学校代码：10246
学　　号：19212010035

復旦大學

硕 士 学 位 论 文

（学术学位）

基于多源知识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 18 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

本章节概述了背景、研究目的与意义[1–2]。

1.1 研究背景

1.2 本文工作概述

1.3 本文篇章结构

如图1-1所示。如表1-1所示。

表 1-1 表格名称

AA \ BB	C1	C2
R1	1	2
R2	3	4

图 1-1 在上海混口饭吃真不容易啊

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)[2]，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 漏洞通告【todo】

漏洞通告（advisory），也称漏洞警报。。。。是指。。。官方或第三方？再顺路引出相关的 sources，作为例子。

2.1.3 漏洞补丁【todo】

是指。。。。.Patch Git, SVN commit

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

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[3]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[4]和 Dashevskyi 等人[5]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[6]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[7]识别受漏洞影响的开源库信息。Chaparro 等人的工作[8]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[9]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[10]。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[11–13]、补丁的存在性测试[14–16]以及秘密补丁识别[17–20]。

此外，研究人员也已为 Java[21]、C/C++[22]以及特定开源项目[23]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[24–27]。在这些工作中，补丁信息多由人工识别[13,15–21,24]，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[12,22–23,25–26]，以及在提交信息（commit message）中搜索 CVE 标识符[22–23,27]。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[28–29]，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[30–33]，以及通过学习漏洞特征[34–37]、通过匹配漏洞签名[38–39]、通过匹配漏洞和补丁检测签名[40–42]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[30–32,41]、基于启发式规则的方法[29,33–35,37]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[38–40]。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【todo】

precision、recall。。。

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[21–23,25–27,43–44]，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[45]、WhiteSource[46]、Veracode[47] 和 Snyk[48] 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。

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.”

- **Snyk**[①]，该公司声称：漏洞数据库[②]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。

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。

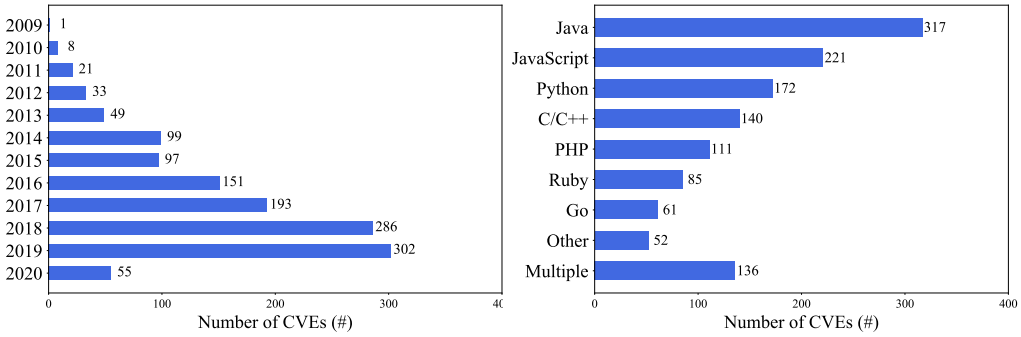
在该深度数据集的构建过程中，为了确保数据集能够涵盖足够多的漏洞用以实验评估，但又不至于在人工识别补丁的阶段产生难以完成的工作量，本文仅将在 DB_A 和 DB_B 都含有补丁信息的漏洞列入该深度数据集，最终，该深度数据集共包含1,417个 CVE 漏洞。

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[49] 标识为其补丁，两

① https://snyk.io/vuln

② https://snyk.io/product/vulnerability-database/

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。

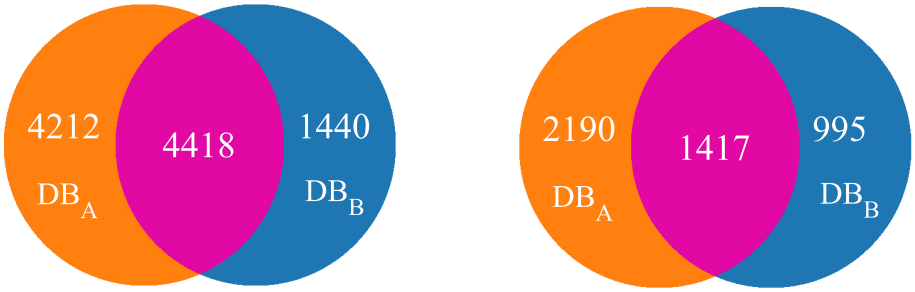


(a) 漏洞年份分布统计　　(b) 程序语言分布统计

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[50]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

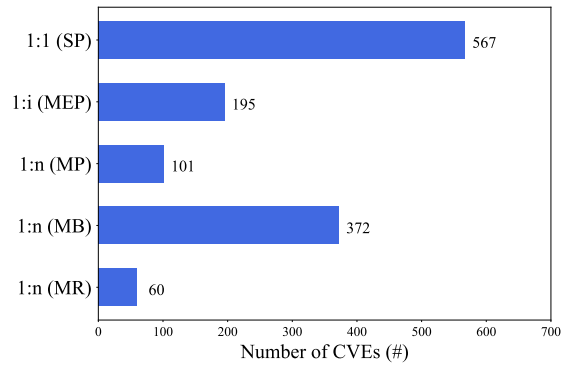


图 3-3 Mapping Cardinalities between CVEs and Patches

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[51]是拉取请求提交（requested commit），python-jose@73007d[52]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[53]与 GitHub commit james-hupa@aff28a[54]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[55]、deltaspike@72e607[56]和 deltaspike@d95abe[57] 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[58]、django@36f580[59]、django@092cd6[60]和 django@11c5e0[61]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[62]和 libusbmuxd@4397b3[63]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。

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

3.7 RQ5：补丁准确性分析

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—补丁定位方法

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

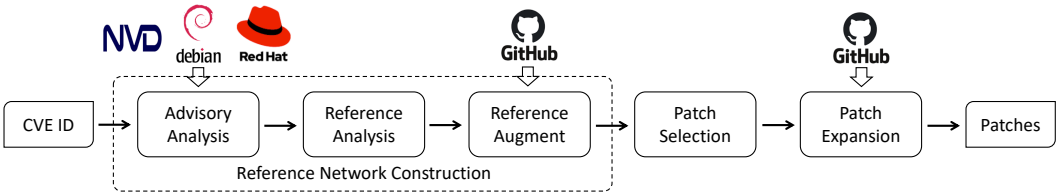


图 4-1　TRACER 方法概览

基于上文经验研究的发现，本文提出了一种名为 TRACER 的自动化方法来查找来源软件漏洞的补丁（Github Commit 形式）。TRACER 的基本思想是：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。

图4-1展示了 TRACER 的方法概览。TRACER 以漏洞的 CVE 标识符作为输入，最终返回其补丁信息。具体分为三步：首先，TRACER 从多个信息源（即：NVD、Debian[64]、RedHat[65]和 GitHub）为输入的 CVE 构建一个相关引用链接的信息网络。该步骤的目的是将 CVE 在报告、讨论和解决阶段的引用链接信息进行建模。这里将 NVD 视为主信息来源，将 Debian、RedHat 和 GitHub 视为次级信息来源，该级信息源可以进一步扩展。其次，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（即：commit）作为该 CVE 的补丁。最后，TRACER 通过搜索同一存储库中其他分支上的相关提交来扩展补丁集。该步骤目的是在 CVE 及其补丁之间建立潜在的一对多映射关系。在本章的其他小节中，将详细阐述每个步骤。

4.2　步骤一：构建多源引用信息网络

TRACER 的步骤一共包括三个子步骤，前两个子步骤为漏洞公告分析和引用分析，通过分析来自 NVD、Debian 和 Red Hat 三个信息源的漏洞公告构建初始

引用信息网络；第三个子步骤为信息增强，通过从 GitHub 搜索相关提交链接来扩充信息网络。

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，然后再添加三个漏洞公告源节点（即：NVD、Debian 和 Red Hat）作为 root 的子节点。这些公告源节点用于追溯最终选定的补丁节点的来源。

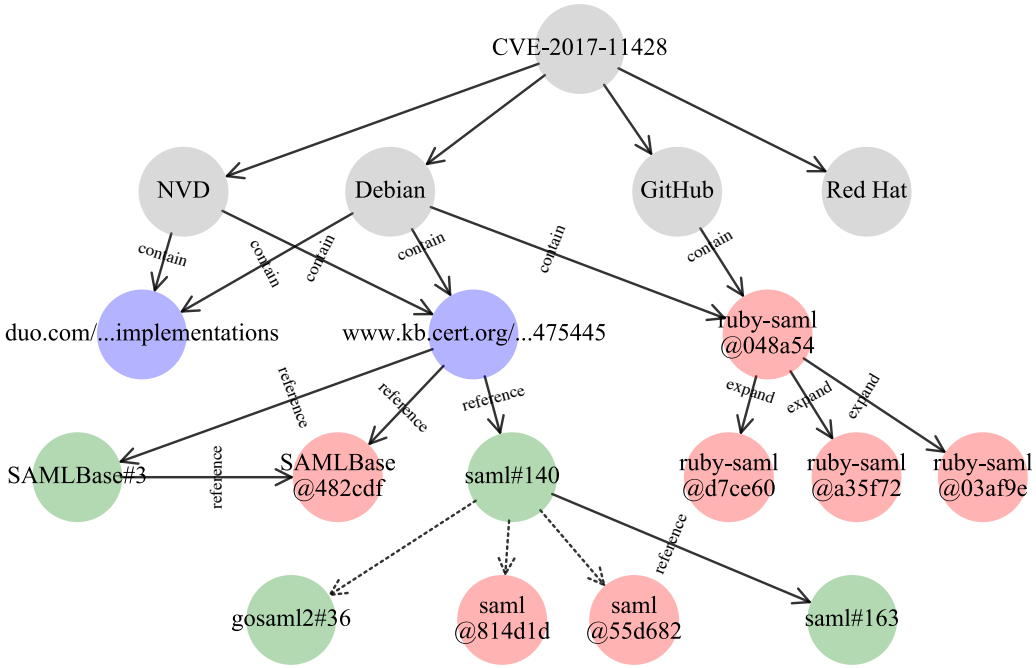


图 4-2　样例 CVE-2017-11428 的多源信息网络

Example 4.1　图4-2为样例 CVE-2017-11428 的完整的多源引用信息网络。其中，顶层显示根节点，第二层显示公告来源节点。

然后，TRACER 基于 CVE-ID 分别获取 NVD、Debian 和 Red Hat 的漏洞公告。其中，NVD 平台以 JSON 形式按年份提供所有漏洞的结构化数据[66]，TRACER 通过下载并解析相应的 JSON 文件即可获得 NVD 中该 CVE 的信息。Debian 平台的漏洞公告存储在仓库[67]中，TRACER 可直接从中解析得 Debian 提供的该 CVE。Red Hat 平台提供了 WebService API[68]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 仅会跟踪部分 CVE 漏洞。

TRACER 从每个信息源的漏洞公告中提取引用信息（即：URL），并将它们添加为相应公告源节点的子节点。对于 NVD 公告，TRACER 从“references”字段中提取出相关 URL；类似地，对于 Debian 公告，TRACER 会从“Notes”字段中提取

出相关 URL；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，这是因为开发人员会在评论区讨论和记录漏洞的解决过程，并可能列出对补丁信息。

Example 4.2 如图4-2中的第三层所示，对于 CVE-2017-11428，NVD 中包含了两个引用链接。链接一引用是对描述此漏洞细节的博客的引用，链接一引用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，不过该公告还包含对修复此漏洞的 GitHub 提交链接 ruby-saml@048a54[69] 的引用。此外，Red Hat 平台并未收录该 CVE。

TRACER 将引用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，是因为该方法的目标是为 CVE 找到补丁。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；此外，问题追踪系统（Issue Tracker）中的报告会为 CVE 分配一个标识符（issue-id），开发人员常常会将 issue-id 写入补丁提交信息（即：commit message）中。未识别为补丁或问题节点的引用链接节点将被视为混合节点，它们多为博客、第三方漏洞公告等网页链接。

对于补丁节点的识别，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 SVN 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。

对于问题节点的识别，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。

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构建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。

如果该引用节点的类型为补丁节点，TRACER 会通过网络请求该提交（commit）的信息并分析该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，则该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，如果文件的后缀不在列表[①]中则判定为非源代码文件。

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 并获取网页信息（即：HTML 文本），并解析出该网页中引用的 URL 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）相同的方式检查提取的 URL 引用，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，在该步骤以及后续层的网络中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。引用信息网络中仅包含直接在 NVD、Debian 和 Red Hat 中被引用的混合节点。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，这会给参考网络带来过多的噪音，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存储库中的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以扩增网络，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引用任何问题或提交信息，另一个混合节点了引用两个问题链接 SAMLBase#3 和 saml#140 以及一个提交链接 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 以及两个提交链接 saml@814d1d 和 saml@55d682。考虑到 gosaml2#36 与 saml#140 并不属于同个的代码仓库，saml@814d1d 和 saml@55d682 也仅涉及测试代码的更改，TRACER 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，但通过虚线箭头线连接。

① todo, 展示常用 list

4.2.3　信息源扩增

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，因为补丁提交通常隐藏在代码仓的提交历史中。因此，在此子步骤中，TRACER 将搜索代码托管平台以获取 CVE 漏洞的补丁提交（即：commit），通过扩增信息源以进一步扩增构建的信息网络。

受经验研究中补丁类型分析（Sec.3.5）的启发，因为大多数补丁都属于 GitHub 提交的类型（93.7% 的补丁都为 GitHub commit 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 漏洞分配一个问题标识符（Issue—id）；同样，软件厂商通常也会为 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了 SECURITY-1573[70] 的标识符，问题跟踪系统为该漏洞分配了 THRIFT-4647[71] 问题标识符。对此，TRACER 使用正则表达式①分别从已构建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[72] 接口全站搜索相关的提交。为了减少噪音，对于 API 返回的提交，TRACER 首先会检查该提交的仓库信息是否与 CVE 漏洞的 CPE 信息匹配。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，可以直接从 NVD 的 JSON 文件中解析得到。对于提交的仓库信息与 CPE 信息匹配的操作，TRACER 遵循 Dong 等人的匹配准则[6] 以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 信息源节点的子节点加入网络。

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 使用 CVE 标识符（CVE-id）来搜索 GitHub 提交。该搜索返回的提交保罗 ruby-saml@048a54，其仓库信息是“onelogin: ruby-saml”；此 CVE 的 CPE 是“onelogin: ruby-saml”，因此实现了名称的完全匹配，TRACER 将会把该节点加入网络。由于此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 源节点的子节点，如图4-2。此外，该次搜索结果中没有其他匹配的提交。

① add 表达式

4.3 步骤二：精选补丁节点

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 基于置信度选择补丁节点

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[12,34–35]都是基于这种启发式来查找补丁。

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时，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 名称可以成功匹配。这种方法也已有的很多工作[29,33]中使用。

Example 4.6 在图4-2示例 CVE-2017-11428 的引用信息网络中，TRACER 将直接选择补丁节点 ruby-saml@048a54，因为它是 GitHub 源节点的子节点，具有较高的置信度作为 CVE-2017-11428 的正确补丁。实际上，该提交也是正确的补丁之一。

4.3.2 基于连通度选择补丁节点

仅仅使用基于置信度的启发式方法通常不足以准确且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连接到网络中的根节点（即：图中的 CVE ID 节点）。因此，本文还设计了基于连通性选择补丁节点的启发式方法。

具体来说，TRACER 从两个角度量化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 包含 NVD 和 GitHub 节点，则路径长度减 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。

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：补丁扩增

受到经验研究中映射分析（Sec.3.6）的启发，TRACER 的步骤三是通过搜索同一存储库中其他分支上的相关提交来扩展在步骤二中选出的补丁集。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为 GitHub Commit 类型。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 类型的补丁，TRACER 将首先定位其代码仓库，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 commit。

具体来说，对于 GitHub Commit 类型的选定补丁，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，TRACER 首先使用 GitHub 的 REST API[73]检索存储库中的所有分支，对于每个分支，TRACER 会通过 GitHub 的 REST API[74]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，是考虑到工具性能和准确性的平衡。然后，对于每个检索到的提交，TRACER 使用以下两个标准来确定该提交是否是正在分析的 CVE 的补丁：一是检索到的提交的提交消息（Commit Message）与已选补丁的提交消息（Commit Message）相同或是包含关系；二是检索到的提交的提交消息（Commit Message）包含 CVE ID、Advisory ID 或 Issue ID。如果

检索到的提交满足两个条件之一，TRACER 会将其作为扩展补丁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输入 CVE 的补丁。此外，TRACER 还提供该 CVE 的引用信息网络，以便于工作人员追溯并确认返回的补丁信息。

Example 4.8 对于在步骤二中为 CVE-2017-11428 选定补丁 ruby-saml@048a54（位于主分支），TRACER 通过步骤三查找到了三个提交—ruby-saml@d7ce60[75]、ruby-saml@a35f72[76]和 ruby-saml@03af9e[77]。它们与 ruby-saml@048a54 具有相同的提交信息（Commit Message），但分别位于分支 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。实际上，这四个补丁也都是 CVE-2017-11428 的正确补丁，且这四个补丁的代码更改并不相同。经验研究3中的数据库 DB_A 和 DB_B 都只提供了一个补丁信息，即：ruby-saml@048a54。

第 5 章 实验验证及结果分析

本章节实验评估。

5.1 实验设计

5.2 RQ6：准确性验证

5.3 RQ7：削弱性分析

5.4 RQ8：敏感度分析

5.5 RQ9：通用性分析

5.6 RQ10: 实用性能分析

5.7 讨论

第 6 章 相关工作

第 7 章 总结与展望

本章节总结与展望。

7.1 本文总结

本文总结本文总结本文总结本文总结本文总结本文总结本文总结本文总结

7.2 未来展望

未来展望未来展望未来展望未来展望未来展望未来展望未来展望未来展望

参考文献

[1] 贾培养, 孙鸿宇, 曹婉莹, 等. 开源软件漏洞库综述[J]. 信息安全研究, 2021: 566-574.

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

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

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

致谢

致谢致谢致谢