和检测性措施。”

威胁建模。 在此活动中，目标是概述系统可能面临的威胁，然后确定可采取哪些对策来防范这些威胁。

通过提供一份详尽的攻击向量清单，该工具能够支持开源软件供应链攻击场景下的威胁建模工作。尤其是在绘制出基础设施的架构图后，可以识别参与者和系统可能面临哪些威胁（即攻击向量）。此外，

[2] https://github.com/SAP/risk-explorer-for-software-supply -chains



(a) (b) (c)

图3：按发表年份统计收集到的参考文献数量。(a) 所有类型的参考文献；(b) 讨论现实世界攻击的参考文献；(c) 经过同行评审的参考文献。

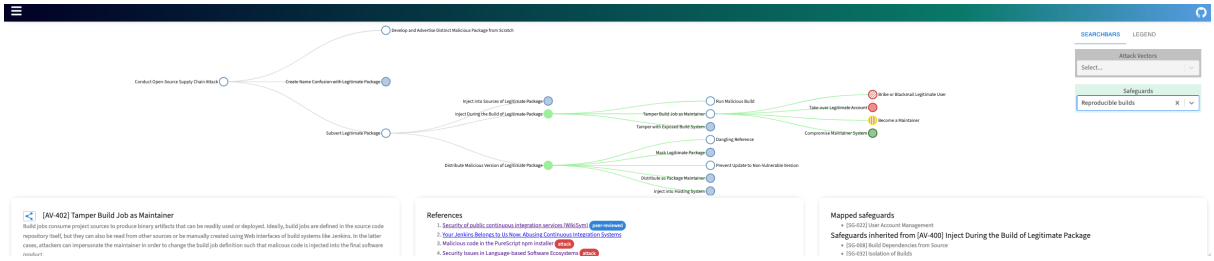


图4：攻击树可视化的截图（绿色部分为防护措施 可复现构建 所覆盖的攻击向量） [6]。

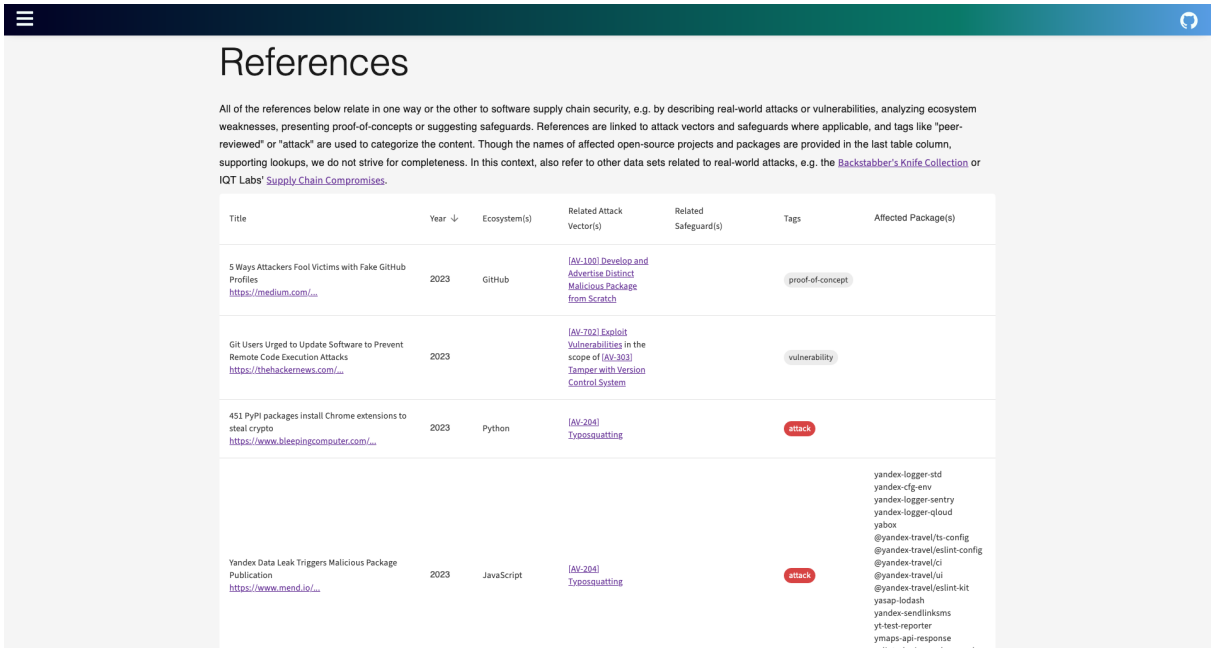


图5：页面展示了按类别整理的所有参考文献的表格视图。可按标题、发表年份、相关攻击向量、相关防护措施和类型进行排序。

与相关对策的交叉引用有助于了解如何保护基础设施免受这些威胁。

界定红队活动范围。红队演练包括模拟现实世界攻击场景，以评估系统的安全能力。

与 ATT&CK Navigator 类似[3]，该工具也可用于规划红队/蓝队活动和开展安全评估。根据不同目标（父节点）列举攻击向量（子节点），可为项目安全评估期间（作为攻击者）可能采取的策略提供一份检查清单。

防护措施的差距分析。通过防护措施搜索栏突出显示现有防护措施所涵盖的攻击向量（参见图 4），有助于进行差距分析，从而评估现有防护措施正在缓解哪些攻击向量，以及哪些攻击向量（如有）仍需实施相应的防护措施。

添加新的攻击参考：以 pytorch-nightly 为例。

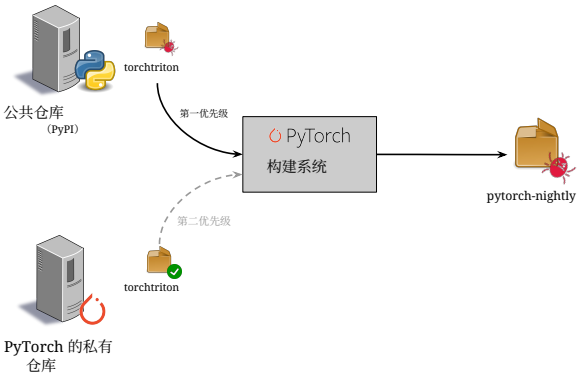


图 6：分类体系中 PyTorch-nightly 遭入侵所使用攻击向量的重点展示。

假设我们了解到一种新的攻击，并希望通过在树状结构中将其归入适当的攻击向量来作出贡献。我们以近期 PyTorch-nightly 遭入侵事件（2022 年 12 月 25 日至 30 日）为例。

安全公告指出：“一个恶意依赖包（*torchtriton*）*[.....]*被上传到了 *Python* 包索引（*PyPI*）代码仓库，其包名与我们在 *PyTorch nightly* 包索引中发布的包相同。由于 *PyPI* 索引具有优先级，因此安装的是这个恶意包，而不是来自我们官方仓库的版本。”[13]该攻击如图6所示。

要将攻击归类到分类体系中的某个节点，我们需要从根节点开始遍历，确定向更深层节点移动时应遵循的路径。从根节点开始，我们首先要问自己，攻击是从头开发恶意软件包、利用与合法项目相似的名称，还是合法软件包遭到了劫持。由于 `torchtriton`是 PyTorch-nightly 使用的合法依赖项，因此属于最后一种情况。从节点 劫持合法软件包 (AV-001) 出发，我们需要区分攻击是通过版本控制系统、构建系统还是分发平台发生的。在 `torchtriton`遭到入侵的事件中，源代码和构建基础设施均未遭到破坏，攻击发生在 PyPI 软件包仓库中，因此我们归入最后一个节点：分发合法软件包的恶意版本 (AV-500)。通过阅读安全公告可知，攻击的根本原因在于，`torchtriton` 仅存在于 PyTorch nightly 软件包索引中（该索引用于构建 PyTorch-nightly 项目），而不在公共 PyPI 仓库中。因此，合法依赖项 `torchtriton` 被隐藏，其手段是滥用依赖解析机制 (AV-509)。因此，PyTorch-nightly 的入侵可归类到节点 (AV-509)（参见图 7）。

每当攻击利用注入恶意代码的新方式时，现有攻击向量可能无法覆盖新识别出的行为。在此类情况下，就需要考虑扩展分类体系本身。

我们并不孤单：相关框架

开源及其他组织已经认识到，在开发安全的软件和服务过程中，确保开源软件和基于开源的供应链安全的重要性。人们已经提出了几个框架，为安全的软件生命周期提供实践指导。

持久安全框架[4] （由美国国家安全局、CISA 和 ODNI 牵头的公私联合跨部门工作组）发布了可操作的指导方针，帮助开发人员、供应商和客户保障整个软件供应链。“保障开发人员的软件供应链”针对软件开发生命周期，考量了安全代码开发、第三方组件验证、强化构建系统以及交付代码过程中面临的威胁，并提供了建议的缓解措施。“保障供应商的软件供应链”重点说明供应商应如何识别可能危及其组织、

[3] https://mitre-attack.github.io/attack-navigator/

[4] https://www.nsa.gov/About/Cybersecurity-Collaboration-Center/Cybersecurity-Partnerships/ESF/

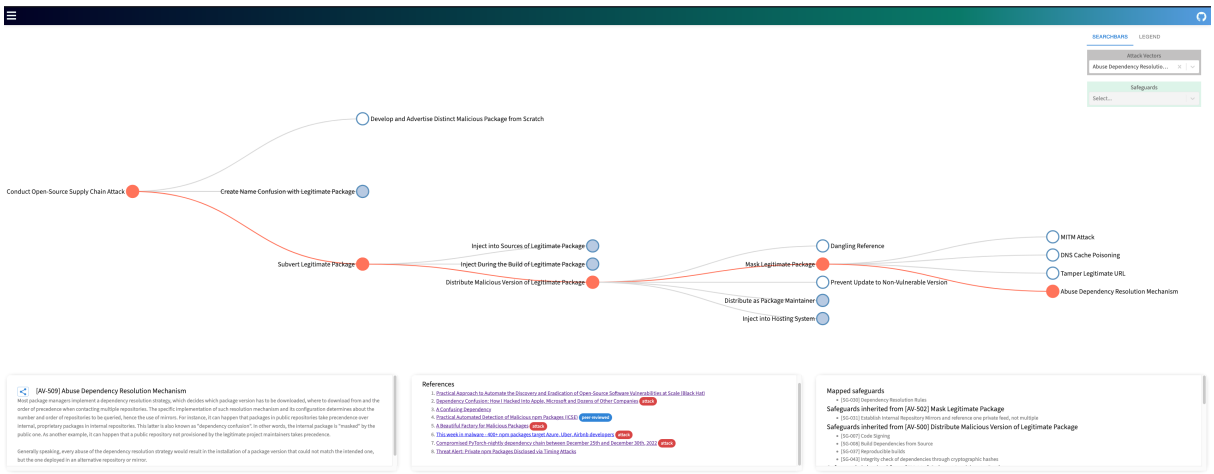


图 7：分类体系中 PyTorch-nightly 入侵所使用攻击向量的重点标示。

软件开发、软件产品和软件交付的威胁。“保障客户的软件供应链”则聚焦于软件产品采购、部署和运行的最佳实践。

Microsoft 开源软件安全供应链消费框架（S2C2F）[5] 结合了相关要求和工具，以降低使用开源软件所带来的风险。该框架基于三个核心概念：控制所有使用的开源软件，使用成熟度模型帮助确定待实施要求的优先级，以及大规模保障软件供应链的安全。与“为开发人员保障软件供应链安全”框架相比，Microsoft S2C2F仅关注开源软件的安全使用，但在这一背景下提供了更为详细的指导。该框架定义了 8 项实践（例如，扫描和更新第三方组件）以及一系列相关的运营要求（例如，扫描开源软件中的恶意软件、对开源软件执行安全审查）。此外，它还配备了一个成熟度模型，将这些要求划分为 4 个不同级别，并列出了可帮助满足这些要求的工具。

OWASP 软件组件验证标准（SCVS）[6] 旨在建立一个框架，用于识别活动、控制措施和最佳实践，帮助识别和降低软件供应链中的风险。与 S2C2F 类似，OWASP SCVS 提供了 6 类控制措施（例如，清单、谱系和来源证明），以及 3 个验证级别；随着保障级别提高，所需满足的要求也逐步增加。不过，该标准适用于一般的软件组件，因此其要求并不像 S2C2F 框架那样，专门针对供应链中使用的开源软件。

软件制品供应链级别（SLSA）[7] 是OpenSSF 的一个项目，由一份标准和控制措施清单组成，旨在防止篡改、提高完整性，并保护项目、业务或企业中的软件包和基础设施。它定义了四个保障级别：从通过记录在案的自动化构建流程提供简单的来源信息，到通过对源代码变更进行同行评审并采用密闭、可复现构建来实现高可信度和高度信任。与MicrosoftS2C2F和 OWASP SCVS相比，SLSA 的范围更窄，因为它专注于完整性，从而确保所使用的代码未被篡改。

不同的框架存在重叠，因为它们都旨在提供指导和建议，并且已经开展了一些将它们进行映射的工作，例如，开源软件 SSC 框架包含了将需求映射到其他相关规范的内容，包括 OWASP SCVS 和 SLSA。然而，所有框架都缺乏对攻击如何实施的系统性描述，因此无法清晰界定所提供建议涵盖了哪些威胁。

本文提出的分类体系补充了此类工作。它从攻击者的视角出发，提供了最完整的攻击分类体系，而上述框架则侧重于

[5] https://www.microsoft.com/en-us/securityengineering/opensource/开源软件ssscframeworkguide[6] https://owasp.org/www-project-software-component-verification-standard/

[7] https://slsa.dev/

通过提供建议来制定防护措施。类似于 MITRE ATT&CK 框架针对基础设施攻击所做的工作，它有助于描述攻击者如何利用软件供应链传播恶意软件。拥有完整的分类体系是确定已识别的建议是否涵盖已知攻击向量的关键。通过提供与攻击相关的知识基础，它有助于设计和研究新型对策，也可能促进对现有框架的比较。

从攻击向量到恶意代码

如前所述，分类体系的范围是描述攻击者如何在上游项目中注入恶意代码，而不论实际注入的恶意内容是什么。作为补充，观察恶意代码可以提供有关攻击者试图实现的行为以及不同生态系统受到影响方式的有价值信息。

恶意软件行为

正如首次提及于 [9], 现有开源软件供应链攻击中观察到的主要目标是：

- 反向 *Shell*，即生成一个 Shell进程，并通过一个开放套接字将其输入和输出都重定向到攻击者的机器。
- 下载器 连接到攻击者控制的主机，以下载随后将被执行的第二阶段载荷。远程有效载荷可以直接通过连接读取，也可以临时存储在本地文件中。
- 数据窃取（最常见的行为 [9]）旨在读取敏感信息（例如环境变量、文件），然后将其发送到攻击者控制的端点。
- 拒绝服务（*DoS*）通常通过耗尽资源（例如分叉炸弹）或删除系统文件来实现。
- 经济利益是通过在目标系统上运行加密货币挖矿程序获得的。

在我们最近对NPM 和 PyPI 生态系统[8]，进行的分析中，我们还观察到存在研究概念验证，旨在测试语言和/或特定包管理器所提供的执行功能，以展示安装此类软件包时的潜在风险。我们观察到的另一种不受欢迎的行为是瑞克摇滚攻击。当我们观察到并报告此类软件包时，相关安全团队决定不予移除，因为他们认为瑞克摇滚不属于恶意行为。

恶意软件技术

我们还观察到一些使用混淆技术的案例。还请注意，代码混淆对攻击者的价值可能因生态系统而异。对于解释型语言，下载的软件包包含恶意软件的源代码，因此与编译型语言相比，分析人员更容易查看这些代码。事实证明，此类软件包中存在编码或加密代码是一个很好的失陷指标 [11]，因为开源软件包几乎没有正当的使用场景（例如代码压缩，主要用于前端 JavaScript 库）。尽管源代码可以访问，但开源软件包及其版本数量庞大，人工检查仍然非常困难。 [9]的研究表明，恶意软件包通常不使用混淆，或仅使用简单技术（例如 Base64）；不过，我们研究的软件包中有一小部分使用了更复杂的混淆机制，例如加密代码（如 AES-256），或对字符串和标识符进行自定义编码。对于编译后的代码，加壳、死代码插入或子程序重排序等知名技术会增加逆向工程和分析的难度。

攻击要成功，恶意代码需要被执行。攻击者可以在安装时、测试用例期间或运行时实现这一点（例如，将有效载荷嵌入特定的函数或初始化器中）。只有在某些生态系统中，才可以在安装时执行代码。对于Python 和 Node.js，这通常通过安装钩子实现，这些钩子会触发执行下载的软件包中提供的代码（例如，Python 中的 setup.py 或JavaScript 中的 package.json）。大多数编译型语言（如 Java 或 C/C++）都不具备类似功能。

为了提高成功的可能性，攻击者可能会开展恶意软件活动，即分发大量实施相同恶意行为的软件包，以扩大对下游用户的影响范围。这些活动可能只影响一个生态系统，也可能是跨语言的（即以不同语言实施相同的恶意行为，并传播到相关生态系统）。

恶意软件报告

恶意软件包如何报告给相应的安全团队，因生态系统而异。例如，NPM 仅通过官方网站中的用户界面提供报告功能：报告者

[8]正在投稿至要求匿名的会议的论文

必须在搜索栏中搜索软件包，导航到项目页面，点击“报告恶意软件”按钮，最后填写报告表单。如果在短时间内提交多份报告，网站将启用验证码检查。对于PyPI，则官方报告恶意软件包的流程是向安全团队发送电子邮件[9]，邮件中包含软件包名称以及（最好还包括）使用 *Inspector*[10]突出显示的恶意软件代码行链接。由于这些流程的结构方式，同时报告多个恶意软件会变得复杂，尤其是考虑到恶意软件活动的普遍性。

> *"[当前的报告机制] 使得同时报告多个恶意软件变得复杂，尤其是考虑到恶意软件活动的普遍性。"*

一经报告并确认，恶意软件包就会从软件包仓库中移除，不再公开访问。鉴于该问题的重要性，以及学术界和业界都在积极开发软件供应链攻击检测方法，维护一个包含这些攻击（包括软件包及相关-联元数据）的数据集将大有裨益。

> *一旦报告并确认，恶意软件包会从软件包仓库中移除，不再公开访问。[……] 维护这些攻击的数据集（包括软件包及相关元数据）将会很有益。*

旅程结束了吗？收益与未解决的挑战。

我们的工作通过抽象、联系上下文并分类现有研究，系统梳理了有关开源软件供应链安全的知识。所提出的分类体系可为未来研究提供一个统一的参考点和一套通用术语。全面的攻击向量和防护措施清单可用于评估开源项目的安全水平，例如，对不同项目、生态系统及不同时期开展比较性实证研究。

开源软件供应链攻击的一项开放挑战是恶意代码的检测。解释型语言生态系统中源代码的可用性，意味着恶意软件分析会更直接一些。不过，近期发表的研究主要关注这些生态系统，尤其是 JavaScript 和 Python [5]，部分原因是它们广受欢迎，另一个原因是现有的恶意软件分析技术难以直接应用。更隐蔽的攻击（例如故意植入漏洞）会加大检测难度，因为必须分析变更的上下文，才能将其与意外引入的漏洞区分开来。此外，代码生成以及难以识别与预构建组件相对应的版本控制系统提交，也使源代码的恶意软件分析变得困难。

参考文献

1. Nicholas Boucher 和 Ross Anderson。特洛伊木马源代码：隐形漏洞，2021。2. 欧洲网络与信息安全局。ENISA 2021年威胁态势，2021。 [在线；访问日期：2021]年10月20日。3. Joseph R. Biden JR. 改善国家网络安全的行政命令，2021。 [在线；访问日期：2021]年10月20日。4. Alexander Kjäll、Stian Kristoffersen 和 Ståle Pettersen。我们如何防范依赖项混淆攻击。网址：*schibsted.com/blog/dependency-confusion-how-we-protected-ourselves/*，2021。5. P. Ladisa、H. Plate、M. Martinez 和 O. Barais。综述：开源软件供应链攻击分类体系。载于*2023*年 *IEEE* 安全与隐私研讨会（*SP*），第167–184页，美国加利福尼亚州洛斯阿拉米托斯，2023年5月。IEEE计算机学会。6. Piergiorgio Ladisa、Henrik Plate、Matias Martinez、Olivier Barais 和 Serena Elisa Ponta。软件供应链风险探索器：了解基于开源软件的开发的攻击面。载于*2022*年 *ACM* 软件供应链进攻性研究与生态系统防御研讨会论文集，SCORED’22，第35–36页，美国纽约州纽约市，2022。计算机协会。

7. Stuart McClure、Shanit Gupta、Carric Dooley、Vitaly Zaytsev、Xiao Bo Chen、Kris Kaspersky、M Spohn 和 R Permeh。保护你的关键资产——

[9] https://pypi.org/security/ [10] https://inspector.pypi.io

极光行动的经验教训。技术报告，2010。8. Ryan Naraine。开源 ProFTPD 遭黑客攻击，源代码中被植入后门。
h
t
t
p
s://
w
w
w.
z
d
n
e
t.
com/article/open-source-proftpd-hacked-backdoor-planted-in-source-code/。 [访问日期：15-Mar-2022]。9. Marc Ohm、Henrik Plate、Arnold Sykosch 和 Michael Meier。《背刺者的刀具收藏：开源软件供应链攻击综述》，2020。10. Sean Peisert、Bruce Schneier、Hamed Okhravi、Fabio Massacci、Terry Benzel、Carl Landwehr、Moham- mad Mannan、Jelena Mirkovic、Atul Prakash 和 James Bret Michael。SolarWinds 事件的观点。*IEEE*《安全与隐私》，19(2):7–13，2021。11. Adriana Sejfia 和 Max Schäfer。恶意 npm 软件包的实用自动化检测。*arXiv* 预印本 *arXiv:2202.13953*，2022。12. Sonatype。《第八份年度软件供应链现状报告》。
https://www.sonatype.com/state-of-the-software-supply-chain/introduction, 2023。[访问日期：01-Feb-2023]。13. PyTorch 团队。2022 年 12 月 25 日至 12 月 30 日期间 pytorch-nightly 依赖链遭入侵。
https://pytorch.org/blog/compromised-nightly-dependency/。 [访问日期：02-Feb-2023]。14. Nikolai Philipp Tschacher。编程语言包管理器中的拼写仿冒。博士论文，汉堡大学，计算机科学系，2016。