学校代码：10246
学　　号：19212010035

復旦大學

硕 士 学 位 论 文

（学术学位）

基于多源信息的开源软件漏洞的补丁识别方法

Finding Patches for Open Source Software Vulnerabiliies from multiple sources

院　　系：　软件学院
专　　业：　软件工程
姓　　名：　许聪颖
指 导 教 师：　陈碧欢　副教授
完 成 日 期：2021 年 12 月 26 日

目 录

摘 要

开源软件 (Open source software, OSS) 漏洞管理已然成为一个热点问题。开源漏洞数据库为解决漏洞问题提供十分有价值的数据信息，因此，漏洞数据库的数据质量也受到越来越多的关注和研究。具体的问题为：现有漏洞数据库中补丁的质量尚未研究清楚，此外，现有的补丁信息多由人工或基于启发式的识别方法进行收集。这种方法人工成本过高，且过于定制化无法应用于全部的 OSS 漏洞。

empirical study 该如何翻译比价好呢？实证研究？经验性研究？为了解决这些问题，首先，我们进行了实证研究，以了解当前商业旗舰漏洞数据库中开源软件漏洞补丁的质量和特征。我们的研究涵盖五个方面，包括：补丁的覆盖度、一致性、类型、基数和准确性。然后，基于研究的发现，我们提出了第一种名为 TRACER 的自动化方法，用于从多个来源查找开源漏洞的补丁。

实验评估表明：i) 与现有的基于启发式的方法相比，TRACER 能够为多达 273.8% 的 CVE 找到补丁；同时，准确性方面，将 F1 数值提高达 116.8%；ii) 与现有的漏洞数据库相比，TRACER 将召回率（recall）提高达 18.4% ；然而，12.0% 的 CVE 补丁未找到，精度（precision）下降约6.4%。

关键字：关键词 1，关键词 2，关键词 3
中图分类号：TP311

Abstract

Open source software (OSS) vulnerability management has become an open problem. Vulnerability databases provide valuable data that is needed to address OSS vulnerabilities. However, there arises a growing concern about the information quality of vulnerability databases. In particular, it is unclear how the quality of patches in existing vulnerability databases is. 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, we first conduct an empirical study to understand the quality and characteristics of patches for OSS vulnerabilities in two state-of-the-art vulnerability databases. Our study is designed to cover five dimensions, i.e., the coverage, consistency, type, cardinality and accuracy of patches. Then, inspired by our study, we propose the first automated approach, named TRACER, to find patches for an OSS vulnerability from multiple sources. Our key idea is that patch commits will be frequently referenced during the reporting, discussion and resolution of an OSS vulnerability.

Our extensive evaluation has indicated that i) TRACER finds patches for up to 273.8% more CVEs than existing heuristic-based approaches while achieving a significantly higher F1-score by up to 116.8%; and ii) TRACER achieves a higher recall by up to 18.4% than state-of-the-art vulnerability databases, but sacrifices up to 12.0% fewer CVEs (whose patches are not found) and 6.4% lower precision. Our evaluation has also demonstrated the generality and usefulness of TRACER.

Keywords: Keyword1, Keyword2, Keyword3

CLC code: TP311

第 1 章 绪论

本章将详细阐述本文的研究背景、主要工作以及本文的篇章结构。

1.1 研究背景及问题

开源软件（Open Source Software，OSS）为开源及闭源应用程序的开发提供了基础，应用程序的开发人员可以直接使用开源软件提供的通用功能而不需重新造轮子。据 Synopsys 公司发布的《开源安全和风险分析报告》[1] 中数据显示，分析的 1,500 个应用程序中有 98% 的应用程序都包含开源软件。然而，大规模使用开源软件加速了应用程序开发的进程，但同时引入了安全风险。例如，Synopsys 公司在 2020 年分析的 1,500 个应用程序中有 84% 的应用程序包含至少一个公共 OSS 漏洞，对比 2019 年时的数据（75%）增加了 9%[1]。更糟糕的是，据 Snyk 公司发布的报告[2] 显示：近些年开源软件中披露的漏洞越来越多，在过去两年中几乎翻了一倍。

为此，大量工作都在研究如何降低开源软件漏洞的安全风险，其中包括：通过学习漏洞特征来检测开源软件中的漏洞[3–6]，通过匹配漏洞及补丁签名[?] 以修复开源软件中的漏洞[?]，或分析软件组成成分以确定应用程序中的开源漏洞是否可达[7–10]。

值得注意的是，漏洞数据库在这些工作中发挥着重要作用，体现在为各种漏洞分析任务提供有价值的数据（例如，漏洞描述、受漏洞影响的软件、版本以及补丁信息）。目前，已有多方力量致力于构建漏洞数据库。在安全社区中，CVE List[11] 和 NVD[12] 是最具影响力的漏洞数据库，该数据库包含应用软件、系统以及硬件的漏洞。在工业界中，BlackDuck[13]、WhiteSource[14]、Veracode[15] 和 Snyk[16] 等公司较为关注开源软件的漏洞，并构建各自的工业数据库。在学术界中，已有很多工作致力于构建漏洞数据集[17–21]，但它们大多针对特定语言的生态系统或针对特定的软件项目而设计的。

研究问题：随着这些漏洞数据库积累越来越多的漏洞，人们也越来越关注这些数据库中漏洞信息的质量。Nguyen、Massacci[22] 和 Dong 等人[23] 发现了漏洞数据库中受漏洞影响的软件版本信息的不准确性 Chaparro 等人[24] 和 Mu 等人[25] 发现了漏洞描述中缺失关键漏洞重现步骤的普遍性。这种不完整或不准确的信息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及其使用的

开源软件中的漏洞。

漏洞补丁，作为捕获漏洞的宝贵信息，可应用于多种程序安全相关的任务，例如，补丁生成和热部署[26–28]、补丁存在测试[29–31]、软件成分分析[8–10]以及漏洞检测[3–4,32–35]。如果漏洞的补丁信息缺失或不准确，那么这些安全应用的准确性将会受到严重的影响。然而，漏洞数据库中补丁信息尚未被系统地研究，目前尚不清楚现有漏洞数据库中补丁的质量如何。

此外，现有的安全应用主要通过人工手动定位漏洞补丁[8–9,28,30–31,34,36–38]，或是通过启发式规则，在 CVE 引用信息中查找代码提交[3,27?]以及并在代码提交历史中搜索 CVE 标识符（CVE-ID）[10,39]，或者从已经为特定项目建立漏洞和补丁之间映射关系的安全公告中搜索漏洞补丁[26,32–33]。以上这些方法的人工成本过高，且过于定制化具体的程序语言或项目无法广泛应用于所有开源软件漏洞。

1.2 本文工作概述

为了解决以上研究问题，本文先进行了一项经验研究，以了解当前顶级工业漏洞数据库中开源软件漏洞补丁的质量和特征。该研究涵盖主要五个方面，包括：补丁的覆盖度、一致性、类型、映射关系和准确性。然后，受经验研究结果的启发，本文提出了第一种名为 TRACER 的基于多源信息的开源软件漏洞的补丁识别方法。本文还设计了大量实验验证了 TRACER 的准确性、通用性、实用性等方面。TRACER 的源代码和所有实验数据已经在`https://patch-tracer.github.io`网站上发布。

1.2.1 经验研究

为了解当前顶级工业漏洞数据库中开源软件漏洞补丁的质量和特征，涵盖：补丁的覆盖度、一致性、类型、映射关系和准确性五个方面。该经验研究选择 Veracode[15] 公司和 Snyk[16] 公司公开的漏洞数据库作为研究对象。

基于经验研究中构建的包含 10，070 个漏洞的广度数据集，本文分析两个漏洞数据库中所有开源软件漏洞补丁信息的覆盖率和一致性。结果表明，10,070个 CVE 漏洞中只有4,602(5.7%)的漏洞至少由一个漏洞数据库提供补丁信息，只有19.7%的漏洞在两个数据库中有一致的补丁信息。可以再写些其他的结果。

此外，基于经验研究中构建的包含1,295个漏洞的深度数据集，本文分析了开源软件漏洞补丁类型、映射关系和准确性。结果表明，1,295个 CVE 漏洞中，1,265(97.7%)的漏洞有 GitHub 和 SVN Commit 提交类型的补丁，533(41.1%)的 CVE 漏洞与其补丁在数量上有一对多的映射关系。这两个数据库的总体补丁精度都高于90%，然而，对于一对多映射类型的 CVE 漏洞，这两个数据库中补丁

的召回率仅约为50%。

这些结果表明，这两个顶级工业漏洞数据库经常会遗漏一些漏洞的补丁信息，尤其是对于有多个补丁的 CVE 漏洞，补丁缺失现象更为严重。这种不完整或不准确的信息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及其使用的开源软件中的漏洞。同时，这也反映出自动化补丁查找方法的需求，以帮助工作人员正确且完全地找到补丁信息。

1.2.2 方法设计

受经验研究结果的启发，本文提出了第一种名为 TRACER 的基于多源信息的开源软件漏洞的补丁识别方法，从多个知识源（即：NVD[12]，Debian[40]，Red Hat[41] 以及 GitHub）。该方法的核心思想是：漏洞的补丁信息（即：Commit URL）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；因此，本文首先设计了一种基于多知识源的漏洞引用信息网络，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。

TRACER 以漏洞的 CVE-ID 作为输入，主要经过三个步骤：首先，构建多源信息网络，该步骤的目的是将该 CVE 在被报告、讨论和解决阶段的引用链接信息进行建模。TRACER 从多个漏洞知识源（即：NVD、Debian[40]、RedHat[41] 和 GitHub）中提取与该 CVE 相关的引用链接信息并构建一个信息网络。然后，精选补丁节点，TRACER 从构建的引用信息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 CVE 的补丁。最后，扩增候选补丁，TRACER 通过搜索同一代码库所有分支中的相关提交（Commit）来扩展候选补丁集。最终，TRACER 将该 CVE 的候选补丁集返回给用户，用户可基于 TRACER 提供的信息选定确认正确的补丁。

1.2.3 实验验证

为了评估 TRACER 的准确性，本文将 TRACER 与三种基于启发式的方法和两个顶级工业漏洞数据库在经验研究中构建地深度数据集上进行了对比。结果表明，（1）与现有的基于启发式的方法相比，TRACER 将能找到补丁的漏洞数提高58.6%到273.8%，同时，将 F1 值提高116.8%。（2）TRACER 具有比 DB_A 和 DB_B 高15.5% 和 18.4%的召回率和5.5% 到 8.6%的 F1 值，同时牺牲了6.4%的精度和12.0%CVE 的补丁信息。

此外，为了评估 TRACER 的通用性，本文还在另外两个分别包含3,185和5,468个 CVE 的数据集上运行 TRACER 。结果表明，TRACER 在两个附加数据集上可查找到67.7%和51.5%CVE 的补丁，通过采样评估，精度分别为0.823和0.888，召回率分别为0.845和0.899，体现出 TRACER 在查找补丁方面具有较好的通用性。

此外，为了评估 TRACER 的实用性，本文还邀请了 10 名参与者进行了用户研究。评估结果表明，在实际使用中，TRACER 有助于安全专家更准确、更快速地查找到补丁信息。

1.2.4 主要贡献

本文主要有以下贡献。

- 本文进行了一项经验研究，以了解当前顶级工业漏洞数据库中开源软件漏洞补丁的质量和特征。该研究涵盖主要五个方面，包括：补丁的覆盖度、一致性、类型、映射关系和准确性。
- 本文提出了第一种名为 TRACER 的基于多源信息的开源软件漏洞的补丁识别方法，可服务于安全社区、工业界和学术界。
- 本文进行了一系列实验验证，以评估 TRACER 的准确性、通用性、在实践中的实用性等方面。

1.3 本文篇章结构

本论文一共六章，结构如下：

第一章绪论，介绍了本文的研究背景及研究问题，然后概述了本文的主要工作及主要贡献，包括：经验研究、方法设计和实验验证，以及本文的篇章结构。

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、方法设计等内容的详细阐述做铺垫；本章还介绍了本文研究主题的相关工作，包括：漏洞信息质量、漏洞补丁分析以及漏洞补丁的应用。

第三章经验研究，介绍了本文为了解当前顶级工业漏洞数据库中开源软件漏洞补丁的质量和特征所开展的实证研究工作，涵盖：补丁的覆盖度、一致性、类型、映射关系和准确性五个方面。

第四章 TRACER 方法设计，介绍了本文提出了第一种名为 TRACER 的基于多源信息的开源软件漏洞的补丁识别方法，包括：多源信息网络构建、补丁节点精选以及候选补丁扩增三个步骤。

第五章实验验证及结果分析，介绍了本文评估 TRACER 的准确性、通用性、在实践中的实用性等方面的实验设计及结果分析。

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及可以改进的地方，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【v1-doing】

本文的研究重点是开源软件漏洞的补丁定位问题，本章将首先介绍漏洞相关的背景知识，包括：通用漏洞披露（CVE）、漏洞通告（advisory）和漏洞补丁（vulnerability patch）；然后，从漏洞披露信息的质量、漏洞补丁的分析及应用三个方面介绍相关的研究工作。

2.1 背景知识

2.1.1 通用漏洞披露 (CVE)

通用漏洞披露 (Common Vulnerabilities and Exposures，CVE)[42]，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给予唯一编号以便于公众查阅及引用。

如图例2-1所示，每一个 CVE 条目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：References），该参考链接多为外部网站且包含与该漏洞相关的更详细的描述信息。

基于 CVE 收录的漏洞条目信息，美国国家漏洞数据库（NVD）、中国国家信息安全漏洞库（CNNVD）等与 CVE 数据完全同步的漏洞数据库被构建，用于为每个 CVE 条目提供更丰富的信息，如：修复信息、严重性评分、影响评级等。

2.1.2 漏洞公告【todo】

漏洞公告（Advisory），也称漏洞通告，是由含漏洞的软件的厂商（Vendor）对外发布的安全漏洞警报，一般包含：漏洞触发描述、漏洞影响结果、漏洞软件名、软件版本等漏洞描述信息，有时也会包含漏洞发现者、漏洞修复记录、漏洞补丁等信息。

如图（log4j 的漏洞所示，）啥啥啥分分别是啥啥信息。

以上由厂商发布的漏洞公告，也常被称为：厂商公告（Vendor Advisory）；但由于某一厂商只会维护与该厂商相关的软件漏洞通告，开源社区的开发人员难以一一关注所有厂商的公告信息，所以，便出现了第三方平台收集并维护所有。称为：第三方公告（Third-party Library）。

CVE-ID

CVE-2017-11428 — Learn more at National Vulnerability Database (NVD)
• CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description

OneLogin Ruby-SAML 1.6.0 and earlier may incorrectly utilize the results of XML DOM traversal and canonicalization APIs in such a way that an attacker may be able to manipulate the SAML data without invalidating the cryptographic signature, allowing the attack to potentially bypass authentication to SAML service providers.

References

Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

- MISC:https://duo.com/blog/duo-finds-saml-vulnerabilities-affecting-multiple-implementations
- MISC:https://www.kb.cert.org/vuls/id/475445

Assigning CNA

Duo Security, Inc.

Date Record Created

20170718 — Disclaimer: The record creation date may reflect when the CVE ID was allocated or reserved, and does not necessarily indicate when this vulnerability was discovered, shared with the affected vendor, publicly disclosed, or updated in CVE.

Phase (Legacy)

Assigned (20170718)

Votes (Legacy)

Comments (Legacy)

Proposed (Legacy)

N/A

This is a record on the CVE List, which provides common identifiers for publicly known cybersecurity vulnerabilities.

SEARCH CVE USING KEYWORDS: Submit
You can also search by reference using the CVE Reference Maps.

For More Information: CVE Request Web Form (select "Other" from dropdown)

图 2-1　CVE-2017-11428

官方或第三方？再顺路引出相关的 sources，作为例子。

Vender advisory，official advisory：CVE NVD，third-party advisory：Redhat debian 啥啥

2.1.3　漏洞补丁【todo】

是指。。。。.Patch Git, SVN commit

2.2　相关工作

2.2.1　漏洞信息质量

漏洞数据库（例如：CVE、NVD）被广泛关注，其中的漏洞信息也被广泛参考使用。随着漏洞数据库积累的漏洞数据越来越多，研究人员也越来越关注其中的漏洞信息的质量。

Nguyen 和 Massacci[22] 最早揭示了 NVD 数据库中漏洞所影响的软件版本信息的不可靠性。为了提高该信息的可靠性，Nguyen[43] 和 Dashevskyi 等人[44] 开发了工具以确定某一旧软件版本是否会受到新披露的漏洞的影响。他们认为：如果旧版本包含修复漏洞所更改的源代码行，则该版本被视为受漏洞影响。Dong

等人[23]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。他们发现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。Chen 等人[45]识别受漏洞影响的开源库信息。Chaparro 等人的工作[24]检测漏洞描述中是否缺少用于重现漏洞的关键步骤或预期效果信息。Mu 等人的工作[25]揭示了漏洞报告丢失重现漏洞信息的普遍性。以上工作已侧重于漏洞信息的多个方面，而本文的工作重点则是研究漏洞的补丁信息，并尝试自动化地从不同来源漏洞报告的综合信息中定位漏洞补丁。

近期，Tan 等人完成了一项与本文的研究问题相似的工作[46]。他们使用深度学习排名算法对代码仓库中的提交（commit）历史进行排名，把排在首位的提交当作为漏洞的补丁提交。他们的工作包含两个假设：（1）CVE 中，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（Sec.3.6）。

2.2.2 漏洞补丁分析

当前，有多中补丁分析相关的任务可用于提高软件安全性，如：补丁的生成和部署[26–28]、补丁的存在性测试[29–31]以及秘密补丁识别[36–38,47]。

此外，研究人员也已为 Java[17]、C/C++[18]以及特定开源项目[19]构建安全补丁数据集。基于这些数据集，研究人员已开展实证研究以表征漏洞及其补丁[48–51]。在这些工作中，补丁信息多由人工识别[17,28,30–31,36–38,47–48]，或通过启发式规则识别，例如：在 CVE 的引用链接中查找补丁提交信息[18–19,27,49–50]，以及在提交信息（commit message）中搜索 CVE 标识符[18–19,51]。这些工作存在的问题为：通过人工收集成本过高，且耗时较长；然而，启发式规则的方法又不足以找到或是找全补丁。

2.2.3 漏洞补丁应用

漏洞补丁信息可被用于多种软件安全性任务。例如，基于漏洞补丁生成漏洞攻击程序[39,52]，通过软件成分分析以确定项目是否使用包含漏洞的第三方库，并判定该漏洞所影响的函数是否被调用[7–10]，以及通过学习漏洞特征[3–6]、通过匹配漏洞签名[32–33]、通过匹配漏洞和补丁检测签名[34–35,53]来检测程序中的漏洞。

与上一小结的补丁分析工作类似，这些工作中的 CVE 补丁主要通过人工识别[7–9,34]、基于启发式规则的方法[3–4,6,10,39]或直接取自为特定项目建立 CVE 和补丁之间映射关系的安全公告[32–33,53]。但是，人工识别的成本过高，而且基于

启发式规则的方法找到或是找全补丁。

第 3 章 经验研究

本章将主要介绍为了解当前漏洞数据库现状，考察其中漏洞补丁的质量和特征所开展的经验研究工作，包括：经验研究的设计、数据准备以及经验研究的结果分析。

3.1 研究设计

3.1.1 研究问题

为了了解已有漏洞数据库中开源软件漏洞补丁的质量和特征，本文所开展的针对当前高质量漏洞数据库的经验研究包含以下研究问题：

- **RQ1 覆盖率分析：**当前高质量漏洞数据库中，漏洞补丁信息的覆盖度如何？即：有多少漏洞包含补丁信息？(Sec. 3.3)
- **RQ2 一致性分析：**不用漏洞库间，漏洞补丁信息的一致性如何？即：有多少漏洞在漏洞数据库中具有相同的补丁信息？(Sec. 3.4)
- **RQ3 补丁类型分析：**开源漏洞补丁的类型有哪些？(Sec. 3.5)
- **RQ4 补丁映射分析：**开源漏洞与其补丁在数量上的映射关系是怎样的？(Sec. 3.6)
- **RQ5 补丁准确性分析：**当前高质量漏洞数据库中，漏洞的补丁信息准确度如何？(Sec. 3.7)

其中，RQ1 可用来评估漏洞数据库中开源软件漏洞的补丁缺失程度，RQ2 用来评估不同漏洞数据库中漏洞补丁的不一致程度，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信息的准确性。总的来说，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：将正样本预测为正样本，即数据库提供的或工具返回的正确补丁信息。
- True Negative（真阴性），表示：将负样本预测为负样本，即数据库未提供的或工具未返回的错误补丁信息。
- False Positive（假阳性），表示：将负样本预测为正样本，即数据库提供的或工具返回的错误补丁信息。
- False Negative（假阴性），表示：将正样本预测为负样本，即数据库未提供的或工具未返回的正确补丁信息。

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：

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{3.1}$$

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：

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{3.2}$$

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：

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3.3}$$

3.2 数据准备

3.2.1 漏洞数据库选择

为挑选高质量的、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。在该章节的经验研究工作中，本文首先排除了来自安全社区的数据库（例如，CVE List 和 NVD）。因为这两个数据库不提供结构化的补丁信息，而补丁链接多是隐藏在参考链接中；此外，CVE List 和 NVD 数据库中不仅仅包含开源软件漏洞，还包括闭源软件、系统及硬件相关的漏洞。本文还排除了来自学术界的数据集[17–21,49–51]，这是因为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），而非面向所有开源软件，缺乏多样性不具有代表性；此外，由于长期缺乏维护，这些漏洞数据集缺失较新的漏洞数据。

对于工业界的数据库，本文首先关注到 BlackDuck[13]、WhiteSource[14]、Veracode[15]和 Snyk[16]四家安全公司提供软件成分分析（Software Composition Analysis）服务，这种服务通过识别并分析当前软件系统中使用的开源成本（即：第三方库），报告所使用的开源成分中的漏洞。因此，这四家公司需要先构建尽可能完整且包含详细漏洞信息的漏洞库作为服务基础，本文便首先将这四家公司的漏洞数据库作为研究对象。截至 2021 年 4 月 5 日，收集的信息为：

- **Black Duck**，该公司的报告显示：该公司的安全公告中共包含 157,000 多个漏洞，涵盖 90 多种编程语言，其中，数千个漏洞尚未被 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，然而，该公司的漏洞数据信息并未对外公开。
- **Sonatype**①，该公司声称：“*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。”② Sonatype 的 OSS Index 支持 20 多个生态系统（如：Maven、npm、Go、PyPI 等）。该公司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 向量和参考链接等信息。
- **WhiteSource链接**，该公司从 NVD 及其他安全公告平台和问题追踪系统（issue tracking system）中共收集的漏洞超过 175,000 个，涵盖 200 多种编程语言。
- **Veracode链接**，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，公开的漏洞信息包括：受漏洞影响的组件和版本范围、库修复说明、参考资料等。
- **Snyk**③，该公司声称：漏洞数据库④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 issue 报告，Github commits 等方式自动识别安全漏洞相关的报告。该公司的数据库涵盖超过 10 个编程语言生态系统，如：Maven、npm、Go、Composer 等。该数据库提供漏洞的详细信息，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，这四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信息，这将无法达成研究目标。最终，本文选定 Veracode 和 Snyk 的漏洞数据库作为研究对象，下文中简称为：DB_A 和 DB_B。

① https://ossindex.sonatype.org

② 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.”

③ https://snyk.io/vuln

④ https://snyk.io/product/vulnerability-database/

3.2.2　广度数据集构建

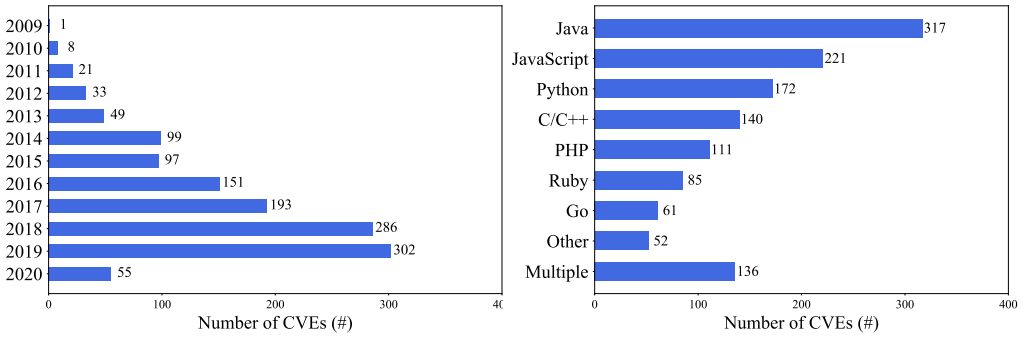
为了评估漏洞数据库中补丁的缺失程度以及不同数据库间补丁的不一致性（即：RQ1 和 RQ2），本文基于 DB_A 和 DB_B 构建了一个开源软件漏洞的广度数据集用以实验分析。截至 2020 年 4 月 7 日，分别从 DB_A 和 DB_B 中分别获取了8,630和5,858个 CVE 漏洞。

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信息的准确性（即：RQ3、RQ4 和 RQ5），本文还基于 DB_A 和 DB_B 的数据，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信息。

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 都含有补丁信息的漏洞列入该深度数据集，最终，该深度数据集共包含1,417个 CVE 漏洞。

然后，对于该深度数据集中的每个 CVE 漏洞，首先分别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信息以及搜索 GitHub 代码仓库的提交历史和其他网络资源等方式，独立得找到其补丁信息；之后，对比由两位研究人员独立查找得到得补丁信息，对于补丁结果不一致的漏洞，两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 CVE 漏洞中的122个 CVE 漏洞无法找到补丁信息，比如：漏洞 CVE-2016-3942 在 NVD 中没有漏洞报告，但 DB_A 和 DB_B 将 jsrender@f984e1[54] 标识为其补丁，两位研究人员无法确认该补丁信息的准确性。最终，该深度数据集共包含了1,295个 CVE 漏洞。

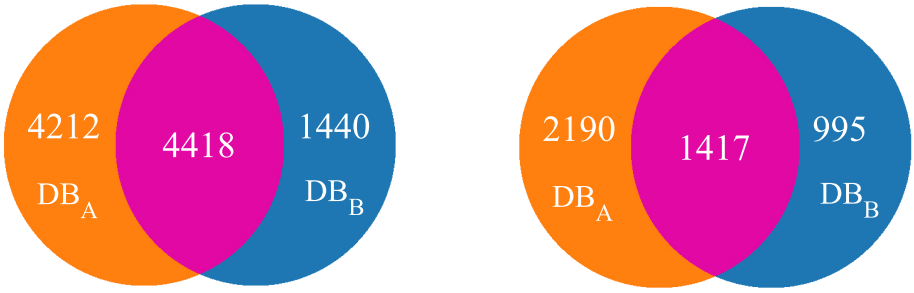


(a) 漏洞年份分布统计　　(b) 程序语言分布统计

图 3-1　数据集中漏洞年份及程序语言分布统计

本文还进一步分析了该深度数据集中1,295个 CVE 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3-1a所示，CVE 的数量逐年增加，这与 Snyk 的报告[2]一致。此外，本文通过分析补丁中更改的源文件类型来确定 CVE 的编程语言。如图3-1b所示，深度数据集中的 CVE 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。

3.3　RQ1：覆盖率分析



(a) 开源软件漏洞　　(b) 含补丁信息的开源软件漏洞

图 3-2　DB_A 与 DB_B 间数据交集

如图3-2a所示，DB_A 和 DB_B 数据库中共有的 CVE 漏洞为4,418个，同时 DB_A 和 DB_B 分别包含4,212和1,440个特有的 CVE 漏洞；如图3-2b所示，DB_A 中3,607(41.8%)的 CVE 漏洞含有补丁信息，DB_B 中2,412(41.2%)的 CVE 漏洞含有补丁信息；DB_A 和 DB_B 数据库共有10,070个开源软件 CVE 漏洞，而其中仅有4,602(45.7%)的漏洞提供了补丁信息。

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较低，分别为 41.8% 和 41.2%，漏洞补丁缺失的情况较为普遍。同时，这也体现出自动化补丁查找方法的必要性，可用于填补数据库中缺失的补丁信息。

3.4　RQ2：一致性分析

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	无漏洞信息	无补丁信息	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%）

为了分析两个数据库之间的补丁信息一致性情况，本节主要关注带有补丁的 CVE 漏洞，即：图3-2b中的 CVE 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，才判定为补丁信息一致。本节将补丁信息不一致分为存在性不一致和内容不一致两种情况。前者是指某一个数据库为该 CVE 漏洞提供了补丁信息，而另一个数据库却不存在该 CVE 信息，或是存在该 CVE 却不存在相关补丁信息；后者是指两个数据库都存在该 CVE 的补丁信息，但它们的补丁集并不完全一致，分为包含关系或非包含关系的不一致。这两种情况分别反映了出数据库 DB_A 和 DB_B 中开源软件漏洞及其补丁信息的不完整性，以及漏洞补丁信息可能是不准确的

表3-1中展示了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 CVE 数量（907，19.7%），第二至四列为补丁存在性不一致的 CVE 数量（3,185，69.2%），最后三列为都存在补丁信息但补丁集内容不一致的 CVE 数量（510，11.1%）。可以发现：4,602个 CVE 中，（1）只有907(19.7%)的漏洞在 DB_A 和 DB_B 中有一致的补丁信息；（2）超过三分之二（即：3,185(69.2%)）的 CVE 漏洞在数据库 DB_A 和 DB_B 中存在补丁信息不一致的情况，其中1,392(30.2%)的 CVE 漏洞不在 DB_A 或 DB_B 中，1,793(39.0%)的 CVE 漏洞都存在于 DB_A 和 DB_B 中但在某一数据库中无补丁信息；（3）510(11.1%)的 CVE 漏洞补丁信息都存在于 DB_A 和 DB_B 中补丁集内容不一致，其中，176(3.8%)CVE 的来自于某一个数据库的补丁集包含来自另一个数据库的补丁集，334(7.3%)CVE 的来自 DB_A 和 DB_B 的补丁集即不同也不包含。

这些结果表明，DB_A 和 DB_B 间存在较多的补丁信息不一致情况，进而表明数据库中补丁信息的准确性也需要进一步评估。

3.5 RQ3：补丁类型分析

在3.2小节中，基于人工收集的深度数据集共包含1,295个 CVE 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。分析结果表明，3,043个补丁中，2,852(93.7%)的补丁都是为 GitHub commit 形式，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 commit 形式，仅有55(1.8%)的补丁为来自其他 Git 平台的 commit 形式。

此外，从 CVE 的角度来看，1,295个 CVE 中1,202(92.8%)的 CVE 有 GitHub commit 类型的补丁，4(0.3%)的 CVE 有 SVN commit 类型的补丁。由于很多项目是从 SVN 切换为 Git 管理，48(3.7%)的 CVE 既有 GitHub commit 又有 SVN commit 类型的补丁。只有30(2.3%)的 CVE 的补丁都为来自其他 Git 平台的 commit 形式。以上分析结果表明，开源软件漏洞的补丁类型主要为 GitHub commit，少部分为 SVN commit，而极小部分为其他 Git 平台的 commit。

3.6 RQ4：补丁映射分析

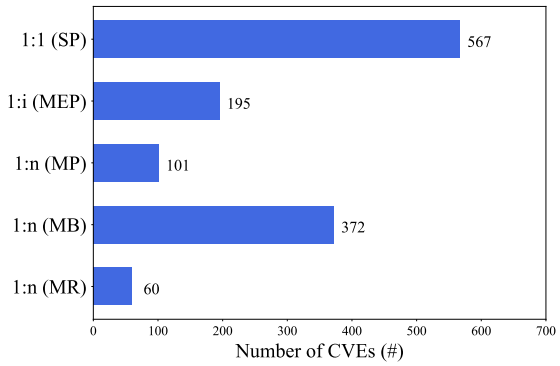


图 3-3 Mapping Cardinalities between CVEs and Patches

基于深度数据集中1,295个 CVE 漏洞及其补丁数据，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 CVE 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 CVE 漏洞与其补丁在数量上为一对一的关系，即：一个 CVE 漏洞只需一个补丁即可修复，后文中简记为：*SP*（Single Patch）。如图3-3所示，深度数据集中567(43.8%)的 CVE 与其补丁具有一对一的映射关系（*SP*）

一对一组, 是指一个 CVE 漏洞有多个补丁信息，CVE 漏洞与其补丁（commit）在数量上非一对一关系，然而，这些 commit 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 commit，其主要有两种类型：（1）Requested Commit VS. Merged Commit，通过 GitHub 中的 Pull Request 功能修补的 CVE 漏洞，请求提交（requested commit）和合并提交（merged commits）是该漏洞的等效补丁集。例如，python-jose@89b463[55]是拉取请求提交（requested commit），python-jose@73007d[56]是用于修复 CVE-2016-7036 的合并提交（merged commits），这两个 commit 是等效的。（2）SVN Commit VS. Github Commit，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 的代码仓库中分别有用于修补该 CVE 的 commit，且这两处的 commit 中代码变更是完全一样且完全等效的。例如，james-hupa 代码仓库从 SVN 迁移到了 GitHub，SVN commit james-hupa@1373762[57]与 GitHub commit james-hupa@aff28a[58]是等效的。如图3-3所示，深度数据集中195(15.1%)的 CVE 与其补丁为一对一组映射关系（*MEP*）

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

- 一个 CVE 是通过一个分支中的多个独立 commit 来修复的。这是因为该 CVE 较难修复需多次提交（commit），或是后期发现初始的补丁不足以修复漏洞便追加了补丁。后文中将此简记为：***Multiple Patch*，*MP***，占比7.8%（101）。例如，CVE-2017-17837 由三个独立的提交 deltaspike@4e2502[59]、deltaspike@72e607[60] 和 deltaspike@d95abe[61] 修复。
- 一个 CVE 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)。例如，CVE-2019-19118 影响了 django 框架的 2.1.x、2.2.x、3.0.x 和 3.2.x 版本，提交 django@103ebe[62]、django@36f580[63]、django@092cd6[64] 和 django@11c5e0[65] 分别修复了受影响的四个版本分支，其中，django@103ebe 与其他提交中的代码变更并不相同。
- 一个 CVE 由多个存储库中的多个补丁集修复。这是因为该 CVE 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)。例如，CVE-2016-5104 影响了 libimobiledevice 和 libusbmuxd 两个开源软件，提交 libimobiledevice@df1f5c[66] 和 libusbmuxd@4397b3[67] 分修复了受影响的两个开源库。

这些结果表明 CVE 及其补丁之间映射关系的多样性。在后文设计自动化补丁查找方法时，应充分考虑该特征。

3.7 RQ5：补丁准确性分析

表 3-2 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
Total	1,295	0.923	0.748	0.793	0.917	0.730	0.771

本文使用精度（precision）、召回率（recall）和 F1 值（F1-score）作为评估补丁准确性的指标。对于具有两个等效补丁的 CVE，若数据库提供两个等效补丁中的任意一个，精度和召回率都为 1；若数据库提供两个等效补丁中的一个和另一个不相关的补丁，那么精度为 0.5，召回率为 1。

表3-2为 DB_A 和 DB_B 的补丁准确性评估结果。其中，第一列为 CVE 与补丁

的映射类型，第二列为每种映射类型的 CVE 数量，最后六列分别为数据库 DB_A 和 DB_B 中 CVE 补丁的准确率、召回率和 F1 值。结果表明，DB_A 和 DB_B 对于 *SP* 和 *MEP* 类型的 CVE 可实现约 90% 的精度和召回率；同时，对于 *MP*、*MB* 和 *MR* 类型的 CVE，可达到 90% 以上的高精度，但仅有约 50% 的召回率。这说明漏洞数据库 DB_A 和 DB_B 经常会遗漏一些漏洞的补丁信息，尤其是对于具有多个补丁的 CVE 漏洞。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；对于漏洞 CVE-2019-19118，DB_A 报告四个补丁中的两个，而 DB_B 仅报告四个补丁中的一个。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挑战。

第 4 章　TRACER 方法设计

本章将详细阐述 TRACER—一种基于多源信息的开源软件漏洞的补丁识别方法的设计。

4.1　方法概述

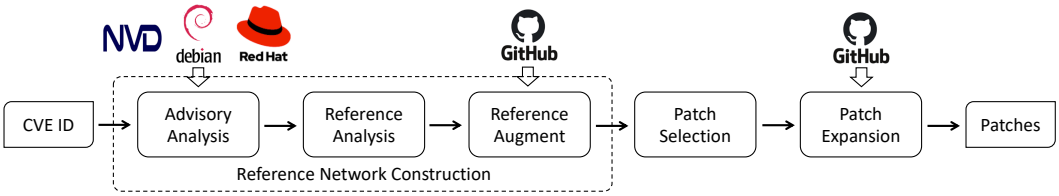


图 4-1　TRACER 方法概览

基于上一章经验研究的发现，本章设计了一种名为 TRACER 的自动化方法来查找开源软件漏洞的补丁。该方法的核心思想是：漏洞的补丁信息（即：Commit URL）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；因此，本文首先设计了一种基于多知识源的漏洞引用信息网络，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返回。

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，最终返回其补丁信息。步骤一：构建多源信息网络，该步骤的目的是将该 CVE 在被报告、讨论和解决阶段的引用链接信息进行建模。TRACER 从多个漏洞知识源（即：NVD、Debian[40]、RedHat[41]和 GitHub）中提取与该 CVE 相关的引用链接信息并构建一个信息网络，这里将 NVD 视为主知识源，将 Debian、RedHat 和 GitHub 视为次知识源，次级知识源列表可以灵活扩展。步骤二：精选补丁节点，TRACER 从构建的引用信息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 CVE 的补丁。步骤三：候选补丁扩增，经验研究中发现 CVE 与其补丁之间存在一对多的映射关系，对此，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提交（Commit）来扩展候选补丁集。最终，TRACER 将该 CVE 的候选补丁集返回给用户，用户可基于 TRACER 提供的信息选定确认正确的补丁。三个步骤的具体设计和实现将在下文中详细阐述。

4.2　步骤一：多源信息网络构建

TRACER 的步骤一为构建漏洞引用信息网络，一共包括三个子步骤。前两个子步骤为漏洞公告节点分析和引用节点分析，通过分析来自 NVD、Debian 和 Red Hat 三个知识源的漏洞公告及其中的引用信息，构建初始引用信息网络；第三个子步骤为知识源扩增，从 GitHub 平台搜索相关的代码提交，作为补丁节点扩充入信息网络。

4.2.1　公告节点分析

首先，TRACER 初始化信息网络，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）中该漏洞的公告作为公告节点添加为根节点的子节点。这些公告节点便于后期追溯选定的补丁的信息来源。

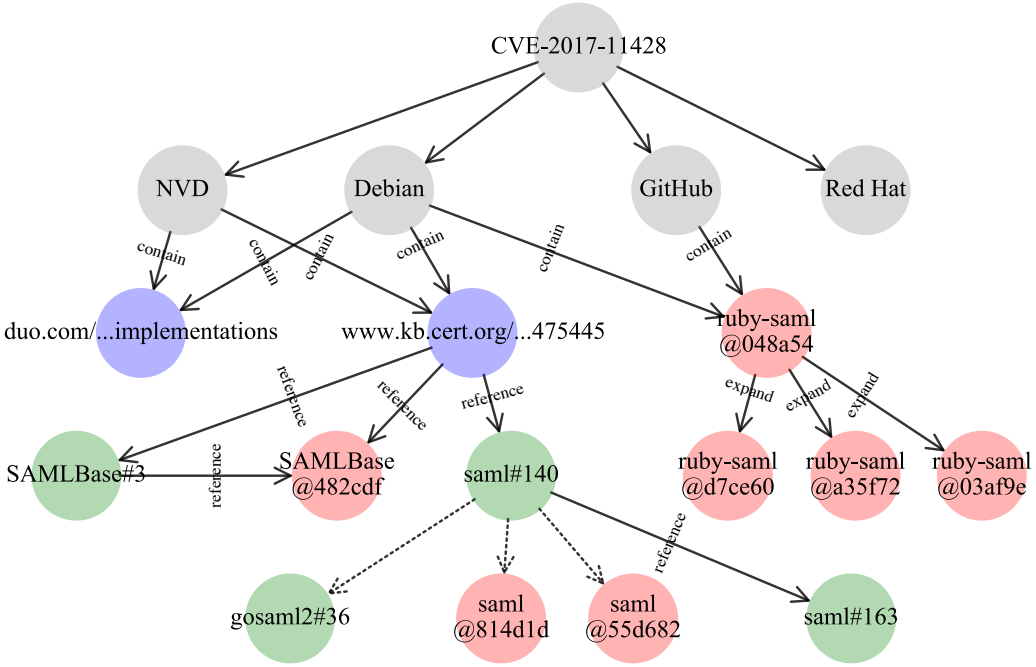


图 4-2　样例 CVE-2017-11428 的多源引用信息网络

Example 4.1　图4-2为样例 CVE-2017-11428 的多源引用信息网络。其中，最顶层为根节点（即：CVE-ID，第二层为公告节点（即：知识源）。

然后，TRACER 通过网络请求分别获取 NVD、Debian 和 Red Hat 平台中该 CVE 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[68]中，TRACER 通过下载并解析相应的 JSON 文件即可获得 NVD 中该 CVE 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[69]中，TRACER 可直接从中解析得 Debian 提供的该 CVE 公告信息。Red Hat 平台提供了

WebService API[70]服务，TRACER 可以直接使用该服务来检索 Red Hat 平台的漏洞公告。值得注意的是，分析发现：Debian 会跟踪 NVD 上的所有 CVE 漏洞，而 Red Hat 并非跟踪所有 CVE 漏洞。

TRACER 从每个知识源的漏洞公告中提取引用的链接信息（即：URL），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”字段中直接获取取出相关 URL；类似地，对于 Debian 公告，TRACER 直接从“Notes”字段中提取出相关 URL；而对于 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 URL，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对补丁信息，但 NVD 和 Debian 平台尚未开发评论区模块。

Example 4.2 如图4-2中的第三层所示，对于 CVE-2017-11428，NVD 公告中包含了两个引用链接。链接一①是对描述此漏洞细节的博客的引用，链接二②是对该漏洞的第三方公告的引用。同时，这两个引用链接也包含在 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[71]。此外，Red Hat 平台并未收录该 CVE，所以也无引用链接信息。

TRACER 基于引用链接信息的内容，将引用节点分为三种类型：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信息。识别补丁节点的方法为：如果 URL 链接中包含“git”字段且可通过正则表达式匹配到 Commit-ID，则该链接为 Git Commit 形式的补丁节点；如果 URL 链接中包含“svn”字段且可通过正则表达式匹配到 Commit-ID，则该链接为 SVN Commit 形式的补丁节点。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）中讨论该 issue 的解决方案并引用补丁链接信息，所以，问题报告（Issue Report）是一种较为特殊且重要的引用信息；此外，问题追踪系统（Issue Tracker）中 CVE 相关的问题报告会有一个标识符（Issue-ID），开发人员常常会将 Issue-ID 写入代码提交信息（即：commit message）中。识别问题节点的方法为：如果 URL 链接中包含 “/github.com/” 和 “/issues/”，则该链接为 GitHub issue 形式的问题节点；如果 URL 链接中包含 “/github.com/” 和 “/pull/”，则该链接为 GitHub pull request 形式的问题节点；如果 URL 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 issue-id，则该链接为通常 issue tracker 形式的问题节点。对于所有未识别为补丁或问题节点的引用链接信息将被视为混合节点，它们多为博客、第三方漏洞公告等。

① https://duo.com/blog/duo-finds-saml-vulnerabilities-affecting-multiple-implementations

② https://www.kb.cert.org/vuls/id/475445

Example 4.3 如图4-2中的第三层所示，NVD 和 Debian 公告中包含的两个引用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 公告中的引用链接被识别为补丁节点（即：图中的红色节点）。

4.2.2 引用节点分析

对于在先前的子步骤中已添加入图的每个引用节点，TRACER 将通过以下两种节点分析方式，以分层的方式继续扩建引用信息网络。

方式一：如果该引用节点的类型为补丁节点，TRACER 会通过网络请求该次代码提交（Commit）的内容并分析该代码提交是否只涉及了测试代码或非源代码文件的修改。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，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。对于非源代码文件的判定，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代码文件类型，即不在列表[①]中则判定为非源代码文件。

方式二：如果该引用节点的类型为问题节点或混合节点，TRACER 会通过网络请求该 URL 的页面信息（即：HTML 文本），并解析出该网页中的 URL 引用信息，将其作为子节点扩入引用网络。首先，对于网页中 URL 引用信息的提取，TRACER 使用正则表达式提取纯文本中的 URL 引用信息，同时使用 HTML 解析器提取超链接（即：**<a>** 标签）中的 URL 引用信息。然后，使用前一子步骤（4.2.1）中相同的方式识别提取的 URL 类型，以识别出补丁和问题引用，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，随着构建的越深噪声也就就越多，所以，除了直接被 NVD、Debian 和 Red Hat 公告节点引用的混合节点，其他节点分析中将不再考虑混合节点。此外，考虑到 GitHub Issue 中通常还包含来自其他软件仓库的问题或提交的引用，这也会给网络带来过多的噪音且使网络空间爆炸。因此，在该步骤中，如果被分析的引用节点为 GitHub Issue，则仅仅将其引用的同一代码仓库中的代码提交或问题引用节点添加到网络中。

对于所有新增的节点，TRACER 将一直重复以上两种节点分析的方式迭代扩增网络，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）。

Example 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，其中一个混合节点未

① todo, 展示常用 list

引用任何问题或提交信息，另一个混合节点了引用了两个问题链接 SAMLBase#3 和 saml#140 以及一个代码提交链接 SAMLBase@482cdf。

在第二次迭代中，TRACER 发现节点 SAMLBase#3 引用了 SAMLBase@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，这些未添加的节点仍显示在图4-2 中，但以虚线连接。

4.2.3 知识源扩增

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，因为补丁信息通常隐藏在代码仓的代码提交历史中。因此，在此子步骤中，TRACER 将搜索代码托管平台以获取 CVE 漏洞的补丁（Github Commit 形式），通过扩增知识来源的方式以进一步扩增该 CVE 的信息网络。

受经验研究中补丁类型分析（Sec.3.5）的启发，93.7% 的都为 GitHub Commit 的形式，所以在该步骤中 TRACER 只搜索 GitHub 平台的提交。此外，问题跟踪系统（Issue Tracker）通常还会为 CVE 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 CVE 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-1573[72]，问题跟踪系统为该漏洞分配的问题标识符为 THRIFT-4647[73]。对此，TRACER 可使用正则表达式①分别从已有的网络节点中提取问题和公告标识符。

然后，TRACER 使用 CVE 标识符和提取出的问题和公告标识符作为关键字，通过 GitHub 提供的 REST API[74]接口全站搜索相关的代码提交。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 CVE 漏洞的 CPE 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[23]，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add example。此外，TRACER 仍会检查该提交是否为测试代码或非源代码文件的更改。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。

① add 表达式

Example 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 CVE 标识符（CVE-ID）来搜索 GitHub 中相关的代码提交。该搜索返回的提交有 ruby-saml@048a54，其仓库信息是“onelogin: ruby-saml”；此 CVE-2017-11428 的 CPE 是“onelogin: ruby-saml”，因此实现了名称的完全匹配，TRACER 将会把 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，将其连接为 GitHub 节点的子节点，如图4-2。此外，该次搜索结果中没有其他匹配的提交。

4.3 步骤二：补丁节点精选

步骤二的目的是尽可能准确且完整地从该 CVE 引用信息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启发式方法。

4.3.1 基于置信度的补丁节点选择方法

该方法中，TRACER 将直接选择具有高置信度的补丁节点作为正确的补丁信息。具体来说，本文认为引用信息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。已有的很多工作[3–4,27]都是基于这种启发式方法来查找补丁。

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至网络中前，经过较为严苛的确认。TRACER 会确保 commit message 中包含该漏洞的 CVE ID、Advisory ID 或 Issue ID，并且其所属的仓库信息与 CPE 名称必须成功匹配。这种方法也在已有的很多工作[10,39]中使用。

Example 4.6 在示例 CVE-2017-11428 的引用信息网络4-2中，TRACER 将直接选择补丁节点 ruby-saml@048a54 加入候选补丁集。因为它是 GitHub 节点的子节点，具有较高的置信度，而实际上，ruby-saml@048a54 也确实是正确的补丁之一。

4.3.2 基于连通度的补丁节点选择方法

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引用信息，且 CVE ID、Advisory ID 或 Issue ID 也很可能不在提交消息（Commit Message）中，因此也无法通过搜索 Github 获取到补丁信息。考虑到漏洞的补丁信息（即：commit）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。一是基于路径数，根节点可到达补丁节点的路径越多，补丁节点与根节点的连通性就越高；二是基于路径长度，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。为了结合这两个维度，TRACER 使用公式4.1计算补丁节点到根节点的连通度，其中 $p = 1, ..., n$ 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。

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

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，加入候选补丁集。

Example 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。另一个是源自 GitHub 的路径，原始长度为 2，调整后为 1，连通性 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。

4.4 步骤三：候选补丁扩增

受经验研究中映射分析（Sec.3.6）的启发，CVE 与其补丁之间存在一对多的映射关系。对此，TRACER 的步骤三通过搜索同一代码库所有分支中的相关提交（Commit）来扩展候选补丁集。映射分析结果表明，超过40%的 CVE 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。对于这些多补丁的情况，TRACER 在前两个步骤中构建的网络通常不能够完整地包含所有补丁。此外，补丁类型分析（Sec.3.5）表明绝大多数的补丁都以 GitHub Commit 形式。因此，TRACER 的步骤三设计如下：对于每个已在步骤二中选定的 GitHub Commit 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（Commit）。

具体来说，对于 GitHub Commit 形式的选定补丁，TRACER 从补丁 URL 中提取出仓库信息，即：所有者（Owner）和存储库（Repository）信息。英文版中，这

里貌似写的有问题. 基于提交的所有者和仓库名信息，首先，TRACER 使用 GitHub 的 REST API[75] 获取存储库中的所有分支信息，对于每个分支，TRACER 会通过 GitHub 的 REST API[76] 检索在选定补丁之前和之后特定时间跨度（默认情况下为30 天）内创建的提交。这里设置了时间跨度，是考虑到工具时间性能和准确性的平衡；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 REST API 返回的提交，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：一是检索返回的提交的提交消息（Commit Message）与已选补丁的提交消息（Commit Message）相同或是包含关系；二是检索返回的提交的提交消息（Commit Message）包含 CVE ID、Advisory ID 或 Issue ID。如果检索返回的提交满足两个条件之一，TRACER 会将其作为扩展补丁节点添加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候选补丁集返回给用户。此外，TRACER 还返回该 CVE 的引用信息网络，以便于工作人员追溯补丁信息的来源及关系，从候选补丁集选定确认正确的补丁。

Example 4.8 在步骤二中为 CVE-2017-11428 选定的补丁为 ruby-saml@048a54（位于主分支),TRACER 通过步骤三查找到了另外三个提交:ruby-saml@d7ce60[77]、ruby-saml@a35f72[78] 和 ruby-saml@03af9e[79]。它们与选定补丁 ruby-saml@048a54 具有相同的提交信息（Commit Message）却具有并不完全相同的代码更改（Diff），且分别位于分支 0.8.3–0.8.17、v0.9.3 和 v1.6.2 中。如图4-2所示，TRACER 将它们添加为 ruby-saml@048a54 的子节点，而实际上，这四个代码提交也确实都是 CVE-2017-11428 的正确补丁，但经验研究3中的数据库 DB_A 和 DB_B 都只提供了 ruby-saml@048a54 这一个补丁信息。

第 5 章 实验验证及结果分析

本章将详细阐述为评估 TRACER 所设计的实验验证及其结果分析。

5.1 实验设计

为了全面评估 TRACER，本章设计了以下四个研究问题。

- **RQ6 准确性验证：**与基于启发式的方法和两个工业漏洞数据库相比，TRACER 在查找漏洞补丁方面的准确性如何？（Sec. 5.2）
- **RQ7 削弱性分析：**TRACER 中各步骤和设置的必要性和贡献度如何？(Sec.5.3)
- **RQ8 敏感度分析：**TRACER 的精度对 TRACER 中参数的敏感性如何？(Sec. 5.4)
- **RQ9 通用性分析：**TRACER 对更大范围的开源软件漏洞的通用性如何？(Sec. 5.4)
- **RQ10 实用性能分析：**TRACER 在实际使用中的实用性如何？(Sec. 5.5)

本文在经验研究（Sec. 3）中构建了深度数据集，本章的实验验证中将继续使用该深度数据集来验证 **RQ6**、**RQ7** 和 **RQ8**。对于 **RQ9**，本章将构造另外两个数据集来进行验证。实验验证中采用了经验研究（Sec. 3.7）中相同的评估指标来衡量准确性，即：Not Found、Precision（精度）、Recall（召回率）和 F1-Score（F1 值）。此外，本章还进行了用户研究以验证 **RQ10**，通过评估用户在有无 TRACER 辅助下查找补丁的用时和准确性。英文版，这里有问题

5.2 RQ6：准确性验证

为了评估 TRACER 的准确性，本小节将 TRACER 与基于启发式的方法和两个工业漏洞数据库（DB_A 和 DB_B）进行比较，并通过人工进一步分析 TRACER 中的误报和误报。

5.2.1 与基于启发式方法对比

本节选择两种广泛使用的启发式方法：（1）检索该漏洞的 NVD 中“Reference”字段中引用信息以获取补丁提交[3–4,27]，（2）在 GitHub 中检索带有 CVE 标识符的代码提交[10,39]。此外，本节将这两种启发式方法结合为第三种启发式方法，即：检索 NVD References 和 GitHub Commits。

表 5-1　TRACER VS. 已有的基于启发式的方法

映射类型	数量	检索 NVD References				检索 GitHub Commits			
		Not Found	Pre.	Rec.	F1	Not Found	Pre.	Rec.	F1
1:1 (SP)	567	285 (50.3%)	0.973	0.986	0.977	472 (83.2%)	0.416	0.642	0.471
1:*i* (MEP)	195	125 (64.1%)	0.932	0.925	0.921	162 (83.1%)	0.472	0.490	0.452
1:*n* (MP)	101	68 (67.3%)	0.980	0.552	0.683	73 (72.3%)	0.536	0.445	0.461
1:*n* (MB)	372	244 (65.6%)	0.979	0.416	0.546	246 (66.1%)	0.445	0.236	0.284
1:*n* (MR)	60	46 (76.7%)	1.000	0.708	0.794	37 (61.7%)	0.627	0.345	0.413
Total	1,295	768 (59.3%)	0.970	0.805	0.842	990 (76.4%)	0.461	0.417	0.386
映射类型	数量	检索 NVD 以及 GitHub				TRACER			
		Not Found	Pre.	Rec.	F1	Not Found	Pre.	Rec.	F1
1:1 (SP)	567	222 (39.2%)	0.839	0.930	0.864	102 (18.0%)	0.860	0.951	0.881
1:*i* (MEP)	195	104 (53.3%)	0.821	0.867	0.820	6 (3.1%)	0.886	0.918	0.888
1:*n* (MP)	101	52 (51.5%)	0.779	0.605	0.647	20 (19.8%)	0.872	0.741	0.761
1:*n* (MB)	372	171 (46.0%)	0.704	0.393	0.465	23 (6.2%)	0.861	0.788	0.795
1:*n* (MR)	60	27 (45.0%)	0.801	0.539	0.604	4 (6.7%)	0.831	0.620	0.659
Total	1,295	576 (44.5%)	0.793	0.732	0.720	155 (12.0%)	0.864	0.864	0.837

表5-1显示了三种启发式方法与 TRACER 的准确率对比结果英文版，这里有问题，可以发现，对于很大一部分 CVE（59.3%、76.4% 和 44.5%），三种启发式方法都无法找到其补丁信息，而仅12.0%CVE 的补丁无法被 TRACER 找到。此外，由于 NVD 引用信息的高置信度，第一种启发式方法找到的补丁具有较高的精度，但对于一对多映射关系的 CVE，召回率较低；第二种启发式方法具有较低的精度和召回率，第三种启发式方式中和了前两种方法的精度和召回率。对比下来，TRACER 的精度比第一种启发式方法更低，但拥有更高的召回率和相似的 F1 值；对于 *MP* 和 *MB* 类型的 CVE，TRACER 的召回率明显更高。考虑到 TRACER 找到补丁的漏洞数比第一种启发式方法多出116.3%，TRACER 中轻微的准确率降低是可以接受的。此外，对比于第二和第三种启发式方法，TRACER 分别将 F1 值提高了116.8%和16.3%。

Highlight：与现有的基于启发式的方法相比，TRACER 将能找到补丁的漏洞数提高58.6%到273.8%，同时，将 F1 值提高116.8%。

5.2.2　与工业漏洞数据库对比

与工业漏洞数据库 DB_A 和 DB_B 进行比较的并非为了证明 TRACER 或是 DB_A 和 DB_B 的优劣，因为在构建漏洞数据库 DB_A 和 DB_B 的过程中涉及的人工工作量并不可知且无法量化，无法进行公平的比较。本节设计的对比是为了评估 TRACER 可以达到的准确性水平和 TRACER 的实用价值，并探索 TRACER 是否可以帮助工作改进或补充现有的工业漏洞数据库。

从表3-2和5-1可以看出，TRACER 为找到补丁的漏洞数比 DB_A 和 DB_B 少12.0%，且精度也分别低了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，也因此比自动化技术拥有更高的准确率。此外，TRACER 具有比 DB_A 和 DB_B 高15.5% 和 18.4%的召回率（尤其是对于一对多映射的 CVE，TRACER 具有更高的召回率），以及比 DB_A 和 DB_B 高5.5% 和 8.6%的 F1 值。结果表明，TRACER 以较低的精度和较少数量未找到补丁的 CVE 为代价，却拥有更为显着召回率。这也说明，TRACER 可用于补充现有漏洞数据库缺失的漏洞数据。

Highlight：TRACER 具有比 DB_A 和 DB_B 高15.5% 和 18.4%的召回率和5.5% 到 8.6%的 F1 值，同时牺牲了6.4%的精度和12.0%CVE 的补丁信息。

5.2.3 TRACER 漏报和误报分析

漏报分析，通过人工分析 TRACER 找不到补丁或遗漏补丁的 CVE 数据，本节共总结出 TRACER 漏报的五个主要原因：（1）对于一些年代久远的 CVE，NVD、Debian 和 Red Hat 中包含的引用信息比较少，有的引用信息甚至是失效的。这种情况下，TRACER 无法构建完整的参考网络。例如，漏洞 CVE-2011-1950，在 NVD[80] 中标记为“补丁（Patch）”和“供应商咨询（Vendor Advisory）”的关键引用链接已经失效。（2）NVD、Debian 和 Red Hat 平台中缺少 CVE 的一些关键引用信息（例如，问题链接（Issue URL）），TRACER 便难以选出正确的补丁。例如，漏洞 CVE-2018-14642，其问题报告[81]不包含在三个咨询来源中的任何一个中，但是，根据此问题报告，我们可以找到补丁[82]。（3）补丁的提交消息与 CVE 描述具有语义相似性，可通过人工识别，但不包含 CVE 标识符，工具难以辨别。因此，TRACER 的知识源扩增步骤也未能捕捉到它。例如，漏洞 CVE-2019-10077[83]，代码提交[84]修复了该漏洞，但没有再提交信息种指明 CVE 标识符。（4）GitHub 平台用于检索代码提交的 REST API 仅返回 1,000 个结果，这会使 TRACER 在知识源扩增步骤种错失正确的补丁提交。（5）TRACER 的补丁精选步骤中，只选择了一个具有最高连通度的补丁，其他已包含在引用信息网络中的正确补丁将会被 TRACER 错失。

误报分析，本节还人工分析了 TRACER 找错补丁的 CVE 数据，并总结了两个主要原因：（1）在讨论和解决漏洞时，相关人员也会引用引入漏洞的代码提交。由于 TRACER 缺乏对引用链接的上下文语义理解，TRACER 错误地将其识别为补丁提交。例如，漏洞 CVE-2020-5249，引入漏洞的代码提交[85]和修复漏洞的代码提交[86]在同一问题报告（Issue Report）的评论中被引用。（2）被引用的页面上列出了多对 CVE 及其问题和补丁信息。由于 TRACER 缺乏语义理解，其他 CVE

的补丁也可能会被 TRACER 错误地识别。例如，漏洞 CVE-2018-15750，其补丁维护在发行说明（Release Note）[87] 中，CVE-2018-15751 的发行说明均被 NVD、Debian 和 RedHat 引用。

Highlight：通过人工分析总结出的五个漏报和两个误报原因，可以用来进一步提高 TRACER 的准确性。

5.3 RQ7：削弱性分析

表5-2展示了削弱性分析的结果，以衡量 TRACER 中的各种设置对于准确性的贡献度。

5.3.1 去除某一知识源

在 TRACER 的步骤一（多源信息网络构建）中，分别删除了四个知识源 NVD、Debian、Red Hat 和 GitHub 中的一个，生成变体 v_1^1、v_1^2、v_1^3 和 v_1^4。可以从表5-2中观察到，这四个变体没有找到补丁的 CVE 数量都在增加。值得注意的是，在精度、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 找到补丁的漏洞数分别比 TRACER 少了30.1%、1.6%、2.2% 和 7.6%。这表明，构建更完整的信息网络，也将取得更好的补丁查找结果。TRACER 中选取的四个知识源都有助于为更多的漏洞找到补丁信息，同时，显然 NVD 和 GitHub 的贡献度最高，重要性也最高。

5.3.2 去除网络构建步骤

在 TRACER 的步骤一（多源信息网络构建）中，不以分层的方式构建信息网络，而是简单地使用包含在四个知识源公告中的直接引用信息。具体实现为：在 TRACER 的步骤一中省略“引用节点分析”步骤，实验结果为表5-2中的变体 v_1^5。可以看到 v_1^5 没有找到补丁的 CVE 数量都在增加。在数值上，v_1^5 找到补丁的漏洞数比 TRACER 少了22.7%，但同时精度高出了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 中直接引用，而是可能隐藏在间接引用信息中，本文构建的信息网络有助于用户以可接受的精度降低为代价而找到隐藏的补丁信息。

5.3.3 去除补丁精选步骤

去除基于连通度和置信度的补丁选择方法在 TRACER 的步骤二（补丁节点精选）中，并非选择具有高置信度和连通度的补丁，而是选择网络中的所有补丁。实验结果为表5-2中的变体 v_2^1，可以看到该变体显着提高了工具的召回率。v_2^1 找到补丁的漏洞数比 TRACER 多了8.8%，特别是对于一对多映射类型的 CVE；同时

精度大幅下降29.3%和 F1 下降了21.4%。该结果表明，引用信息网络中确实包含了大部分正确的补丁，基于启发式的补丁选择方法有助于实现精度和召回之间的平衡。

去除基于连通度或置信度的补丁选择方法在 TRACER 的步骤二（补丁节点精选）中，并非选择具有高置信度和连通度的补丁，而是采用的两个启发式方法之一，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启发式方法时，v_2^2 找到补丁的漏洞数比 TRACER 少了41.7%，但同时精度提升了5.3%，召回率提高了2.9%，F1 值提高了4.1%。这些结果表明，基于连通度的启发式有助于为更多 CVE 找到补丁信息，但同时也会引入少部分噪声。

如果没有基于置信度的启发式方法，v_2^3 的召回率和 F1 值会略有下降，尤其是对于 *MR* 类型的 CVE 影响较大。这些结果表明，基于置信度的启发式方法有助于在所有基数之间实现平衡的准确性。

变更基于连通度的方法设置通过仅考虑路径长度（即：选择到根节点的路径最短的补丁）和仅考虑路径数量（即：选择具有最多路径数的补丁）来削弱基于连通度的启发式方法，分别生成变体 v_2^4 和 v_2^5。v_2^4 和 v_2^5 的精度分别比 TRACER 低了4.3%和5.2%，召回率高了2.1%和1.0%，F1 值降低了3.0%和3.3%。这些结果表明，基于连通性的启发式方法考虑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 的精度。

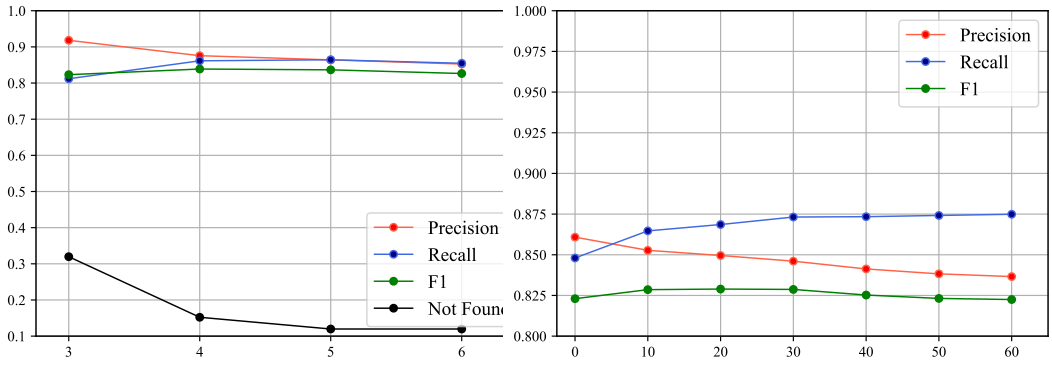
5.3.4 去除补丁扩增步骤

去除 TRACER 的步骤三（候选补丁扩增），为表5-2中的变体 v_3。可以发现，v_3 比 TRACER 的召回率下降了10.8%，F1 值下降了7.3%，尤其是对于一对多映射类型的 CVE，下降更显著。这表明补丁扩增步骤有助于更完整地为漏洞找到多个补丁信息。

Highlight：TRACER 中的知识源、网络构建、补丁选择和补丁扩增步骤都具有一定的重要性，都对 TRACER 补丁查找的准确性做出了贡献。

5.4 RQ8：敏感度分析

TRACER 中有两个可配置的参数，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。验证 **RQ6**、**RQ7**、**RQ9** 和 **RQ10** 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 层。为了评估 TRACER 对这两个参数的敏感性，该小节将先固定一个参数为默认设置，在改变另一个参数的设置，并在深度数据集上重新运



(a) 网络深度限制　　(b) 提交时间跨度

图 5-1　网络深度限制和提交时间跨度的敏感性分析结果

行 TRACER 。网络深度限制具体设置为 3 到 6 层，步长为 1，提交时间跨度具体设置为 0 到 60 天，步长为 10。

图5-1a和5-1b分别显示了两个参数对 TRACER 精度的影响，其中 *x*-axis 表示参数的值，*y*-axis 表示 TRACER 的精度。总体来看，随着网络深度限制的增加，网络中包含了将更多的潜在补丁。TRACER 未发现补丁的漏洞数量和精度都在降低，而召回率和 F1-score 先增加后降低。因此，可以认为网络深度限制设置为 5 层时最优。随着代码提交时间跨度的增加，TRACER 会搜索更广的代码提交。TRACER 的精度降低，召回率提高，F1-score 先升后降。其中，TRACER 未找到补丁的漏洞数量不会改变，所以没有出现在图5-1b中。因此，可以认为提交时间跨度设置为 30 天时最优。

Highlight：总的来说，TRACER 的精度对两个可配置参数的敏感性是可以接受的。

5.5　RQ9：通用性分析

为了进一步评估 TRACER 的通用性，本节还构建了两个漏洞数据集。第一个数据集为：两个漏洞数据库 DB_A 和 DB_B 中只有一个数据库提供补丁的 CVE（图3-2b），共有3,185个 CVE。第二个数据集为：两个漏洞数据库都没有提供补丁的 CVE（图3-2），共有5,468个 CVE。对于第一个数据集中的3,185个 CVE，TRACER 找到了2,155(67.7%)漏洞的补丁，而两个漏洞数据库 DB_A 和 DB_B 仅分别提供了2,190(68.8%)和995(31.2%)漏洞的补丁。在 TRACER 找到补丁的2,155CVE 中，DB_A 和 DB_B 仅提供了其中的1,455和700个 CVE 的补丁。对于第二个数据集中的5,468个 CVE，TRACER 找到了2,816(51.5%)漏洞的补丁，而两个漏洞数据库 DB_A 和 DB_B 没有提供补丁信息。这些结果表明，TRACER 可以通过查找 CVE 的补丁来极大地补充或增强现有的工业漏洞数据库。

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 CVE 中分别采样了100个，并采用“数据准备”（Sec. 3.2）中同样的方式手动找到它们的补丁信息以测量 TRACER 的准确性，表5-3为评估结果。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不能确定补丁的 CVE。在第一个数据集取样的91个 CVE 中，TRACER 的 F1 值为0.784，而 DB_A 的 F1 值较高，而 DB_B 的 F1 值较低。与 Sec. 5.2中的结果类似，漏洞数据库 DB_A 和 DB_B 拥有更高的精度，但召回率也更低。在第二个数据集取样的89CVE 中，DB_A 和 DB_B 不提供补丁信息，而 TRACER 可取得达到0.867的 F1 值。这些结果表明，在更大范围的开源漏洞中，TRACER 都能取得不错的效果，通用性较好。

Highlight：TRACER 在两个附加数据集上可查找到67.7%和51.5%CVE 的补丁，通过采样评估，精度分别为0.823和0.888，召回率分别为0.845和0.899，体现出 TRACER 在查找补丁方面具有较好的通用性。

5.6 RQ10: 实用性能分析

在实践中，为了确保补丁的准确性，即使使用了较准确的自动工具查找补丁，安全专家仍然需要对漏洞补丁进行确认。为了评估 TRACER 在这种使用场景中的实用性，本节邀请了 10 名参与者进行了一项用户研究，他们将分别在有和没有 TRACER 的辅助下找到 10 个 CVE 的补丁。

本文作者从多所大学和科技公司的安全实验室中招募了 10 名参与者，他们中有软件安全方向的博士后、博士生、硕士研究人员以及工程师。本节从深度数据集中随机选择 10 个 CVE 作为实验任务，其中，2 个 CVE 属于 *SP* 类型，3 个 CVE 属于 *MEP* 类型，1 个 CVE 属于 *MP* 类型，4 个 CVE 属于 *MB* 类型。为了公平比较，该用户研究将参与者分为两组（即：A 组和 B 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 TRACER 的辅助下完成前 5 个任务，并不用 TRACER 完成剩余的任务。

表5-4显示了 10 个任务的平均时间消耗和补丁准确性。本节将属于 *SP* 和 *MEP* 的 5 个 CVE 归类为单补丁任务，将属于 *MP* 和 *MB* 的 5 个 CVE 归类为多补丁任务。总体而言，在 TRACER 的帮助下，参与者在完成每个任务中约节省了17.7%时间，并提高了将补丁的准确率在 precision、recall 和 F1-score 方面分别提高了11.7%，35.9%和24.3%。对于 5 个单补丁任务而言，节省的时间较为显著，但对于 5 个多补丁任务来说则不显着。这是因为 TRACER 为多补丁任务返回的信息网络和补丁比单补丁任务更复杂，参与者需要花费更多的时间以理解信息网络以及验证补丁。此外，对于 5 个多补丁任务而言，准确度的提高较为显着，但

对于 5 个单补丁任务则并不显着。从这个意义上说，TRACER 对于具有多个补丁的 CVE 的作用更为明显。

本文作者还采访了每位参与者以获取他们对 TRACER 的使用感受。总的来说，他们都认为漏洞的信息网络具有较高的价值，因为它总结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信息。其中，一家高科技公司的安全工程师评论："信息网络作为一个证据链，有助于定位和验证补丁"[①]。此外，他们还建议 TRACER 添加更多的知识源以及提供更多补丁节点的信息，例如，提交消息和代码差异，这些信息有助于人工审查补丁信息。

Highlight：在实际使用中，TRACER 有助于安全专家更准确、更快速地查找到补丁信息。

5.7 讨论

本节将主要讨论 TRACER 的局限性和重要性。

5.7.1 TRACER 的局限性

首先，TRACER 以 CVE-ID 作为输入，因此该方法可能不适用于没有 CVE-ID 的开源漏洞（即：不在 CVE List 中的开源漏洞），比如一些秘密修补的漏洞[47]。对此，一种可能的解决错失是将 Advisory-ID 作为 TRACER 的输入，因为某些第三方漏洞平台（例如，WhiteSource）可能具有比 NVD 更多的漏洞。

其次，TRACER 仅将 Debian 和 Red Hat 视为次级知识源，但 TRACER 具有较好的可扩展性，它可以在轻松地囊括其他知识源（例如，SecurityFocus[88]）。此外，正如（Sec. xxx）TRACER 漏报和误报分析中所阐述的，缺失 CVE 与补丁间的语义理解和分析将会限制 TRACER 的准确性。

5.7.2 TRACER 的重要性

TRACER 将会有益于安全社区、学术界和工业界的用户。对于安全社区而言，TRACER 可以发现并向 NVD 管理员报告缺少补丁信息的 CVE 漏洞，以提高 CVE 的信息质量并加速 CVE 条目的更新，使得 NVD 平台的用户受益。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[4–5]）和经验研究工作提供补丁数据。它还可以帮助通过分析漏洞补丁信息以确定受影响的库的工作[23]。对于工业界来说，TRACER 可以帮助安全工程师提高工业漏洞数据库的补丁覆盖率和准确性，从而提高软件成分分析的准确性。值

① 原文为：the network graph, as a chain of evidence, is helpful to localize and verify patches

得一提的是，TRACER 已经部署到一家高科技公司。但是，由于保密协议，本文不便分享相关数据。

表 5-2　TRACER VS. 削弱变体

映射类型	数量	TRACER				v_1^1: TRACER w/o NVD			
		Not Found	Pre.	Rec.	F1	Not Found	Pre.	Rec.	F1
1:1 (SP)	567	102 (18.0%)	0.860	0.951	0.881	286 (50.4%)	0.820	0.936	0.846
1:*i* (MEP)	195	6 (3.1%)	0.886	0.918	0.888	79 (40.5%)	0.882	0.935	0.886
1:*n* (MP)	101	20 (19.8%)	0.872	0.741	0.761	41 (40.6%)	0.881	0.728	0.766
1:*n* (MB)	372	23 (6.2%)	0.861	0.788	0.795	84 (22.6%)	0.876	0.780	0.800
1:*n* (MR)	60	4 (6.7%)	0.831	0.620	0.659	8 (13.3%)	0.848	0.551	0.624
Total	1,295	155 (12.0%)	0.864	0.864	0.837	498 (38.5%)	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			
		Not Found	Pre.	Rec.	F1	Not Found	Pre.	Rec.	F1
1:1 (SP)	567	110 (19.4%)	0.847	0.943	0.869	113 (19.9%)	0.853	0.943	0.874
1:*i* (MEP)	195	8 (4.1%)	0.880	0.912	0.882	7 (3.6%)	0.883	0.918	0.886
1:*n* (MP)	101	22 (21.8%)	0.851	0.716	0.739	21 (20.8%)	0.880	0.736	0.760
1:*n* (MB)	372	28 (7.5%)	0.838	0.760	0.771	35 (9.4%)	0.844	0.761	0.767
1:*n* (MR)	60	5 (8.3%)	0.819	0.613	0.651	4 (6.7%)	0.738	0.640	0.618
Total	1,295	173 (13.4%)	0.848	0.849	0.821	180 (13.9%)	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			
		Not Found	Pre.	Rec.	F1	Not Found	Pre.	Rec.	F1
1:1 (SP)	567	149 (26.3%)	0.898	0.943	0.908	177 (31.2%)	0.910	0.972	0.925
1:*i* (MEP)	195	19 (9.7%)	0.887	0.921	0.892	78 (40.0%)	0.956	0.959	0.941
1:*n* (MP)	101	28 (27.7%)	0.873	0.690	0.726	40 (39.6%)	0.943	0.669	0.743
1:*n* (MB)	372	39 (10.5%)	0.874	0.752	0.773	109 (29.3%)	0.908	0.575	0.659
1:*n* (MR)	60	7 (11.7%)	0.816	0.545	0.604	10 (16.7%)	0.920	0.641	0.712
Total	1,295	242 (18.7%)	0.883	0.841	0.835	414 (32.0%)	0.918	0.812	0.823
映射类型	数量	v_2^1: TRACER w/o Selection				v_2^2: TRACER w/o Connectivity			
		Not Found	Pre.	Rec.	F1	Not Found	Pre.	Rec.	F1
1:1 (SP)	567	102 (18.0%)	0.632	0.961	0.680	245 (43.2%)	0.892	0.978	0.913
1:*i* (MEP)	195	6 (3.1%)	0.622	0.976	0.682	111 (56.9%)	0.929	0.939	0.915
1:*n* (MP)	101	20 (19.8%)	0.615	0.933	0.656	56 (55.4%)	0.953	0.685	0.764
1:*n* (MB)	372	23 (6.2%)	0.616	0.903	0.657	191 (51.3%)	0.927	0.787	0.821
1:*n* (MR)	60	4 (6.7%)	0.368	0.891	0.394	27 (45.0%)	0.885	0.722	0.772
Total	1,295	155 (12.0%)	0.611	0.940	0.658	630 (48.6%)	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			
		Not Found	Pre.	Rec.	F1	Not Found	Pre.	Rec.	F1
1:1 (SP)	567	102 (18.0%)	0.860	0.942	0.879	102 (18.0%)	0.833	0.957	0.859
1:*i* (MEP)	195	6 (3.1%)	0.888	0.913	0.889	6 (3.1%)	0.848	0.945	0.867
1:*n* (MP)	101	20 (19.8%)	0.880	0.722	0.751	20 (19.8%)	0.849	0.760	0.742
1:*n* (MB)	372	23 (6.2%)	0.871	0.765	0.784	23 (6.2%)	0.830	0.798	0.770
1:*n* (MR)	60	4 (6.7%)	0.849	0.462	0.550	4 (6.7%)	0.652	0.747	0.590
Total	1,295	155 (12.0%)	0.869	0.844	0.826	155 (12.0%)	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			
		Not Found	Pre.	Rec.	F1	Not Found	Pre.	Rec.	F1
1:1 (SP)	567	102 (18.0%)	0.805	0.951	0.837	102 (18.0%)	0.871	0.948	0.889
1:*i* (MEP)	195	6 (3.1%)	0.849	0.920	0.858	6 (3.1%)	0.910	0.914	0.902
1:*n* (MP)	101	20 (19.8%)	0.801	0.756	0.726	20 (19.8%)	0.873	0.696	0.732
1:*n* (MB)	372	23 (6.2%)	0.833	0.811	0.791	23 (6.2%)	0.860	0.506	0.590
1:*n* (MR)	60	4 (6.7%)	0.789	0.630	0.644	4 (6.7%)	0.847	0.567	0.629
Total	1,295	155 (12.0%)	0.819	0.873	0.809	155 (12.0%)	0.873	0.771	0.776

表 5-3 Generality of TRACER over Two Additional Datasets

Dataset	Number	TRACER			DB_A				DB_B			
		Pre.	Rec.	F1	Not Found	Pre.	Rec.	F1	Not Found	Pre.	Rec.	F1
First Dataset	91	0.823	0.845	0.784	29 (31.8%)	0.935	0.827	0.858	62 (68.1%)	0.885	0.664	0.725
Second Dataset	89	0.888	0.899	0.867	–	–	–	–	–	–	–	–

表 5-4 用户研究中 10 个任务的用时和准确率对比结果

Approach	All 10 Tasks				5 Single-Patch Tasks				5 Multiple-Patches Tasks		
	Time (mins)	Pre.	Rec.	F1	Time (mins)	Pre.	Rec.	F1	Time (mins)	Pre.	Rec.
w/o TRACER	5.66	0.880	0.677	0.765	5.60	0.960	0.960	0.960	5.72	0.800	0.393
with TRACER	4.66	0.983	0.920	0.951	3.84	1.000	1.000	1.000	5.48	0.967	0.840

第 6 章 总结

6.1 本文总结

本文首先从补丁覆盖率、补丁一致性、补丁类型、漏洞与补丁之间的映射基数以及补丁准确性五个方面进行了一项经验研究，以了解两个工业漏洞数据库中开源软件漏洞补丁的质量和特征。

受经验研究结果的启发，本文提出一种名为 TRACER 的自动化补丁查找方法，以从多个知识源中查找开源软件漏洞的补丁。本文还设计了大量实验验证了 TRACER 的准确性、通用性、实用性等方面。TRACER 的源代码和所有实验数据已经在`https://patch-tracer.github.io`网站上发布。

参考文献

[1] CENTER S C R. Open source security and risk analysis report[EB/OL]. 2021. https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/rep-ossra-2021.pdf.

[2] SNYK. State of the open source security report[EB/OL]. 2019. https://snyk.io/wp-content/uploads/sooss_report_v2.pdf.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] PASHCHENKO I, PLATE H, PONTA S E, et al. Vuln4real: A methodology for counting actually vulnerable dependencies[J]. IEEE Transactions on Software Engineering, 2020.

[10] Wang Y, Chen B, Huang K, et al. An empirical study of usages, updates and risks of third-party libraries in java projects[C]//ICSME. 2020: 35-45.

[11] Cve list[EB/OL]. https://cve.mitre.org/cve/.

[12] National vulnerability database[EB/OL]. https://nvd.nist.gov.

[13] Black duck[EB/OL]. https://www.synopsys.com/content/dam/synopsys/sig-assets/datasheets/bdknowledgebase-ds-ul.pdf.

[14] Whitesource[EB/OL]. https://www.whitesourcesoftware.com/vulnerability-database/.

[15] Veracode[EB/OL]. https://sca.veracode.com/vulnerability-database/search.

[16] Snyk[EB/OL]. https://snyk.io/vuln.

[17] 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.

[18] FAN J, LI Y, WANG S, et al. Ac/c++ code vulnerability dataset with code changes and cve summaries[C]//Proceedings of the 17th International Conference on Mining Software Repositories. 2020: 508-512.

[19] JIMENEZ M, LE TRAON Y, PAPADAKIS M. Enabling the continous analysis of security vulnerabilities with vuldata7[C]//Proceedings of the IEEE International Working Conference on Source Code Analysis and Manipulation. 2018: 56-61.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] ZHANG H, QIAN Z. Precise and accurate patch presence test for binaries[C]// Proceedings of the 27th USENIX Security Symposium. 2018: 887-902.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] Debian security tracker[EB/OL]. https://security-tracker.debian.org/tracker/.

[41] Red hat bugzilla[EB/OL]. https://bugzilla.redhat.com/.

[42] MITRE. About cve[EB/OL]. 2021. https://cve.mitre.org/.

[43] 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.

[44] 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.

[45] CHEN Y, SANTOSA A E, SHARMA A, et al. Automated identification of libraries from vulnerability data[C]//Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice. 2020: 90-99.

[46] 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.

[47] XU Z, CHEN B, CHANDRAMOHAN M, et al. Spain: security patch analysis for binaries towards understanding the pain and pills[C]//Proceedings of the IEEE/ACM 39th International Conference on Software Engineering. 2017: 462-472.

[48] ZAMAN S, ADAMS B, HASSAN A E. Security versus performance bugs: a case study on firefox[C]//Proceedings of the 8th working conference on mining software repositories. 2011: 93-102.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

[54] [EB/OL]. https://github.com/BorisMoore/jsrender/commit/f984e139deb0a7648d5b543860ec652c21f6dcf6.

[55] [EB/OL]. https://github.com/mpdavis/python-jose/commit/89b46353b9f611e9da38de3d2fedf52331167b93.

[56] [EB/OL]. https://github.com/mpdavis/python-jose/commit/73007d6887a7517ac07c6e755e494baee49ef513.

[57] [EB/OL]. https://svn.apache.org/viewvc?view=revision&revision=1373762.

[58] [EB/OL]. https://github.com/apache/james-hupa/commit/aff28a8117a49969b0fc8cc9926c19fa90146d8d.

[59] [EB/OL]. https://github.com/apache/deltaspike/commit/4e2502358526b944fc5514c206d306e97ff271bb.

[60] [EB/OL]. https://github.com/apache/deltaspike/commit/72e607f3be66c30c72b32c24b44e9deaa8e54608.

[61] [EB/OL]. https://github.com/apache/deltaspike/commit/d95abe8c01d256da2ce0a5a88f4593138156a4e5.

[62] [EB/OL]. https://github.com/django/django/commit/103ebe2b5ff1b2614b85a52c239f471904d26244.

[63] [EB/OL]. https://github.com/django/django/commit/36f580a17f0b3cb087deadf3b65eea024f479c21.

[64] [EB/OL]. https://github.com/django/django/commit/092cd66cf3c3e175acce698d6ca2012068d878fa.

[65] [EB/OL]. https://github.com/django/django/commit/11c5e0609bcc0db93809de2a08e0dc3d70b393e4.

[66] [EB/OL]. https://github.com/libimobiledevice/libimobiledevice/commit/df1f5c4d70d0c19ad40072f5246ca457e7f9849e.

[67] [EB/OL]. https://github.com/libimobiledevice/libusbmuxd/commit/4397b3376dc4e4cb1c991d0aed61ce6482614196.

[68] Nvd data feeds[EB/OL]. https://nvd.nist.gov/vuln/data-feeds.

[69] Debian repository for advisories[EB/OL]. https://salsa.debian.org/security-tracker-team/security-tracker/-/tree/master/data/CVE.

[70] Webservice api of red hat[EB/OL]. https://bugzilla.redhat.com/docs/en/html/api/index.html.

[71] [EB/OL]. https://github.com/onelogin/ruby-saml/commit/048a544730930f86e46804387a6b6fad50d8176f.

[72] [EB/OL]. https://www.jenkins.io/security/advisory/2019-09-25/#SECURITY-1573.

[73] [EB/OL]. https://issues.apache.org/jira/browse/THRIFT-4647.

[74] Github rest api[EB/OL]. https://docs.github.com/en/rest/reference/search#search-commits.

[75] Github rest api[EB/OL]. https://docs.github.com/en/rest/reference/repos#list-branches.

[76] Github rest api[EB/OL]. https://docs.github.com/en/rest/reference/repos#list-commits.

[77] [EB/OL]. https://github.com/onelogin/ruby-saml/commit/d7ce607d9f9d996e1046dde09b675c3cf0c01280.

[78] [EB/OL]. https://github.com/onelogin/ruby-saml/commit/a35f7251b86aa3b7caf4a64d8a3451f925e8855c.

[79] [EB/OL]. https://github.com/onelogin/ruby-saml/commit/03af9e33d2d87f4ac9a644c5b0981ded4dca0bb8.

[80] [EB/OL]. https://nvd.nist.gov/vuln/detail/CVE-2011-1950.

[81] [EB/OL]. https://issues.redhat.com/browse/UNDERTOW-1430.

[82] [EB/OL]. https://github.com/undertow-io/undertow/commit/c46b7b49c5a561731c84a76ee52244369af1af8a.

[83] [EB/OL]. https://nvd.nist.gov/vuln/detail/CVE-2019-10077.

[84] [EB/OL]. https://github.com/apache/jspwiki/commit/87c89f0405d6b31fc165358ce5d5bc4536e32a8a.

[85] [EB/OL]. https://github.com/ethereum/go-ethereum/commit/fb9f7261ec51e38eedb454594fc19f00de1a6834.

[86] [EB/OL]. https://github.com/ethereum/go-ethereum/commit/83e2761c3a13524bd5d6597ac08994488cf872ef.

[87] [EB/OL]. https://docs.saltstack.com/en/latest/topics/releases/2018.3.3.html.

[88] Securityfocus[EB/OL]. https://www.securityfocus.com.

致谢

致谢致谢致谢