

学校代码：10246

学 号：

復旦大學

硕 士 学 位 论 文

（学术学位）

基于多源知识的开源软件漏洞的补丁识别方法

**Finding Patches for Open Source Software Vulnerabilities from  
Multi-Source Knowledge**

院 系： 软件学院

专 业： 软件工程

姓 名：

指 导 教 师：

完 成 日 期：2022 年 2 月 28 日

# 指导小组成员

—

目 录

摘 要	I
Abstract	III
第 1 章 绪论	1
1.1 研究背景	1
1.2 研究问题	2
1.3 本文工作	2
1.3.1 开源软件漏洞补丁的经验研究	3
1.3.2 开源软件漏洞的补丁识别方法	3
1.3.3 实验评估	4
1.4 主要贡献	4
1.5 本文篇章结构	5
第 2 章 背景知识及相关工作	6
2.1 背景知识	6
2.1.1 CVE 及 NVD	6
2.1.2 漏洞公告	8
2.1.3 漏洞补丁	10
2.2 相关工作	11
2.2.1 漏洞信息质量	11
2.2.2 漏洞补丁识别	11
2.2.3 漏洞补丁应用	12
第 3 章 开源软件漏洞补丁的经验研究	14
3.1 研究设计	14
3.1.1 研究问题	14
3.1.2 评估标准	15
3.2 数据准备	15
3.2.1 漏洞数据库选择	15

3.2.2	广度数据集构建 . . . . .	17
3.2.3	深度数据集构建 . . . . .	17
3.3	RQ1: 补丁覆盖率分析 . . . . .	19
3.4	RQ2: 补丁一致性分析 . . . . .	19
3.5	RQ3: 补丁类型分析 . . . . .	20
3.6	RQ4: 补丁映射分析 . . . . .	21
3.7	RQ5: 补丁准确性分析 . . . . .	23
3.8	研究发现 . . . . .	24
<b>第 4 章</b>	<b>开源软件漏洞的补丁识别方法</b>	<b>25</b>
4.1	方法概述 . . . . .	25
4.2	参考链接网络构建 . . . . .	25
4.2.1	公告分析 . . . . .	26
4.2.2	参考链接分析 . . . . .	29
4.2.3	参考链接扩增 . . . . .	31
4.3	补丁选择 . . . . .	33
4.3.1	基于置信度的补丁选择方法 . . . . .	33
4.3.2	基于连通度的补丁选择方法 . . . . .	33
4.4	补丁扩增 . . . . .	34
<b>第 5 章</b>	<b>实验评估</b>	<b>37</b>
5.1	实验问题设计 . . . . .	37
5.2	实验环境准备 . . . . .	38
5.3	RQ6: 准确性评估 . . . . .	38
5.3.1	与基于启发式规则的方法对比 . . . . .	38
5.3.2	与商业漏洞数据库对比 . . . . .	40
5.3.3	漏报和误报分析 . . . . .	41
5.4	RQ7: 削弱性分析 . . . . .	43
5.4.1	去除某一知识源 . . . . .	43
5.4.2	去除网络构建步骤 . . . . .	43
5.4.3	去除补丁选择步骤 . . . . .	44
5.4.4	去除补丁扩增步骤 . . . . .	46
5.5	RQ8: 敏感度分析 . . . . .	46
5.6	RQ9: 通用性分析 . . . . .	48
5.7	RQ10: 实用性分析 . . . . .	49
5.8	实验结论 . . . . .	50

5.9 讨论 . . . . .	50
5.9.1 TRACER 的局限性 . . . . .	51
5.9.2 TRACER 的价值 . . . . .	51
<b>第 6 章 总结与展望</b>	<b>52</b>
6.1 本文总结 . . . . .	52
6.2 展望 . . . . .	53
<b>参考文献</b>	<b>54</b>
<b>致谢</b>	<b>59</b>

## 摘 要

开源软件（Open Source Software, OSS）漏洞管理已成为一个备受关注的研究问题。开源软件漏洞数据库作为各项软件安全任务的基础设施，数据库中漏洞知识的质量已受到越来越多的关注和研究。然而，现有的漏洞数据库中补丁知识的质量和特征尚未被系统地研究；此外，漏洞数据库中的补丁也多是由人工或基于启发式规则的方法半自动化收集。这些方法人工成本高并且为任务定制化，无法应用于所有开源软件漏洞。

针对上述问题，本文先开展了一项针对开源软件漏洞补丁的经验研究，以了解当前商业漏洞数据库中开源软件漏洞补丁的质量和特征。该经验研究涵盖五个方面，包括补丁覆盖度分析、补丁一致性分析、补丁类型分析、补丁映射关系以及补丁准确性分析。研究发现：（1）商业漏洞数据库中开源软件漏洞补丁的质量并不理想。补丁缺失情况较为普遍，商业数据库中漏洞补丁覆盖率仅为 41.0% 左右。对于有多个补丁的漏洞，商业数据库经常会遗漏部分补丁。（2）开源软件漏洞补丁在类型、映射关系方面有一定的特殊性。93.7% 的补丁类型都是 GitHub 的代码提交，且超过 40% 的漏洞与其补丁是一对多的映射关系。

基于经验研究的发现，本文提出了一种名为 TRACER 的基于多源知识的开源软件漏洞的补丁识别方法。该方法可以识别代码提交类型的补丁，并构建一对多的漏洞补丁映射关系。该方法的核心思想是：漏洞补丁会在讨论和解决漏洞的、多种来源的漏洞公告、分析报告等参考链接中被频繁地提及和引用。因此，本文首先设计了一种基于多知识源的漏洞参考链接网络，再从该网络中选出具有最高置信度和连通度的补丁节点作为结果，并基于选定的补丁进行补丁扩增，从而构建一对多的漏洞补丁映射关系。

本文通过五个研究问题，从准确性、通用性、实用性等多个方面对 TRACER 进行了实验评估。实验结果表明：（1）在包含 1,295 个漏洞的实验数据集上，TRACER 可以达到 88.0% 的补丁覆盖率、0.864 的补丁精确率和 0.864 的补丁召回率。（2）与现有的基于启发式规则的方法相比，TRACER 可以将补丁覆盖率提高到 273.8%，将 F1 值提高 116.8%。（3）与商业漏洞数据库相比，TRACER 的召回率高出 18.4%。这表明，TRACER 可用于补充现有漏洞数据库缺失的漏洞补丁数据。（4）在更大范围包含 3,185 和 5,468 个漏洞的数据集上，TRACER 具有较好的通用性。即使在商业漏洞数据库都没有补丁的情况下，TRACER 仍能达到 67.7% 和 51.5% 的补丁覆盖

率。这表明，TRACER 可以极大地补充或增强现有的商业漏洞数据库。在实际工作场景中，TRACER 有助于用户更准确、更快速地识别到补丁。

**关键字：**开源软件，漏洞，补丁

**中图分类号：**TP311

# Abstract

Open source software (OSS) vulnerability management has become an open problem of great concern. OSS vulnerability databases, as the basis of security-related tasks, have arisen a growing concern about their information quality. However, it is unclear how the quality and characteristics of patches in existing vulnerability databases are. Further, existing manual or heuristic-based approaches for patch identification are either too expensive or too specific to be applied to all OSS vulnerabilities.

To address these problems, this thesis first conducts an empirical study to understand the quality and characteristics of patches for OSS vulnerabilities in two commercial vulnerability databases. Our study is designed to cover five dimensions, i.e., the coverage, consistency, type, cardinality and accuracy of patches. The result shows that (1) the quality of vulnerability patches in commercial vulnerability databases is not great. The lack of patches is prevalent, and the coverage of patches for vulnerabilities is only about 41.0%. Patches in commercial vulnerability databases are of high accuracy, but for vulnerabilities with multiple patches, patches are often not complete. (2) In terms of patch types and mapping relationships, 93.7% of patches are in the form of GitHub commits, and more than 40% of vulnerabilities have a one-to-many mapping relationship with patches.

Then, inspired by our study, this thesis proposes an automated approach, named TRACER, to identify patches for an OSS vulnerability from multi-source knowledge. TRACER is designed to identify patch commits and build a one-to-many mapping between vulnerabilities and patches. Our key idea is that patch commits will be frequently referenced during the reporting, discussion and resolution of an OSS vulnerability. Therefore, we first design a reference network based on multi-source knowledge, and then select patch nodes with the highest confidence and connectivity from the network. Based on selected patches, TRACER expands the patch list.

With five research questions, TRACER is evaluated in terms of accuracy, generality, usefulness, etc. The extensive evaluation has indicated that (1) on the dataset of 1,295 CVEs, TRACER can achieve the coverage, precision, and recall in 88.0%, 0.864, and 0.864 respectively. (2) Compared with existing heuristic-based approaches, TRACER improves the patch coverage by 273.8%, and the F1 value by 116.8%. (3) Compared



with commercial vulnerability databases, TRACER improves the recall by 18.4%. This suggests that TRACER can be used to supplement the patch missing from commercial vulnerability databases. (4) TRACER is general to larger datasets containing 3,185 and 5,468 OSS vulnerabilities. TRACER can still achieve the patch coverage of 67.7% and 51.5%, while commercial vulnerability databases provide no patch for these vulnerabilities. This suggests that TRACER can greatly enhance existing commercial vulnerability databases. In practice, TRACER can assist users in identifying patches more accurately and quickly.

**Keywords:** Open Source Software, Vulnerabilities, Patches

**CLC code:** TP311

# 第 1 章 绪论

本章将阐述本文的研究背景、研究问题、主要工作、主要贡献以及本文的篇章结构。

## 1.1 研究背景

开源软件（Open Source Software, OSS）为开源及闭源应用程序的快速开发提供了基础。得益于开源社区的蓬勃发展，在软件开发过程中，开发人员经常会使用开源软件中已实现的功能，节省开发时间，加快开发速度<sup>[1]</sup>。然而，伴随着开发效率的提高，开源软件中的安全漏洞也会被引入软件系统<sup>[2-3]</sup>。据 Synopsys 公司发布的《开源安全和风险分析报告》<sup>①</sup>显示，在该公司分析的 1,500 个应用程序中，98% 的应用程序都使用了开源软件。报告还指出，高达 84% 的应用程序包含至少一个已知的开源软件漏洞，该数据对比于前一年（2019 年）增加了 9%。此外，据 Snyk 公司发布的报告<sup>②</sup>显示，近些年所披露的开源软件漏洞越来越多，过去两年间几乎翻了一倍。

针对以上问题，大量工作都在研究如何降低开源软件漏洞带来的安全风险，包括通过学习漏洞特征来检测开源软件中的漏洞<sup>[4-7]</sup>、通过匹配漏洞及补丁签名来检测开源软件漏洞<sup>[8-12]</sup>以及进行软件成分分析以确定应用程序中的开源软件漏洞是否在执行路径上<sup>[1,13-15]</sup>。在这些工作中，准确且完整的漏洞知识十分重要。例如，漏洞描述、受漏洞影响的软件、版本以及补丁等知识都是这些工作得以开展的基础。目前，已有多方人员致力于构建安全漏洞数据库。在安全社区中，由美国政府资助的 CVE List<sup>③</sup>（Common Vulnerabilities & Exposures）、NVD<sup>④</sup>（National Vulnerability Database）和由中国政府资助的 CNVD<sup>⑤</sup>（国家信息安全漏洞共享平台）、CNNVD<sup>⑥</sup>（国家信息安全漏洞数据库）是最具影响力的漏洞数据

① <https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/rep-ossra-2021.pdf>

② [https://snyk.io/wp-content/uploads/sooss\\_report\\_v2.pdf](https://snyk.io/wp-content/uploads/sooss_report_v2.pdf)

③ <https://cve.mitre.org/cve/>

④ <https://nvd.nist.gov>

⑤ <https://www.cnvd.org.cn/>

⑥ <http://www.cnnvd.org.cn/>

库。在工业界中, BlackDuck<sup>①</sup>、WhiteSource<sup>②</sup>、Veracode<sup>③</sup>和 Snyk<sup>④</sup>等公司较为关注开源软件中的安全漏洞, 并已经构建各自的商业漏洞数据库, 作为安全服务的基础。在学术界中, 也有很多工作致力于构建漏洞数据集<sup>[16–20]</sup>, 但这些数据集大多是针对特定程序语言的生态系统或针对特定的软件项目而设计。

## 1.2 研究问题

随着构建的漏洞数据库越来越多, 数据库中积累的漏洞数据也越来越多, 研究人员开始关注数据库中漏洞知识的质量。Dong 等人<sup>[21]</sup>发现了漏洞数据库中受漏洞影响的软件版本信息不准确的情况, Chaparro 等人<sup>[22]</sup>和 Mu 等人<sup>[23]</sup>发现了漏洞描述中普遍缺失漏洞重现的步骤描述。这种信息不完整或不准确的情况使得安全工作人员难以及时地识别、重现和修复应用程序中的安全漏洞。

漏洞补丁作为刻画漏洞特征的重要知识, 可应用于多种安全相关的任务, 包括补丁生成和热部署<sup>[24–26]</sup>、补丁存在测试<sup>[27–29]</sup>、软件成分分析<sup>[1,14–15]</sup>、漏洞检测<sup>[4–5,8–9,11–12]</sup>等。如果漏洞数据库中的补丁知识缺失或不准确, 那么这些安全任务的准确性将会受到严重影响。然而, 漏洞数据库中的补丁知识尚未被系统地研究和评估, 目前尚不清楚现有漏洞数据库中补丁的质量情况。

此外, 现有的漏洞补丁识别方法主要有三种: (1) 人工手动查找漏洞补丁<sup>[11,14–15,26,28–32]</sup>。(2) 通过启发式规则识别漏洞补丁, 比如在 NVD 参考链接中查找代码提交<sup>[4,25]</sup>或是在代码仓的提交历史中搜索漏洞标识符 (CVE ID)<sup>[1,33]</sup>。(3) 在特定项目的安全公告中搜索漏洞补丁<sup>[8–9,24]</sup>。以上方法的人工成本较高, 且针对特定的程序语言或项目设计, 无法广泛应用于所有开源软件漏洞。

综上, 目前的问题是, 漏洞数据库中补丁的特征及质量尚未被系统地研究和评估, 并且已有的漏洞补丁采集方法通用性较差且人工成本过高。

## 1.3 本文工作

为解决上述研究问题, 本文先开展了一项针对开源软件漏洞补丁的经验研究, 以了解当前商业漏洞数据库中开源软件漏洞补丁的质量和特征。然后, 基于经验研究的发现, 本文提出了一种名为 TRACER 的基于多源知识的开源软件漏洞的补丁识别方法。该方法通过构建漏洞的多源参考链接网络来识别补丁。本文还进行了大量实验, 从准确性、通用性、实用性等多个方面对 TRACER 进行了评估。

① <https://www.synopsys.com/content/dam/synopsys/sig-assets/datasheets/bdknowledgebase-ds-ul.pdf>

② <https://www.whitesourcesoftware.com/vulnerability-database/>

③ <https://sca.veracode.com/vulnerability-database/search>

④ <https://snyk.io/vuln>

### 1.3.1 开源软件漏洞补丁的经验研究

为了了解当前商业漏洞数据库中开源软件漏洞补丁的质量和特征,本文挑选了两个认可度较高的商业漏洞数据库作为研究对象。该经验研究涵盖五个方面,包括补丁覆盖度分析、补丁一致性分析、补丁类型分析、补丁映射关系以及补丁准确性分析。

本文首先构建了一个广度数据集,该数据集包含 10,070 个开源软件漏洞。在此数据集上,本文分析两个商业漏洞数据库中开源软件漏洞补丁的覆盖率和一致性。结果表明,在漏洞补丁覆盖率方面,10,070个漏洞中只有4,602 (5.7%)的漏洞在商业漏洞数据库中提供了补丁;在漏洞补丁一致性方面,只有19.7%的漏洞在两个商业漏洞数据库中有一致的补丁。

基于广度数据集中含补丁的漏洞数据,本文还通过人工构建了一个深度数据集。该数据集包含 1,295 个开源软件漏洞,且这些漏洞都有补丁。在此数据集上,本文分析了开源软件漏洞补丁的类型和漏洞补丁的映射关系,并评估了商业数据库中漏洞补丁的准确性。结果表明,在漏洞补丁类型方面,1,265 (97.7%)漏洞的补丁类型都是 GitHub 或 SVN 的代码提交;在漏洞补丁映射关系方面,533 (41.1%)的漏洞与其补丁有一对多的映射关系;在漏洞补丁准确性方面,两个商业数据库的补丁精确率都高于90%,但对于一对多映射类型的漏洞,两个商业数据库中补丁的召回率都仅为50%左右。

这些结果表明,现有的商业漏洞数据库中缺失了许多漏洞的补丁,尤其是对于有多个补丁的漏洞,补丁缺失情况更为严重。这种信息不完整或不准确的情况,使得安全工作人员难以及时地识别、重现和修复开源软件中的漏洞。同时,这也反映出自动化的补丁识别方法的需求,自动化的方法可以帮助工作人员识别并补全缺失的补丁。

### 1.3.2 开源软件漏洞的补丁识别方法

基于经验研究的发现,本文提出了一种名为 TRACER 的基于多源知识的开源软件漏洞的补丁识别方法。该方法从多个知识源(即 NVD<sup>①</sup>、Debian<sup>②</sup>、Red Hat<sup>③</sup>以及 GitHub<sup>④</sup>)构建漏洞的参考链接网络并识别补丁(参考链接,即 URL 网址)。该方法的核心思想是:漏洞补丁会在讨论和解决漏洞的、多种来源的漏洞公告、分析报告等参考链接中被频繁地提及和引用。因此,本文首先设计了一种基于多知识源的漏洞参考链接网络,然后再从该网络中选出具有最高置信度

① <https://nvd.nist.gov>

② <https://security-tracker.debian.org/tracker/>

③ <https://bugzilla.redhat.com/>

④ <https://github.com/>

和连通度的补丁节点作为结果，并基于选定的补丁进行补丁扩增，从而构建一对多的漏洞补丁映射关系。

TRACER 以漏洞的 CVE ID 作为输入，经过三个步骤，输出该漏洞的补丁。步骤一：多源参考链接网络构建，该步骤的目的是将该漏洞在被报告、讨论和解决阶段的参考链接进行建模。TRACER 从多个漏洞知识源（即 NVD、Debian、Red Hat 和 GitHub）中提取引用的参考链接信息并构建一个参考链接网络。步骤二：补丁选择，TRACER 从构建的参考链接网络中选择中具有高连通性和高置信度的补丁节点作为该漏洞的补丁。步骤三：补丁扩增，基于前一步骤选定的补丁，TRACER 通过搜索同一代码库所有分支中的相关提交来扩展补丁集，构建一对多的漏洞补丁映射关系。最终，返回所有选中及扩展的补丁。

### 1.3.3 实验评估

本文进行了大量实验，通过五个研究问题，从准确性、通用性、实用性等多个方面对 TRACER 进行了评估。为了评估 TRACER 的准确性，本文将 TRACER 与三种基于启发式规则的方法以及两个商业漏洞数据库进行了比较。结果表明，在包含 1,295 个漏洞的深度数据集上，（1）TRACER 可以达到 88.0% 的补丁覆盖率、0.864 的补丁精确率和 0.864 的补丁召回率。（2）与现有的基于启发式规则的方法相比，TRACER 将补丁覆盖率提高 58.6% 到 273.8%；同时，在补丁的准确性上，TRACER 的 F1 数值也比基于启发式规则的方法高 116.8%。（3）与现有的商业漏洞数据库  $DB_A$  和  $DB_B$  相比，TRACER 的补丁召回率高出 18.4%；但仍有 155（12.0%）的漏洞 TRACER 未能找到补丁，补丁精确率也低了 6.4%。

此外，为了评估 TRACER 的通用性，本文还另外构建了两个更大的、包含 3,185 和 5,468 个漏洞的数据集，并在这两个数据集上运行 TRACER。结果表明，TRACER 在两个数据集上分别可以找到 67.7% 和 51.5% 个漏洞的补丁，补丁精确率分别为 0.823 和 0.888，补丁召回率分别为 0.845 和 0.899。这表明 TRACER 在识别补丁方面具有较好的通用性。

此外，为了评估 TRACER 在实际工作种的实用性，本文还邀请了 10 名实验人员进行了用户研究。评估结果表明，在实际使用中，TRACER 有助于用户更准确、更快速地识别到补丁。

## 1.4 主要贡献

本文主要有以下贡献：

- （1）本文开展了一项针对开源软件漏洞补丁的经验研究，以了解当前商业漏洞数据库中开源软件漏洞补丁的质量和特征。该经验研究涵盖五个方面，包括补丁覆盖度分析、补丁一致性分析、补丁类型分析、补丁映射关系以及

补丁准确性分析。

- (2) 本文提出了一种名为 TRACER 的基于多源知识的开源软件漏洞的补丁识别方法，该方法可服务于安全社区、工业界和学术界的多种安全任务。
- (3) 本文进行了针对 TRACER 准确性、通用性、实用性等多个方面的实验评估。

## 1.5 本文篇章结构

本文共包含六个章节，结构如下：

第一章绪论，介绍了本文的研究背景及研究问题，简述了本文的主要工作（包括经验研究、方法设计和实验评估）、主要贡献以及篇章结构。

第二章背景知识及相关工作，介绍了本文所涉及的背景知识，包括通用漏洞披露 CVE、漏洞公告、漏洞补丁等信息，为后文经验研究、方法设计等内容的展开做铺垫。本章还介绍了与本文研究主题的相关工作，包括漏洞信息质量、漏洞补丁识别以及漏洞补丁应用。

第三章开源软件漏洞补丁的经验研究，介绍了本文为了解当前商业漏洞数据库中开源软件漏洞补丁的质量和特征，针对开源软件漏洞补丁所开展的经验研究，涵盖补丁覆盖度分析、补丁一致性分析、补丁类型分析、补丁映射关系以及补丁准确性分析五个方面。

第四章开源软件漏洞的补丁识别方法，介绍了本文提出的一种名为 TRACER 的基于多源知识的开源软件漏洞的补丁识别方法。该方法包括多源参考链接网络构建、补丁选择以及补丁扩增三个步骤。

第五章实验评估，介绍了本文针对 TRACER 的准确性、通用性、实用性等五个方面所进行的实验评估。

第六章总结与展望，对本文的工作内容及研究成果进行总结，讨论本文研究工作中存在的不足，并展望了未来可以进行的工作。

## 第2章 背景知识及相关工作

本章将介绍开源软件漏洞相关的背景知识以及相关的研究工作。

### 2.1 背景知识

本文介绍的开源软件漏洞相关的背景知识，包括通用漏洞披露（CVE）、国家漏洞数据库（NVD）、漏洞公告（Advisory）和漏洞补丁（Vulnerability Patch）。

#### 2.1.1 CVE 及 NVD

通用漏洞披露 (Common Vulnerabilities and Exposures, CVE)<sup>[34]</sup>，是一个与网络安全有关的漏洞字典，收集各种信息安全漏洞并分配唯一编号以便公众查阅及引用<sup>①</sup>。在实际使用中，当人们提及某个 CVE 时，其实是指某个被分配了 CVE ID 的安全漏洞<sup>②</sup>。

如图2-1所示，每一个 CVE 条目（CVE Entry）都有唯一通用标识符（即 CVE ID）、一段漏洞描述（即 Description）以及至少一个参考链接（即 Reference）。引用的参考链接多为外部网站，包含与该漏洞相关的更详细的描述信息。图2-1为漏洞 CVE-2021-44228<sup>③</sup>，该漏洞的描述信息为“Apache Log4j2 2.0-beta9 through 2.12.1 and 2.13.0 through 2.15.0 JNDI features used in configuration, ..... Note that this vulnerability is specific to log4j-core and does not affect log4net, log4cxx, or other Apache Logging Services projects.”，该 CVE 条目引用了多个参考链接，如：“<https://www.kb.cert.org/vuls/id/930724>”、“<https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-apache-log4j-qRuKNEbd>”等。

CVE 被设计作为漏洞字典用于收录并编号漏洞，这也导致 CVE 中漏洞信息过于精简，仅有一段漏洞描述和参考链接信息。基于 CVE 平台收录的漏洞条目信息，美国国家漏洞数据库（NVD）<sup>④</sup>、中国国家信息安全漏洞库（CNNVD）<sup>⑤</sup>等漏洞数据库被构建。这些数据库与 CVE 平台数据完全同步，并为每个漏洞条目（CVE Entry）提供更丰富的信息，如：影响的软件名及版本、修复信息、严重性

<sup>①</sup> <https://cve.mitre.org/cve/>

<sup>②</sup> <https://www.redhat.com/en/topics/security/what-is-cve>

<sup>③</sup> <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-44228>

<sup>④</sup> <https://nvd.nist.gov/>

<sup>⑤</sup> <http://www.cnnvd.org.cn/>

CVE-ID	
CVE-2021-44228	<a href="#">Learn more at National Vulnerability Database (NVD)</a> • CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information
Description	
Apache Log4j2 2.0-beta9 through 2.12.1 and 2.13.0 through 2.15.0 JNDI features used in configuration, log messages, and parameters do not protect against attacker controlled LDAP and other JNDI related endpoints. An attacker who can control log messages or log message parameters can execute arbitrary code loaded from LDAP servers when message lookup substitution is enabled. From log4j 2.15.0, this behavior has been disabled by default. From version 2.16.0, this functionality has been completely removed. Note that this vulnerability is specific to log4j-core and does not affect log4net, log4cxx, or other Apache Logging Services projects.	
References	
<b>Note:</b> <a href="#">References</a> are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.  <ul style="list-style-type: none"><li>CERT-VN:VU#930724</li><li><a href="https://www.kb.cert.org/vuls/id/930724">URL:https://www.kb.cert.org/vuls/id/930724</a></li><li>CISCO:20211210 A Vulnerability in Apache Log4j Library Affecting Cisco Products: December 2021</li><li><a href="https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-apache-log4j-qRuKNEbd">URL:https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-apache-log4j-qRuKNEbd</a></li><li>CISCO:20211210 Vulnerabilities in Apache Log4j Library Affecting Cisco Products: December 2021</li><li><a href="https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-apache-log4j-qRuKNEbd">URL:https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-apache-log4j-qRuKNEbd</a></li><li>CISCO:20211210 Vulnerability in Apache Log4j Library Affecting Cisco Products: December 2021</li><li><a href="https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-apache-log4j-qRuKNEbd">URL:https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-apache-log4j-qRuKNEbd</a></li></ul>	

图 2-1 CVE 平台中漏洞 CVE-2021-44228

CVE-2021-44228 Detail

Current Description


Apache Log4j2 2.0-beta9 through 2.12.1 and 2.13.0 through 2.15.0 JNDI features used in configuration, log messages, and parameters do not protect against attacker controlled LDAP and other JNDI related endpoints. An attacker who can control log messages or log message parameters can execute arbitrary code loaded from LDAP servers when message lookup substitution is enabled. From log4j 2.15.0, this behavior has been disabled by default. From version 2.16.0, this functionality has been completely removed. Note that this vulnerability is specific to log4j-core and does not affect log4net, log4cxx, or other Apache Logging Services projects.

[View Analysis Description](#)

Severity

CVSS Version 3.xCVSS Version 2.0

CVSS 3.x Severity and Metrics:

 NIST: NVD

Base Score: 10.0 CRITICAL

Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

NVD Analysts use publicly available information to associate vector strings and CVSS scores. We also display any CVSS information provided within the CVE List from the CNA.

Note: NVD Analysts have published a CVSS score for this CVE based on publicly available information at the time of analysis. The CNA has not provided a score within the CVE List.

QUICK INFO

**CVE Dictionary Entry:**  
CVE-2021-44228  
**NVD Published Date:**  
12/10/2021  
**NVD Last Modified:**  
12/20/2021  
**Source:**  
Apache Software Foundation

Weakness Enumeration

CWE-ID	CWE Name	Source
CWE-502	Deserialization of Untrusted Data	 NIST  Apache Software Foundation
CWE-400	Uncontrolled Resource Consumption	 Apache Software Foundation
CWE-20	Improper Input Validation	 Apache Software Foundation

Known Affected Software Configurations [Switch to CPE 2.2](#)







<b>Configuration 1</b> <a href="#">(hide)</a>		
 <b>cpe:2.3:a:apache:log4j2.0:-:*:*:*:*</b>	<a href="#">Show Matching CPE(s)</a>	
 <b>cpe:2.3:a:apache:log4j2.0:beta9:*:*:*:*</b>	<a href="#">Show Matching CPE(s)</a>	
 <b>cpe:2.3:a:apache:log4j2.0:rc1:*:*:*:*</b>	<a href="#">Show Matching CPE(s)</a>	
 <b>cpe:2.3:a:apache:log4j2.0:rc2:*:*:*:*</b>	<a href="#">Show Matching CPE(s)</a>	
 <b>cpe:2.3:a:apache:log4j:*:*:*:*:*</b>	From (including) 2.0.1	Up to (excluding) 2.12.2
 <b>cpe:2.3:a:apache:log4j:*:*:*:*:*</b>	From (including) 2.13.0	Up to (excluding) 2.15.0

图 2-2 NVD 平台中漏洞 CVE-2021-44228



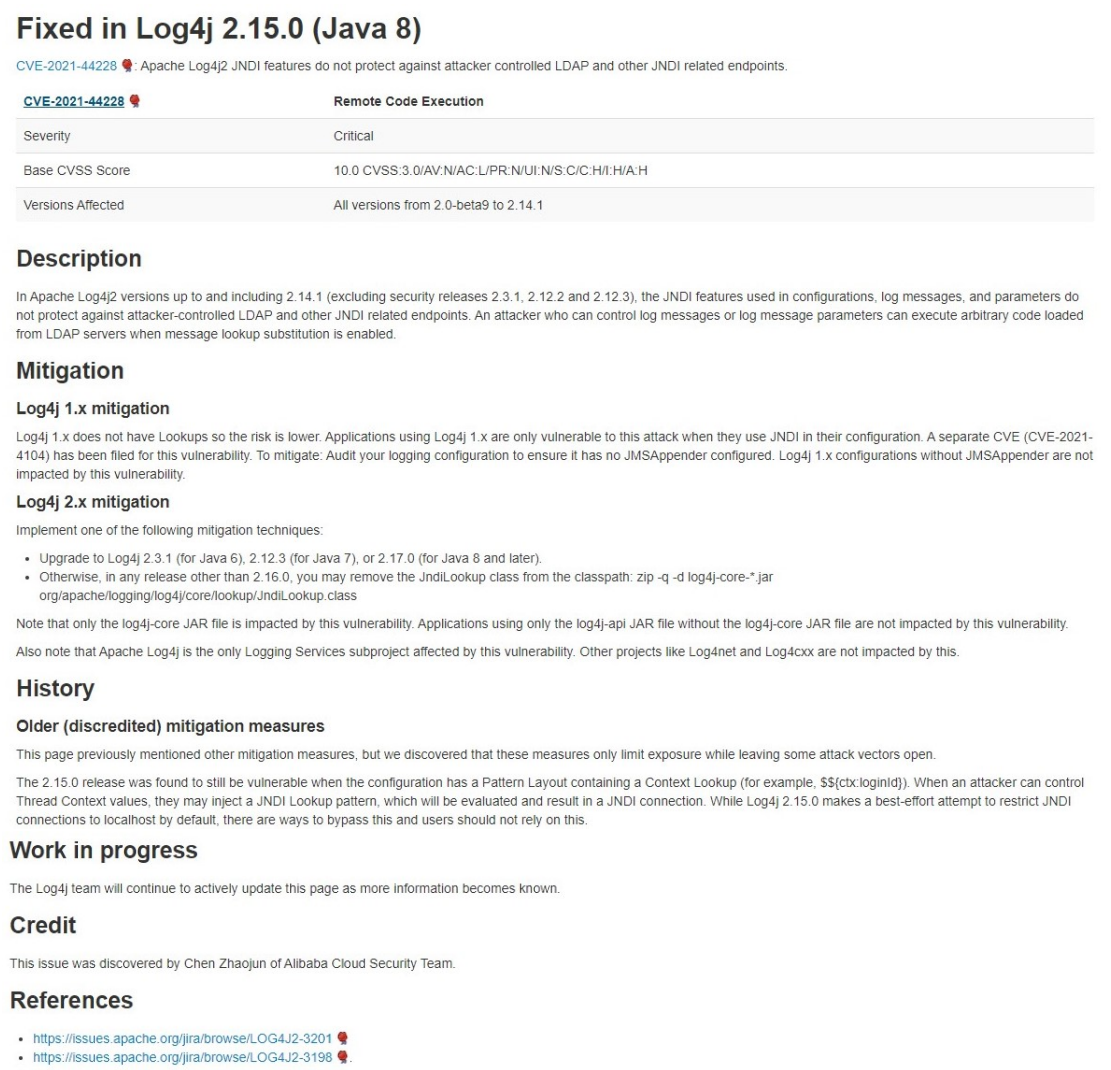


图 2-3 Apache Log4j 发布的漏洞 CVE-2021-44228 公告

评分、影响评级等。图2-2所示为 NVD 平台中漏洞 CVE-2021-44228<sup>①</sup>。

### 2.1.2 漏洞公告

漏洞公告（Advisory），也被称为漏洞通告，一般是由受漏洞影响的软件的厂商（Vendor）对外发布的安全漏洞警报，通常包含：漏洞触发描述、漏洞影响结果、漏洞软件名、软件版本等描述信息，有时也会包含漏洞发现者、漏洞问题报告（Issue Report）、漏洞补丁等知识。

图2-3为厂商 Apache 发布的关于安全漏洞 CVE-2021-44228 的公告<sup>②</sup>，包含修复方式—“Log4j 2.x mitigation ..... Upgrade to Log4j 2.3.1 (for Java 6), 2.12.3 (for Java 7), or 2.17.0 (for Java 8 and later)”、漏洞发现者—“Credit: This issue was discovered by Chen Zhaojun of Alibaba Cloud Security Team.”、参考链接—

① <https://nvd.nist.gov/vuln/detail/CVE-2021-44228>  
② <https://logging.apache.org/log4j/2.x/security.html>



图 2-4 Debian 平台中漏洞 CVE-2021-44228 的公告

“<https://issues.apache.org/jira/browse/LOG4J2-3198>”等信息。

以上介绍的由厂商发布的漏洞公告，也被称为：厂商公告（Vendor Advisory）。由于某一厂商只会发布、维护与该厂商相关的软件漏洞通告，而开源社区的开发人员难以一一关注所有厂商的公告信息，因此，出现了许多致力于收集并维护所有漏洞公告信息的第三方平台。这些由第三方平台收集并维护的漏洞公告也被称为：第三方公告（Third-Party Advisory）。目前，认可度较高、影响力较大的第三方漏洞公告平台有 Red Hat<sup>①</sup>、Debian<sup>②</sup>、Gentoo<sup>③</sup>等。

图2-4为 Debian 平台中漏洞 CVE-2021-44228 的公告<sup>④</sup>，其中还包括了该漏洞

① <https://access.redhat.com/errata/>

② <https://www.debian.org/security/>

③ <https://security.gentoo.org/>

④ <https://security-tracker.debian.org/tracker/CVE-2021-44228>

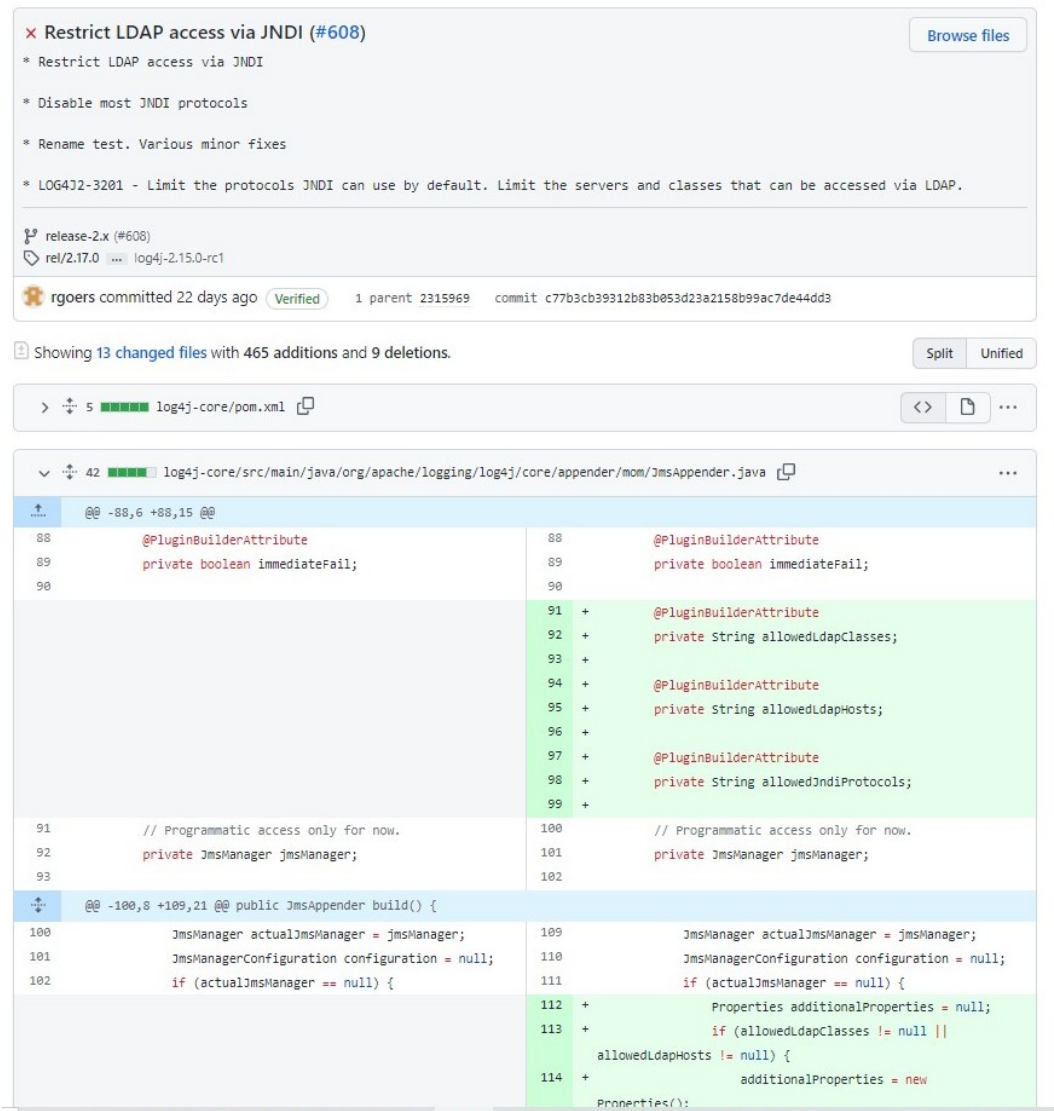


图 2-5 漏洞 CVE-2021-44228 的 Github Commit 补丁

的补丁 `c77b3c@logging-log4j2`<sup>①</sup>（即图中“Note”引用的参考链接），这在 NVD 和 Vendor Advisory 中都未出现。

### 2.1.3 漏洞补丁

补丁（Patch），也称：补丁程序，是指对计算机程序进行的一组更改，旨在更新其功能或修复其缺陷。漏洞补丁（Vulnerability Patch）则指为修复程序中的安全漏洞所开发的补丁，通常是 Git 和 SVN 中的代码提交（Commit），或是文本文件（.patch 文件）形式。对于代码提交形式的补丁，后文中将简称为补丁提交。

例如，上图2-4中的参考链接“<https://github.com/apache/logging-log4j2/commit/c77b3cb39312b83b053d23a2158b99ac7de44dd3>”，即为漏洞 CVE-2021-44228 的补丁提交，该提交的内容见图2-5。

① <https://github.com/apache/logging-log4j2/commit/c77b3cb39312b83b053d23a2158b99ac7de44dd3>

## 2.2 相关工作

本文将从漏洞信息质量、漏洞补丁识别和漏洞补丁应用三个方面介绍相关的研究工作。

### 2.2.1 漏洞信息质量

随着社区、工业界及学术界构建的漏洞数据库越来越多，数据库中积累的漏洞数据也越来越多，研究人员开始关注数据库中漏洞信息的质量。

Nguyen 和 Massacci<sup>[35]</sup>最早揭示了 NVD 数据库中软件版本信息的不准确情况。为了提高这项信息的可靠性，Nguyen<sup>[36]</sup>和 Dashevskyi 等人<sup>[37]</sup>提出方法，用以确定某一软件的旧版本是否会受到新披露漏洞的影响。他们认为：如果软件的旧版本包含修复漏洞所更改的代码行，则该版本被视为受新披露的漏洞影响。不同于以上工作的方法，Dong 等人<sup>[21]</sup>从漏洞的描述信息中识别受漏洞影响的软件名称和版本，并与漏洞报告所提供的软件名称和版本信息进行对比，以此发现不一致、不准确的软件版本信息。他们发现，NVD 等多个漏洞数据库中会遗漏真正受漏洞影响的版本，也会错误地包含了不受漏洞影响的版本。

除了受漏洞影响的软件版本信息，Mu 等人<sup>[23]</sup>揭示了漏洞报告中，普遍缺失用于重现漏洞的关键步骤。Chaparro 等人<sup>[22]</sup>提出方法用于检测漏洞描述中是否缺少用于重现漏洞的关键步骤及预期结果信息。以上工作主要侧重于漏洞的软件版本及漏洞重现的信息，而本文则重点关注漏洞数据库中的补丁知识，并尝试自动化地从多个知识源的漏洞公告信息中定位漏洞补丁。

Tan 等人完成了一项与本文的研究问题较为相似的工作<sup>[38]</sup>。他们使用深度学习排名算法对代码仓库中的代码提交进行排名，把排在首位的代码提交作为漏洞的补丁提交。他们的工作包含两个假设：（1）受漏洞影响的软件的代码仓库已知，（2）漏洞与其补丁提交在数量上是一对一的映射关系。然而，事实上，受漏洞影响的软件的代码仓库并不全部已知，而需人工识别。此外，漏洞与其补丁提交在数量上存在一对多的关系（见章节3.6）。所以，该方法对于开源软件漏洞不具有通用性。

### 2.2.2 漏洞补丁识别

随着披露的开源软件漏洞越来越多、漏洞影响越来越大，为了能够及时获取补丁知识，许多自动化的补丁识别方法被提出。

周鹏等人<sup>[39]</sup>提出了一种针对 Linux 内核的安全漏洞修复补丁自动识别方法，该方法通过持续收集并标注安全漏洞补丁数据，以及定义并抽取原始补丁文件的特征，来训练一个可识别安全漏洞修复补丁的分类器。邹雅毅等人<sup>[40]</sup>针对 Linux 内核、Ffmpeg、Wireshark 三个项目设计了一种 DIFF 文件形式的补丁采集方法。

该方法通过定义的 DIFF 特征来识别补丁链接,并基于与补丁链接出现在同一页面的 CVE ID 来完成漏洞及其修复补丁映射关系的确认。

李瑞科等人<sup>[41]</sup>构建了一个包括近 20 年漏洞数据的安全漏洞数据集,通过程序自动化和人工采集结合的方法采集了多个国内外知名漏洞平台 1999-2018 年间的安全漏洞数据,并对采集的漏洞数据进行切片和格式化操作。Li 和 Vern<sup>[42]</sup>基于 NVD 平台的漏洞数据,通过收集带有 Git 代码提交链接的漏洞,实现漏洞补丁数据集的构建。同样, Liu 等人<sup>[43]</sup>也是基于 CVE 和 NVD 漏洞报告中参考链接的数据,解析并识别 GitHub 代码提交的链接作为漏洞补丁。Ponta 等人<sup>[16]</sup>通过人工手动收集的方式,构建一个针对 Java 语言,包含 205 个开源项目、624 个漏洞、1282 个补丁的数据集。

Fan 等人<sup>[17]</sup>针对 Google Chrome、Linux 和 Google Android 项目构建了一个 C 及 C++ 语言的漏洞数据集。通过分析 CVE Details 漏洞报告中带有 Git 仓库的参考链接,确定该漏洞所影响软件的仓库,并以 CVE ID 及 Bug ID 作为关键词在该仓库的代码提交历史中搜索修复该漏洞的补丁。类似地, Jimenez 等人<sup>[18]</sup>提出了一种自动补丁识别框架-VulData7,并构建了一个针对 Linux Kernel、WireShark、OpenSSL 和 SystemD 四个项目的漏洞补丁数据集。该框架也是以 CVE ID 及 Bug ID 作为关键词在该仓库的代码提交历史中搜索修复该漏洞的补丁。

上述工作提出的补丁识别方法,多是通过基于启发式规则从漏洞公告的参考链接或项目的历史代码提交中识别补丁。这些方法多是针对特定的程序语言或项目而设计,较为局限,无法适用于所有安全漏洞。

### 2.2.3 漏洞补丁应用

漏洞补丁可被用于多种软件安全性任务。例如,补丁存在性检测<sup>[44]</sup>、基于漏洞补丁生成漏洞攻击程序<sup>[33,45]</sup>、基于漏洞补丁进行漏洞影响分析<sup>[1,13-15,46-47]</sup>以及基于漏洞补丁进行漏洞检测<sup>[4-7]</sup>。

文琪等人<sup>[44]</sup>提出一个名为 PatchChecker 的基于关键路径和语义的漏洞补丁存在性检测方法。该方法通过识别漏洞的修复路径及其程序修复前后的语义特征,生漏洞补丁的签名。基于补丁签名,在目标程序中识别对应路径并进行比较,以判断漏洞补丁是否被应用。Li 等人<sup>[4]</sup>提出了一种基于代码相似度分析的自动化漏洞检测方法。给定漏洞,该方法先抽取该漏洞补丁中的代码特征,再利用代码相似度算法检测项目中相似的代码,实现漏洞检测。之后, Li 等人<sup>[5]</sup>又提出了一种基于深度学习系统的漏洞检测方法。Kim 等人<sup>[9]</sup>提出了一种可扩展的克隆代码的安全漏洞检测方法。该方法利用函数级代码和基于长度过滤技术,能够高效准确地检测大型软件程序中的安全漏洞。

Xu 等人<sup>[10]</sup>提出了一种名为 BinXray 基于补丁的漏洞匹配方法,以准确有效

地识别程序中的“1-day”漏洞。该方法首先通过比较漏洞修复前后的程序，基于程序块轨迹来提取补丁的签名。检测时，基于补丁语义加快签名搜索，并通过计算程序轨迹的相似性识别目标程序是否已修补。Xiao 等人<sup>[11]</sup>提出了一种检测重现漏洞 (Recurring Vulnerability) 的方法，MVP。该方法基于使用程序切片从漏洞函数的句法及语义中提取漏洞和补丁签名；在检测阶段，如果目标函数与漏洞签名匹配成功，但与补丁签名匹配不成功，则被识别为潜在漏洞。李赞等人<sup>[48]</sup>提出了一种利用补丁来提高相似性检测准确性的漏洞发现方法。该方法基于漏洞补丁，通过程序切片技术去除漏洞函数中与漏洞无关的代码，并利用程序切片生成漏洞特征来检测漏洞。李元成等人<sup>[49]</sup>提出一种基于深度聚类算法的软件源代码漏洞检测方法。该方法先通过构造软件代码属性图，遍历代码中 API 序列，并将其向量化；再以关键代码为中心进行聚类，根据聚类结果判断异常函数并匹配漏洞库，从而实现漏洞代码的检测。

本文提出的方法可以为以上补丁存在性检测、漏洞影响分析、漏洞挖掘等工作提供漏洞补丁知识。

## 第3章 开源软件漏洞补丁的经验研究

本章将介绍本文所开展的针对开源软件漏洞补丁的经验研究，以了解当前商业漏洞数据库中开源软件漏洞补丁的质量和特征。主要包括研究设计、数据准备以及结果分析三个部分。

### 3.1 研究设计

研究设计部分包含研究问题和评估标准的设计。

#### 3.1.1 研究问题

为了探究已有漏洞数据库中开源软件漏洞补丁的质量和特征，本文设计以下研究问题：

- **RQ1 补丁覆盖率分析：**当前商业漏洞数据库中，漏洞补丁的覆盖度如何？即，有多少漏洞含有补丁知识？（见章节3.3）
- **RQ2 补丁一致性分析：**不同漏洞库间，漏洞补丁的一致性如何？即，有多少漏洞在不同漏洞数据库中有相同的补丁？（见章节3.4）
- **RQ3 补丁类型分析：**开源软件漏洞补丁有哪些类型？（见章节3.5）
- **RQ4 补丁映射分析：**开源软件漏洞与其补丁在数量上有怎样的映射关系？（见章节3.6）
- **RQ5 补丁准确性分析：**当前商业漏洞数据库中，漏洞补丁的准确性如何？（见章节3.7）

上述研究问题中，RQ1 用来评估漏洞数据库中开源软件漏洞的补丁缺失情况，RQ2 用来评估不同漏洞数据库间漏洞补丁的不一致情况，RQ3 和 RQ4 用来表征常见的补丁类型以及开源漏洞及其补丁之间的映射关系，RQ5 可用来评估不同漏洞数据库中漏洞补丁的准确性。总得来说，RQ1、RQ2 和 RQ5 的结果旨在从不同的角度评估补丁质量，并挖掘出对自动化补丁识别方法的需求；RQ3 和 RQ4 旨在从不同角度捕捉开源软件漏洞补丁的特征，并为自动化补丁识别方法的设计提供启发。



### 3.1.2 评估标准

本章经验研究使用了信息检索中常用的评估标准—Precision（精确率）、Recall（召回率）和 F1-Score（F1 值）。

对于一个搜索或分类问题来说，样本分为“Positive（正）”和“Negative（负）”两个类别。那么，模型搜索或分类的结果就会有四种情况：

- **True Positive（真阳性）**，表示：将正样本预测为正，即数据库提供的或工具返回的补丁正确。
- **False Positive（假阳性）**，表示：将负样本预测为正，即数据库提供的或工具返回的补丁错误。
- **True Negative（真阴性）**，表示：将负样本预测为负，即数据库未提供的或工具未返回的错误补丁。
- **False Negative（假阴性）**，表示：将正样本预测为负，即数据库未提供的或工具未返回的正确补丁。

**Precision（精确率）**，用于衡量数据库提供的补丁中正确补丁的比率，或是工具返回结果中正确补丁的比率，计算公式如下：

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \quad (3.1)$$

**Recall（召回率）**，用于衡量正确补丁在数据库中被提供的比率，或是正确补丁被工具返回的比率，计算公式如下：

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \quad (3.2)$$

**F1-Score（F1 值）**，用以中和表示 Precision 及 Recall 的结果。因为 Precision 仅仅反映精确率，Recall 仅仅反映召回率，无法反映总体评估结果，F1-Score 计算公式如下：

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (3.3)$$

## 3.2 数据准备

数据准备部分包含漏洞数据库选择、广度数据集构建和深度数据集构建。

### 3.2.1 漏洞数据库选择

为了挑选出具有代表性的漏洞数据库作为研究对象，本文前期调研了来自安全领域的社区、工业界和学术界的漏洞数据库。

对于来自安全社区的数据库，本文着重关注 CVE、NVD 以及 CVE Details<sup>①</sup>。进一步分析发现，这三个数据库不提供结构化的补丁数据，补丁多是隐藏在参考

① [www.cvedetails.com](http://www.cvedetails.com)



链接列表中。此外，这三个数据库中不仅包含开源软件漏洞，还包括闭源软件、操作系统及硬件相关的漏洞。本文难以从中识别出开源软件漏洞，在经验研究中，本文排除了这类数据库。

对于来自学术界的数据集<sup>[16–20,42–43,50]</sup>，这些数据集中的漏洞通常限定于特定的一两种程序语言（例如 Python、Java 等），而非面向所有开源软件，不具有代表性。此外，在论文发表后，由于长期缺乏维护，这些漏洞数据集缺失较新的漏洞数据。本文也排除了这类漏洞数据集。

对于工业界的数据库，本文首先关注到 BlackDuck、Sonatype、WhiteSource、Veracode 和 Snyk 五家安全公司提供软件成分分析（Software Composition Analysis）服务。这种服务需要先构建尽可能完整且包含详细漏洞知识的漏洞库作为数据基础，通过识别并分析当前软件系统中使用的开源成分（即第三方库），报告系统中开源成分中的漏洞。本文首先选择这五家公司的漏洞数据库作为研究对象。截至 2021 年 4 月 5 日，收集的信息如下：

- **Black Duck**<sup>①</sup>，该公司的报告显示：数据库收录了 157,000 多个漏洞，涵盖 90 多种编程语言。其中，由于更新迅速，该数据库收录了很多较新的漏洞，这些漏洞甚至尚未被 NVD 收录。该公司的漏洞数据库由特定的专家团队进行维护，以确保漏洞数据的完整性和准确性。
- **Sonatype**<sup>②</sup>，该公司称：“OSS Index 是一个免费的开源组件目录，其中的扫描工具可帮助开发人员识别漏洞、了解风险并确保其软件安全”<sup>③</sup>。Sonatype 的漏洞数据库覆盖 20 多个程序语言生态系统（如：Maven、npm、Go、PyPI 等）。该数据库公开的漏洞信息包括漏洞描述、受漏洞影响的组件和版本、CVSS 向量、参考链接等信息。
- **WhiteSource**<sup>④</sup>，该公司从 NVD 及其他安全公告平台和问题追踪系统（Issue Tracking System）中共收集超过 175,000 个漏洞，涵盖 200 多种编程语言。
- **Veracode**<sup>⑤</sup>，该公司的漏洞数据库涵盖 10 多种编程语言相关的 18,000 多个漏洞。该数据库公开的信息包括受漏洞影响的组件和版本范围、补救措施、参考链接等。
- **Snyk**<sup>⑥</sup>，该公司称：漏洞数据库是由经验丰富的安全研究团队持续维护，通过关注安全公告、Jira 问题报告、GitHub 中代码提交等方式自动识别安全漏洞

① <https://www.synopsys.com/software-integrity/security-testing/software-composition-analysis.html>

② <https://ossindex.sonatype.org>

③ 英文原文为：“OSS Index is a free catalogue of open source components and scanning tools to help developers identify vulnerabilities, understand risk, and keep their software safe.”

④ <https://www.whitesourcesoftware.com/vulnerability-database/>

⑤ <https://sca.analysiscenter.veracode.com/vulnerability-database>

⑥ <https://snyk.io/vuln>

相关的报告。该公司的数据库覆盖超过 10 个程序语言生态系统，如：Maven、npm、Go、Composer 等。该数据库公开的漏洞信息包括受漏洞影响的组件、版本范围、修复方法、参考链接等。

进一步调研后发现，这五家公司中某些公司并未公开漏洞数据库，或是公开的漏洞信息中不包含用于修复漏洞的补丁知识，这将无法达成研究目标。最终，本文选择了其中两家公司的漏洞数据库作为研究对象，下文中简称为： $DB_A$  和  $DB_B$ 。

### 3.2.2 广度数据集构建

为了评估漏洞数据库中补丁的缺失程度以及不同数据库间补丁的不一致性（即 RQ1 和 RQ2），本文基于  $DB_A$  和  $DB_B$  构建了一个开源软件漏洞的广度数据集用以实验分析。截至 2020 年 4 月 7 日，分别从  $DB_A$  和  $DB_B$  中分别获取了 8,630 和 5,858 个开源软件漏洞，该广度数据集中包含 10,070 漏洞。

### 3.2.3 深度数据集构建

为了表征漏洞补丁的类型、映射关系以及尽可能准确地评估补丁的准确性（即 RQ3、RQ4 和 RQ5），本文还基于广度数据集，构建了一个开源软件漏洞的深度数据集。该数据集的漏洞数量少于广度数据集，但每个漏洞都包含由人工确认的补丁。

在该深度数据集的构建过程中，为了确保数据集能够涵盖足够多的漏洞用以实验评估，但又不至于在人工识别补丁的阶段产生难以完成的工作量，本文仅将在  $DB_A$  和  $DB_B$  中都含有补丁的漏洞列入该深度数据集，最终，该深度数据集包含 1,417 个漏洞。

对于该深度数据集中的每个漏洞，首先，由两位研究人员通过分析  $DB_A$  和  $DB_B$  数据库报告的补丁、查看 NVD 中的漏洞描述和参考链接、搜索 GitHub 代码仓库的提交历史、检索网络资源等方式，独立地找到其补丁。然后，再对比由两位研究人员独立查找得到的补丁，对于补丁不一致的漏洞结果，两位研究人员再一起分析讨论，直到达成共识。这两位研究人员分别是本文作者和与本文作者同课题组的学生。由于公开的信息有限，1,417 个漏洞，122 个漏洞无法明确其补丁。例如，对于 CVE-2016-3942， $DB_A$  和  $DB_B$  将提交 jsrender@f984e1<sup>①</sup> 标识为其补丁，但 NVD 中并没有该漏洞的信息，两位研究人员也无法确认该补丁的准确性。诸如此类，最终，该深度数据集还剩余 1,295 个漏洞。

本文还进一步分析了该深度数据集中 1,295 个开源软件漏洞的年份和程序语言分布情况，以评估该数据集是否具有代表性。如图 3-1 所示，漏洞的数量逐年

① <https://github.com/BorisMoore/jsrender/commit/f984e139deb0a7648d5b543860ec652c21f6dcf6>

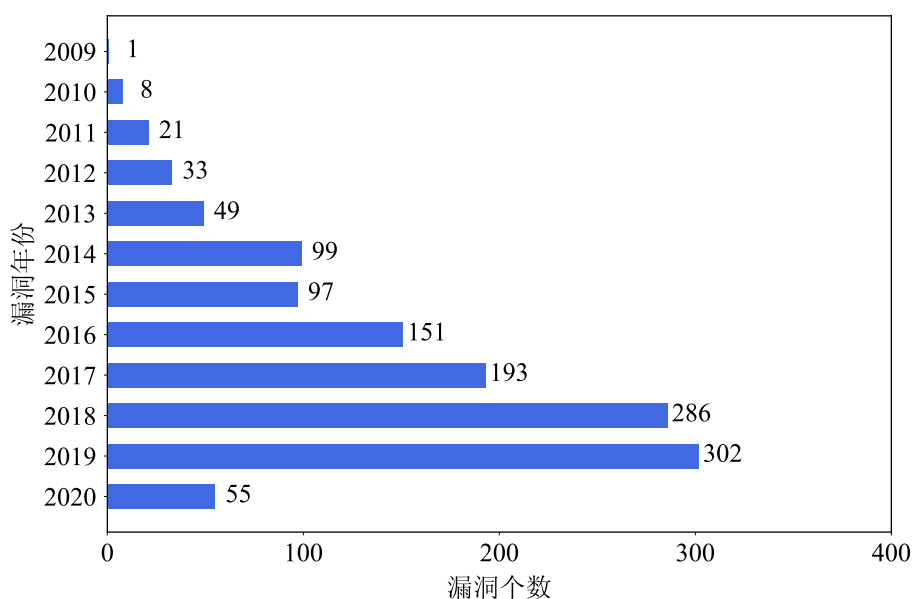


图 3-1 数据集中漏洞年份统计

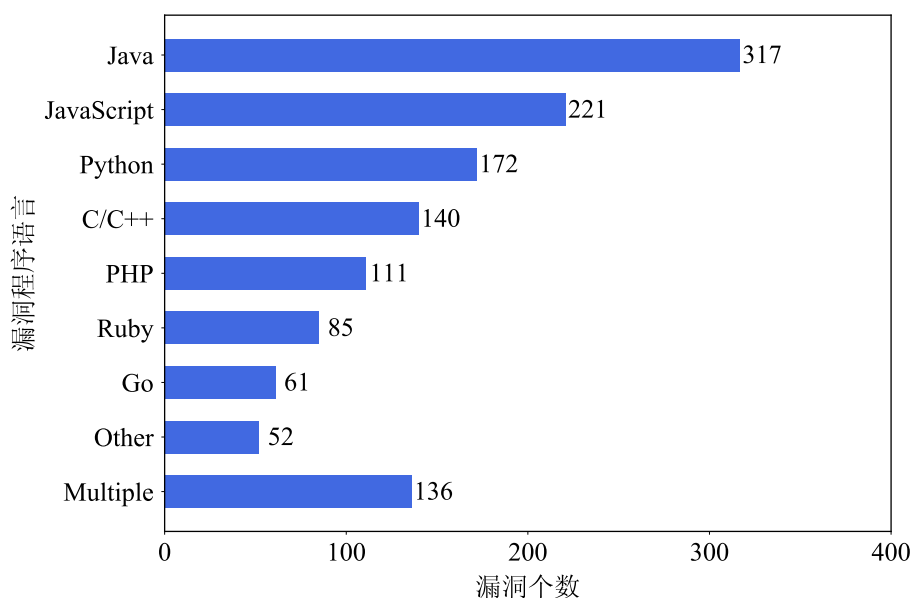


图 3-2 数据集中漏洞程序语言分布统计

增加，这与 Snyk 报告<sup>①</sup>的结果一致。此外，本文通过分析补丁中更改的源文件类型来确定漏洞的程序语言。如图3-2所示，深度数据集中的漏洞涵盖了七种较为常用的程序语言，具有较好的语言多样性。因此，可以认为该深度数据集对于开源软件漏洞数据库具有较好的代表性。

<sup>①</sup> [https://snyk.io/wp-content/uploads/sooss\\_report\\_v2.pdf](https://snyk.io/wp-content/uploads/sooss_report_v2.pdf).

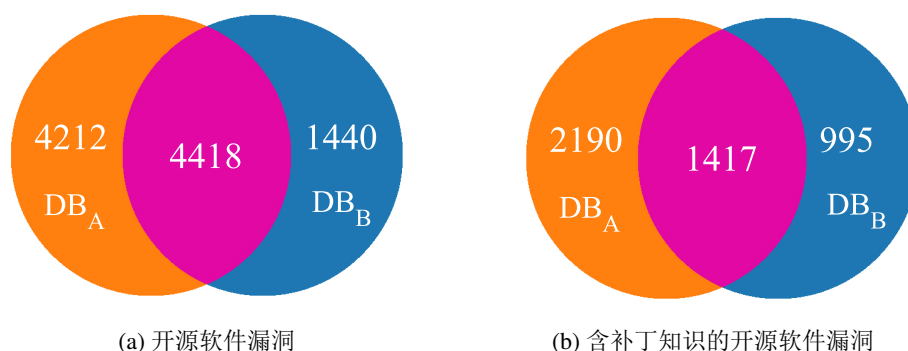
图 3-3  $DB_A$  与  $DB_B$  漏洞数据交集

表 3-1 补丁一致性结果

补丁一致	存在性不一致			内容不一致		
	总数	某一数据库 中无漏洞	某一数据库 中无补丁	总数	包含关系	非包含关系
907 (19.7%)	3,185 (69.2%)	1,392 (30.2%)	1,793 (39.0%)	510 (11.1%)	176 (3.8%)	334 (7.3%)

### 3.3 RQ1：补丁覆盖率分析

RQ1—补丁覆盖率分析，旨在评估漏洞数据库中开源软件漏洞的补丁缺失情况。对于包含 10,070 个开源软件漏洞的广度数据集，仅有 4,602 个漏洞含有补丁，补丁覆盖率为 45.7%。本文还进一步分析了数据库  $DB_A$  和  $DB_B$  的补丁覆盖率。如图 3-3a 所示，数据库  $DB_A$  和  $DB_B$  中共同拥有 4,418 个漏洞，同时  $DB_A$  包含 4,212 个特有的漏洞， $DB_B$  包含 1,440 个特有的漏洞。如图 3-3b 所示， $DB_A$  中，3,607 (41.8%) 的漏洞含有补丁； $DB_B$  中，2,412 (41.2%) 的漏洞含有补丁。 $DB_A$  和  $DB_B$  数据库共有的 4,418 个开源软件漏洞中，仅有 1,417 (32.0%) 的漏洞含有补丁。 $DB_A$  和  $DB_B$  数据库一共有 10,070 个开源软件漏洞（即广度数据集），而其中仅有 4,602 (45.7%) 的漏洞含有补丁。

由此可见，数据库  $DB_A$  和  $DB_B$  中开源软件漏洞的补丁覆盖率都比较低，仅为 41.8% 和 41.2%，这说明开源软件漏洞补丁缺失情况较为普遍。同时，这也体现出自动化补丁识别方法的必要性，可用于填补数据库中缺失的补丁。

### 3.4 RQ2：补丁一致性分析

RQ2—补丁一致性分析，旨在评估不同漏洞数据库间漏洞补丁的不一致情况。为了分析两个数据库之间的补丁一致性情况，本节仅分析带有补丁的漏洞，即图 3-3b 中的漏洞。考虑到漏洞补丁的个数可能不唯一（即可能为一组补丁集），所以仅当两个数据库针对同一漏洞提供的补丁集完全相同时，才判定为补丁一

表 3-2 补丁类型分析结果

补丁总数	GitHub 代码提交	SVN 代码提交	其他 Git 平台代码提交
3,043	2,852 (93.7%)	136 (4.5%)	55 (1.8%)
漏洞总数	仅 GitHub 代码提交	仅 SVN 代码提交	仅其他 Git 平台代码提交
1,295	1,202 (92.8%)	4 (0.3%)	30 (2.3%)

致。本节将补丁不一致分为，存在性不一致和内容不一致两种情况。存在性不一致是指，某一个数据库存在该漏洞的补丁，而另一个数据库却不存在该漏洞，或是存在该漏洞却不存在相关补丁。内容不一致是指，两个数据库都存在该漏洞的补丁，但它们的补丁集并不完全一致，可以是包含关系或非包含关系。以上两种情况，分别反映了出数据库  $DB_A$  和  $DB_B$  中开源软件漏洞补丁的不完整性及不一致性。

表3-1为补丁一致性分析的结果。其中，第一列为在  $DB_A$  和  $DB_B$  中具有一致补丁集的漏洞数量，第二至四列为补丁存在性不一致的漏洞数量，最后三列为都存在补丁但补丁集内容不一致的漏洞数量。可以发现：

- (1) 4,602个漏洞中，只有907 (19.7%) 的漏洞在  $DB_A$  和  $DB_B$  中有一致的补丁。
- (2) 超过三分之二 (即3,185 69.2%) 的漏洞在数据库  $DB_A$  和  $DB_B$  中存在补丁不一致的情况。其中1,392 (30.2%) 的漏洞不在  $DB_A$  或  $DB_B$  中，1,793 (39.0%) 的漏洞都存在于  $DB_A$  和  $DB_B$  中但在另一数据库中无补丁。
- (3) 510 (11.1%) 的漏洞补丁都存在于  $DB_A$  和  $DB_B$  中，但补丁集不一致。其中，176 (3.8%) 的漏洞，某一个数据库的补丁集包含另一个数据库的补丁集；334 (7.3%) 的漏洞， $DB_A$  和  $DB_B$  中的补丁集即不相同也不包含。

这些结果表明， $DB_A$  和  $DB_B$  间存在较多的补丁不一致情况，这很可能是由补丁不准确所导致，进而表明数据库中补丁的准确性需要进一步评估。

### 3.5 RQ3：补丁类型分析

RQ3—补丁类型分析，旨在表征开源软件漏洞的补丁类型。在章节3.2中，基于人工收集的深度数据集共包含1,295个漏洞及3,043个补丁。本小节将基于该数据集分析漏洞的补丁类型。如表3-2所示，3,043个补丁中，2,852 (93.7%) 的补丁都是 GitHub 代码提交，这可能是因为 GitHub 在开源软件中被广泛使用；另外136 (4.5%) 的补丁为 SVN 代码提交，仅有55 (1.8%) 的补丁为来自其他 Git 平台的代码提交。

此外，从漏洞的角度来看，1,295个漏洞中1,202 (92.8%) 的漏洞仅有 GitHub 提交类型的补丁，4 (0.3%) 的漏洞仅有 SVN 提交类型的补丁。只有30 (2.3%) 的

漏洞，补丁仅为来自其他 Git 平台的代码提交。由于很多项目是从 SVN 切换为 Git 管理，48（3.7%）的漏洞既有 GitHub 又有 SVN 提交类型的补丁。另外的11（0.8%）的漏洞，补丁为 GitHub 和其他 Git 平台的代码提交。

以上结果表明，开源软件漏洞的补丁类型主要为 GitHub 的代码提交（占比 93.7%），少部分为 SVN 代码提交，而极小部分为其他 Git 平台的代码提交。

## 3.6 RQ4：补丁映射分析

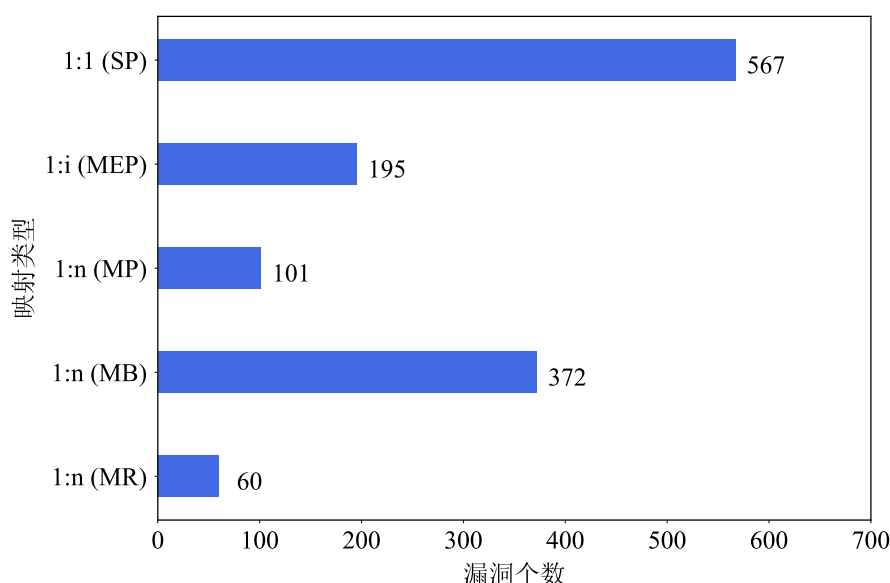


图 3-4 漏洞及其补丁映射类型统计

RQ4—补丁映射分析，旨在表征开源漏洞及其补丁之间的映射关系。基于深度数据集中1,295个漏洞及其补丁，本节将分析开源漏洞及其补丁间在数量上的映射关系。本文将漏洞与其补丁之间的映射关系分为三种类型：一对一、一对一组及一对多。

**一对一**，是指一个漏洞与其补丁在数量上为一对一的关系，即一个漏洞只需一个补丁即可修复，后文中简记为：*SP*（Single Patch）。如图3-4所示，深度数据集中567（43.8%）的漏洞与其补丁都是一对一的映射关系（*SP*）。

**一对一组**，是指一个漏洞有多个补丁，漏洞与其补丁在数量上非一对一关系。然而，这些补丁又都是等效的，任何一个补丁都足以修复该漏洞，后文中简记为：*MEP*（Multiple Equivalent Patch）。等效补丁，是指代码变更完全一样的两个补丁提交，其主要有两种类型：

- （1）**请求提交（Requested Commit）VS. 合并提交（Merged Commits）**，是指通过 GitHub 中的抓取（Pull Request）功能提交修复漏洞的代码更改。此时，

请求提交和合并提交是该漏洞的等效补丁。例如, `python-jose@89b463`<sup>①</sup>是请求提交, `python-jose@73007d`<sup>②</sup>是用于修复 CVE-2016-7036 的合并提交, 这两个提交的代码变更完全相同, 完全等效。

- (2) **SVN 中代码提交 VS. GitHub 中代码提交**, 一些开源软件的仓库是后期由 SVN 迁移到 GitHub 的, 因此, 同一软件的 SVN 和 GitHub 的代码仓库中都有用于修补该漏洞的代码提交, 且这两处提交的代码变更是完全一样且完全等效的。例如, 软件 `james-hupa` 代码仓库从 SVN 迁移到了 GitHub, SVN 代码提交 `james-hupa@1373762`<sup>③</sup>与 GitHub 代码提交 `james-hupa@aff28a`<sup>④</sup>是等效的。

如图3-4所示, 深度数据集中195 (15.1%) 的漏洞与其补丁为一对一组映射关系 (MEP)。

**一对多**, 是指一个漏洞与其补丁在数量上为一对多的关系, 即一个漏洞需多个非等效的补丁来修复。如图3-4所示, 深度数据集中533 (41.2%) 的漏洞与其补丁为一对多的映射关系, 其可以再分为三种类型:

- (1) **多补丁**, 一个漏洞需通过一个分支中的多个代码提交来修复。这是因为该漏洞较难修复需多次提交, 或是后期发现初始的补丁提交不足以修复漏洞便追加了补丁提交。后文中将此简记为: **Multiple Patch, MP**, 占比7.8% (101 个漏洞)。例如, CVE-2017-17837 由三个独立的提交 `deltaspikes@4e2502`<sup>⑤</sup>、`deltaspikes@72e607`<sup>⑥</sup>和 `deltaspikes@d95abe`<sup>⑦</sup>修复。
- (2) **多分支**, 一个漏洞由多个分支中的多个补丁集修复。这是因为该漏洞影响了开源软件的多个版本, 而每个版本又都在独立的分支上维护。后文中将此简记为: **Multiple Branches, MB**, 占比28.7% (372 个漏洞)。例如, CVE-2019-19118 影响了软件框架 `django` 的 2.1.x、2.2.x、3.0.x 和 3.2.x 版本, 每个版本又都在独立的分支上维护。分布在四个分支的提交 `django@103ebe`<sup>⑧</sup>、`django@36f580`<sup>⑨</sup>、`django@092cd6`<sup>⑩</sup>和 `django@11c5e0`<sup>⑪</sup>分别修复了受影响的四个版本。其中, `django@103ebe` 与其他提交中的代码变更并不相同。

① <https://github.com/mpdavis/python-jose/commit/89b46353b9f611e9da38de3d2fedf52331167b93>

② <https://github.com/mpdavis/python-jose/commit/73007d6887a7517ac07c6e755e494baee49ef513>

③ <https://svn.apache.org/viewvc?view=revision&revision=1373762>

④ <https://github.com/apache/james-hupa/commit/aff28a8117a49969b0fc8cc9926c19fa90146d8d>

⑤ <https://github.com/apache/deltaspikes/commit/4e2502358526b944fc5514c206d306e97ff271bb>

⑥ <https://github.com/apache/deltaspikes/commit/72e607f3be66c30c72b32c24b44e9deaa8e54608>

⑦ <https://github.com/apache/deltaspikes/commit/d95abe8c01d256da2ce0a5a88f4593138156a4e5>

⑧ <https://github.com/django/django/commit/103ebe2b5ff1b2614b85a52c239f471904d26244>

⑨ <https://github.com/django/django/commit/36f580a17f0b3cb087deadf3b65eea024f479c21>

⑩ <https://github.com/django/django/commit/092cd66cf3c3e175acce698d6ca2012068d878fa>

⑪ <https://github.com/django/django/commit/11c5e0609bcc0db93809de2a08e0dc3d70b393e4>

表 3-3  $DB_A$  和  $DB_B$  补丁准确性评估结果

映射类型	数量	$DB_A$			$DB_B$		
		Pre.	Rec.	F1	Pre.	Rec.	F1
1:1 (SP)	567	0.908	0.915	0.910	0.900	0.921	0.906
1:i (MEP)	195	0.935	0.898	0.902	0.924	0.909	0.906
1:n (MP)	101	0.923	0.483	0.616	0.911	0.520	0.638
1:n (MB)	372	0.941	0.510	0.620	0.932	0.436	0.555
1:n (MR)	60	0.913	0.610	0.695	0.964	0.526	0.636
总计	1,295	0.923	0.748	0.793	0.917	0.730	0.771

(3) **多仓库**，一个漏洞由多个代码仓库中的多个补丁集修复。这是因为该漏洞影响了多个开源软件或一个开源库的多个版本，而这些版本是分布在独立的代码仓库中维护的，所以会有来自不同仓库的提交。后文中将此简记为：**Multiple Repositories, MR**，占比4.6%（60个漏洞）。例如，CVE-2016-5104影响了 libimobiledevice 和 libusbmuxd 两个开源软件，提交 libimobiledevice@df1f5c<sup>①</sup>和 libusbmuxd@4397b3<sup>②</sup>分别修复了受影响的两个开源库。

这些结果体现了开源软件漏洞与其补丁之间映射关系的多样性，超过40%的漏洞与其补丁具有一对多的映射关系。在后文设计自动化补丁识别方法时，应充分考虑该特征。

### 3.7 RQ5：补丁准确性分析

RQ5—补丁准确性分析，旨在评估不同漏洞数据库中漏洞补丁的准确性。本文使用精确率（precision）、召回率（recall）和 F1 值（F1-score）作为评估补丁准确性的指标。对于具有等效补丁的漏洞，则进行特殊计算。例如，对于具有两个等效补丁的漏洞，若数据库提供两个等效补丁中的任意一个，精确率和召回率都为 1；若数据库提供两个等效补丁中的一个补丁和另一个不相关的补丁，那么精确率为 0.5，召回率为 1。其他情况，依此类推。

表3-3为  $DB_A$  和  $DB_B$  的补丁准确性评估结果。其中，第一列为漏洞与补丁的映射类型，第二列为该映射类型的漏洞数量，最后六列分别为数据库  $DB_A$  和  $DB_B$  中漏洞补丁的准确率、召回率和 F1 值。结果表明，对于 SP 和 MEP 类型的漏洞， $DB_A$  和  $DB_B$  可达到约 90% 的精确率和召回率；对于 MP、MB 和 MR 类型的漏洞， $DB_A$  和  $DB_B$  可达到 90% 以上的高精确率，但仅有约 50% 的召回率。例如，对于漏洞 CVE-2017-17837， $DB_A$  和  $DB_B$  仅报告三个补丁中的一个；对

① <https://github.com/libimobiledevice/libimobiledevice/commit/df1f5c4d70d0c19ad40072f5246ca457e7f9849e>

② <https://github.com/libimobiledevice/libusbmuxd/commit/4397b3376dc4e4cb1c991d0aed61ce6482614196>



于漏洞 CVE-2019-19118,  $DB_A$  报告了四个补丁中的两个, 而  $DB_B$  仅报告四个补丁中的一个。

这说明漏洞数据库  $DB_A$  和  $DB_B$  具有较高的精确率, 但经常会遗漏一些漏洞的补丁, 尤其是对于具有多个补丁的漏洞。对于漏洞数据库用户来说, 这会给漏洞的及时检测和修复带来较大困难。

### 3.8 研究发现

本文针对开源软件漏洞补丁的经验研究, 共涉及五个研究问题。其中, RQ1 补丁覆盖率分析、RQ2 补丁一致性分析和 RQ5 补丁准确性分析的结果旨在从不同的角度评估补丁质量, 并挖掘出对自动化补丁识别方法的需求。RQ3 补丁类型分析和 RQ4 补丁准确性分析旨在从不同角度捕捉开源软件漏洞补丁的特征, 并为自动化补丁识别方法的设计提供启发。该经验研究中, 本文主要有以下发现:

- (1) 商业漏洞数据库中开源软件漏洞补丁的质量并不理想, 体现在: (i) 开源软件漏洞补丁缺失情况较为普遍, 商业数据库  $DB_A$  和  $DB_B$  中开源软件漏洞的补丁覆盖率仅为 41.8% 和 41.2%。(ii) 商业漏洞数据库  $DB_A$  和  $DB_B$  具有较高的精确率, 但经常会遗漏一些漏洞的补丁, 尤其是对于具有多个补丁的漏洞。对于安全服务用户来说, 这会给漏洞的及时检测和修复带来较大困难。这体现出当前开源软件漏洞数据库的不足, 以及利用自动化补丁识别方法完善漏洞数据的需求。
- (2) 开源软件漏洞补丁在类型、映射关系方面有一定的特殊性, 设计自动化补丁识别方法时应充分考虑。体现在: (i) 93.7% 的补丁都为 GitHub 代码提交的形式。(ii) 开源软件漏洞与其补丁之间映射关系具有多样性, 超过 40% 的漏洞与其补丁具有一对多的映射关系。

## 第4章 开源软件漏洞的补丁识别方法

本章将详细阐述 TRACER——一种基于多源知识的开源软件漏洞补丁识别方法的设计。

### 4.1 方法概述

基于前一章经验研究的发现，本文设计了一个名为 TRACER 的自动化开源软件漏洞的补丁识别方法。该方法可用于识别代码提交类型的补丁（即补丁提交，详见章节3.5），并构建一对多的漏洞补丁映射关系。该方法的核心思想是：漏洞补丁会在讨论和解决漏洞的、多种来源的漏洞公告、分析报告等参考链接中被频繁地提及和引用。因此，本文首先设计了一种基于多知识源的漏洞参考链接网络，再从该网络中选出具有最高置信度和连通度的补丁节点作为结果，并基于选定的补丁进行补丁扩展，从而构建一对多的漏洞补丁映射关系。

图4-1为 TRACER 的方法概览。其中，TRACER 以漏洞的 CVE ID 作为输入，主要经过三个步骤，最终返回补丁列表。步骤一，多源参考链接网络构建。该步骤的目的是将该漏洞在被报告、讨论和解决阶段引用的参考链接进行建模（参考链接，即引用的 URL 网址）。TRACER 从多个漏洞知识源（即 NVD、Debian<sup>①</sup>、Red Hat<sup>②</sup>和 GitHub）中提取与该漏洞相关的参考链接并构建一个参考链接网络。这里将 NVD 视为主知识源，将 Debian、Red Hat 和 GitHub 视为次知识源，次级知识源列表可以不断扩增，以确保该方法的可扩展性和灵活性。步骤二，补丁选择。TRACER 从构建的参考链接网络中选择具有高连通性和高置信度的补丁节点作为该漏洞的补丁。步骤三，补丁扩增。经验研究中发现漏洞与其补丁之间存在一对多的映射关系。针对这一特征，基于步骤二中选定的补丁，TRACER 通过搜索同一代码库所有分支中的相关代码提交来扩展补丁集。最终，TRACER 返回该漏洞的补丁列表。三个步骤的具体设计和实现将在下文中详细阐述。

### 4.2 参考链接网络构建

TRACER 的步骤一为漏洞参考链接网络构建，一共包括三个子步骤——公告分析、参考链接分析和参考链接扩增。前两个子步骤通过分析来自 NVD、Debian

<sup>①</sup> <https://security-tracker.debian.org/tracker/>

<sup>②</sup> <https://bugzilla.redhat.com/>

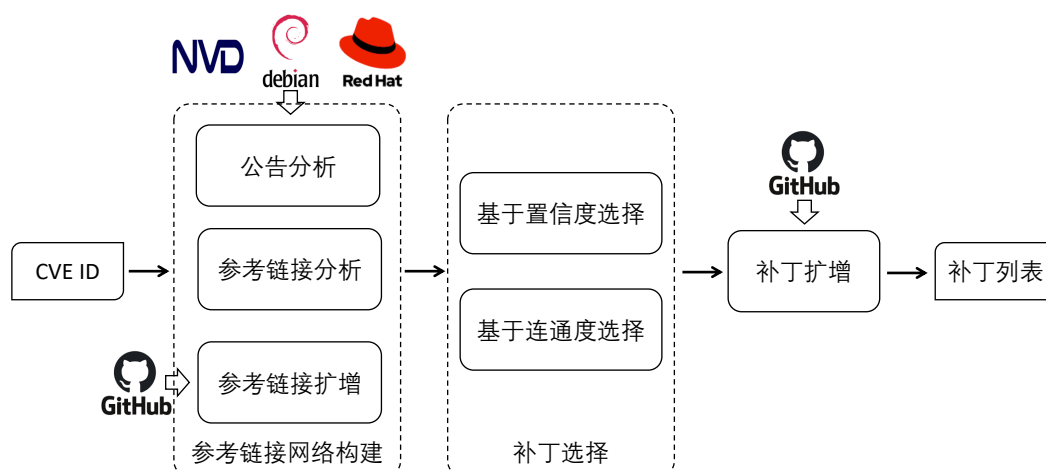


图 4-1 TRACER 方法概览

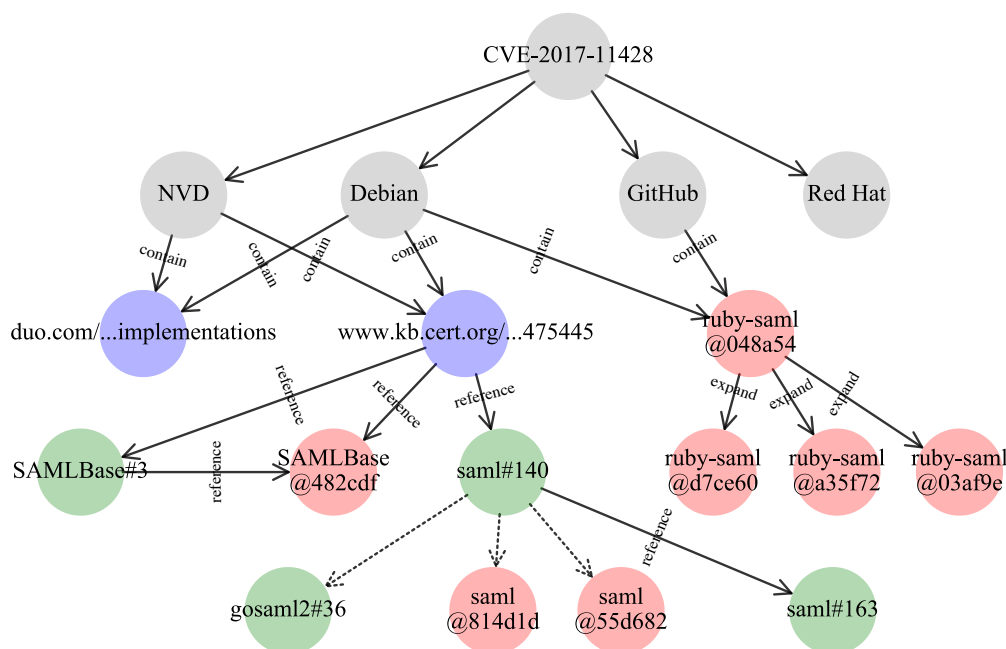


图 4-2 漏洞 CVE-2017-11428 的多源参考链接网络

和 Red Hat 三个知识源的漏洞公告及其中的参考链接，构建初始参考链接网络。第三个子步骤为从 GitHub 平台中搜索相关的代码提交，作为补丁节点扩入参考链接网络。

### 4.2.1 公告分析

输入 CVE ID 后，TRACER 初始化参考链接网络，将输入的 CVE ID 设置为根节点，并将三个漏洞知识源（即 NVD、Debian 和 Red Hat）中该漏洞的公告添加为根节点的子节点。这些公告节点可用于后期追溯补丁的来源。

**样例 4.1** 图4-2为漏洞 CVE-2017-11428<sup>①</sup>的多源参考链接网络。其中，最顶层为根节点（即 CVE ID），第二层为公告节点（即知识源 NVD<sup>②</sup>、Debian<sup>③</sup>和 Red Hat<sup>④</sup>）。

然后，TRACER 通过网络请求分别获取 NVD、Debian 和 Red Hat 平台中该漏洞的漏洞公告。NVD 平台中，漏洞公告以 JSON 形式按年份存储于结构化数据<sup>⑤</sup>中，TRACER 通过下载并解析相应的 JSON 文件即可获得 NVD 中该漏洞的公告信息。Debian 平台中，漏洞公告也以结构化数据的形式存储在仓库<sup>⑥</sup>中，TRACER 可直接从中解析 Debian 提供的该漏洞公告信息。Red Hat 平台提供了 WebService API<sup>⑦</sup>服务，TRACER 可以直接使用该服务来检索 Red Hat 平台的漏洞公告。其中，Debian 会跟踪 NVD 上的所有漏洞，而 Red Hat 仅跟踪 NVD 上的部分漏洞。

TRACER 从每个知识源的漏洞公告中提取引用的参考链接信息，并将它们添加为相应公告节点的子节点。对于 NVD 公告，TRACER 从“References”字段中直接获取相关参考链接。类似地，对于 Debian 公告，TRACER 直接从“Notes”字段中提取出相关参考链接。对于 Red Hat 公告，TRACER 使用正则表达式从评论区（“comments”字段）中提取出相关参考链接。这是因为开发人员常常会在评论区讨论和记录漏洞的解决过程并列出补丁。

**样例 4.2** 如图4-2中的第三层所示，对于 CVE-2017-11428，NVD 公告中包含了两个参考链接。链接一<sup>⑧</sup>是描述此漏洞细节的博客，链接二<sup>⑨</sup>是该漏洞的第三方公告。如图4-3所示，这两个参考链接也包含在 Debian 公告中，不过 Debian 公告还包含修复此漏洞的 GitHub 提交链接 ruby-saml@048a54<sup>⑩</sup>。此外，Red Hat 平台并未收录该漏洞，所以也无参考链接信息。

基于参考链接网页的内容，TRACER 将参考链接节点分为三种类型：补丁节点（Patch Node）、问题节点（Issue Node）和混合节点（Hybrid Node）。这里区分出“补丁节点”，是因为该方法的根本目的就是在多个知识源中找到所引用的补丁。识别补丁节点的方法为，如果参考链接中包含“git”字段且可通过正则表达式匹配到代码提交 ID（Commit ID），则该链接为 Git 提交形式的补丁节点；如

① <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-11428>

② <https://nvd.nist.gov/vuln/detail/CVE-2017-11428>

③ <https://security-tracker.debian.org/tracker/CVE-2017-11428>

④ [https://bugzilla.redhat.com/show\\_bug.cgi?id=CVE-2017-11428](https://bugzilla.redhat.com/show_bug.cgi?id=CVE-2017-11428)

⑤ <https://nvd.nist.gov/vuln/data-feeds>

⑥ <https://salsa.debian.org/security-tracker-team/security-tracker/-/tree/master/data/CVE>

⑦ <https://bugzilla.redhat.com/docs/en/html/api/index.html>

⑧ <https://duo.com/blog/duo-finds-saml-vulnerabilities-affecting-multiple-implementations>

⑨ <https://www.kb.cert.org/vuls/id/475445>

⑩ <https://github.com/onelogin/ruby-saml/commit/048a544730930f86e46804387a6b6fad50d8176f>

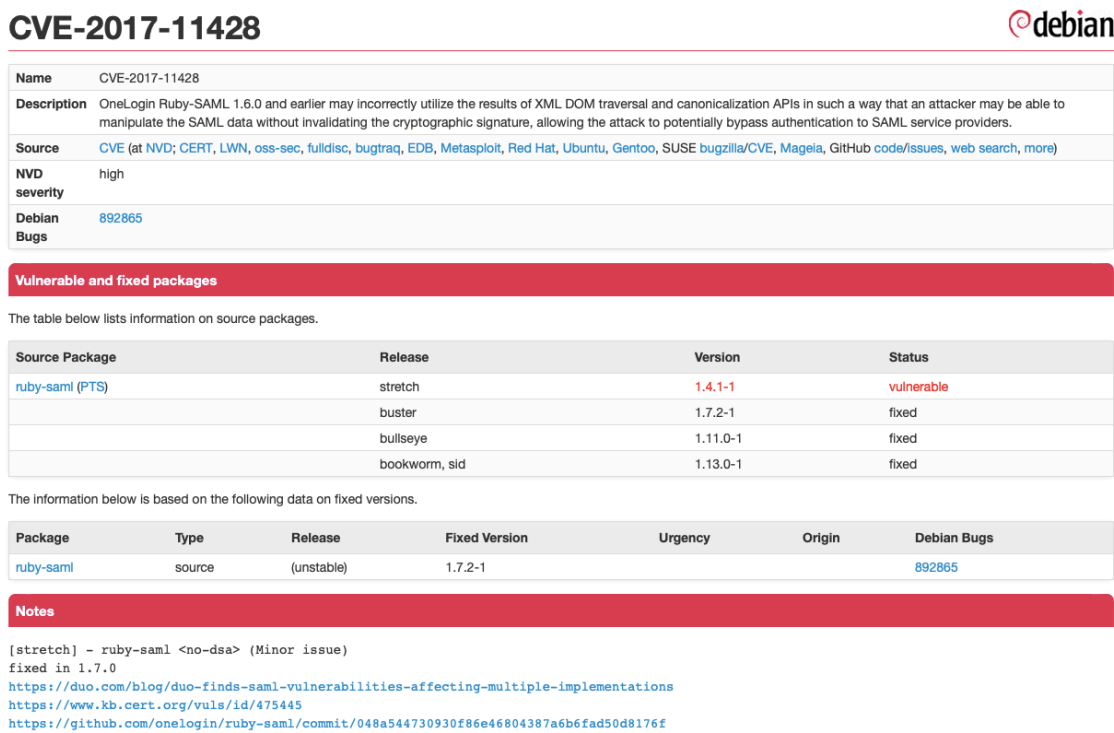


图 4-3 Debian 平台中漏洞 CVE-2017-11428 的公告

果参考链接中包含“svn”字段且可通过正则表达式匹配到代码提交 ID（Commit ID），则该链接为 SVN 提交形式的补丁节点。

这是还区分出“问题节点”，是因为开发人员常常会在问题追踪系统（Issue Tracker）中为漏洞申请一个问题报告（Issue Report，也称为工单），并在该报告的评论区讨论解决方案并引用补丁链接。所以，问题报告是一种较为特殊且重要的参考链接。此外，问题追踪系统中的问题报告会有一个问题标识符（Issue ID），开发人员常常会将此问题标识符写入代码提交信息（Commit Message）中。这有利于识别漏洞相关的代码提交，详见章节4.2.3。识别问题节点的方法为，如果参考链接中包含“/github.com/”和“/issues/”，则该链接为 GitHub Issue 类型的问题节点。如果参考链接中包含“/github.com/”和“/pull/”，则该链接为 GitHub 抓取（Pull Request）类型的问题节点。如果参考链接中包含“bugzilla”、“jira”、“issues”、“bugs”、“tickets”和“tracker”中的某一个字段且可通过正则表达式匹配到问题标识符（Issue ID），则该链接为通用类型的问题节点。对于所有未识别为补丁或问题节点的参考链接将被视为“混合节点”，它们多为博客、第三方漏洞公告等。

**样例 4.3** 如图4-2中的第三层所示，NVD 和 Debian 公告中包含的两个参考链接被标识为混合节点（即图中的两个紫色节点），仅有一个在 Debian 公告中的参考链接被识别为补丁节点（即图中的红色节点）。该漏洞在该步骤中未识别到问题

节点。

## 4.2.2 参考链接分析

对于前序步骤中已添加入图的每个参考链接节点，TRACER 将通过以下两种节点分析方式，以分层迭代的方式继续扩建参考链接网络。

**方式一**，如果该参考链接节点的类型为补丁节点，TRACER 会通过网络请求该代码提交的内容并分析该代码提交是否只涉及了测试代码或非代码文件的修改。如果是的话，则表明该代码提交一定不是用于修复漏洞的补丁提交，TRACER 会从网络中删除该节点。对于测试代码的判定，TRACER 通过检查修改的文件路径中是否含有“test”字段来判断，如果文件路径含有“test”字段则判定为测试文件。对于源代码文件的判定，TRACER 通过检查修改文件的后缀来识别该文件是否为代码文件，如果文件的后缀非常见的代码文件类型，即不在列表<sup>①</sup>中则判定为非源代码文件。

**方式二**，如果该参考链接节点的类型为问题节点或混合节点，TRACER 会通过网络请求该参考链接的页面信息（即 HTML 文本），并解析出该网页中引用的参考链接，将其作为子节点扩入参考链接网络。首先，对于网页中参考链接的提取，TRACER 使用正则表达式提取纯文本中的 URL，同时使用 HTML 解析器提取超链接（即 <a> 标签）中的 URL。然后，使用章节4.2.1中相同的方式识别参考链接类型，并将这些引用添加为当前节点的子节点。值得注意的是，在该步骤及以后流程中参考链接网络将不再加入混合节点。这是因为混合节点极易引入噪声，随着网络构建得越深，噪声也就越多。所以，除了直接被 NVD、Debian 和 Red Hat 公告节点引用的混合节点，其他参考链接节点分析中将不再考虑混合节点，仅识别补丁节点和问题节点。此外，考虑到 GitHub Issue 中通常会引用来自其他软件仓库的问题或提交链接，这给网络带来过多的噪音且使网络搜索空间爆炸。因此，在该步骤中，如果被分析的参考链接节点为 GitHub Issue 类型，则仅仅将该节点引用的同一代码仓库中的代码提交或问题节点添加到网络中。

对于所有新增的节点，TRACER 将一直重复以上两种节点分析方式迭代扩增网络，直到没有任何新增的节点，或者网络深度达到设定的阈值（网络深度默认为5层）时，该步骤结束。

**样例 4.4** 在第一次迭代中，因为 ruby-saml@048a54 并非仅涉及测试代码或非源代码文件的更改，所以 TRACER 将补丁节点 ruby-saml@048a54 保留在图4-2中第三层。此外，TRACER 标识还出第三层中的两个混合节点，其中一个混合节

<sup>①</sup> 常见代码文件后缀列表：['java', 'py', 'c', 'cc', 'cpp', 'cxx', 'c++', 'js', 'go', 'rb', 'cs', 'as', 'php', 'pl', 'coffee', 'h', 'm', 'ts', 'kt', 'mjs', 'twig', 'htaccess', 'tpl', 'pt', 'scala', 's', 'erb', 'swf', 'asm', 'groovy', 'jsp', 'sh', 'hpp', 'phps', 'script', 'wscript', 'ldif', 'Tokens', 'nsi', 'tcl', 'idl', 'pyx', 'ps1', 'toml', 'inl', 'x', 'S', 'hbs']

CERT Coordination Center

Home

Notes

Search

Report a Vulnerability

Disclosure Guidance

VINCE

Home > Notes > VU#475445

Multiple SAML libraries may allow authentication bypass via incorrect XML canonicalization and DOM traversal

Vulnerability Note VU#475445

Original Release Date: 2018-02-27 | Last Revised: 2018-06-05

Wizkunde B.V.

Affected

Updated: April 05, 2018

Statement Date: April 03, 2018

Status

Affected

Vendor Statement

We've got notified about this bug on Monday 3-4-2018 and immediately took actions to fix the ability to exploit this at implementations of our library.

The patch is written in this commit:  
<https://github.com/Wizkunde/SAMLBase/commit/482cdf8c090e0f1179073034ebcb609ac7c3f5b3>

Vendor Information

Wizkunde SAMLBase prior to version 1.2.7 is affected, the issue was addressed in version 1.2.7. CVE-2018-5387 has been assigned.

Vendor References

<https://github.com/Wizkunde/SAMLBase/issues/3>  
<https://github.com/Wizkunde/SAMLBase/commit/482cdf8c090e0f1179073034ebcb609ac7c3f5b3>

SAML (golang)

Not Affected

Notified: March 16, 2018 • Updated: March 19, 2018

Statement Date: March 19, 2018

Status

Not Affected

Vendor Statement

We have not received a statement from the vendor.

Vendor Information

We are not aware of further vendor information regarding this vulnerability.

Vendor References

<https://github.com/crewjam/saml/pull/140>

CONTACT US ABOUT THIS VULNERABILITY >

PROVIDE A VENDOR STATEMENT >

CONTACT US ABOUT THIS VULNERABILITY >

PROVIDE A VENDOR STATEMENT >

图 4-4 混合节点 “www.kb.cert.org/...475445”

点未引用任何问题或提交链接，另一个混合节点引用了两个问题链接 SAML-Base#3 和 saml#140 以及一个代码提交链接 SAMLBase@482cdf。图4-4为混合节点 “www.kb.cert.org/...475445” 引用的代码提交 SAMLBase@482cdf<sup>①</sup>、问题链接

① <https://github.com/GoGentoOSS/SAMLBase/commit/482cdf8c090e0f1179073034ebcb609ac7c3f5b3>

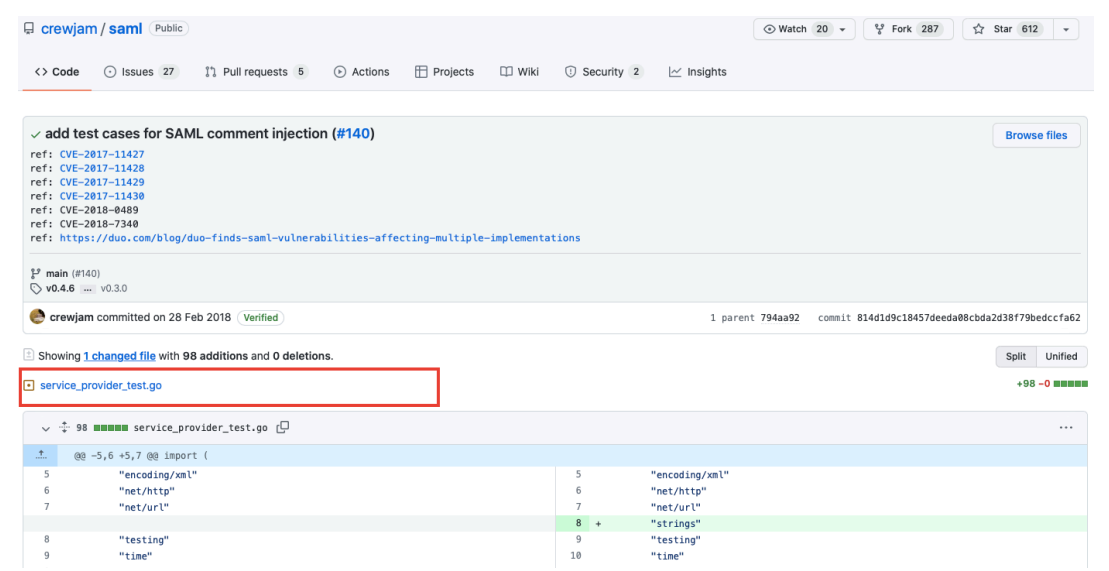


图 4-5 “saml@814d1d” 代码提交

SAMLBa#3<sup>①</sup>和问题链接 saml#140<sup>②</sup>。

在第二次迭代中,TRACER 发现节点 SAMLBa#3 引用了 SAMLBa@482cdf,而节点 saml#140 引用了两个问题链接 gosaml2#36 和 saml#163 以及两个代码提交链接 saml@814d1d 和 saml@55d682。考虑到 gosaml2#36 与 saml#140 并不属于同个代码仓库, saml@814d1d 和 saml@55d682 也仅涉及测试代码的更改, TRACER 便不将它们添加到网络中。为了便于示例讲解, 这些未加入网络的节点仍显示在图4-2中, 但以虚线连接。图4-5为代码提交链接 saml@814d1d<sup>③</sup>, 可以发现, 该提交仅修改了“service\_provider\_test.go”一个文件, 且该文件为测试代码文件。

4.2.3 参考链接扩增

除了 NVD、Debian 和 Red Hat 等漏洞公告平台可作为漏洞知识来源之外, 代码托管平台也可以被视为隐含的知识来源, 因为补丁通常隐藏在代码仓的代码提交历史中。因此, 在该步骤中, TRACER 将搜索代码托管平台以获取漏洞的补丁提交, 并将检索到的补丁扩增到该漏洞的参考链接网络。

受经验研究中补丁类型分析结果的启发, 93.7% 的补丁都为 GitHub 代码提交的形式 (见章节3.5), 所以在该步骤中 TRACER 只搜索 GitHub 平台的代码提交。此外, 问题跟踪系统通常还会为漏洞相关的问题分配一个问题标识符 (Issue ID)。同样, 软件厂商通常也会为该漏洞分配一个漏洞公告标识符 (Advisory ID)。例如, 受漏洞 CVE-2019-10426 影响的软件厂商为漏洞分配的标识符为 SECURITY-

① <https://github.com/GoGentoOSS/SAMLBa/issues/3>  
② <https://github.com/crewjam/saml/pull/140>  
③ <https://github.com/crewjam/saml/commit/814d1d9c18457deeda08cbda2d38f79bedccfa62>



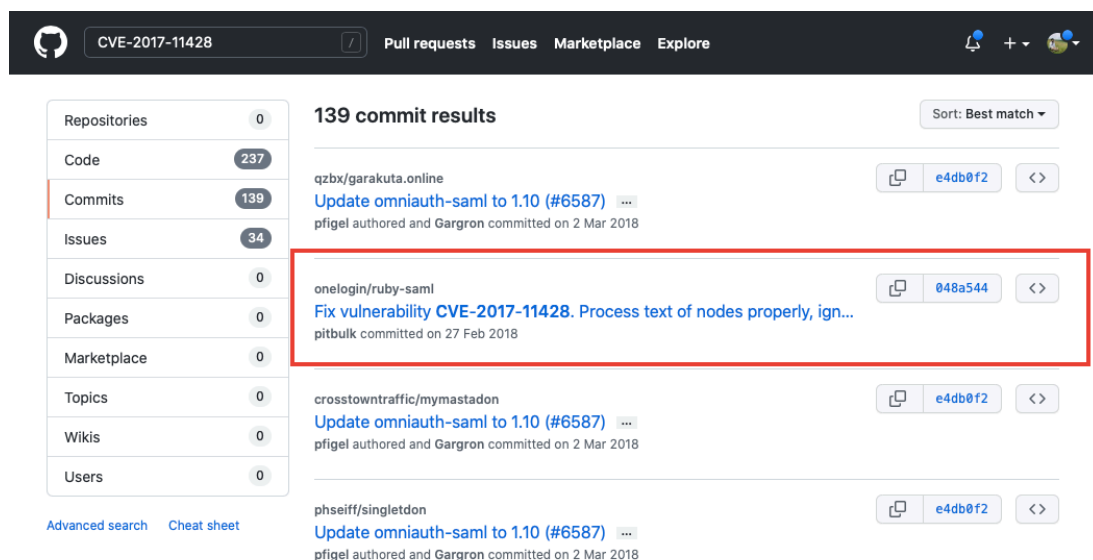


图 4-6 关键词“CVE-2017-11428”的搜索结果

1573<sup>①</sup>，问题跟踪系统为该漏洞分配的问题标识符为 THRIFT-4647<sup>②</sup>。这些标识符会包含在修复补丁的代码提交中。针对这一特征，TRACER 使用正则表达式从已有的网络节点中分别提取问题标识符和公告标识符，并与漏洞标识符一起作为关键词，检索 GitHub 中的代码提交。

TRACER 通过 GitHub 提供的 REST API<sup>③</sup>接口，全站搜索 GitHub 中相关的代码提交。为了减少噪音，对于 API 返回的提交，TRACER 首先会检查该代码提交的仓库信息是否与漏洞的 CPE<sup>④</sup>信息匹配。其中，CPE 是针对受漏洞影响软件的结构化命名方案，包括供应商和产品名称信息，可以直接从 NVD 的 JSON 文件中解析得到。对于代码提交的仓库信息与 CPE 信息匹配的判定，TRACER 沿用了 Dong 等人的匹配准则<sup>[21]</sup>，该准则可灵活地应对同一个软件名称存在不同别名的情况。具体来说，给定两个软件的名称，如果匹配的单词数不小于不匹配的单词数，则这两个软件的名称匹配成功，视为同一软件。此外，TRACER 仍会检查该提交是否为测试代码或非代码文件的更改。如果这两个检查都通过，TRACER 会将其作为 GitHub 节点的子节点加入网络。

**样例 4.5** 对于样例 CVE-2017-11428，TRACER 无法从构建的网络中提取任何问题或公告标识符。因此，TRACER 仅使用漏洞标识符（CVE ID）来搜索 GitHub 中相关的代码提交。如图4-7所示，搜索返回的提交包含提交 ruby-saml@048a54，该提交的仓库信息是“onelogin: ruby-saml”，此 CVE-2017-11428 的 CPE 是“onelogin:

① <https://www.jenkins.io/security/advisory/2019-09-25/#SECURITY-1573>

② <https://issues.apache.org/jira/browse/THRIFT-4647>

③ <https://docs.github.com/en/rest/reference/search#search-commits>

④ <https://nvd.nist.gov/products/cpe>

ruby-saml”，因此实现了名称的完全匹配，TRACER 会将 ruby-saml@048a54 节点加入网络。由于该节点已包含在参考网络中，TRACER 便不再新增节点，仅将其连接为 GitHub 节点的子节点，见图4-2。此外，该次搜索结果中没有其他匹配的代码提交。

## 4.3 补丁选择

步骤二的设计目标是尽可能准确且完整地从该漏洞参考链接网络中选择出补丁节点。为了实现这一目标，本小节设计了以下两种补丁选择方法。

### 4.3.1 基于置信度的补丁选择方法

在该方法中，TRACER 将直接选择具有高置信度的补丁作为正确的补丁。具体来说，本文认为参考链接网络中的两种补丁节点具有比较高的置信度。

第一种，为被 NVD 直接引用的补丁节点，即图中作为 NVD 子节点的补丁节点，被认为具有高置信度。这是因为 NVD 数据库建立在强大的社区支持下，每个漏洞的信息都经过多个流程的人工确认，且初始漏洞报告在发布后还会不断维护更新。因此，本文认为 NVD 中的补丁具有较高的精度。此外，已有的很多工作<sup>[4-5,25]</sup>都是基于这种方法来识别补丁。

第二种，为从 GitHub 直接搜索出的补丁节点，即图中作为 GitHub 子节点的补丁节点，被认为具有高置信度。这是因为在将此类补丁节点添加到网络前，经过了较为严苛的确认。TRACER 会确保代码提交信息中包含该漏洞的 CVE ID、Advisory ID 或 Issue ID，并且其所属的仓库信息必须与该漏洞的 CPE 名称成功匹配。这种方法也在已有的很多工作<sup>[1,33]</sup>中使用。

**样例 4.6** 在漏洞 CVE-2017-11428 的参考链接网络中（图4-2），TRACER 将直接选择补丁节点 ruby-saml@048a54 作为补丁结果之一。因为该节点是 GitHub 节点的子节点，具有较高的置信度。实际上，ruby-saml@048a54 也确实是正确的补丁之一。

### 4.3.2 基于连通度的补丁选择方法

仅仅使用基于置信度的启发式方法通常不足以完整地找出漏洞补丁集。因为 NVD 中很可能不包含补丁链接，且 CVE ID、Advisory ID 或 Issue ID 也很可能不在代码提交信息中，因此也无法通过搜索 GitHub 获取到补丁。考虑到漏洞的补丁会在与该漏洞相关的各种来源的漏洞公告、分析报告、讨论和解决的过程中被频繁提及和引用。这也意味着：正确的补丁节点将会广泛地连接到网络中的根节点（即图中的 CVE ID 节点）上。因此，本文还设计了基于连通度的补丁节点选择方法。

具体来说, TRACER 从两个角度衡量网络中补丁节点与根节点间的连通度。一是路径数, 从根节点到补丁节点的路径越多, 补丁节点与根节点的连通性就越高。二是路径长度, 从根节点到补丁节点的路径越短, 补丁节点到根节点的连通性就越高。为了结合这两个方面, TRACER 使用公式4.1计算补丁节点到根节点的连通度。其中,  $p = 1, \dots, n$  表示从根节点到补丁节点的  $n$  条路径,  $d_p$  表示路径  $p$  的长度。考虑到 NVD 和 GitHub 子节点的高置信度, 如果路径  $p$  源于 NVD 和 GitHub 节点, 则路径长度自动减 1。

$$connectivity = \sum_{p=1}^n \frac{1}{2^{(d_p-1)}} \quad (4.1)$$

基于每个补丁节点到根节点的连通度, TRACER 选择连通度最高的节点作为该漏洞的补丁, 加入补丁集。

**样例 4.7** 在图4-2中, 从根节点到补丁节点 ruby-saml@048a54 的路径有两条。一是源自 Debian 的路径, 长度为 2, 连通度为 0.5。另一条是源自 GitHub 的路径, 原始长度为 2, 调整后为 1, 连通度为 1。因此, ruby-saml@048a54 节点到根节点的总连通度为 1.5。同理, 从根节点到补丁节点 SAMLBase@482cdf 存在四条路径, 连通性分别为 0.5、0.25、0.25 和 0.125。因此, SAMLBase@482cdf 到根节点的连通度为 1.125, 低于 ruby-saml@048a54 节点, 所以 TRACER 选择 ruby-saml@048a54 作为补丁。

完成以上两种补丁选择方法后, TRACER 已从该漏洞的参考链接网络中选出补丁集。由于构建该网络的过程中可能没有识别到补丁链接, 所以该补丁集可能为空。同时, 基于以上两种补丁选择方法, TRACER 也有可能已获得该漏洞的多个补丁。

## 4.4 补丁扩增

受经验研究中映射分析结果的启发, 漏洞其补丁之间存在一对多的映射关系。章节3.6中, 映射分析结果表明, 超过40%的漏洞与其补丁具有一对多的映射关系, 且这些补丁通常是位于同一代码库的某一个或多个分支。对于这种多补丁的漏洞, TRACER 在前两个步骤中构建的网络无法确保能完整地包含所有补丁。针对这一情况, 基于步骤二中选出的补丁, 步骤三通过搜索同一代码库所有分支中的相关提交来扩展补丁集。此外, 补丁类型分析(见章节3.5)表明绝大多数的补丁都以 GitHub 代码提交的形式。因此, TRACER 的步骤三具体设计如下: 对于每个已在步骤二中选定的 GitHub 代码提交形式的补丁, TRACER 将首先获取其 GitHub 代码仓库信息, 并获取该仓库中的所有分支信息, 在每个分支的特定时间范围内搜索相关的代码提交。

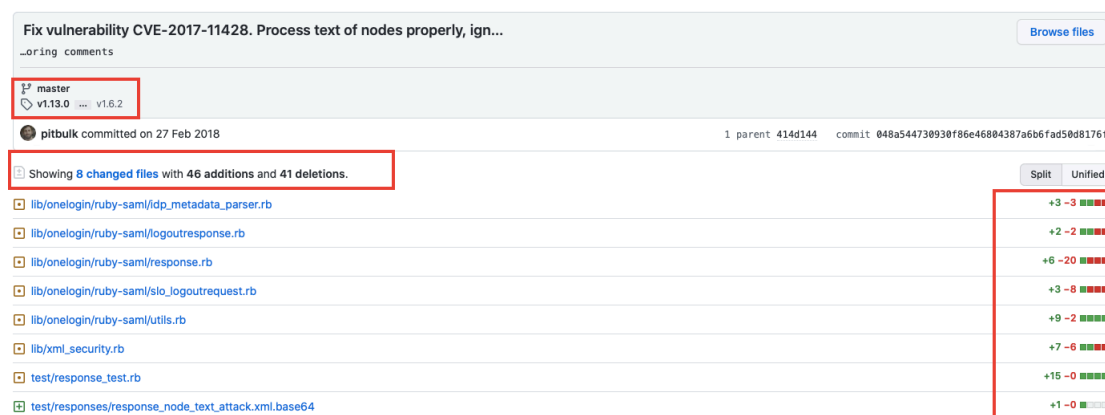


图 4-7 “ruby-saml@048a54” 代码提交

具体来说，对于 GitHub 代码提交形式的补丁，TRACER 从补丁链接中提取出仓库信息，即所有者（Owner）和存储库（Repository）信息。基于补丁提交的所有者和仓库名信息，TRACER 先使用 GitHub 的 REST API<sup>①</sup>获取存储库中的所有分支信息。对于每个分支，TRACER 再通过 GitHub 的 REST API<sup>②</sup>检索选定补丁之前和之后特定时间跨度内的提交，时间跨度默认为30 天。这里设置时间跨度，是考虑到工具时间性能和准确性之间的平衡。当项目历史较长且不设置时间限制时，TRACER 运行时长也会无限增长。

然后，对于 API 返回的特定时间跨度内的提交，TRACER 使用以下两个标准来确定该提交是否与选定补丁具有相关性：一是提交信息（Commit Message）与已选补丁的提交消息相同或是包含关系；二是提交消息包含 CVE ID、Advisory ID 或 Issue ID。如果检索返回的提交满足这两个条件之一，TRACER 会将该补丁扩展为选定补丁的子节点。

最后，TRACER 将返回步骤二中选定的补丁和步骤三中扩展的补丁作为补丁列表返回给用户。此外，TRACER 还返回该漏洞的参考链接网络，以便于工作人员追溯补丁的来源及关系。

**样例 4.8** 在步骤二中，TRACER 为漏洞 CVE-2017-11428 选出的补丁为 ruby-saml@048a54（位于主分支）。在步骤三中，TRACER 识别到了另外三个提交—ruby-saml@d7ce60<sup>③</sup>、ruby-saml@a35f72<sup>④</sup>和 ruby-saml@03af9e<sup>⑤</sup>。图 4-7、4-8、4-9和4-10所示，它们与选定补丁 ruby-saml@048a54 具有相同的提交信息却不同的代码更改。这三个提交分别位于分支 0.8.3–0.8.17、v0.9.3 和 v1.6.2 中。如图4-2所示，

① <https://docs.github.com/en/rest/reference/repos#list-branches>

② <https://docs.github.com/en/rest/reference/repos#list-commits>

③ <https://github.com/oneLogin/ruby-saml/commit/d7ce607d9f9d996e1046dde09b675c3cf0c01280>

④ <https://github.com/oneLogin/ruby-saml/commit/a35f7251b86aa3b7caf4a64d8a3451f925e8855c>

⑤ <https://github.com/oneLogin/ruby-saml/commit/03af9e33d2d87f4ac9a644c5b0981ded4dca0bb8>

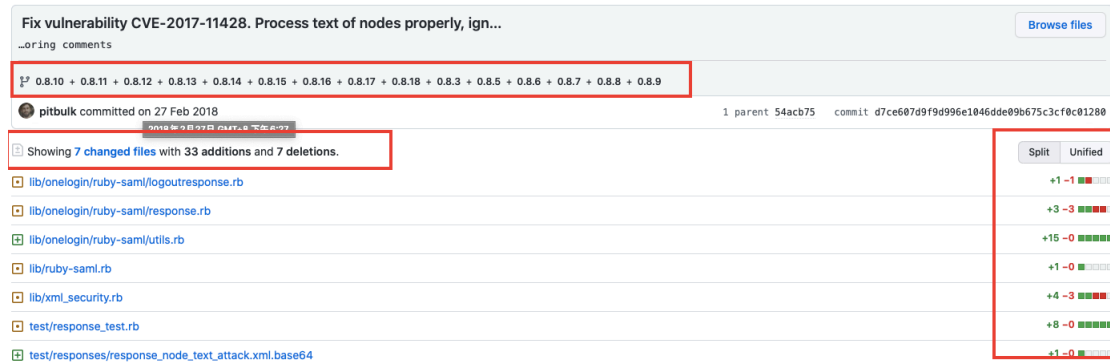


图 4-8 “ruby-saml@d7ce60” 代码提交

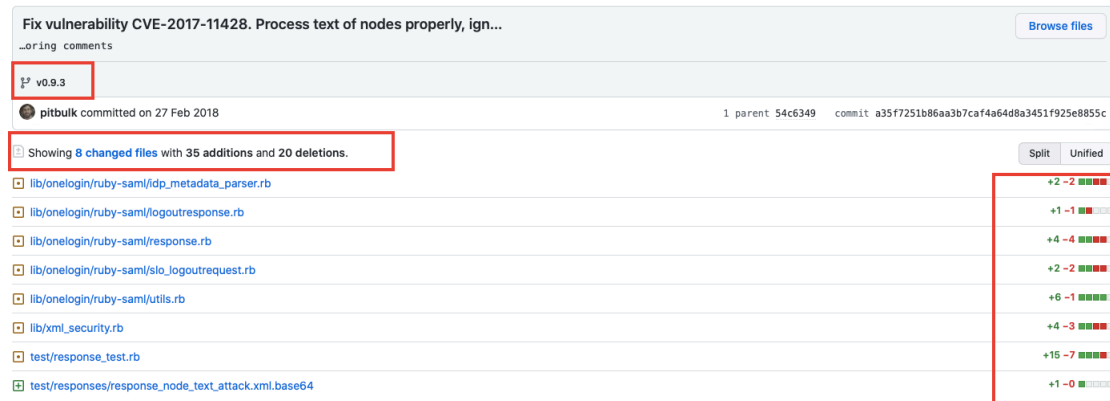


图 4-9 “ruby-saml@a35f72” 代码提交

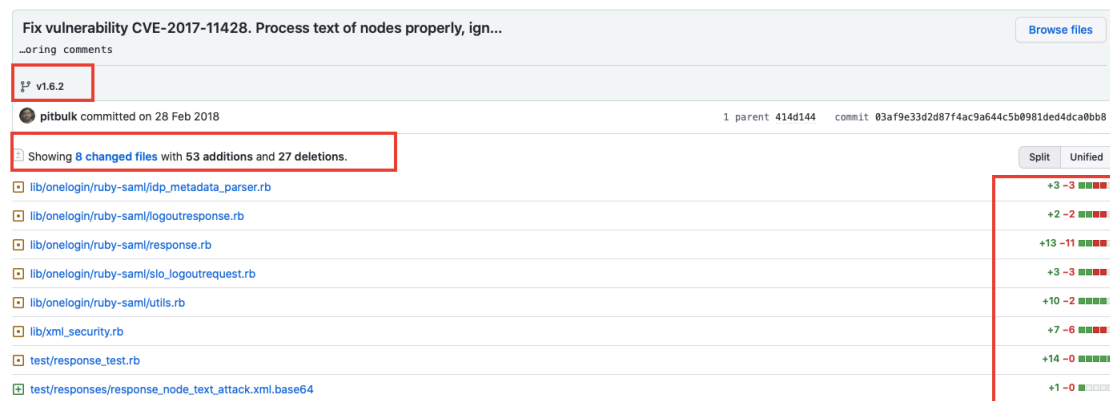


图 4-10 “ruby-saml@03af9e” 代码提交

TRACER 将三个提交添加为 ruby-saml@048a54 的子节点。实际上，这四个代码提交也确实都是 CVE-2017-11428 的正确补丁，而数据库  $DB_A$  和  $DB_B$  都只提供了 ruby-saml@048a54 一个补丁。

## 第 5 章 实验评估

本章将介绍针对 TRACER 所开展的实验评估，讨论并分析实验的结果。首先，本文从五个方面提出五个研究问题，包括准确性、削弱性、敏感度、通用性和实用性。然后，针对这五个问题再逐一评估，讨论并分析结果的原因及意义。最后，基于实验评估的结果，讨论 TRACER 的价值及局限性。

### 5.1 实验问题设计

本文从准确性、削弱性、敏感度、通用性及实用性五个方面设计了以下五个研究问题，以尽可能全面地评估 TRACER。

- **RQ6 准确性评估：**与已有的基于启发式规则的方法相比，TRACER 识别漏洞补丁的准确性如何？与两个商业漏洞数据库相比，TRACER 识别的漏洞补丁准确性又如何？这个研究问题旨在对比评估 TRACER 的准确性（见章节5.3）。
- **RQ7 削弱性分析：**TRACER 中各个步骤及环节的设计有着怎样的效果？或有着怎样的必要性？如果去除 TRACER 中的某些环节，或削弱 TRACER，对于最终结果会有怎样的影响？这个问题旨在评估 TRACER 中各个步骤及环节设计的实际效果及必要性（见章节5.4）。
- **RQ8 敏感度分析：**TRACER 对设计参数的敏感性如何？这个问题旨在评估 TRACER 的健壮性及鲁棒性，探究 TRACER 中的参数是否对结果的准确性有过大影响（见章节5.5）。
- **RQ9 通用性分析：**TRACER 在更大范围的开源软件漏洞上表现如何？这个问题旨在评估 TRACER 的通用性（见章节5.5）。
- **RQ10 实用性分析：**TRACER 在实际使用中表现如何？这个问题旨在评估 TRACER 在实际工作中实用性（见章节5.6）。

为解答以上研究问题，本章的实验评估将继续采用经验研究（见章节3.7）中的评估指标，即 Coverage（覆盖率）、Precision（精确率）、Recall（召回率）和 F1-Score（F1 值）。本章的实验评估还将继续使用在经验研究（见章节3）中构建的深度数据集，以探究 **RQ6 准确性评估**、**RQ7 削弱性分析**和 **RQ8 敏感度分析**。

对于 **RQ9 通用性分析**，本文将另外构造两个更大范围的数据集来进行评估。此外，本文还进行了用户研究，通过分析用户在有和没有 TRACER 辅助下查找补丁的用时和准确性，并基于用户反馈，来探究 **RQ10 实用性分析**。

References to Advisories, Solutions, and Tools

By selecting these links, you will be leaving NIST webspace. We have provided these links to other web sites because they may have information that would be of interest to you. No inferences should be drawn on account of other sites being referenced, or not, from this page. There may be other web sites that are more appropriate for your purpose. NIST does not necessarily endorse the views expressed, or concur with the facts presented on these sites. Further, NIST does not endorse any commercial products that may be mentioned on these sites. Please address comments about this page to [nvd@nist.gov](mailto:nvd@nist.gov).

Hyperlink	Resource
<a href="https://chartkick.com">https://chartkick.com</a>	Product
<a href="https://github.com/ankane/chartkick.js/issues/117">https://github.com/ankane/chartkick.js/issues/117</a>	Third Party Advisory
<a href="https://github.com/ankane/chartkick/blob/master/CHANGELOG.md">https://github.com/ankane/chartkick/blob/master/CHANGELOG.md</a>	Product Release Notes
<a href="https://github.com/ankane/chartkick/commit/b810936bbf687bc74c5b6dba72d2397a399885fa">https://github.com/ankane/chartkick/commit/b810936bbf687bc74c5b6dba72d2397a399885fa</a>	Patch
<a href="https://github.com/ankane/chartkick/commits/master">https://github.com/ankane/chartkick/commits/master</a>	Patch
<a href="https://rubygems.org/gems/chartkick/">https://rubygems.org/gems/chartkick/</a>	Product Release Notes

图 5-1 NVD 平台中漏洞 CVE-2019-18841 的参考链接

5.2 实验环境准备

本文作者使用 Python 3.6 搭建实验平台，并完成工具 TRACER 的开发，TRACER 共包括 6117 行代码。实验评估中，TRACER 运行在含有 16 核 Intel Xeon 型 CPU（2.10GHz）、62GB 内存的服务器上。基于从广度数据集中随机抽样的 200 个漏洞，运行结果表明，TRACER 在该实验环境下完成一次漏洞补丁定位的平均用时约为 22 分钟。

5.3 RQ6：准确性评估

为了评估 TRACER 的准确性，本节将 TRACER 分别与三种基于启发式规则的方法和两个商业漏洞数据库（DB<sub>A</sub> 和 DB<sub>B</sub>）进行比较，并进一步分析了 TRACER 中误报和漏报的原因。

5.3.1 与基于启发式规则的方法对比

本节首先选择了两种被广泛使用的基于启发式规则的方法（检索 NVD<sup>[4-5,25]</sup> 和检索 GitHub<sup>[1,33]</sup>），此外，将这前两种方法结合为第三种基于启发式规则的方法（检索 NVD 以及 GitHub）。

**检索 NVD:** 检索 NVD 平台上该漏洞“Reference”字段中引用的参考链接以获取补丁提交。如图5-1所示为 NVD 平台上 CVE-2019-18841 的参考链接<sup>①</sup>，可以从中提取出补丁提交“https://github.com/ankane/chartkick/commit/b810936bbf687bc74c5b6dba72d2397a399885fa”。

**检索 GitHub:** 检索 GitHub 中，带有漏洞标识符的代码提交。如图5-2所示，漏洞 CVE-2018-8014 的补丁提交信息<sup>②</sup>为“Fix https://bz.apache.org/bugzilla/sho

① <https://nvd.nist.gov/vuln/detail/CVE-2019-18841>  
② <https://github.com/apache/tomcat80/commit/2c9d8433bd3247a2856d4b255>



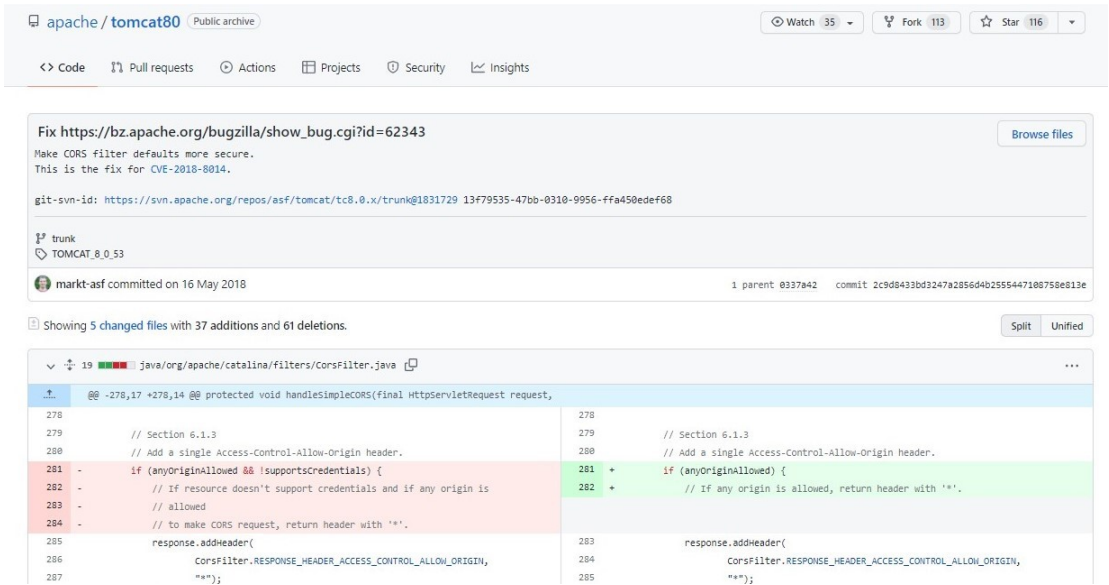


图 5-2 CVE-2018-8014 标识符出现在提交信息中

w\_bug.cgi?id=62343 Make CORS filter defaults more secure. This is the fix for CVE-2018-8014.”，漏洞标识符“CVE-2018-8014”出现在该提交信息中。以“CVE-2018-8014”为输入，通过检索 GitHub 即可获取该补丁提交。

**检索 NVD 以及 GitHub：**将前两种方法结合为第三种基于启发式规则的方法，即检索 NVD 中的参考链接以及 GitHub 中的代码提交。

表5-1显示了三种基于启发式规则的方法与 TRACER 的准确率对比结果。

**补丁覆盖率方面：**可以发现，在包含 1,295 个漏洞的深度数据集上，三种基于启发式规则方法的补丁覆盖率都较低。对于基于检索 NVD 的启发式方法，768（59.3%）的漏洞无法找到补丁，补丁覆盖率为 40.7%。对于基于检索 GitHub 的启发式方法，990（76.4%）的漏洞无法找到补丁，补丁覆盖率为 23.6%。对于基于检索 NVD 以及 GitHub 的启发式方法，576（44.5%）的漏洞无法找到补丁，补丁覆盖率为 55.5%。然而，仅155（12.0%）漏洞的补丁无法被 TRACER 找到，TRACER 的补丁覆盖率为 88.0%。

**补丁准确性方面：**基于检索 NVD 的启发式方法具有较高的精确率（0.970），这是因为 NVD 平台的信息都经过人工核对，参考链接的置信度比较高。但对于一对多映射关系的漏洞，基于检索 NVD 的启发式方法召回率就比较低，分别为0.552、0.416 和 0.708。这也反映出 NVD 平台漏洞补丁的不完整情况。与基于检索 NVD 的启发式方法对比，TRACER 的精确率更低，但拥有更高的召回率和相似的 F1 值. 尤其是对于 *MP* 和 *MB* 类型的漏洞，TRACER 的召回率也明显更高。考虑到 TRACER 的补丁覆盖率比基于检索 NVD 的启发式方法多出116.3%，TRACER 中轻微的准确率降低是可以接受的。

基于检索 GitHub 的方法具有较低的精确率和召回率，分别为0.461和0.417，



表 5-1 TRACER VS. 基于启发式规则的方法和商业数据库

映射类型	数量	TRACER				检索 NVD			
		Coverage	Pre.	Rec.	F1	Coverage	Pre.	Rec.	F1
1:1 (SP)	567	465 (82.0%)	0.860	0.951	0.881	282 (49.7%)	0.973	0.986	0.977
1:i (MEP)	195	189 (96.9%)	0.886	0.918	0.888	70 (35.9%)	0.932	0.925	0.921
1:n (MP)	101	81 (80.2%)	0.872	0.741	0.761	33 (32.7%)	0.980	0.552	0.683
1:n (MB)	372	349 (93.8%)	0.861	0.788	0.795	148 (39.8%)	0.979	0.416	0.546
1:n (MR)	60	56 (93.3%)	0.831	0.620	0.659	14 (23.3%)	1.000	0.708	0.794
总计	1,295	1,140 (88.0%)	0.864	0.864	0.837	527 (40.7%)	0.970	0.805	0.842
映射类型	数量	检索 GitHub				检索 NVD 以及 GitHub			
		Coverage	Pre.	Rec.	F1	Coverage	Pre.	Rec.	F1
1:1 (SP)	567	95 (16.8%)	0.416	0.642	0.471	345 (60.8%)	0.839	0.930	0.864
1:i (MEP)	195	33 (16.9%)	0.472	0.490	0.452	91 (46.7%)	0.821	0.867	0.820
1:n (MP)	101	28 (27.8%)	0.536	0.445	0.461	49 (48.5%)	0.779	0.605	0.647
1:n (MB)	372	126 (33.9%)	0.445	0.236	0.284	201 (54.0%)	0.704	0.393	0.465
1:n (MR)	60	23 (38.3%)	0.627	0.345	0.413	33 (55.0%)	0.801	0.539	0.604
总计	1,295	305 (23.6%)	0.461	0.417	0.386	719 (55.5%)	0.793	0.732	0.720
映射类型	数量	$DB_A$				$DB_B$			
		Coverage	Pre.	Rec.	F1	Coverage	Pre.	Rec.	F1
1:1 (SP)	567	100.0%	0.908	0.915	0.910	100.0%	0.900	0.921	0.906
1:i (MEP)	195	100.0%	0.935	0.898	0.902	100.0%	0.924	0.909	0.906
1:n (MP)	101	100.0%	0.923	0.483	0.616	100.0%	0.911	0.520	0.638
1:n (MB)	372	100.0%	0.941	0.510	0.620	100.0%	0.932	0.436	0.555
1:n (MR)	60	100.0%	0.913	0.610	0.695	100.0%	0.964	0.526	0.636
总计	1,295	100.0%	0.923	0.748	0.793	100.0%	0.917	0.730	0.771

而基于检索 NVD 以及 GitHub 的方法中和了前两种方法，精确率和召回率分别为0.793和0.732。与基于检索 GitHub 的启发式方法和基于检索 NVD 以及 GitHub 的启发式方法对比，TRACER 在精确率、召回率和 F1 值上全面胜出。TRACER 的 F1 值分别高出了116.8%和16.3%。

综上，与现有的基于启发式规则的方法相比，TRACER 能够显著的提高补丁覆盖率和 F1 值。TRACER 将补丁覆盖率提高58.6%到273.8%，同时，将 F1 值提高116.8%。

### 5.3.2 与商业漏洞数据库对比

考虑到漏洞数据库  $DB_A$  和  $DB_B$  并非由工具自动化构建，构建的过程涉及了大量人工工作。因此，漏洞数据库  $DB_A$  和  $DB_B$  中漏洞知识的质量较高，自

自动化补丁识别工具难以全面超越。本文将 TRACER 与数据库  $DB_A$  和  $DB_B$  进行对比, 可以对比评估 TRACER 所达到的准确性水平, 并探究 TRACER 用于改进或补充现有的商业漏洞数据库的可能性。

从表5-1可以看出, TRACER 能找到补丁的漏洞数比  $DB_A$  和  $DB_B$  少了12.0%, 补丁覆盖率仅为 88.0%, 且精确率也分别低了6.4%和5.8%。这是因为漏洞数据库  $DB_A$  和  $DB_B$  构建过程中涉及了安全专家的人工工作, 许多补丁由人工收集得来, 难以被自动化工具找到。此外, 收集的补丁还会经过安全专家验证, 因此  $DB_A$  和  $DB_B$  具有比自动化工具更高的准确率。

此外, TRACER 的精确率、召回率和 F1 值分别为 0.864、0.864 和 0.837,  $DB_A$  的精确率、召回率和 F1 值分别为 0.923、0.748 和 0.793,  $DB_B$  的精确率、召回率和 F1 值分别为 0.917、0.730 和 0.771。可以发现 TRACER 的召回率比  $DB_A$  和  $DB_B$  高15.5% 和 18.4%, 尤其是对于一对多映射类型的漏洞, TRACER 具有更高的召回率。TRACER 的 F1 值也比  $DB_A$  和  $DB_B$  高5.5% 和 8.6%。

这些结果表明, 与漏洞数据库  $DB_A$  和  $DB_B$  相比, TRACER 以略低的补丁精确率和覆盖率为代价, 拥有更为显著的召回率。这也说明, TRACER 可用于补充现有漏洞数据库缺失的漏洞补丁数据。

### 5.3.3 漏报和误报分析

基于前两节的实验结果, 本小节主要分析造成 TRACER 漏报和误报的原因, 即 TRACER 未找到补丁或找到的补丁不正确的原因。

**漏报分析:** 漏报是指 TRACER 未能找到补丁或找全补丁, 体现在补丁覆盖率 (Coverage) 和召回率 (Recall) 小于 1。通过人工分析 TRACER 未能找到补丁或找全补丁的漏洞数据, 本文共总结出造成 TRACER 漏报的五个主要原因:

- (1) 对于一些年代久远的漏洞, NVD、Debian 和 Red Hat 中包含的参考链接比较少, 有的参考链接甚至是失效的。这种情况下, TRACER 无法构建完整的参考链接网络。例如, 对于 TRACER 未能找到补丁的漏洞 CVE-2011-1950, 该漏洞有一个关键的参考链接在 NVD<sup>①</sup>中标记为“补丁 (Patch)”和“供应商咨询 (Vendor Advisory)”。这个链接中很有可能含有补丁, 但网址已失效。
- (2) NVD、Debian 和 Red Hat 平台缺失关键参考链接 (如, 问题链接), TRACER 便难以选出正确的补丁。例如, 对于 TRACER 未能找到补丁的漏洞 CVE-2018-14642, 该漏洞的问题报告 UNDERTOW-1430<sup>②</sup>不包含在任何一个知

① <https://nvd.nist.gov/vuln/detail/CVE-2011-1950>

② <https://issues.redhat.com/browse/UNDERTOW-1430>

识源中。人工分析发现，基于这个问题报告可以找到该漏洞的补丁<sup>①</sup>。

- (3) 补丁的提交信息不包含漏洞标识符，但与漏洞描述具有语义相似性。这种情况人工可识别，但自动化工具难以识别。因此，TRACER 的参考链接扩增步骤也未能找到这类型的补丁。例如，漏洞 CVE-2019-10077<sup>②</sup>，代码提交 87c89f@jspwiki<sup>③</sup>为该漏洞的补丁提交，但提交信息中未提及漏洞标识符，TRACER 也未能找到该补丁。
- (4) GitHub 平台提供的用于检索代码提交的 REST API 仅返回前 1,000 个结果。如果补丁提交排在 1000 个以后的话，会使 TRACER 在参考链接扩增步骤中错失一些补丁提交。
- (5) 在补丁选择步骤中，TRACER 只选择了具有最高连通度的补丁节点。当其他已包含在参考链接网络中的正确补丁非最高连通度时，会被 TRACER 遗漏掉，造成漏报。

**误报分析：**误报是指 TRACER 识别的结果非该漏洞的补丁，体现在精确率（Precision）小于 1。通过人工分析 TRACER 返回错误补丁的漏洞数据，本文节总结出造成 TRACER 误报的两个主要原因：

- (1) 在讨论和解决漏洞时，相关人员会引用引入漏洞的代码提交（Vulnerability-Inducing Commit）。由于 TRACER 缺乏对参考链接上下文的语义理解，TRACER 会错误地将引入漏洞的代码提交识别为补丁提交。例如，对于漏洞 CVE-2020-5249，引入漏洞的代码提交<sup>④</sup>和修复漏洞的代码提交<sup>⑤</sup>在同一问题报告的评论中被引用，造成了 TRACER 的误报。
- (2) 参考链接的页面上列出了多对漏洞及其问题和补丁链接。由于 TRACER 缺乏语义理解能力，其他漏洞的补丁也可能被 TRACER 错误地识别。例如，漏洞 CVE-2018-15750 的补丁和 CVE-2018-15751 的补丁都维护在发行说明（Release Note）<sup>⑥</sup>中，该发行说明还都被 NVD、Debian 和 Red Hat 引用，造成了 TRACER 的误报。

综上，通过人工分析共总结出 TRACER 漏报的五个主要原因和误报的两个主要原因。

① <https://github.com/undertow-io/undertow/commit/c46b7b49c5a561731c84a76ee52244369af1af8a>

② <https://nvd.nist.gov/vuln/detail/CVE-2019-10077>

③ <https://github.com/apache/jspwiki/commit/87c89f0405d6b31fc165358ce5d5bc4536e32a8a>

④ <https://github.com/ethereum/go-ethereum/commit/fb9f7261ec51e38eedb454594fc19f00de1a6834>

⑤ <https://github.com/ethereum/go-ethereum/commit/83e2761c3a13524bd5d6597ac08994488cf872ef>

⑥ <https://docs.saltstack.com/en/latest/topics/releases/2018.3.3.html>

## 5.4 RQ7：削弱性分析

为了探究 TRACER 中各个步骤及环节设计的实际效果及必要性，本小节通过对比 TRACER 及其削弱部分环节后生成的变体的准确率，以量化分析各步骤及环节的实际效果。

针对步骤一——参考链接网络构建，本小节分别进行了“去除某一知识源”和“去除网络构建步骤”的削弱处理。针对步骤二——补丁选择，本小节分别进行了“去除基于连通度和置信度的补丁选择方法”、“去除基于连通度或置信度的补丁选择方法”和“变更基于连通度的方法设置”的削弱处理。针对步骤三——补丁扩增，本小节进行了“去除补丁扩增步骤”的削弱处理。表5-2和5-3为削弱性分析的结果。

### 5.4.1 去除某一知识源

在 TRACER 的步骤一（参考链接网络构建）中，通过分别删除四个知识源 NVD、Debian、Red Hat 和 GitHub 中的一个，生成削弱后的变体  $v_1^1$ 、 $v_1^2$ 、 $v_1^3$  和  $v_1^4$ 。如表5-2所示，这四个变体的补丁覆盖率都小于 TRACER。在精确率、召回率和 F1 值相当的情况下， $v_1^1$ 、 $v_1^2$ 、 $v_1^3$  和  $v_1^4$  未找到补丁的漏洞数分别为 498（38.5%）、173（13.4%）、180（13.9%）和 242（18.7%），比 TRACER 分别多了30.1%、1.6%、2.2%和 7.6%。这表明，集成更多的知识源，构建更完整的参考链接网络，有助于取得更好的补丁识别结果。TRACER 中选取的四个知识源都有助于构建更完整的参考链接网络，也更有助于为更多的漏洞找到补丁。同时，可以发现，删除知识源 NVD 和 GitHub 后未找到补丁的漏洞数增到了 498（38.5%）和 242（18.7%），补丁覆盖率减少到了 61.5% 和 81.3%。这也说明知识源 NVD 和 GitHub 的贡献度是最高的，效果也是最好的。

### 5.4.2 去除网络构建步骤

在 TRACER 的步骤一（参考链接网络构建）中，通过不以逐层迭代的方式构建参考链接网络，而是仅仅使用包含在四个知识源公告中直接引用的参考链接，生成削弱后的变体  $v_1^5$ 。具体实现为，在 TRACER 的步骤一中跳过“参考链接分析”环节，实验结果为表5-2中的  $v_1^5$ 。可以发现， $v_1^5$  没有找到补丁的漏洞数量增加到了 414（32.0%），补丁覆盖率下降到了 68.0%。但同时精确率却高出了6.3%，召回率低了6.0%，F1 值较为相似。这些结果表明，补丁并不总是在 NVD、Debian 和 Red Hat 中被直接引用，而是可能隐藏在多层参考链接之中，去除网络中深层参考链接后，会丢失补丁。同时说明，本文构建的参考链接网络有助于用户找到隐藏的补丁，但需要以小幅的精确率降低为代价。

表 5-2 TRACER 削弱性分析结果 (1)

映射类型	数量	TRACER				$v_1^1$ : TRACER w/o NVD			
		Coverage	Pre.	Rec.	F1	Coverage	Pre.	Rec.	F1
1:1 (SP)	567	465 (82.0%)	0.860	0.951	0.881	281 (49.6%)	0.820	0.936	0.846
1:i (MEP)	195	189 (96.9%)	0.886	0.918	0.888	116 (59.5%)	0.882	0.935	0.886
1:n (MP)	101	81 (80.2%)	0.872	0.741	0.761	60 (59.4%)	0.881	0.728	0.766
1:n (MB)	372	349 (93.8%)	0.861	0.788	0.795	288 (77.4%)	0.876	0.780	0.800
1:n (MR)	60	56 (93.3%)	0.831	0.620	0.659	52 (86.7%)	0.848	0.551	0.624
总计	1,295	1,140 (88.0%)	0.864	0.864	0.837	797 (61.5%)	0.856	0.839	0.815
映射类型	数量	$v_1^2$ : TRACER w/o Debian				$v_1^3$ : TRACER w/o Red Hat			
		Coverage	Pre.	Rec.	F1	Coverage	Pre.	Rec.	F1
1:1 (SP)	567	457 (80.6%)	0.847	0.943	0.869	454 (80.1%)	0.853	0.943	0.874
1:i (MEP)	195	187 (95.6%)	0.880	0.912	0.882	188 (96.4%)	0.883	0.918	0.886
1:n (MP)	101	79 (78.2%)	0.851	0.716	0.739	80 (79.2%)	0.880	0.736	0.760
1:n (MB)	372	344 (92.5%)	0.838	0.760	0.771	337 (90.6%)	0.844	0.761	0.767
1:n (MR)	60	55 (91.7%)	0.819	0.613	0.651	56 (93.3%)	0.738	0.640	0.618
总计	1,295	1,122 (86.6%)	0.848	0.849	0.821	1,115 (86.1%)	0.851	0.853	0.823
映射类型	数量	$v_1^4$ : TRACER w/o GitHub				$v_1^5$ : TRACER w/o Network			
		Coverage	Pre.	Rec.	F1	Coverage	Pre.	Rec.	F1
1:1 (SP)	567	418 (73.7%)	0.898	0.943	0.908	390 (68.8%)	0.910	0.972	0.925
1:i (MEP)	195	176 (90.3%)	0.887	0.921	0.892	117 (60.0%)	0.956	0.959	0.941
1:n (MP)	101	73 (72.3%)	0.873	0.690	0.726	61 (60.4%)	0.943	0.669	0.743
1:n (MB)	372	333 (89.5%)	0.874	0.752	0.773	263 (70.7%)	0.908	0.575	0.659
1:n (MR)	60	53 (88.3%)	0.816	0.545	0.604	50 (83.3%)	0.920	0.641	0.712
总计	1,295	1,053 (81.3%)	0.883	0.841	0.835	881 (68.0%)	0.918	0.812	0.823

### 5.4.3 去除补丁选择步骤

**去除基于连通度和置信度的补丁选择方法:** 在 TRACER 的步骤二 (补丁选择) 中, 通过选择网络中的所有补丁, 而非选择具有高置信度的补丁和具有高连通度的补丁, 生成的变体  $v_2^1$  (见表5-3)。可以发现, 该变体显著提高了 TRACER 的召回率, 由 0.864 提高到了 0.940 多了8.8%, 尤其是对于一对多映射类型的漏洞。但同时精确率大幅下降29.3%, 由 0.864 降到了 0.611; F1 值也大幅下降了21.4%, 由 0.837 降到了 0.658。该结果表明, 参考链接网络中确实包含了大部分漏洞的正确补丁, 且基于连通度和置信度的补丁选择方法有助于实现精确率和召回率

表 5-3 TRACER 削弱性分析结果 (2)

映射类型	数量	$v_2^1$ : TRACER w/o Selection				$v_2^2$ : TRACER w/o Connectivity			
		Coverage	Pre.	Rec.	F1	Coverage	Pre.	Rec.	F1
1:1 (SP)	567	465 (82.0%)	0.632	0.961	0.680	322 (56.8%)	0.892	0.978	0.913
1:i (MEP)	195	189 (96.9%)	0.622	0.976	0.682	84 (43.1%)	0.929	0.939	0.915
1:n (MP)	101	81 (80.2%)	0.615	0.933	0.656	45 (44.6%)	0.953	0.685	0.764
1:n (MB)	372	349 (93.8%)	0.616	0.903	0.657	181 (48.7%)	0.927	0.787	0.821
1:n (MR)	60	56 (93.3%)	0.368	0.891	0.394	33 (55.0%)	0.885	0.722	0.772
总计	1,295	1,140 (88.0%)	0.611	0.940	0.658	665 (51.4%)	0.910	0.889	0.871
映射类型	数量	$v_2^3$ : TRACER w/o Confidence				$v_2^4$ : TRACER with Path Length			
		Coverage	Pre.	Rec.	F1	Coverage	Pre.	Rec.	F1
1:1 (SP)	567	465 (82.0%)	0.860	0.942	0.879	465 (82.0%)	0.833	0.957	0.859
1:i (MEP)	195	189 (96.9%)	0.888	0.913	0.889	189 (96.9%)	0.848	0.945	0.867
1:n (MP)	101	81 (80.2%)	0.880	0.722	0.751	81 (80.2%)	0.849	0.760	0.742
1:n (MB)	372	349 (93.8%)	0.871	0.765	0.784	349 (93.8%)	0.830	0.798	0.770
1:n (MR)	60	56 (93.3%)	0.849	0.462	0.550	56 (93.3%)	0.652	0.747	0.590
总计	1,295	1,140 (88.0%)	0.869	0.844	0.826	1,140 (88.0%)	0.827	0.882	0.812
映射类型	数量	$v_2^5$ : TRACER with Path Number				$v_3$ : TRACER w/o Expansion			
		Coverage	Pre.	Rec.	F1	Coverage	Pre.	Rec.	F1
1:1 (SP)	567	465 (82.0%)	0.805	0.951	0.837	465 (82.0%)	0.871	0.948	0.889
1:i (MEP)	195	189 (96.9%)	0.849	0.920	0.858	189 (96.9%)	0.910	0.914	0.902
1:n (MP)	101	81 (80.2%)	0.801	0.756	0.726	81 (80.2%)	0.873	0.696	0.732
1:n (MB)	372	349 (93.8%)	0.833	0.811	0.791	349 (93.8%)	0.860	0.506	0.590
1:n (MR)	60	56 (93.3%)	0.789	0.630	0.644	56 (93.3%)	0.847	0.567	0.629
总计	1,295	1,140 (88.0%)	0.819	0.873	0.809	1,140 (88.0%)	0.873	0.771	0.776

之间的平衡。

**去除基于连通度或置信度的补丁选择方法：**在 TRACER 的步骤二（补丁选择）中，并非选择具有高置信度和连通度的补丁，而是采用两个补丁选择方法之一，即选择具有高置信度或连通度的补丁，生成两个变体  $v_2^2$  和  $v_2^3$ 。

在不使用基于连通度的补丁选择方法时， $v_2^2$  找到补丁的漏洞数比 TRACER 少了 41.7%，覆盖率下降到了 51.4%，但同时精确率提升了 5.3%，召回率提高了 2.9%，F1 值提高了 4.1%。这些结果表明，基于连通度的补丁选择方法有助于为更多漏洞找到补丁，但同时也会引入少部分噪声。

在不使用基于置信度的补丁选择方法时， $v_2^3$  的召回率和 F1 值会略有下降，

分别由 0.864 和 0.837 降到了 0.844 和 0.826；尤其是对于 *MR* 类型的漏洞，影响更为明显，召回率由 0.620 下降到 0.462。这些结果表明，基于置信度的补丁选择方法有助于为漏洞找到更多的补丁，且几乎不会引入少噪声。

**变更基于连通度的方法设置：**在 TRACER 的步骤二（补丁选择）基于连通度的补丁选择方法中，通过仅考虑路径长度（即选择到根节点的路径最短的补丁）或仅考虑路径数量（即选择具有最多路径数的补丁）来削弱基于连通度的补丁选择方法，分别生成了变体  $v_2^4$  和  $v_2^5$ 。

可以发现， $v_2^4$  和  $v_2^5$  的精确率分别比 TRACER 低了 4.3% 和 5.2%，召回率分别高了 2.1% 和 1.0%，F1 值分别降低了 3.0% 和 3.3%。这些结果表明，基于连通度的补丁选择方法中所设计的补丁节点到根节点的路径长度和路径数都有助于提高 TRACER 的准确率，但同时会使召回率略微降低。

#### 5.4.4 去除补丁扩增步骤

去除 TRACER 的步骤三（补丁扩增）后，生成变体  $v_3$ 。可以发现， $v_3$  的召回率为 0.771，比 TRACER 的召回率下降了 10.8%。 $v_3$  的 F1 值为 0.776，比 TRACER 的 F1 值下降了 7.3%。尤其是对于一对多映射类型的漏洞，召回率和 F1 值下降更为显著。这些结果表明，补丁扩增步骤有助于 TRACER 更完整地漏洞找到多个补丁。

综上，TRACER 中的知识源、网络构建、补丁选择和补丁扩增步骤的设计对最终结果都有一定的贡献度和必要性。

### 5.5 RQ8：敏感度分析

本小节将评估 TRACER 的准确性对参数的敏感性，及 TRACER 中参数设置合理性。TRACER 中有两个可配置参数，分别是：TRACER 步骤一网络构建时网络深度限制的设置，以及步骤三补丁扩增时的相关代码提交时间跨度的设定。

在评估 **RQ6 准确性评估**、**RQ7 削弱性分析**、**RQ9 通用性分析** 和 **RQ10 实用性性能分析** 时，网络深度限制默认设置为 5 层，代码提交时间跨度默认设置为 30 天。为了评估 TRACER 对这两个参数的敏感性，该小节将固定其中一个参数为默认设置，改变另一个参数的设置，并在深度数据集上重新运行 TRACER。网络深度限制具体设置为 3、4、5 和 6 层，提交时间跨度具体设置为 0、10、20、30、40、50 和 60 天。

图 5-3 和 5-4 分别显示了两个参数对 TRACER 准确性的影响，其中横坐标表示参数的取值，纵坐标表示 TRACER 的精确率。总体来看，随着网络层数的增加，网络中将包含更多补丁。此时，TRACER 覆盖率在上升，精确率在降低，而召回率和 F1 值先增加后降低。基于图 5-3 中数据，可以认为网络深度限制设置为 5 层时，

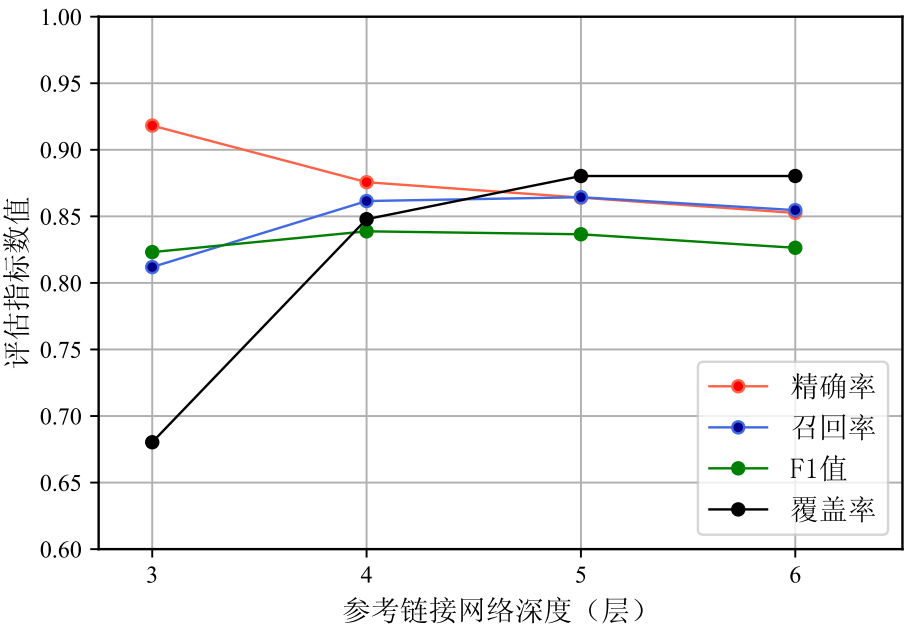


图 5-3 网络深度限制（层数）敏感性分析结果

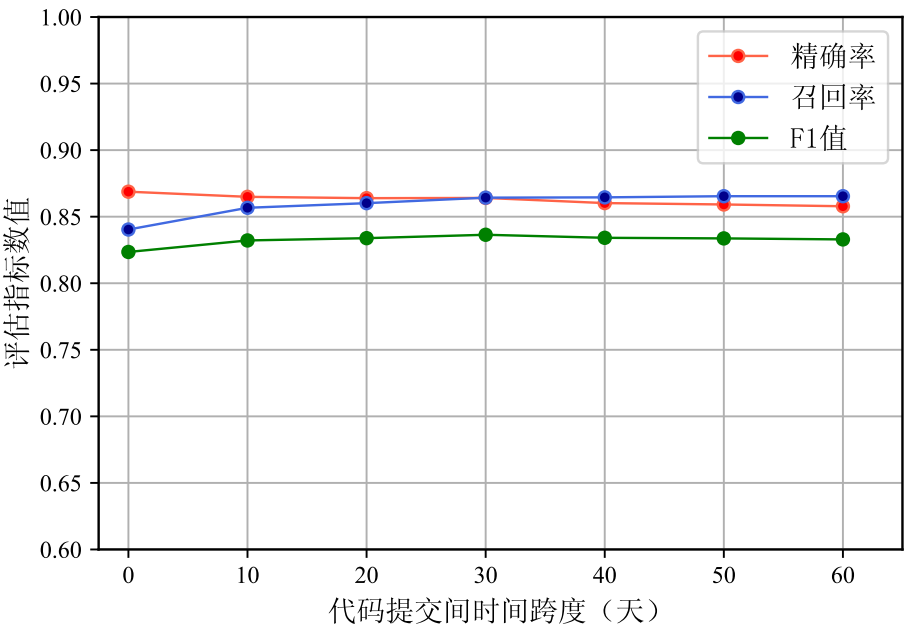


图 5-4 提交时间跨度的敏感性分析结果

TRACER 效果最优。

随着代码提交时间跨度的增加，TRACER 会搜索到更广的代码提交。TRACER 的精确率也会降低，召回率会提高，F1 值先上升后下降。其中，TRACER 未找到补丁的漏洞数量并不改变，即补丁覆盖率并未改变，所以没有在图5-4中显示。基于图5-4中数据，可以认为提交时间跨度设置为 30 天时，TRACER 效果最优。

综上，TRACER 的准确率对两个可配置参数的变化不是非常敏感，且网络深



表 5-4 TRACER 通用性分析结果

评估对象	数据集一（91 个漏洞）				数据集二（89 个漏洞）			
	Coverage	Pre.	Rec.	F1	Coverage	Pre.	Rec.	F1
TRACER	100.0%	0.823	0.845	0.784	100.0%	0.888	0.899	0.867
$DB_A$	62 (68.1%)	0.935	0.827	0.858	0.0%	—	—	—
$DB_B$	29 (31.8%)	0.885	0.664	0.725	0.0%	—	—	—

度参数设置为 5 层、提交时间跨度设置为 30 天时效果相对最优。

## 5.6 RQ9：通用性分析

为了评估 TRACER 在更大范围开源软件漏洞上的表现（即 TRACER 的通用性），本节还另外构建了两个漏洞数据集。第一个数据集为，漏洞数据库  $DB_A$  和  $DB_B$  中只有一个数据库提供补丁的漏洞（图3-3b中的橙色和蓝色部分），共有3,185个漏洞。第二个数据集为：两个漏洞数据库  $DB_A$  和  $DB_B$  中都没有提供补丁的漏洞（图3-3b在3-3a中的补集），共有5,468个漏洞。

在第一个包含3,185个漏洞的数据集上，TRACER 找到了2,155（67.7%）漏洞的补丁（补丁覆盖率为 67.7%）；而两个漏洞数据库  $DB_A$  和  $DB_B$  仅提供了2,190（68.8%）和995（31.2%）漏洞的补丁。在 TRACER 找到补丁的2,155个漏洞中， $DB_A$  和  $DB_B$  仅提供了其中的1,455和700个漏洞的补丁。对于第二个数据集中的5,468个漏洞，TRACER 找到了2,816（51.5%）漏洞的补丁（补丁覆盖率为 51.5%）；而两个漏洞数据库  $DB_A$  和  $DB_B$  没有提供补丁，补丁覆盖率为 0.0%。

此外，本节还从 TRACER 在第一个和第二个数据集上能找到补丁的漏洞中分别采样了100个。采用“数据准备”（见章节3.2）中同样的方式人工找到它们的补丁以评估 TRACER 的准确性。由于公开的信息有限，人工分析的两份数据集中分别有9和11个漏洞不能确定其补丁。

表5-4为评估结果，在第一个数据集取样的91个漏洞中，TRACER 的 F1 值为0.784， $DB_A$  的 F1 值较高为 0.858， $DB_B$  的 F1 值较低为 0.725。与小节5.3中的结果类似，漏洞数据库  $DB_A$  和  $DB_B$  都拥有更高的精确率（0.935 和 0.885），但召回率也更低（0.827 和 0.664）。在从第二个数据集取样的89个漏洞中， $DB_A$  和  $DB_B$  中没有任何漏洞的补丁，而 TRACER 可取得达到0.867的 F1 值，0.888 的精确率和 0.899 的召回率。

这些结果表明，即使对于更大范围的开源软件漏洞，TRACER 的准确率也基本稳定。在开源软件漏洞的补丁定位方面，TRACER 具有较好的通用性。尤其在第二个数据集上的结果，在漏洞数据库  $DB_A$  和  $DB_B$  都没有补丁时，TRACER 仍可以找到大量漏洞的补丁。这表明，TRACER 可以极大地补充或增强现有的商业

表 5-5 用户研究中任务的用时和准确率

任务	w/o TRACER				with TRACER			
	用时	Pre.	Rec.	F1	用时	Pre.	Rec.	F1
全部 10 个任务	5.66 mins	0.880	0.677	0.765	4.66 mins	0.983	0.920	0.951
5 单补丁任务	5.60 mins	0.960	0.960	0.960	3.84 mins	1.000	1.000	1.000
5 多补丁任务	5.72 mins	0.800	0.393	0.527	5.48 mins	0.967	0.840	0.899

漏洞数据库。

5.7 RQ10: 实用性分析

本小节旨在评估 TRACER 在实际工作场景下的实用性。在实际工作中，安全专家在使用较准确的自动工具识别到补丁后，他们仍然需要对补丁进行多次确认，以确保补丁的准确性。为了评估 TRACER 在这种使用场景中的实用性，本文开展了一项用户研究，对比分析用户在有和没有 TRACER 的辅助下找到 10 个漏洞补丁的用时和准确性。

本文作者从国内外多所大学和科技公司的安全实验室中共招募了 10 名实验人员，他们中有软件安全方向的博士后、博士生、硕士研究生人员以及工程师。本文还从深度数据集中随机选择了 10 个漏洞作为实验任务。其中，2 个漏洞属于 *SP* 类型，3 个漏洞属于 *MEP* 类型，1 个漏洞属于 *MP* 类型，4 个漏洞属于 *MB* 类型。为了公平比较，实验人员被平均分为 A、B 两组。A 组人员需要在不使用 TRACER 的情况下完成前五项任务，并使用 TRACER 完成剩余的任务。反之，B 组人员需要在不使用 TRACER 的情况下完成前 5 个任务，并使用 TRACER 完成剩余的任务。这样可以实现公平比较，在有和没有 TRACER 时用户完成 10 项漏洞补丁查找任务的用时和准确性。

表5-5显示了 10 个任务的平均时间消耗和补丁的准确性。本节将属于 *SP* 和 *MEP* 类型的 5 个漏洞归类为单补丁任务，将属于 *MP* 和 *MB* 类型的 5 个漏洞归类为多补丁任务。总体而言，对于全部 10 个任务，在 TRACER 的帮助下，实验人员完成每个任务的用时较少（平均 4.66 分钟），比在没有 TRACER 时少用了 17.7% 时间。在 TRACER 的帮助下，补丁的精确率为 0.983，召回率为 0.920，F1 值为 0.951。对比没有 TRACER 时，分别提高了 11.7%，35.9% 和 24.3%。

**用时方面：**对于 5 个单补丁任务，在 TRACER 的帮助下，实验人员用时显然更短；但对于 5 个多补丁任务，实验人员用时基本持平。这是因为 TRACER 为多补丁任务返回的参考链接网络比单补丁任务的更复杂，实验人员需要花费更多的时间理解参考链接网络，并验证其中的补丁。

**准确性方面：**在 TRACER 的帮助下，5 个多补丁任务的准确度提高也较为显

着,但对于5个单补丁任务则并不显著。这是因为,单补丁任务相比多补丁任务较为简单,实验人员无需 TRACER 的帮助即可完成补丁识别。这也表明,TRACER 对于具有多个补丁的开源漏洞作用更为明显。

本文作者还采访了每位实验人员,以获取实验人员对 TRACER 使用感受的评价。总体来说,实验人员都认为漏洞的参考链接网络具有较高的价值,因为它囊括了来自多个知识源的信息,其中不同类型的节点和关系有助于人工定位和验证补丁。一家来自国际知名科技公司的安全工程师评论:“参考链接网络作为一个证据链,有助于定位和验证补丁”。此外,实验人员还建议:TRACER 可以添加更多的知识源,并提供更多关于补丁节点的信息,如提交消息、代码变更内容等,这些信息有助于人工审查和确认补丁。

综上,在实际使用中,TRACER 有助于用户更准确、更快速地识别到补丁。

## 5.8 实验结论

本文从准确性、削弱性、敏感度、通用性及实用性五个方面评估 TRACER 进行了评估。结果表明:

- (1) **准确性:** TRACER 可以达到 88.0% 的补丁覆盖率、86.4% 的精确率和 86.4% 的召回率。与现有的基于启发式规则的方法相比,TRACER 能够显著地提高补丁覆盖率和 F1 值,将补丁覆盖率提高 58.6% 到 273.8%,将 F1 值提高 116.8%。与商业漏洞数据库  $DB_A$  和  $DB_B$  相比,TRACER 以略低的补丁精确率和覆盖率为代价,拥有更为显著的召回率。这表明,TRACER 可用于补充现有漏洞数据库缺失的漏洞补丁数据。
- (2) **削弱性:** TRACER 中的知识源、网络构建、补丁选择和补丁扩增步骤的设计对最终结果都有一定的贡献度和必要性。
- (3) **敏感度:** TRACER 的准确率对两个可配置参数的变化不是非常敏感,且网络深度参数设置为 5 层、提交时间跨度设置为 30 天时效果相对最优。
- (4) **通用性:** 在更大范围的开源软件漏洞上,TRACER 具有较好的通用性,覆盖率和准确率比较稳定。在商业漏洞数据库  $DB_A$  和  $DB_B$  都没有补丁时,TRACER 仍可以找到大量漏洞的补丁。这表明,TRACER 可以极大地补充或增强现有的商业漏洞数据库。
- (5) **实用性:** 在实际工作场景下,TRACER 有助于用户更准确、更快速地识别到补丁。

## 5.9 讨论

基于上文实验评估的结果,以及 TRACER 的设计,本节将主要讨论 TRACER 存在的短板和潜在的价值。

### 5.9.1 TRACER 的局限性

TRACER 以 CVE ID 作为输入，因此该方法不适用于没有 CVE ID 的开源漏洞（即不在 CVE 平台中的开源漏洞），比如一些秘密修补的漏洞并没有收录在 CVE 平台中，也没有 CVE ID。针对这种情况，一种可能的解决方案是将 Advisory ID 作为 TRACER 的输入。

TRACER 仅仅包含 Debian、Red Hat 和 GitHub 三个次级知识源，也因此会遗漏在其他知识源中出现的补丁。但 TRACER 具有较好的可扩展性，它可以灵活地扩增其他知识源（如，SecurityFocus<sup>①</sup>）。

此外，章节5.3.3中阐述了造成 TRACER 漏报和误报的多个原因，针对这些原因，可进一步提高 TRACER 的准确性。。

### 5.9.2 TRACER 的价值

TRACER 有益于软件安全领域相关的社区、学术界和工业界。对于安全社区而言，TRACER 可以发现缺少补丁的漏洞，并向 CVE 管理员报告该情况，以提高 CVE 的信息质量并加速 CVE 条目的更新，使得 CVE、NVD 平台的广大用户受益。对于学术界而言，TRACER 可以为大规模数据驱动的安全分析工作（例如，基于学习的漏洞检测<sup>[5-6]</sup>）和经验研究工作提供的补丁数据作为研究基础。对于工业界来说，TRACER 可以帮助安全工程师提高商业漏洞数据库的补丁覆盖率和准确性，从而提高软件安全服务的表现。值得一提的是，TRACER 已经部署到一家国际知名科技公司。

---

① <https://www.securityfocus.com>

## 第 6 章 总结与展望

本章将对本文的工作内容及研究成果进行总结，讨论本文研究工作中存在的不足，并展望了未来可以进行的工作。

### 6.1 本文总结

开源软件（Open Source Software, OSS）加快了开源及闭源应用程序的开发进程，但也将安全风险引入了应用程序中。随着开源软件中被披露的漏洞越来越多，开源软件漏洞管理也逐渐成为一个热点问题，大量软件供应链安全相关的工作都是基于漏洞补丁知识开展。然而，开源软件漏洞数据库中补丁的特征及质量尚未被系统地研究和评估，并且已有的漏洞补丁采集方法通用性较差且人工成本过高。

为解决上述研究问题，本文先开展了一项针对开源软件漏洞补丁的经验研究，以了解当前商业漏洞数据库中开源软件漏洞补丁的质量和特征。该经验研究涵盖五个方面，包括补丁覆盖度分析、补丁一致性分析、补丁类型分析、补丁映射关系以及补丁准确性分析。研究发现：

- (1) 商业漏洞数据库中开源软件漏洞补丁的质量并不理想，体现在：(i) 开源软件漏洞补丁缺失情况较为普遍，商业数据库  $DB_A$  和  $DB_B$  中开源软件漏洞的补丁覆盖率仅为 41.8% 和 41.2%。(ii) 商业漏洞数据库  $DB_A$  和  $DB_B$  具有较高的精确率，但经常会遗漏一些漏洞的补丁，尤其是对于具有多个补丁的漏洞。对于安全服务用户来说，这会给漏洞的及时检测和修复带来较大困难。这体现出当前开源软件漏洞数据库的不足，以及利用自动化补丁识别方法完善漏洞数据的需求。
- (2) 开源软件漏洞补丁在类型、映射关系方面有一定的特殊性，设计自动化补丁识别方法时应充分考虑。体现在：(i) 93.7% 的补丁都为 GitHub 代码提交的形式。(ii) 开源软件漏洞与其补丁之间映射关系具有多样性，超过 40% 的漏洞与其补丁具有一对多的映射关系。

基于经验研究的发现，本文提出了一种名为 TRACER 的基于多源知识的开源软件漏洞的补丁识别方法。该方法用于识别代码提交类型的补丁，并构建一对多的漏洞补丁映射关系。该方法的核心思想是：漏洞补丁会在讨论和解决漏洞的、多种来源的漏洞公告、分析报告等参考链接中被频繁地提及和引用。因此，本文

首先设计了一种基于多知识源的漏洞参考链接网络，再从该网络中选出具有最高置信度和连通度的补丁节点作为结果，并基于选定的补丁进行补丁扩展，从而构建一对多的漏洞补丁映射关系。

本文还进行了大量实验，通过五个研究问题，从准确性、通用性、实用性等多个方面对 TRACER 进行了评估。研究发现：

- (1) **准确性**：TRACER 可以达到 88.0% 的补丁覆盖率、86.4% 的精确率和 86.4% 的召回率。与现有的基于启发式规则的方法相比，TRACER 能够显著的提高补丁覆盖率和 F1 值。与商业漏洞数据库  $DB_A$  和  $DB_B$  相比，TRACER 以略低的补丁精确率和覆盖率为代价，拥有更为显著的召回率。这表明，TRACER 可用于补充现有漏洞数据库缺失的漏洞补丁数据。
- (2) **削弱性**：TRACER 中的知识源、网络构建、补丁选择和补丁扩增步骤，及其环节中环节的设计对最终结果都有一定的贡献度和必要性。
- (3) **敏感度**：TRACER 的准确率对两个可配置参数的变化不是非常敏感，且参数采用默认设置时效果相对最优。
- (4) **通用性**：在更大范围的开源软件漏洞上，TRACER 具有较好的通用性，覆盖率和准确率比较稳定。结果表明，TRACER 可以极大地补充或增强现有的商业漏洞数据库。
- (5) **实用性**：在实际工作场景下，TRACER 有助于用户更准确、更快速地识别到补丁。

## 6.2 展望

TRACER 以 CVE ID 作为输入，该方法不适用于没有 CVE ID 的开源漏洞，即没有收录在 CVE 平台中的漏洞。未来可以考虑以 Advisory ID、Issue ID 作为 TRACER 的输入，进一步挺高补丁覆盖率。

此外，TRACER 中仅仅包含了四个知识源（NVD、Debian、Red Hat 和 GitHub），未来可以扩增更多的知识源（如，SecurityFocus<sup>①</sup>、CERT<sup>②</sup>等），以构建更完整的漏洞参考链接网络。

当前，TRACER 使用基于置信度和连通度的方法从参考链接网络中选择补丁，未来可以尝试使用自然语言和程序分析等技术，以实现基于语义的补丁识别方法。

---

① <https://www.securityfocus.com>

② <https://www.kb.cert.org/vuls/>

## 参考文献

- [1] WANG Y, CHEN B, HUANG K, et al. An empirical study of usages, updates and risks of third-party libraries in java projects[C]//Proceedings of the 36th IEEE International Conference on Software Maintenance and Evolution. 2020: 35-45.
- [2] 何熙巽, 张玉清, 刘奇旭. 软件供应链安全综述[J]. 信息安全学报, 2020, 5: 57-73.
- [3] 刘剑, 苏璞睿, 杨珉, 等. 软件与网络安全研究综述[J]. 软件学报, 2018, 29: 42-68.
- [4] LI Z, ZOU D, XU S, et al. Vulpecker: an automated vulnerability detection system based on code similarity analysis[C]//Proceedings of the 32nd Annual Conference on Computer Security Applications. 2016: 201-213.
- [5] LI Z, ZOU D, XU S, et al. Vuldeepecker: A deep learning-based system for vulnerability detection[C]//Proceedings of the 25th Annual Network and Distributed System Security Symposium. 2018.
- [6] ZHOU Y, LIU S, SIOW J, et al. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks[C]//Proceedings of the 32nd Annual Conference on Neural Information Processing Systems. 2019: 10197-10207.
- [7] JIMENEZ M, RWEMALIKA R, PAPADAKIS M, et al. The importance of accounting for real-world labelling when predicting software vulnerabilities[C]//Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2019: 695-705.
- [8] JANG J, AGRAWAL A, BRUMLEY D. Redebug: finding unpatched code clones in entire os distributions[C]//Proceedings of the IEEE Symposium on Security and Privacy. 2012: 48-62.

- [9] KIM S, WOO S, LEE H, et al. Vuddy: A scalable approach for vulnerable code clone discovery[C]//Proceedings of the IEEE Symposium on Security and Privacy. 2017: 595-614.
- [10] XU Y, XU Z, CHEN B, et al. Patch based vulnerability matching for binary programs[C]//Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2020: 376-387.
- [11] XIAO Y, CHEN B, YU C, et al. {MVP}: Detecting vulnerabilities using patch-enhanced vulnerability signatures[C]//Proceedings of the 29th USENIX Security Symposium. 2020: 1165-1182.
- [12] CUI L, HAO Z, JIAO Y, et al. Vuldetector: Detecting vulnerabilities using weighted feature graph comparison[J]. IEEE Transactions on Information Forensics and Security, 2021, 16: 2004-2017.
- [13] PASHCHENKO I, PLATE H, PONTA S E, et al. Vulnerable open source dependencies: Counting those that matter[C]//Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. 2018: 1-10.
- [14] PONTA S E, PLATE H, SABETTA A. Detection, assessment and mitigation of vulnerabilities in open source dependencies[J]. Empirical Software Engineering, 2020, 25(5): 3175-3215.
- [15] PASHCHENKO I, PLATE H, PONTA S E, et al. Vuln4real: A methodology for counting actually vulnerable dependencies[J]. IEEE Transactions on Software Engineering, 2020.
- [16] PONTA S E, PLATE H, SABETTA A, et al. A manually-curated dataset of fixes to vulnerabilities of open-source software[C]//Proceedings of the IEEE/ACM 16th International Conference on Mining Software Repositories. 2019: 383-387.
- [17] FAN J, LI Y, WANG S, et al. A c/c++ code vulnerability dataset with code changes and cve summaries[C]//Proceedings of the 17th International Conference on Mining Software Repositories. 2020: 508-512.
- [18] JIMENEZ M, LE TRAON Y, PAPADAKIS M. Enabling the continuous analysis of security vulnerabilities with vuldata7[C]//Proceedings of the IEEE International Working Conference on Source Code Analysis and Manipulation. 2018: 56-61.



- [19] GKORTZIS A, MITROPOULOS D, SPINELLIS D. Vulinoss: a dataset of security vulnerabilities in open-source systems[C]//Proceedings of the 15th International Conference on Mining Software Repositories. 2018: 18-21.
- [20] NAMRUD Z, KPODJEDO S, TALHI C. Androvul: a repository for android security vulnerabilities[C]//Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering. 2019: 64-71.
- [21] DONG Y, GUO W, CHEN Y, et al. Towards the detection of inconsistencies in public security vulnerability reports[C]//Proceedings of the 28th USENIX Security Symposium. 2019: 869-885.
- [22] CHAPARRO O, LU J, ZAMPETTI F, et al. Detecting missing information in bug descriptions[C]//Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. 2017: 396-407.
- [23] MU D, CUEVAS A, YANG L, et al. Understanding the reproducibility of crowd-reported security vulnerabilities[C]//Proceedings of the 27th USENIX Security Symposium. 2018: 919-936.
- [24] MULLINER C, OBERHEIDE J, ROBERTSON W, et al. Patchdroid: Scalable third-party security patches for android devices[C]//Proceedings of the 29th Annual Computer Security Applications Conference. 2013: 259-268.
- [25] DUAN R, BIJLANI A, JI Y, et al. Automating patching of vulnerable open-source software versions in application binaries.[C]//Proceedings of the 26th Annual Network and Distributed System Security Symposium. 2019.
- [26] XU Z, ZHANG Y, ZHENG L, et al. Automatic hot patch generation for android kernels[C]//Proceedings of the 29th USENIX Security Symposium. 2020: 2397-2414.
- [27] ZHANG H, QIAN Z. Precise and accurate patch presence test for binaries[C]//Proceedings of the 27th USENIX Security Symposium. 2018: 887-902.
- [28] JIANG Z, ZHANG Y, XU J, et al. Pdiff: Semantic-based patch presence testing for downstream kernels[C]//Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. 2020: 1149-1163.

- [29] DAI J, ZHANG Y, JIANG Z, et al. Bscout: Direct whole patch presence test for java executables[C]//Proceedings of the 29th USENIX Security Symposium. 2020: 1147-1164.
- [30] ZHOU Y, SHARMA A. Automated identification of security issues from commit messages and bug reports[C]//Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. 2017: 914-919.
- [31] SABETTA A, BEZZI M. A practical approach to the automatic classification of security-relevant commits[C]//Proceedings of the IEEE International Conference on Software Maintenance and Evolution. 2018: 579-582.
- [32] CHEN Y, SANTOSA A E, YI A M, et al. A machine learning approach for vulnerability curation[C]//Proceedings of the 17th International Conference on Mining Software Repositories. 2020: 32-42.
- [33] YOU W, ZONG P, CHEN K, et al. Semfuzz: Semantics-based automatic generation of proof-of-concept exploits[C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017: 2139-2154.
- [34] MITRE. About cve[EB/OL]. 2021. <https://cve.mitre.org/>.
- [35] NGUYEN V H, MASSACCI F. The (un) reliability of nvd vulnerable versions data: An empirical experiment on google chrome vulnerabilities[C]//Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security. 2013: 493-498.
- [36] NGUYEN V H, DASHEVSKYI S, MASSACCI F. An automatic method for assessing the versions affected by a vulnerability[J]. Empirical Software Engineering, 2016, 21(6): 2268-2297.
- [37] DASHEVSKYI S, BRUCKER A D, MASSACCI F. A screening test for disclosed vulnerabilities in foss components[J]. IEEE Transactions on Software Engineering, 2019, 45(10): 945-966.
- [38] TAN X, ZHANG Y, MI C, et al. Locating the security patches for disclosed oss vulnerabilities with vulnerability-commit correlation ranking[C]//Proceedings of the 28th ACM Conference on Computer and Communications Security. 2021.
- [39] 周鹏, 武延军, 赵琛. 一种 Linux 安全漏洞修复补丁自动识别方法[J]. 计算机研究与发展, 2022, 59: 197-208.

- [40] 邹雅毅, 李珍. 开源软件漏洞补丁的采集与整理[J]. 河北省科学院学报, 2016, 33: 18-22.
- [41] 李瑞科, 刘元, 廖雷, 等. 1999–2018 年安全漏洞数据集[J]. 中国科学数据 (中英文网络版), 2019, 4: 123-132.
- [42] LI F, PAXSON V. A large-scale empirical study of security patches[C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017: 2201-2215.
- [43] LIU B, MENG G, ZOU W, et al. A large-scale empirical study on vulnerability distribution within projects and the lessons learned[C]//Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering. 2020: 1547-1559.
- [44] 文琪, 江喆越, 张源. 基于关键路径测试的安全补丁存在性检测[J]. 计算机应用与软件, 2020, 37: 1-7.
- [45] BRUMLEY D, POOSANKAM P, SONG D, et al. Automatic patch-based exploit generation is possible: Techniques and implications[C]//Proceedings of the IEEE Symposium on Security and Privacy. 2008: 143-157.
- [46] 李舟军, 张俊贤, 廖湘科, 等. 软件安全漏洞检测技术[J]. 计算机学报, 2015, 38: 717-732.
- [47] 李韵, 黄辰林, 王中锋, 等. 基于机器学习的软件漏洞挖掘方法综述[J]. 软件学报, 2020, 31: 2040-2061.
- [48] 李赞, 边攀, 石文昌, 等. 一种利用补丁的未知漏洞发现方法[J]. 软件学报, 2018, 29: 1119-1212.
- [49] 李元诚, 黄戎, 来风刚, 等. 基于深度聚类的开源软件漏洞检测方法[J]. 计算机应用研究, 2020, 37: 1107-1110.
- [50] ANTAL G, KELETI M, HEGEDŰS P. Exploring the security awareness of the python and javascript open source communities[C]//Proceedings of the 17th International Conference on Mining Software Repositories. 2020: 16-20.

# 致谢