开源软件高级漏洞扫描：Log4j 漏洞的检测与缓解

Victor Wen 和 Zedong Peng

美国蒙大拿州米苏拉市，蒙大拿大学，
victor.wen@umconnect.umt.edu，
zedong.peng@mso.umt.edu

arXiv:2601.00235v1 [cs.SE] 1 Jan 2026

摘要——**自动检测软件漏洞仍然是软件安全领域的一项关键挑战。Log4j 是一款工业级 Java 日志框架，被列为 100 个关键开源项目之一。2021年12月10日，严重漏洞 Log4Shell 被披露，直到2021年12月18日发布 Log4j2 2.17.0版后才得到完整修复。然而，时至今日，过去 7 天内约有 410 万次 Log4j下载量（占总量的 33%）仍包含易受攻击的软件包。此后，许多 Log4Shell扫描器相继问世，用于检测用户安装的 Log4j 版本是否存在漏洞。目前的检测工具主要着眼于识别已安装的 Log4j 版本，因此产生了大量误报，因为它们并未检查所扫描的软件是否确实可被恶意攻击者利用。本研究旨在开发一种先进的 Log4j扫描工具，以评估软件在真实环境中的可利用性，从而减少误报。我们的方法首先识别漏洞，然后针对检测到的漏洞提供有针对性的缓解建议，并即时向用户反馈。借助 GitHub Actions，我们的工具可实现自动化持续扫描，确保在代码发生变更时及时发现漏洞。将该工具集成到现有开发工作流中，可实现实时监控，并更快地响应潜在威胁。我们通过评估不同发行版本中的28个开源软件项目，验证了该方法的有效性；在140次扫描的样本中，准确率达到91.4%。我们的 GitHub Action 实现已发布在 GitHub Marketplace，任何有意提升软件安全性或开展后续研究的人都可以使用。本工具为检测和缓解开源项目中的漏洞提供了一种可靠的方法。**索引词——**开源软件、漏洞检测、Log4j、GitHub Action**

I. 引言

如今，绝大多数代码库都包含开源软件（OSS）。根据《开源软件安全性与风险分析（OSSRA）2024年报告》，多个行业中约有 96% 的代码库使用开源代码，而在这些代码库中，平均有 77% 的代码包含 OSS [1]。然而，令人担忧的是，在接受风险评估的代码仓库中，有 84% 至少包含一个开源漏洞。鉴于 OSS 发挥着至关重要的作用，了解其中的漏洞变得尤为重要。OSS 通常通过直接下载引入代码，或者在许多情况下作为软件生态系统的一部分进行安装例如 NPM、NuGet 和 Maven。尽管在部署安全补丁后，更新任何过时的软件包都很容易，但在许多情况下，组织可能并不知道其软件包中存在漏洞。许多组织通常会在其流水线中制定软件安全治理规则，但这些规则只有在对特定代码仓库进行更改时才会触发。因此，如果项目未进行更新，流水线可能要到数月甚至数年后才会发现漏洞。由于项目的依赖项也可能包含漏洞，这一问题可能会扩展到原始代码仓库之外 [2]。

一种方法是依据治理规则扫描代码库，以检测当前存在的通用漏洞披露（CVE） [3]。这些开源组件治理规则可以在 Azure DevOps、Maven Enforcer 中设置并启用，也可以作为 GitHub Action 使用。组件治理能够检测过时或易受攻击的软件包，这些问题可以通过升级到不存在漏洞的版本来解决。然而，有时开发者可能需要更新代码库的其他部分，以适应升级，因为如果需要进行主版本升级，可能会出现一些破坏性变更。重构代码可能会耗费大量时间，因为开发者需要准确找出所有破坏性变更的位置。开源软件通常保持向后兼容，但如果包含漏洞的软件包中的旧类也需要更新，这也可能引发其他问题。

这就引出了本研究的重点——Log4j，这是一款由 Apache 开发的工业级日志软件。Log4j 尤为重要，因为它被应用于全球最大的 Java 软件包仓库 Maven 生态系统中超过 8% 的项目。关键开源项目（如 Log4j）之所以被视为生态系统中的“关键”项目，是基于多项因素，例如总下载量、贡献者数量以及对该项目的依赖项数量。2021年12月10日，一项严重漏洞 CVE-2021-44228（称为 Log4Shell）被公开披露；针对 Log4j 的 JNDI 查找可被利用，从 LDAP（轻量级目录访问协议）服务器加载并执行任意代码2 [4]。Apache 于 2021年12月17日彻底修复了该漏洞，但在此之前，已有数百万台设备

遭到漏洞利用。2021年发现该漏洞时，数百万台设备受到影响，超过 35,000 个 Java 软件包受到 Log4j 漏洞的影响 [5]。

虽然 Netty 和 MyBatis 等代码库分别于2021年12月19日和2021年12月18日 [6],更新至 Log4j2 2.17.0版，但许多组织仍花费数月时间苦于应用必要的补丁。对于使用 Log4j 1.x 版的组织而言，情况尤其如此；该版本自2015年以来便不再受支持且已被弃用[7]。此外，代码仓库中开源漏洞的平均存在时间达到令人担忧的2.8年，其中近一半代码库在过去两年内处于不活跃状态[8]。这表明，无论规模大小，组织通常都缺乏及时解决这些漏洞所需的资源，因为它们需要在漏洞修复与新功能开发之间进行权衡。

此外，NextGen Health-care 的 Mirth Connect 案例还凸显了另一个方面。在 Log4Shell披露时，Mirth Connect 使用的是不受支持的 Log4j 版本 1.2.16 [9], ，该版本已有六年多未获得更新。尽管其中包含易受各种安全风险影响的已弃用软件包，例如允许数据反序列化攻击的 CVE-2021-4104，但由于 Log4j 1.x 版中缺少 JndiLookup 类，Mirth Connect 并不易受 CVE-2021-44228 影响。此外，由于未使用 JMSAppender 类进行日志记录，它也未暴露于 CVE-2021-4104 [9]。Mirth Connect 直到2022年7月26日才过渡到安全的 Log4j2 2.17.2 版。

这引出了一个关键问题：这些被标记为至少存在一个开源软件漏洞的代码库中，有多少确实存在漏洞？CVE 报告又有多可靠？例如，让我们来研究一下由 Micromatch 维护的 braces 软件包。今年早些时候，braces 被曝存在 CVE-2024-4068（CVE 评分：7.5），该漏洞未限制程序能够处理的字符数量，从而导致内存耗尽。尽管有相关 CVE 报告，braces 实际上并不可利用，除非用户在本地自行利用它，因为无法通过直接输入提交任何正则表达式 [10]。因此，CVE-2024-4068 并不是真正的安全漏洞，充其量只是一次性能提升——这是在 Micromatch 审查并测试这些软件包之后得出的结论。

我们的研究旨在开发一种先进的扫描工具，不仅能够通过 GitHub Action 和本地计算机扫描代码仓库，还能评估 Log4Shell 等漏洞在这些代码库中的实际可利用性。借助 GitHub Actions，我们的工具能够提供自动化持续扫描功能，确保在代码发生更改时及时识别漏洞，并根据 CVE 报告提供缓解方案。

我们通过在 28 个公开提供源代码的开源软件项目上评估该方法，展示其有效性，准确率达到 91.4%。本研究的主要贡献是开发了一种先进的扫描工具，它不仅能检测易受攻击的软件包，还能针对检测到的漏洞提供有针对性的缓解建议，为软件安全管理提供新的支持。接下来，我们将在第二节介绍背景信息，然后讨论开源软件的重要作用以及第三节中漏洞检测面临的挑战。第四节详细介绍我们的先进扫描方法，第五节阐述实证评估，最后，第六节总结全文。

II. 背景

本节将讨论开源软件的背景及其潜在安全风险。随后，我们将探讨本项目的方法和挑战。

A. 开源软件

目前有数百万个开源项目，其中绝大多数代码仓库托管在 GitHub 等网站上。开源软件以公开方式开发，任何人都可以依据开源许可证使用其源代码和文档。开源许可证有许多种，例如 MIT 和 Apache 2.0，但它们都遵循相同的基本标准。采用开源许可证的项目必须免费公开，包含源代码，并允许任何人出于任何目的修改代码，且不得有所歧视 [11]。因此，较受欢迎的开源软件项目可能会衍生出数千种变体，以满足不同组织的使用需求。这种灵活性带来了诸多好处，例如降低成本；同时，由于可以直接访问源代码并提高代码可复用性，也有助于提升安全性 [2]。然而，开源软件的广泛采用和修改也带来了重大挑战 [12]。其中一个主要问题是，难以在项目的多个版本和变体中维护安全性。由于贡献者成千上万，分支版本不计其数，漏洞可能被引入却未被发现，从而造成安全风险，并在不同版本间传播。此外，组织可能难以跟上最新安全补丁，尤其是在涉及多个开源软件项目的依赖项时 [13]。许多组织缺乏全面的开源软件组件跟踪和管理系统，这使问题更加严重，导致过时且可能存在漏洞的软件仍在使用。这些挑战凸显了强大的漏洞检测和管理工具的重要性，以确保依赖开源组件的软件安全可靠。

B. 开源代码中的安全漏洞

开源软件灵活性提高是一把双刃剑，因为第三方不仅更容易发现并报告漏洞，恶意行为者也更容易检查源代码并利用这些漏洞 [14]。之所以令人担忧，是因为利用公开报告的漏洞通常比打补丁更快。一些开源软件项目维护不善，导致用于解决安全漏洞的安全补丁部署延迟。由于开源项目是去中心化的，组织需要自行负责维护其代码库中的版本管理。然而，许多组织由于开发人员资源有限，导致补丁迟迟无法实施。开发者

不仅必须直接为受影响的软件打补丁，还必须处理使用易受攻击的版本的项目依赖项。

含有易受攻击依赖项的项目，其影响不容小觑，因为单个漏洞就可能导致大规模服务中断。7月19日，一家名为 CrowdStrike 的网络安全公司发布了一项更新，导致全球 IT 故障，约 850 万台 Windows 设备出现蓝屏死机（BSOD），这被称为“史上最大规模的 IT 服务中断” [15]。尽管这并非由漏洞利用所致，但这表明，如果开源软件中的安全漏洞未得到修补并被恶意行为者利用，也可能造成连锁反应。

我们来看看 Apache 的 Log4j2 软件包中的 Log4Shell 漏洞 CVE-2021-44228，该漏洞影响了数亿台设备 [16]。查看官方 CVE 报告可知，该报告涉及 Log4Shell 漏洞，攻击者可通过 Log4j2 的 JNDI 功能执行远程代码执行（RCE） [17]。这为黑客提供了利用使用易受攻击版本 Log4j 的软件所需的全部信息，因为报告明确列出了受影响的功能，使恶意行为者更容易将攻击集中在 JNDI 端点上。由于 Java 项目广泛使用 Log4j 日志框架，这一 CVE 也体现了开发者尝试修补漏洞时面临的挑战之一。可复用库可能会对任何依赖该项目的软件产生安全影响 [2]：Arduino IDE、Netty、MyBatis 和 Elasticsearch 只是受 CVE-2021-44228 影响的其他依赖 Log4j 的开源软件项目中的几个例子。

C. 现有的漏洞检测方法

Log4Shell 漏洞披露后，人们开发了扫描器，用于扫描软件中受影响的软件包。早期的一些版本非常简单，只会检查软件包版本，例如 log4j-checker[18]。后来，人们开发了更先进的扫描器，用于扫描托管应用程序中的 Log4Shell，例如 log4j-scanner [19]。这些扫描器主要针对 HTTP 相关端口，通过添加有效载荷来检测漏洞 [20]。

除了这些基础扫描器外，人们还开发了更先进的检测方法。例如，Veracode 平台提供静态和动态应用程序测试，以识别第一方代码和开源依赖项中的漏洞。Veracode 还提供工具来监控网络流量，检查是否有可疑的 Java 类下载，这可能表明存在利用尝试 [21]。此外，Data-dog 等平台集成了检测 Log4Shell 漏洞利用尝试的功能：通过监控可疑的 Java 类下载，并分析网络流量以查找异常的 LDAP 连接[22]。虽然这些扫描器和工具在识别漏洞方面十分有效，但也面临误报率高、需要持续更新以应对新型漏洞利用变体等挑战。结合使用这些工具，并维护一份使用 Log4j 的资产清单，有助于加强对此类漏洞的检测和缓解。

D. 当前检测方法的不足

目前的漏洞扫描器通常侧重于通过标记已知存在漏洞的过时软件包（例如受 Log4Shell 影响的软件包），来识别代码库中的高风险软件包。另一种做法是使用 log4j-scanner 等专用工具，在 JNDI 查找字符串中模拟有效载荷；如果模拟攻击导致远程代码执行，就会将该软件标记为存在漏洞 [19]。

然而，这些扫描器存在显著局限。首先，它们无法解释特定代码库为何容易受到漏洞影响 [23, 24]。它们主要用于识别是否存在漏洞，却无法分析和说明暴露于漏洞的根本原因。缺乏详细分析，使开发者和安全团队难以了解漏洞的根源并采取适当的补救措施。此外，这些工具通常会产生大量误报：仅仅因为存在特定版本的软件包，就将软件标记为存在漏洞，而不考虑该软件在其运行环境中的实际可利用性。

E. GitHub Action

GitHub Actions 是一种强大的 CI/CD 工具，可在持续集成和持续部署流水线中促进软件构建、测试和部署的自动化[25]。GitHub Action 的步骤通常配置在代码仓库的 .github/workflows 目录中的 YAML 文件内。这些 YAML 文件定义了在特定事件触发时要执行的作业，例如代码推送、拉取请求或定时间隔触发的事件。

先前的研究强调了 GitHub Actions 在软件开发中的影响和应用。例如，Kinsman 等人[25] 研究了 3,000 多个代码仓库，以调查采用 GitHub Actions 后各种开发活动指标的变化。Valenzuela-Toledo 和 Bergel [26] 对热门 GitHub 代码仓库中 GitHub Actions 工作流的使用和维护实践进行了研究，并确定了各种类型的工作流修改。使用 GitHub Actions 进行漏洞扫描的主要优势在于，它能够向开发者提供即时反馈。代码发生新的更改并推送到代码仓库后，GitHub Action 会立即触发，运行配置的安全扫描并报告检测到的任何问题。这使开发者能够在开发过程的早期处理漏洞，从而降低生产环境中发生安全漏洞的风险。同样，我们开发了一个定制的 GitHub Action，用于自动扫描代码仓库中的潜在 Log4j 漏洞。

III. 方法论

在许多情况下，尽管代码仓库中可能包含易受攻击的软件包，但这些软件包未必能被利用。为弥补扫描器能力上的这一不足，我们提出了一种直接分析代码库的扫描器。为实现这一目标，我们检查了 28 个开源代码仓库，涵盖数百个项目版本，包括 Netty、Mirth Connect [27],MyBatis，以及各种

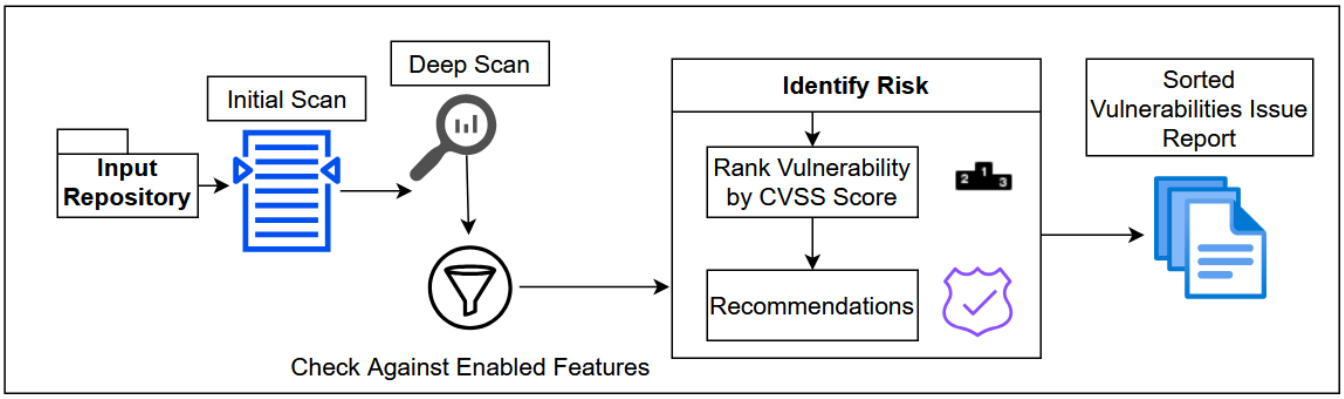

图 1. 扫描器流程

依赖 Log4j 的 Apache 基金会代码仓库。通过使用同一项目的不同发行版本，我们可以将代码仓库中存在漏洞的版本和已修补版本直接与扫描器的结果进行比较。图 1 概述了我们的方法。我们的方法主要分为三个步骤：初步扫描、高级深度扫描，以及通过漏洞排序和验证进行风险识别。

A. 开源软件的收集

我们方法的第一步是收集一个全面的开源软件仓库数据集。此过程包括：

1）筛选标准：

- **对 Log4j 的依赖**：我们将目标设为依赖 Log4j 的代码仓库，以确保分析所用数据具有相关性和意义。
- **多样化的项目类型**：涵盖来自不同领域、规模各异的项目，包括 Netty、Mirth Connect 和 MyBatis 等热门项目，以及 Apache 基金会的多个代码仓库。
- **版本历史**：优先选择版本历史较长的代码仓库，以便对不同版本进行比较分析。*2)* 数据收集过程：
- **GitHub API 的使用**：我们使用 GitHub API，以编程方式收集元数据，并克隆符合筛选标准的代码仓库。
- **本地存储**：将收集到的代码仓库克隆并存储在本地，以便进行详细分析和重复扫描。*3)* 初步筛选：
- **初步扫描 Log4j 的使用情况**：我们对收集到的代码仓库进行了初步扫描，以确认其中是否存在 Log4j 依赖项。扫描过程中，我们解析了项目配置文件，例如 Maven 项目的 pom.xml，以识别 Log4j 版本。

每个选定的代码仓库都选取了 5 个版本进行扫描，每个项目共计选取 140 个版本。

次扫描。对每个代码仓库扫描相同数量的版本，可以确保所收集数据的一致性，因为每次扫描结果在各个代码仓库中所占的权重相同。每个项目只扫描 5 个版本的另一个原因是 Apache Ozone [28] 的版本数量有限。Apache Ozone 在其生命周期内总共只发布了 5 个版本（1.0.0、1.1.0、1.2.1、1.3.0 和 1.4.0），因此将每个项目的扫描总数限制为 5 次，可确保各代码仓库中每次扫描的权重一致。

B. 初步扫描

我们对算法 1 中展示的收集到的代码仓库进行了初步扫描，以确认其中是否存在 Log4j 依赖项。此过程涉及解析项目配置文件，例如 Maven 项目的 pom.xml，以识别所使用的 Log4j 版本。扫描旨在检测可能存在漏洞的 Log4j 版本，并为代码仓库的深入分析做好准备。初步扫描过程可概括如下：

1) 解析配置文件：解析 pom.xml 文件，以创建文档的结构化表示。

2) 提取依赖项：检查 XML 树的根元素，以识别依赖元素。

3) 识别 *Log4j* 版本：对于每个依赖项，提取制品 ID 和版本。如果制品 ID 中包含“Log4j”，则将其版本与已知的易受攻击模式进行比对：

- 以“1.”开头的版本会被标记为 Log4j 1.x 版漏洞。
- 版本在 2.0 至 2.17.2 之间（不包括安全补丁 2.3.1 和 2.12.3）时，会被标记为 Log4j 2.x 版漏洞。

4) 处理异常：如果解析 pom.xml 文件失败或 pom.xml 不存在，扫描器将默认对整个代码仓库进行深度扫描。

此初步扫描通过识别包含可能使用易受攻击的 Log4j 版本的代码库，提供了一个基准。例如，扫描 MyBatis 3.5.16 版本时，由于其使用的是 Log4j 2.23.1 版本，因此扫描结果未发现漏洞。在

相比之下，MyBatis 3.5.8 版本因使用易受攻击的 Log4j 版本而被标记为存在漏洞，从而触发对源文件夹中所有文件的深度扫描。

通过进行这次初步扫描，我们筛除不含漏洞的代码仓库，从而将高级分析集中在存在潜在安全风险的代码仓库上。此步骤可确保高效利用资源，并使深度扫描流程专注于最相关的代码库。

算法 1 检查 pom.xml 中易受攻击的 Log4j 版本

要求： 文件路径：pom.xml 文件的路径
确保： 发现的漏洞列表

```
函数检查 pom 文件中的易受攻击版本（文件路径）
tree ← 解析文件路径中的 XML 文件
root ← 从树中获取根元素
发现漏洞 ← []
对根元素中的每个依赖项 执行
    制品 ID、版本 ←从依赖项中提取
    如果制品 ID 包含“log4j”则
        如果版本以“1.”开头则
            将发现的 Log4j 1.x 版添加到发现的漏洞
        否则，如果版本匹配易受攻击的模式则
            Add Log4j v2 易受攻击 to 发现的漏洞
        结束 if
    结束 if
结束 for
返回发现的漏洞
结束函数
异常处理：如果解析失败，则打印错误并返回 []
```

算法 2 Log4j 漏洞深度扫描

要求： 目录：要扫描的目录路径
确保： 已发现漏洞列表

```
函数深度扫描 Log4j 漏洞（目录）
已发现漏洞 ← []
对目录和子目录中的每个文件执行
    如果文件不是 ‘pom.xml’那么
        内容 ←读取文件内容
        漏洞 ← 检查 Log4j 漏洞（内容）
        将漏洞追加到发现的漏洞
    结束 if
结束 for
如果 found 漏洞为空则
    返回 “未发现漏洞。”
否则
    返回 found 漏洞
结束 if
结束函数
```

C. 深度扫描

如果初步扫描在配置文件中检测到易受攻击的软件包，流程便会进入更彻底的“深度扫描”阶段。深度扫描会仔细评估软件中 Log4j 已启用和未启用的特性，并按照算法 2 所示，搜索源代码中每个文件里的特定关键词和模式。这些模式根据相应的通用漏洞和暴露（CVE）以及正则表达式定义，其中包含有关特定 Log4j 特性的关键信息。如果启用了这些特性，软件就可能因此受到外部威胁的漏洞利用。在此次深度扫描中，我们针对与已知漏洞相关的特定模式进行检查。具体而言，所有 Log4j 1.x 版均会被标记，因为它们本身存在漏洞。此外，所有低于 2.17.2 版本的 Log4j 2.x 版也会被标记，但安全性已修补的 2.3.2 版（适用于 Java 6）和 2.12.4 版（适用于 Java 7）除外。扫描器不仅会识别版本，还会根据 Log4j 库中已知易受攻击组件的清单进行检查，包括查找在 CVE 中被认定为潜在攻击途径的特定类和配置

1) JNDI 查找类：此模式专门查找 org.apache.logging.log4j.core.lookup.JndiLookup 类，该类与 CVE- 2021-44228 和 CVE-2021-45046 [29] 等严重漏洞有关。CVE-2021-44228 允许攻击者通过 LDAP 服务器远程执行代码 [17],，而 CVE-2021-45046 则允许恶意输入数据利用 JNDI 查找执行远程代码，并发起拒绝服务攻击 [30]。

2) SockerServer 类：查找 org.apache.log4j.net.SocketServer 是否存在，该类与 CVE-2019-17571 [7] 有关。此漏洞影响 Logj1 1.2 至 1.2.17 版本，且 SockerServer 类容易受到不可信数据反序列化攻击，攻击者可通过远程执行代码加以利用 [31]。

3) SMTPAppender 类：识别 org.apache.log4j.net.SMTPAppender 下 SMTPAppender 类的使用情况，该类与 CVE-2020-9488 有关，此漏洞已在 Log4j2.12.3 和 2.13.1 中修复。由于主机不匹配，CVE-2020-9488 会导致 SMTPS 连接遭受中间人攻击，从而被拦截 [32]。这是由 SslConfiguration 中的错误导致的，可能造成日志通过追加器泄露 [29]。

4) JMSAppender 类：此模式用于检测与 CVE-2021-4104 相关的 JMSAppender 类 org.apache.log4j.net.JMSAppender 的使用。攻击者可利用 TopicBindingName 和 TopicConnectionFactoryBindingName 配置，诱使 JMSAppender 发出 JDNI 请求，通过操纵 LDAP [33] 实现远程代码执行。此漏洞影响 Logj1，并与 Log4Shell 漏洞有关。

5) JMSSink 类：搜索与 CVE-2022-23302[7] 相关的 JMSSink 类 org.apache.log4j.net.JMSSink。此 CVE 与 CVE-2021-4104 类似，攻击者可利用 JMSSink 发起 JNDI 请求，由于不可信数据反序列化而导致远程代码执行。只有在恶意行为者拥有

Log4j 配置的写入权限或引用了 LDAP 服务 [34] 时，才可能发生此漏洞利用。

6) JDBCAppender 类：最后，JDBCAppender 类与 CVE-2022-23305 相关，对应 org.apache.log4j.jdbc.JDBCAppender。默认情况下，JDBCAppender 接受作为配置的 SQL 查询，并且几乎总是使用 %m 消息转换器 [7]。因此，JDBCAppender 类容易受到 SQL 注入攻击，攻击者可借此执行非预期的 SQL 脚本 [35]

通过分析活动配置与已知漏洞之间的相互作用，深度扫描可以准确评估软件面临的实际风险。经过这一全面分析后，软件会被标记为“无漏洞”或“存在漏洞”。采用这种深度扫描方法，我们能够准确识别超出简单版本检查范围的潜在漏洞。全面评估对于了解这些漏洞带来的实际风险至关重要，也为我们方法的后续步骤——CVE 排名、建议和人工验证——奠定了基础。接下来的步骤包括量化已识别漏洞的严重性并确保其准确性，从而帮助开发者有效地确定优先级并处理最关键的安全问题。

D. CVE 排名

深度扫描识别出潜在漏洞后，会使用通用漏洞评分系统（CVSS）的基础分 [36]，根据严重程度对每个漏洞进行排名。CVSS 基础分范围为 0 到 10，10 代表最严重的漏洞。该分数以标准化的方式评估漏洞的固有特性，并考虑可利用性和影响等因素。可利用性考量攻击向量、复杂度和所需权限等要素；影响则评估漏洞遭利用时对系统机密性、完整性和可用性的潜在影响。CVSS 基础分的计算方法如下：

$$\text{Base Score} = \lceil \text{Impact} + \text{Exploitability} \rceil \tag{1}$$

其中：

- **影响：** 表示成功利用漏洞对目标系统造成的整体影响。
- **可利用性：** 反映攻击者利用该漏洞的难易程度。

通过使用 CVSS 基础分，我们可以客观衡量每个漏洞的严重程度，帮助开发者有效地安排修复工作的优先级。这确保将注意力集中在对系统安全性构成最大风险的漏洞上，从而实现更高效、更有针对性的方法来进行漏洞管理。

E. 建议操作与报告生成

除了标记存在漏洞的功能，并为每项标注相关的 CVE 编号和 CVSS 基础分外，报告还将向用户提供以下有用信息：可采取哪些措施来缓解漏洞，包括修复漏洞所需的最低 Log4j 版本，以及通过更新配置或禁用有问题的功能来消除安全风险的步骤。

根据检测到的 CVE，扫描器将提供以下建议操作：

1) “CVE-2021-44228：” 升级到 Log4j 2.17.1 或更高版本。如果无法升级，可将系统属性 log4j2.formatMsgNoLookups [37]设置为 true，或从类路径中移除 JndiLookup 类来缓解风险。

2) “CVE-2021-45046”：升级到 Log4j 2.17.0 [4] 或更高版本。此漏洞是 CVE-2021-44228 的扩展，必须更新才能有效降低风险 [30]。也可以通过移除 JndiLookup 类来缓解此漏洞。

3) “CVE-2021-45105”：升级到 Log4j 2.17.0 或更高版本。此漏洞与自引用查找导致的失控递归有关。更新配置，移除对 Context 查找的引用，或将其替换为 Thread Context Map 模式，即可修复此漏洞[4]。

4) “CVE-2021-44832”：升级到 Log4j 2.17.1 或更高版本。此漏洞影响 Log4j 2.0-beta9 至 2.17.0，涉及因配置不当而导致的远程代码执行漏洞 [17]。如果目前无法升级，也可以将 JDBCAppender 设置为仅接受来自 Java 协议的 JNDI 数据源 [4]。

5) “CVE-2019-17571”：升级到 Log4j 2.8.2 或更高版本。此漏洞影响 Log4j 1.x，允许对不可信数据进行反序列化，从而导致远程代码执行 [31]。可通过从 JAR 包中删除 SocketServer.class 来缓解此问题。

6) “CVE-2020-9488”：升级到 Log4j 2.13.2 或更高版本。此漏洞涉及一种可能导致拒绝服务或远程代码执行的错误配置 [32]。可通过将 mail.smtp.ssl.checkserveridentity 设置为 true 来启用主机名验证，从而缓解此主机不匹配漏洞 [4]。

7) “CVE-2021-4104”：此漏洞影响 Log4j 1.x 版本。由于 Log4j 1.x 已不再受支持，最佳做法是升级到 Log4j 2.x。如果无法升级，请确保日志配置文件中未配置 JMSAppender [7]。

8) “CVE-2022-23302”：升级到 Log4j 2.17.1 或更高版本。此漏洞允许远程攻击者通过构造的输入执行代码，可通过在配置中禁用或移除 JMSSink [34]来缓解。

9) “CVE-2022-23305”：升级到 Log4j 2.17.1 或更高版本。此漏洞可能通过失控递归导致拒绝服务（DoS），应通过更新或移除配置中的 JDBCAppender 来缓解。

10) “CVE-2022-23307：升级到 Log4j 2.17.1 或更高版本。此严重漏洞允许远程代码执行，应立即修复。可以通过从配置中移除对 Apache Chainsaw [38]项目的依赖项来缓解风险。

11) “潜在的配置错误”： 移除所有不需要的附加器，并确保所有配置均为最新版本。

开发者可以优先采取措施，立即缓解已发现的漏洞，例如禁用受影响的功能，或从 Log4j 配置文件中移除存在漏洞的类和追加器。这样可以确保项目在官方安全补丁发布前不会受到已检测到的 CVE 的漏洞利用。本文的评估章节将进一步详细探讨这一点。扫描器还会生成一份报告，列出所有已启用且可能遭到漏洞利用的易受攻击功能。用户随后可以采取措施，将软件包更新到不存在漏洞的版本，或通过精简功能、移除未使用或存在风险的代码来禁用特定功能。

F. 用于扫描器的 *GitHub Action*

下一步是创建一个公开可用的 GitHub Action，以便轻松扫描任何 GitHub 仓库。我们创建了一个名为 Log4jDeepScanAction 的 GitHub Action，可轻松集成到 CI/CD 流水线中。它会使用现有的检出操作 [39] ，将代码仓库检出到“$GITHUB WORKSPACE”下，使我们的工作流能够访问该仓库并针对其运行扫描。这使开发者能够在代码变更时实时使用扫描器进行检查，因为扫描器的输出会显示打补丁后代码库是否仍存在漏洞。我们的 GitHub Action 实现可在 GitHub Marketplace 上获取（ https://github.com/marketplace/actions/log4j-vulnerability-scanner）。

G. 验证与挑战

在此项目中，我们收集并下载了所选代码仓库及发行版本的源代码到本地，以便使用脚本进行扫描。这是因为开源软件并非全部存储在 GitHub 上，因此除非有可用的最新发行版，否则并不总能使用 GitHub Action。随后，我们记录每个版本的结果，并与官方发行说明、问题列表和文档进行比较。例如，我们安装并扫描了 MyBatis v3.5.16（最新版本）、v3.5.8 和 v3.5.9。根据发行说明，扫描器应判定 v3.5.9 和 v3.5.16 不存在漏洞，同时将 MyBatis v3.5.8 [40] 标记为存在漏洞。这是因为我们可以看到，官方问题列表和发行说明指出，MyBatis 在 v3.5.9 中升级到了 Log4j 2.17.0。然而，扫描器报告称 MyBatis v3.5.9 存在 CVE-2021-45105 漏洞，该漏洞可能导致拒绝服务 [41]；但根据发行说明，此版本理应已不再存在漏洞，因为 MyBatis 可以在不使用 Log4j 的情况下运行，而且 pom.xml 中的 Log4j 配置是可选的。针对不同发行版本进行扫描，可以将扫描器的效果与 Log4j 漏洞补丁的官方发行说明进行比较。

在排名过程中，我们优先使用 CVSS 的基础分对开源项目中的漏洞进行排序。虽然 CVSS 框架还包含时间分和环境分，但要准确评估这些指标在开源软件的背景下颇具挑战性 [36, 42–44]。

时间评分旨在反映漏洞的当前状态，同时考虑漏洞利用可用性、修复级别和报告置信度等因素[44]。然而，开源项目的动态特性使得难以持续准确地评估这些因素。开源项目通常会快速变化，频繁进行更新和发布补丁，因此很难始终如一地评估修复级别。同样，由于开源软件 [44] 的部署环境多种多样，准确评估环境评分也存在困难。这些项目可能采用各种配置和安全措施进行部署，因此很难概括出适用于所有用户的环境评估结果。在分析开源项目时，这些上下文细节往往未知或不可用，使得环境评分难以确定。

IV. 评估

本节将考察我们的扫描器在检测存在漏洞的 Log4j 软件包及其特性方面的有效性。此外，我们还将分析生成的报告，以根据官方发行说明验证扫描器结果的准确性。在对该高级漏洞扫描工具进行评估时，我们重点分析了扫描器针对各类开源代码仓库的有效性和准确性。这些代码仓库是根据其与 Log4j漏洞的相关性及历史情况选定的，从而确保能够针对不同版本和配置进行全面评估。

A. 输入代码仓库

为了评估我们的高级漏洞扫描工具，我们选取了一组多样化的开源代码库。这些代码库因其与 Log4j 漏洞的相关性及历史记录而入选。本次全面评估涵盖了 Apache 基金会以及其他重要开源贡献者的多个项目。所有实验材料均可公开获取：https://doi.org/10.5281/zenodo.13188600。

表 I 总结了所分析的 28 个代码仓库，并提供了代码仓库名称、作者、发布日期、最后更新日期以及被确定为存在 Log4j 漏洞的具体版本等关键信息。选择这些代码仓库时，我们考虑了几个关键因素。首先，每个代码仓库都有与 Log4j 相关的漏洞记录，为测试扫描器的有效性提供了丰富的数据集。其次，这些代码仓库既包含中型项目，也包含大型项目，从而确保全面评估扫描器在不同软件复杂度下的有效性。在这些代码仓库中，有相当一部分（28 个中有 21 个）被归类为大型项目 [69], , 每个项目的代码行数（LOC）都超过 100,000 行。这些大型项目，如 Apache Spark 和 Elasticsearch，代表了企业环境中常用的大型复杂系统。28 个代码仓库中有 7 个被归类为中型项目，代码行数为 20,000 至 100,000 [69]。这类中型项目包括

表 I 代码仓库及其相应信息列表

代码仓库名称	Ref.	作者	发布版本	最后更新	易受攻击的版本
Apache Spark	[45]	Apache	2014年5月26日	2024年4月18日	Apache Spark：1.0.0 - 3.2.0
Arduino IDE	[46]	Arduino	2011年11月30日	2024年2月20日	Arduino IDE：1.8.18 之前
Apache Hive	[47]	Apache	2013年1月11日	2024年5月20日	Apache Hive：0.6.0 - 3.1.2
Apache Wicket	[48]	Apache	2005年6月	2024年6月17日	Apache Wicket 8.13.0 及更早版本
MyBatis-3	[40]	MyBatis	2010年5月19日	2024年4月4日	MyBatis-3：5.8 及更早版本
Netty	[49]	Netty	2004	2024年7月19日	4.1.72 及更早版本
Mirth Connect	[9]	NextGen Healthcare	2006年7月18日	2024年7月27日	4.0.0 及更低版本
Elasticsearch	[50]	Elastic	2010年2月10日	2024年3月11日	低于 7.16.1 / 6.8.21
Log4j2	[29]	Apache	2013年1月8日	2024年3月10日	2.0-beta7至2.17.0（不包括2.3.2和2.12.4）
phoss-smp	[51]	phax	2016年8月7日	2024年5月24日	phoss SMP 5.5.0及更早版本
Apache Pulsar	[52]	Apache	2016年8月31日	2024年6月5日	Apache Pulsar：2.8.2及更早版本
Apache Tapestry	[53]	Apache	2002年4月	2024年4月16日	Apache Tapestry：5.0 - 5.7.3
Apache NiFi	[54]	Apache	2015年7月26日	2024年7月1日	Apache NiFi：0.1 - 1.15.0
Apache Traffic Control	[55]	Apache	2018年1月22日	2024年4月3日	Apache Traffic Control：1.1.2 - 6.0.1
Apache SkyWalking	[56]	Apache	2015年12月30日	2024年5月29日	SkyWalking：3.0.3 - 8.9.0
Apache OFBiz-Framework	[57]	Apache	2010年4月	2024年5月	OFBiz：4.0 - 18.12.02
Apache JMeter	[58]	Apache	1998年12月15日	2024年1月9日	Apache JMeter：2.0 - 5.4
Apache Jena	[59]	Apache	2000年8月28日	2024年7月12日	Jena：2.7.1 - 4.3.0
Apache Geode	[60]	Apache	2016年10月15日	2024年5月	Apache Geode：1.12.0 - 1.14.0
Apache Fortress	[61]	Apache	2015年4月15日	2023年9月6日	Apache Fortress：2.0.6
Apache Druid	[62]	Apache	2014年6月18日	2024年6月16日	Apache Druid：0.22.1之前
Apache Calcite Avatica	[63]	Apache	2013年11月6日	2024年5月6日	Apache Calcite Avatica：1.10.0 - 1.19.0
Apache Archiva	[64]	Apache	2005年11月	2023年3月20日（已退役）	Apache Archiva：1.0 - 2.2.5
Apache Tika	[65]	Apache	2007年3月27日	2024年7月15日	Apache Tika：1.0 - 2.2.0
Apache Solr	[66]	Apache	2006年12月22日	2024年5月29日	Apache Solr：7.4.0 - 8.11.0
Apache Flink	[67]	Apache	2014年8月26日	2024年3月18日	Apache Flink：1.10.0 - 1.14.0
Apache EventMesh	[68]	Apache	2020年8月20日	2023年12月19日	Apache EventMesh：1.0.0 - 1.2.0
Apache Ozone	[28]	Apache	2020年9月2日	2024年1月19日	Apache Ozone：1.0.0 - 1.2.0

MyBatis-3 和 Log4j2，它们是广泛使用的框架，在复杂度与可管理性之间实现了良好平衡。

所选代码仓库的发布日期和最后更新日期跨度较大，表明这些项目既具有较长的生命周期，也保持着当前活跃度。例如，Apache Spark[45], 于2014年首次发布，并于2024年最后更新，代表了一个维护良好且持续更新的项目。同样，纳入Apache JMeter[58], 等较早的项目（其首次发布于1998年），凸显了Log4j漏洞在不同软件发展阶段中的长期相关性。

通过将扫描器的检测能力与已记录的漏洞和官方发行说明进行比较，我们旨在验证其准确性和可靠性。

B. CVE 排名与分析

表 II 详细列出了与Log4j及其他相关软件有关的各种 CVE，并按照其CVSS 评分进行排序。CVE-2021-44228和 CVE-2022-23307 [17] [38],的评分均为10.0，被确定为最严重的漏洞，凸显了它们对系统安全性的重大影响。CVE-2022-23307与涉及CVE-2020-9493的反序列化问题有关；Apache Chainsaw项目中的Java 反序列化可能导致恶意代码执行 [70]。该问题影响所有低于2.1.0版本的 Apache Chainsaw，而由于Log4j1依赖低于2.0.0版本的 Apache Chainsaw，因此也会受到影响。其他值得注意的CVE（如CVE-2021-45046和CVE-2022-23302）评分也较高，均超过9.0，进一步凸显了它们带来的重大风险。

我们对 140 个代码仓库进行扫描的结果如图 2 所示，并在 https://doi.org/10.5281/zenodo.13188600 分享了更详细的实验材料。我们发现了几种严重漏洞的大量实例。值得注意的是，CVE-2021-44228 和 CVE-2021-45046 均为严重程度极高的漏洞，CVSS 评分分别为 10.0 和 9.0，在 50 个和 54 个代码仓库中被发现，凸显了它们广泛遭到利用的风险。相比之下，尽管 CVE-2021-44832 的 CVSS 评分较低，却出现在 65 个代码仓库中，表明这一威胁可能未受到足够重视。CVE-2022-23307 仅在 15 个代码仓库中被发现，出现频率较低，这可能意味着它在实际环境中较少遭到利用，也可能说明这些环境中已采取了更有效的缓解策略。

表 II 按评分排序的 CVE 评分

CVE 标识符	评分
CVE-2021-44228	10.0
CVE-2022-23307	10.0
CVE-2021-45046	9.0
CVE-2022-23302	9.0
CVE-2022-23305	9.1
CVE-2019-17571	9.8
CVE-2021-45105	7.5
CVE-2020-9488	7.5
CVE-2021-4104	7.5
CVE-2021-44832	6.6
潜在的配置错误	5.0

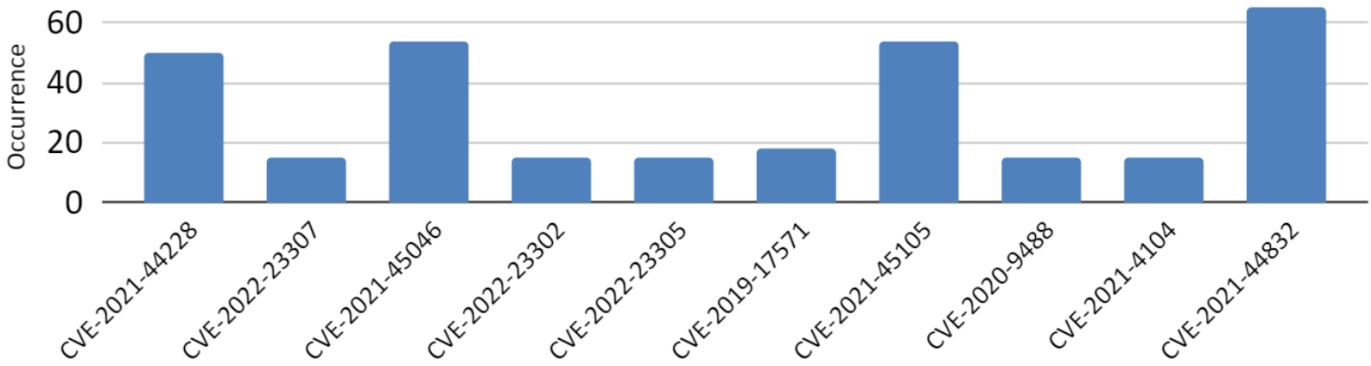

图 2. CVE 出现情况

C. 结果与建议

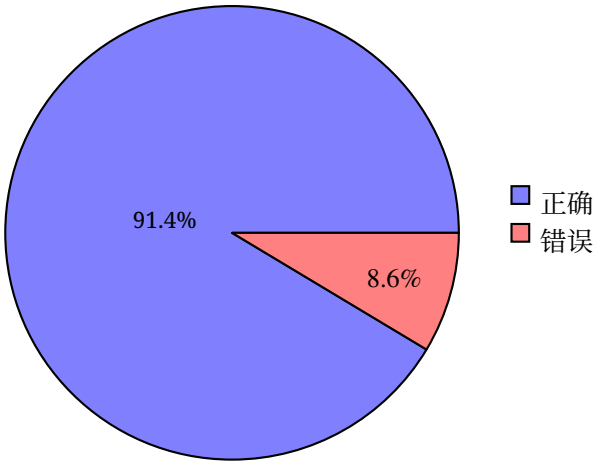

图 3. 正确扫描与错误扫描的分布

对 Log4Shell 漏洞（CVE-2021-44228）补丁发布日期的分析 [17] 显示，不同开源项目的响应时间存在显著差异。大多数项目在一周内修复了该漏洞，平均补丁发布时间约为 5.83 天；Apache Spark、Apache Traffic Control、Apache Wicket 和 Mirth Connect 除外，因为这些项目依赖的是 Log4j 1.x，而不是 Log4j 2.x。对 Log4Shell 补丁时间线的评估揭示了不同项目在漏洞管理方面的成效。如此迅速的响应，体现了研究方法强调的一个重点：必须迅速采取行动，缓解 Log4j 等广泛使用的库所带来的安全风险。Elasticsearch 和 Apache SkyWalking 等项目表现出色，在漏洞披露后的一两天内便发布了补丁，凸显了建立完善的安全协议和高效部署流程的重要性。

这项评估还凸显了响应时间上的差异：Arduino IDE 和 Apache Tapestry 等项目分别花了 12 天和 11 天才为 Log4Shell 实施补丁。这表明，资源限制、需要进行广泛测试或项目特有的复杂性等因素，可能会延迟补丁发布。造成延迟的部分原因是，最初针对 Log4Shell 发布的补丁 Log4j2.15.0 被发现无法充分应对非默认配置，因此在 2.16.0 中默认禁用了 JNDI 类，以解决这一问题 [29]。此外，Apache 软件基金会内部展现出的协调合作——多个项目在短时间内相继发布补丁——体现了强大的内部通信网络和共享资源所带来的益处；这些因素此前被认为是主动管理开源软件安全风险的关键组成部分。

在评估过程中，扫描器应用于每个代码仓库的 5 个发行版本，重点检查已知存在与 Log4j 相关漏洞的版本，以及已知不存在漏洞的最新发行版本。随后，将结果与官方发行说明文档和 CVE 数据库进行交叉比对，以验证扫描器的发现结果。

我们方法的准确性如图 3 所示。在总计扫描的 140 个代码仓库中，有 128 个得出了准确结果，并正确标记代码仓库为“无漏洞”或“存在漏洞”；而有 12 次扫描出现了错误，包括扫描器错误地将代码仓库标记为“存在漏洞”或“无漏洞”，或为发现的漏洞标记了错误的 CVE。扫描生成的报告通过以下内容进行验证：Log4j 漏洞得到修复的已知发行版本、GitHub 问题列表以及发行说明。生成的报告会向开发者提供相关 CVE、发现的受影响软件包版本，以及检测到软件包引用的文件路径。

最后，扫描器还会根据我们方法中检测到的存在漏洞的 Log4j 版本提供建议。例如，如图4所示，扫描器在扫描 MyBatis 后会提供具体建议。开发者随后可以通过禁用或移除受影响的类来采取措施，降低安全风险，在官方安全补丁发布之前作为临时措施，并防止恶意行为者利用该漏洞。这意味着代码仓库可以迅速得到保护

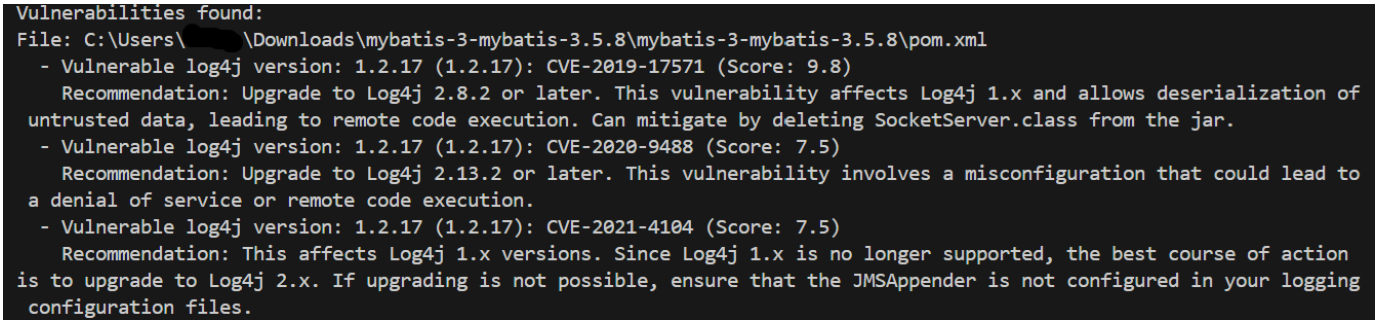

图 4. MyBatis 漏洞的建议安全缓解措施

通过禁用相关功能，使用受影响开源软件的开发者无需等待官方补丁。如果该扫描器在 CVE 发布当天就已推出，更多代码库本可通过更新配置来降低漏洞利用风险，从而迅速得到保护。此类措施包括将 log4j2.formatMsgNoLookups 设置为 true，从而有效禁用攻击点，并移除 JNDILookup 等受影响的类，

在最初应对 Log4Shell 之后，修补 CVE-2021-44228 和 CVE-2021-45046 后，Log4j 版本升级发布的速度有所放缓，因为与 Log4j 2.x 相关的其余 CVE 严重程度较低。本项目分析的大多数项目直到 2022年第二季度之后才升级到 Log4j 2.17.1 或 2.17.2（当前最低的非易受攻击版本[29]）。

D. 扫描器误报结果

在用户验证后发现，12 份扫描报告中有信息不准确的情况，其中没有假阴性结果，共有 7 份产生了假阳性结果，另外 5 份将代码仓库标记为存在错误的 CVE。在 7 份假阳性结果中，全部都包含仍残留在源代码中的 Log4j 1.x 依赖。这些扫描涉及 Mirth Connect、Apache Spark 和 MyBatis。针对 MyBatis 3.5.9 的扫描出现问题，是因为扫描器无法判断在该版本之后 Log4j 已成为可选项，且不再是 MyBatis 的默认日志记录器。此外，Log4j 1.x 也在 3.5.9 中被完全弃用；后续对 3.5.10 及更高版本的扫描均未发现 Log4j 漏洞。

其余 5 条错误报告是由于 CVE 标注不正确所致。其中一个例子是 Apache Pulsar 的 2.9.1 和 2.9.2 版本，它们分别升级到了 Log4j 2.16.0 和 2.17.1。不过，尽管这两个版本都已修补源代码以修复 Log4Shell 漏洞，但它们都依赖于 Netty 4.1.72Final，而该版本仍在使用 Log4j 2.15.0 [49] [52]。这是一次不准确的扫描，因为该扫描器的缺点之一是目前只能扫描源代码中的文件和文件夹，无法直接检查其他外部依赖。我们在验证阶段发现了这个漏洞，因为 Apache Pulsar 的发行说明提到，2.9.1 和 2.9.2 版本升级到了 Netty 4.1.72Final，但我们已通过此前对 Netty 发行版本的扫描和分析，确认 Netty 4.1.72Final 版本存在漏洞。

V. 有效性威胁

我们的研究中，构念效度的一个潜在威胁来自用于确定漏洞是否存在的方法。扫描器的建议基于检测到的易受攻击的 Log4j 版本，并依赖于可能无法涵盖所有现实场景的配置。尽管我们的工具旨在通过将 CVE 与检测到的漏洞进行交叉比对来提供准确结果，但在不同环境中，扫描器的输出与实际系统配置之间可能存在差异。这可能会影响我们研究结果的召回率和精确率，因为这些建议取决于可能并不普遍适用的特定配置。

两名研究人员开展了为期 30 小时的详细分析，对我们的评估进行了严格的人工评估，从而支持了研究的内部效度。这项人工评估确保了扫描器的检测能力能够根据已知漏洞和官方发行说明得到准确评估。然而，人工评估过程中的主观判断可能会影响结果，这构成了一项威胁；尽管如此，研究人员通过独立核验研究结果等措施，尽量减少了偏差。

外部效度方面的一个威胁是，我们的结果能否推广到研究中纳入的 28 个开源代码库之外。尽管这些代码库涵盖了多种规模和复杂度，但它们可能无法充分体现所有可能受到 Log4j 漏洞影响的开源项目的多样性。

VI. 未来工作与结论

本文介绍了一款基于 GitHub Action 的扫描器，旨在检测并缓解 Log4j 漏洞，提供可操作的见解并减少误报。我们对 28 个开源代码库进行了 140 次扫描，评估结果表明，该扫描器识别 CVE-2021-44228 和 CVE-2021-45105 等关键 CVE 的准确率达 91.4%，并能及时提供缓解建议。我们的自动化方案现已在 GitHub Marketplace 上提供 [71]。

我们的工作可以沿着多个方向进一步拓展。将扫描器的适用范围扩展到其他编程语言

和平台，将提升其在更广泛软件项目中的实用性。可以利用自然语言处理（NLP）分析提交消息、问题报告和安全公告中的非结构化数据，自动识别潜在风险，并根据严重程度对其进行优先级排序。

参考文献

[1] Synopsys，《2024年开源软件安全与风险分析报告：开源供应链安全指南》，2024年2月。最后访问于2024年4月18日。[2] J. Song、Q. Li、H. Wang 和 J. Liu，《Pkvic：补充安全漏洞报告中缺失的软件包信息》，*IEEE*可靠与安全计算汇刊，第21卷，第4期，第3785–3800页，2024年。[3] MITRE，《术语表》，2024年。最后访问于2024年7月22日。[4] Apache，《安全性》，2024年。最后访问于2024年4月15日。[5] Google，《了解 apache log4j 漏洞的影响》，2021年。最后访问于2024年4月11日。[6] D. Zhang、Z. Wei 和 Y. Yang，《基于 spring mvc 和 mybatis 的轻量级 mvc 框架研究》，载于*2013*年第六届国际计算智能与设计研讨会，第1卷，第350–353页，IEEE，2013年。[7] Apache，《生命周期终止》，2024年。最后访问于2024年4月22日。[8] Sonatype，《Log4j漏洞利用更新》，2024年。最后访问于2024年4月10日。[9] NextGen Healthcare，《connect》，2024年。最后访问于2024年7月27日。[10] Micromatch，《在 micromatch 和 braces 中发现的漏洞》，2024年。最后访问于2024年7月14日。[11]开放源代码促进会，《开源定义》，2024年。最后访问于2024年7月16日。[12] K.-J. Stol 和 M. Ali Babar，《在产品开发中使用开源软件的挑战：文献综述》，载于第三届自由/*Libre*/开源软件研究与开发新兴趋势国际研讨会论文集，第17–22页，2010年。[13] R. G. Kula、D. M. German、A. Ouni、T. Ishio 和 K. Inoue，《开发者会更新其库依赖项吗？安全公告对库迁移影响的实证研究》，《实证软件工程》，第23卷，第384–417页，2018年。[14] R. Rajala、M. Westerlund 和 K. M¨oller，《开放式创新中的战略灵活性——为开源软件设计商业模式》，《欧洲市场营销杂志》，第46卷，第10期，第1368–1388页，2012年。[15]J. Parsons，《CrowdStrike全球Windows崩溃最新消息——史上最大IT服务中断的后续影响》，2024年7月。最后访问于2024年7月22日。[16] C. Adam、M. F. Bulut、D. Sow、S. Ocepek、C. Bedell 和 L. Ngweta，《漏洞的攻击技术与威胁识别》，*arXiv*预印本 *arXiv:2206.11171*，2022年。[17] MITRE，《CVE-2021-44228》，2021年12月。最后访问于2024年7月17日。[18]Occamsec，“Occamsec/log4j-checker”，2021年。最后访问于2024年4月20日。[19]美国网络安全和基础设施安全局，“Cisagov/log4j-scanner”，2021年。最后访问于2024年4月20日。[20]R. Hiesgen、M. Nawrocki、T. C. Schmidt 和 M. W¨ahlisch，“奔向漏洞：测量 log4j shell 事件”，载于网络流量测量与分析会议，第1–9页，国际信息处理联合会，2022年。[21] Y. Chen、A. E. Santosa、A. Sharma 和 D. Lo，“基于漏洞数据自动识别库”，载于*ACM/IEEE*第*42*届软件工程国际会议论文集：软件工程实践，第90–99页，2020年。[22] Z. Allen 和 C. Tafani-Dereeper，“log4j Log4Shell 漏洞：概述、检测与修复”，2021年。最后访问于2024年7月14日。[23] R. Amankwah、P. K. Kudjo 和 S. Y. Antwi，“软件漏洞检测方法和工具评估：综述”，《国际计算机应用杂志》，第169卷，第8期，第22–27页，2017年。[24]S. Chakraborty、R. Krishna、Y. Ding 和 B. Ray，“基于深度学习的漏洞检测：我们是否已达到目标？”，《*IEEE* 软件工程汇刊》，第48卷，第9期，第3280–3296页，2021年。[25] T. Kinsman、M. Wessel、M. A. Gerosa 和 C. Treude，“软件开发人员如何使用 GitHub Actions 自动化其工作流？”，载于*2021*年*IEEE/ACM*第*18*届软件仓库挖掘国际会议（*MSR*），第420–431页，IEEE，2021年。[26] P. Valenzuela-Toledo 和 A. Bergel，“GitHub Action 工作流的演化”，载于*2022*年*IEEE*软件分析、演化与再工程国际会议（*SANER*），第123–127页，IEEE，2022年。[27] J. C. Camacho Rodriguez、S. St¨aubert 和 M. L¨obe，“使用 Mirth Connect 将 HL7 消息中的临床数据自动导入 OpenClinica 和 TranSMART”，载于《探索健康复杂性：跨学科系统方法》，第317–321页，IOS出版社，2016年。[28] Apache，“Apache Ozone”，2024年。最后访问于2024年7月27日。[29] Apache，“Apache Log4j”，2024年。最后访问于2024年4月15日。[30] MITRE，“CVE-2021-45046”，2021年。最后访问于2024年7月30日。[31] MITRE，“CVE-2019-17571”，2019年。最后访问于2024年7月30日。[32] MITRE，“CVE-2020-9488”，2020年。最后访问于2024年7月30日。[33] MITRE，“CVE-2021-4104”，2021年。最后访问于2024年7月30日。[34] MITRE，“CVE-2022-23302”，2022年。最后访问于2024年7月30日。[35] MITRE，“CVE-2022-23305”，2022年。最后访问于2024年7月30日。[36]P. Mell、K. Scarfone 和 S. Romanosky，“通用漏洞评分系统”，《*IEEE* 安全与隐私》，

第4卷，第6期，第85–89页，2006年。[37] Apache，“安全性”，2024年。最后访问于2024年8月1日。[38] MITRE，“CVE-2022-23307”，2022年。最后访问于2024年7月30日。[39] GitHub Actions，“checkout”，2024年。最后访问于2024年7月22日。[40]MyBatis，“MyBatis-3”，2024年。最后访问于2024年7月22日。[41] MITRE，“CVE-2021-45105”，2021年12月。最后访问于2024年7月22日。[42] R. Gifford、L. Scannell、C. Kormos、L. Smolova、A. Biel、S. Boncu、V. Corral、H. G¨untherf、K. Hanyu、D. Hine，等，“环境评估中的时间悲观主义与空间乐观主义：18国研究”，《环境心理学杂志》，第29卷，第1期，第1–12页，2009年。[43] M. Walkowski、J. Oko 和 S. Sujecki，“基于通用漏洞评分系统的漏洞管理模型”，《应用科学》，第11卷，第18期，第8735页，2021年。[44]P. Mell、K. Scarfone、S. Romanosky，等，“通用漏洞评分系统2.0版完整指南”，载于由*FIRST*事件响应与安全团队论坛出版，第1卷，第23页，2007年。[45] Apache，“Apache Spark”，2024年。最后访问于2024年7月27日。[46] Arduino，“Arduino IDE”，2024年。最后访问于2024年7月27日。[47] Apache，“Apache Hive”，2024年。最后访问于2024年7月27日。[48] Apache，“Apache Wicket”，2024年。最后访问于2024年7月27日。[49] Netty项目社区，“Netty”，2024年。最后访问于2024年7月27日。[50]Elastic，“Elasticsearch”，2024年。最后访问于2024年7月27日。[51] phax，“phoss-smp”，2024年。最后访问于2024年7月27日。[52] Apache，“Apache Pulsar”，2024年。最后访问于2024年7月27日。[53] Apache，“Apache Tapestry”，2024年。最后访问于2024年7月27日。[54]Apache，“Apache NiFi”，2024年。最后访问于2024年7月27日。[55] Apache，“Apache Traffic Control”，2024年。最后访问于2024年7月27日。[56] Apache，“Apache SkyWalking”，2024年。最后访问于2024年7月27日。[57] Apache，“Apache OFBiz”，2024年。最后访问于2024年7月27日。[58] Apache，“Apache JMeter”，2024年。最后访问于2024年7月27日。[59] Apache，“Apache Jena”，2024年。最后访问于2024年7月27日。[60] Apache，“Apache Geode”，2024年。最后访问于2024年7月27日。[61] Apache，“Apache Fortress”，2024年。最后访问于2024年7月27日，2024.[62] Apache，“Apache druid”，2024年。最后访问于2024年7月27日。[63] Apache，“Apache calcite avatica”，2024年。最后访问于2024年7月27日。[64]Apache，“Apache archiva repository”，2024年。最后访问于2024年7月27日。[65] Apache，“Apache tika”，2024年。最后访问于2024年7月27日。[66] Apache，“Apache solr”，2024年。最后访问于2024年7月27日。[67] Apache，“Apache flink”，2024年。最后访问于2024年7月27日。[68] Apache，“Apache eventmesh”，2024年。最后访问于2024年7月27日。[69] T. Winters、T. Manshreck 和 H. Wright，*Google* 的软件工程：长期编程的经验教训。O’Reilly Media，2020年。[70] MITRE，“CVE-2020-9493”，2020年。最后访问于2024年7月31日。[71] Log4jDeepScanAction，“Log4j漏洞扫描器”，2021年。最后访问于2024年8月3日。