软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(1):19−37 [doi: 10.13328/j.cnki.jos.006935]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

开源软件漏洞感知技术综述*

詹 奇[1], 潘圣益[1], 胡 星[2], 鲍凌峰[1], 夏 鑫[3]

[1](浙江大学 计算机科学与技术学院, 浙江 杭州 310027)

[2](浙江大学 软件学院, 浙江 宁波 315048)

[3](华为技术有限公司 软件工程应用技术实验室, 浙江 杭州 310053)

通信作者: 胡星, E-mail: xinghu@zju.edu.cn

摘 要: 随着现代软件规模不断扩大, 软件漏洞给计算机系统和软件的安全运行、可靠性造成了极大的威胁, 进而给人们的生产生活造成巨大的损失. 近年来, 随着开源软件的广泛使用, 其安全问题受到广泛关注. 漏洞感知技术可以有效地帮助开源软件用户在漏洞纰漏之前提前感知到漏洞的存在, 从而进行有效防御. 与传统软件的漏洞检测不同, 开源漏洞的透明性和协同性给开源软件的漏洞感知带来巨大的挑战. 因此, 有许多学者和从业人员提出多种技术, 从代码和开源社区中感知开源软件中潜在的漏洞和风险, 以尽早发现开源软件中的漏洞从而降低漏洞所带来的损失. 为了促进开源软件漏洞感知技术的发展, 对已有研究成果进行系统的梳理、总结和点评. 选取 45 篇开源漏洞感知技术的高水平论文, 将其分为 3 大类: 基于代码的漏洞感知技术、基于开源社区讨论的漏洞感知技术和基于软件补丁的漏洞感知技术, 并对其进行系统地梳理、归纳和总结. 值得注意的是, 根据近几年最新研究的总结, 首次提出基于开源软件漏洞生命周期的感知技术分类, 对已有的漏洞感知技术分类进行补充和完善. 最后, 探索该领域的挑战, 并对未来研究的方向进行展望.

关键词: 开源软件; 漏洞感知; 软件安全

中图法分类号: TP311

中文引用格式: 詹奇, 潘圣益, 胡星, 鲍凌峰, 夏鑫. 开源软件漏洞感知技术综述. 软件学报, 2024, 35(1): 19–37. http://www.jos.org.cn/1000-9825/6935.htm

英文引用格式: Zhan Q, Pan SY, Hu X, Bao LF, Xia X. Survey on Vulnerability Awareness of Open Source Software. Ruan Jian Xue Bao/Journal of Software, 2024, 35(1): 19–37 (in Chinese). http://www.jos.org.cn/1000-9825/6935.htm

Survey on Vulnerability Awareness of Open Source Software

ZHAN Qi[1], PAN Sheng-Yi[1], HU Xing[2], BAO Ling-Feng[1], XIA Xin[3]

[1](College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China)

[2](School of Software Technology, Zhejiang University, Ningbo 315048, China)

[3](Software Engineering Application Technology Lab, Huawei Technologies Co. Ltd., Hangzhou 310053, China)

Abstract: As the modern software scale expands, software vulnerabilities bring a great threat to the security and reliability of computer systems and software, causing huge damage to people's production and life. In recent years, as open source software (OSS) is widely used, the vulnerability issues of OSS have received much attention. Vulnerability awareness techniques can effectively help OSS users to identify vulnerabilities at the early stage for timely defense. Different from the vulnerability detection techniques for traditional software, the transparency and cooperativity of OSS vulnerabilities bring great challenges to vulnerability awareness. Therefore, various techniques are proposed by scholars and developers to perceive potential vulnerabilities and risks in OSS from the code and open source community,

* 基金项目: 国家重点研发计划 (2021YFB2701102); 国家自然科学基金 (62141222, U20A20173); 中央高校基本科研专项资金 (226-2022-00064)
收稿时间: 2022-10-21; 修改时间: 2023-01-13, 2023-02-28; 采用时间: 2023-03-15; jos 在线出版时间: 2023-08-23
CNKI 网络首发时间: 2023-08-28

so as to find OSS vulnerabilities as early as possible and reduce the losses caused by the vulnerabilities. To boost the development of OSS vulnerability awareness techniques, this study conducts a systematic literature review of existing research works. The study selects 45 high-level papers on open source vulnerability awareness techniques, including code-based, open source community discussion-based, and patch-based vulnerability awareness techniques. The results of these papers are systematically summarized. Especially, this study proposes the category of techniques based on the OSS vulnerability life cycle for the first time according to the most recent publications, which supplements and improves the existing taxonomy of vulnerability awareness techniques. Finally, the study discusses the challenges in the field and predicts future research direction.

Key words: open source software (OSS); vulnerability awareness; software security

1 引 言

当前, 信息技术发展正在进入人类社会、物理世界和信息社会融合发展的泛在时代. 开源软件本身往往涉及诸多其他开源软件. 这些开源软件彼此组合、依赖, 连同为各个开源软件做贡献的维护者和开发者、上游社区、源码包、二进制包、包管理器、存储仓库等共同形成一个包含上万个节点的供应关系网络, 即开源软件供应链. 因此开源软件生态的安全运行和演化依赖于开源软件供应链的漏洞管理.

软件漏洞是指在软件的规范、开发或配置中出现错误, 并导致软件执行违反一定安全策略的情况[1]. 随着软件技术的不断发展, 软件漏洞也愈发成为不容低估的问题, 如果不及时发现处理, 会对人们的生产生活和社会安全造成极大的破坏. 具体而言, 软件漏洞可能会导致用户隐私 (如密码、手机号、验证码) 泄露、服务端程序无法正常工作乃至崩溃、提供错误的服务和权限认证或执行任意攻击者指定程序, 进而造成用户和服务商的经济损失和安全风险. 2022 年 1 月, 中国信息通信研究院发布《中国网络安全产业白皮书》, 其中提到: “2020 年我国网络安全产业规模较 2019 年增长 10.6%”, 同时指出在产业链上游, 我国操作系统、数据库等软件系统方面的技术基础仍然较为薄弱, 也证实了软件安全问题日益突出, 并且受到学术界和工业界的广泛关注.

正是由于漏洞在软件中广泛存在和其巨大风险, 学界和工业界提出了各种技术来检测代码漏洞. 总体而言, 早期学术界和工业界往往根据软件本身进行漏洞扫描, 主要采用静态分析与动态分析技术来探测漏洞. 静态分析技术基于对源代码的静态语义理解和对漏洞的抽象表征, 通过分析静态语义满足漏洞抽象得出该代码存在某种特定漏洞. 动态分析方法则是通过捕捉在给定输入下程序实际运行的各种属性来判断是否含有漏洞. 而随着机器学习和深度神经的发展与兴起, 基于机器学习和深度学习的技术逐渐被用于漏洞挖掘. 研究者从源代码中提取各种不同的特征, 将源代码最终抽象为向量表示, 使用训练得到的能“理解”程序漏洞的模型用于预测.

近年来, 开源软件被广泛应用于现代软件开发. 开源软件具有代码公开、易获取的特点, 因此可以突破技术壁垒, 推动创新. 然而, 正因为其透明开放的特点, 同时也会导致漏洞更易被发现、传播和利用, 给软件开发者、使用者带来更大的安全风险. 在现有项目的新版本不断发布以及全新项目创建的推动下, 开源库的全球供应持续呈指数级增长. 随着开源供应链的广泛使用, 开源供应链安全漏洞问题也随之而来. 在传统软件供应链中, 攻击者利用公开披露的漏洞来直接攻击软件系统. 然而, 在开源软件供应链中, 攻击者不再等待公开的漏洞披露来进行攻击, 而是主动将新漏洞注入全球供应链提供支持的开源软件中, 然后在这些漏洞被发现之前利用它们. 通过转移攻击“上游”, 攻击者可以获得影响力和关键的时间优势, 使恶意软件能够在整个供应链中传播, 从而对“下游”用户进行更具扩展性的攻击. 根据 Synopsys 发布的《2022 开源安全和风险分析报告》, 81% 的被审代码库中包含至少一个漏洞[2]. 以 Log4Shell 漏洞 (CVE-2021-44228), 即一个 Apache 服务器软件中开源软件 Log4j 库中的漏洞为例. 该漏洞利用 Log4j 允许对任意 LDAP (lightweight directory access protocol) 和 JNDI (Java naming and directory interface) 服务器发起请求的特点, 使得攻击者可以在服务器或其他计算机上执行任意 Java 代码. 调查显示, Log4Shell 漏洞自 2013 年就已经存在, 直到 2021 年才被阿里云团队发现. 在此期间, 大量服务商 (如亚马逊, 腾讯等) 都受到该漏洞的影响. 据 Wiz 等人[3]统计, 该漏洞使得 93% 的企业云端环境面临安全风险, 造成了大量损失. 因此, 及时感知到开源软件中的漏洞风险, 可以尽可能地缩短软件暴露在漏洞风险下的时间窗口, 进而进行主动防御. 而随着开源软件的兴起, 传统的技术[4,5]无法及时感知开源社区中已经暴露的大量漏洞, 我们总结了以下两方面原因.

一方面, 开源软件的透明性 (transparency) 使得源代码及所有开发活动中产生的元数据 (如缺陷报告、代码变更的提交记录) 都公开可获取, 这使得攻击者更容易发现开源软件中的漏洞并构建可行的攻击. 此外, 开源软件的协同开发的特质使得统一的安全规范和准则难以实践, 而开发者的安全知识、经验和风险意识通常参差不齐, 且与开源软件漏洞信息相关的讨论可能以多种方式在不同的频道开展 (如缺陷报告、问答网站), 这进一步增加了软件开发者和维护者及时获取并避免漏洞信息扩散的难度, 而使得潜在的攻击者受益 (利用泄露的信息在漏洞的生命周期早期发起攻击). 为应对上述开源软件中特有的安全挑战, 开源软件社区需要能够尽早感知潜在的漏洞, 从而采取即时的消减措施以尽可能缩小将漏洞暴露给潜在攻击者的时间窗口. 而传统的代码分析技术开销大, 误报率高, 并且开源软件迭代频繁, 导致这些技术在开源软件漏洞感知上能力有限. 这要求研究者在现有从源代码角度利用各种代码分析执行技术检测漏洞的基础上, 也要基于开源软件的特性, 能够利用其开发过程中公开可见的各类元数据 (如缺陷报告、安全补丁) 及时感知漏洞.

另一方面, 开源软件广泛采用协同漏洞披露政策 (coordinated vulnerability disclosure, CVD), 其要求漏洞相关信息在正式披露前对公众隐藏, 以留出时间让项目管理者及时采取消减和修复措施. 开源软件用户通过安全披露平台 (例如 national vulnerability database, NVD) 获取新漏洞信息, 并应用发布的补丁来消除安全隐患. 然而在实践中, 部分开源社区开发者未能严格遵守 CVD (缺乏软件安全的实践经验或风险管理的意识[6]), 如在公开的缺陷报告中报告新发现的漏洞[7], 在提交日志中提及漏洞相关信息等[8], 这些泄露的漏洞信息可能被潜在的攻击者利用, 从而在公众未察觉前发动危险的零日攻击 (zero-day attack). 除此之外, 为应对攻击者的攻击, 开源软件用户同样需要尽早感知潜在的安全漏洞来以提前防御. 此外, 漏洞信息的公开披露可能存在较大的延迟甚至被刻意隐瞒 (如隐秘的漏洞修复)[9], 这导致 OSS (open source software) 用户无法及时知道特定漏洞的存在并修复, 因此这也要求其可以通过分析提交日志和代码变更来感知潜在的安全补丁以及时修复.

正是由于传统漏洞检测技术在开源软件漏洞感知上的局限, 在传统检测技术基础上结合各类元数据的漏洞感知技术受到了广泛的关注. 开源软件漏洞感知旨在漏洞被公开数据库 (如 NVD) 正式披露前, 用户提前感知到开源软件中潜在的漏洞风险. 图 1 展示了协同披露机制下开源软件漏洞的生命周期.

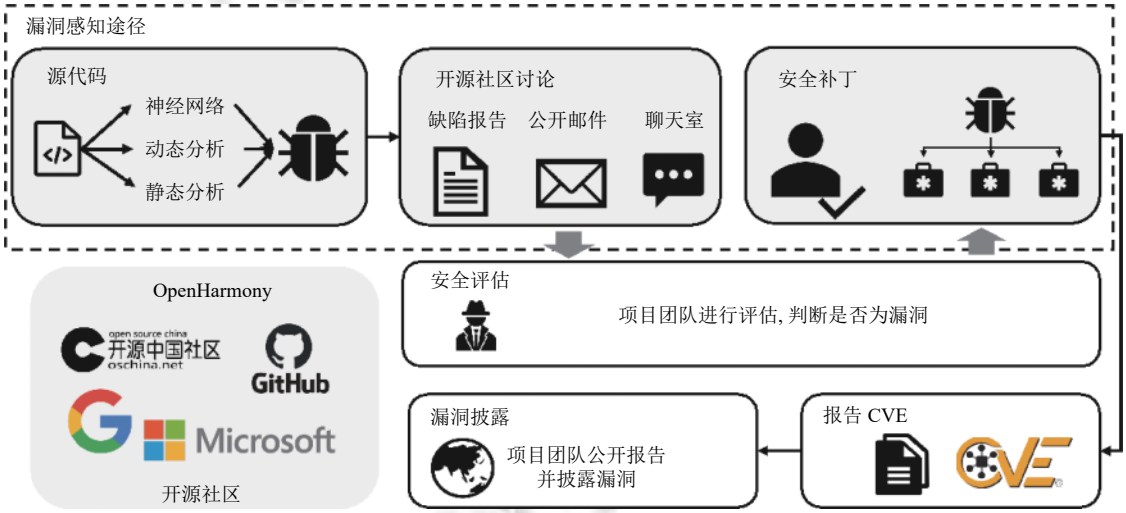

图 1 协同披露机制下的漏洞生命周期

基于这一周期, 在漏洞被正式披露前, 可以在以下 3 个阶段开发自动化技术对其进行检测, 以实现提前感知的目的: ① 可使用传统的代码漏洞检测技术检测开源项目中的漏洞, 具体包括基于静态分析技术、基于动态分析技术和基于机器学习和深度神经网络的技术. 这一类技术仅依赖源码作为输入, 不受特定的披露阶段的限制. 这同时也导致这类检测技术面临的挑战最大, 具有开销大、误报率高等弊端, 无法仅依赖该技术完成对开源软件漏洞的实时、可靠的感知. ② 开源软件的终端用户发现潜在漏洞后, 可能通过公共缺陷报告、公开邮件列表报告或在聊

天室、问答网站讨论相关信息, 因此可以通过监测公开频道中的敏感安全信息感知潜在漏洞. ③ 开源项目团队在收到用户的报告后, 会对其进行安全评估, 判断该漏洞是否真实存在并其进行修复, 并在代码仓库中提交相应的补丁. 在漏洞修复后, 经过若干的延迟, 项目团队才会正式公开报告并披露漏洞. 因而也可以通过分析提交日志和代码变更来感知潜在的安全补丁以感知漏洞并及时修复.

本文从开源社区中使用协同漏洞披露机制 (CVD) 的漏洞管理流程出发, 综述各个阶段的漏洞感知技术, 总结论文作者的核心思想、创新方法和最新的研究成果. 根据开源软件漏洞生命周期阶段不同, 本文将现有的开源软件漏洞感知技术可分为基于代码的漏洞感知技术, 基于开源社区讨论的漏洞感知技术, 和基于安全补丁的漏洞感知技术. 基于源代码的漏洞感知技术利用静态分析技术、动态分析技术和机器学习技术来扫描软件中的漏洞; 基于开源社区讨论的漏洞感知技术通过分析开源社区中的缺陷报告、讨论等进行漏洞的提前感知; 基于安全补丁的漏洞感知技术通过分析代码变更来感知隐秘的漏洞修复. 本文基于开源软件的特点, 创新性地从开源软件漏洞生命周期的角度来总结不同阶段的漏洞感知技术. 除此之外, 本文对开源软件漏洞感知常用的工具和数据集进行了整理和分类, 以方便后来的研究者. 最后, 本文根据选取的论文和其他材料总结分析开源软件漏洞感知领域当前所面临的挑战, 并提出未来可能的发展方向.

总的来说, 本文贡献如下.

• 从开源软件漏洞生命周期的角度, 对国内外开源软件漏洞感知技术的最新研究进行系统的分析和介绍.

• 梳理了漏洞感知常用的 9 个数据集和近年来提出的 12 个工具, 便于后续研究者使用.

• 总结当前领域所面临的挑战, 如高质量漏洞数据集的稀缺以及缺少多来源、细粒度漏洞感知技术带来的挑战, 并提出未来可能的发展方向.

本文第 2 节介绍研究方法与相关工作. 第 3 节介绍代码级别的漏洞感知技术. 第 4 节介绍基于开源社区讨论的漏洞感知技术. 第 5 节介绍基于代码补丁的漏洞感知技术. 第 6 节总结公开的漏洞检测工具与数据集. 第 7 节分析当前领域的挑战和未来的研究方向. 第 8 节总结全文.

2 研究方法与相关工作

2012 年, Shahriar 等人[10]先介绍了如缓冲区溢出, SQL 注入等常见漏洞及与编程语言的对应关系, 然后对基于测试, 静态分析和混合分析的漏洞检测方法进行了详细介绍和对比分析. Shahriar 等人指出大多数技术只能检测一类或几类的漏洞, 有些技术严格限制在某些程序语言上, 并认为利用混合技术来检测各种程序语言的各种漏洞会是未来的发展方向. 之后李韵等人[4]对基于机器学习的软件漏洞挖掘方法进行了综述, Lin 等人[11]在 2020 年的综述则考虑基于深度神经网络的漏洞挖掘技术并将重点放在代码语义与漏洞检测的讨论上, Lin 等人根据神经网络学到的特征表示将技术分为基于图的特征表示, 基于序列的特征表示, 基于文本的特征表示以及混合特征表示 4 类. Lin 等人认为深度神经网络技术在漏洞检测中应用的发展趋势是含漏洞的代码片段的特性, 从而建立起使用者所理解的漏洞和神经网络学习到的语义特征的桥梁. 原有的综述都只总结了基于代码的漏洞检测技术, 而忽略了基于开源社区讨论和基于安全补丁的漏洞感知技术, 尤其是面对开源软件独有特质及其所带来的种种新挑战, 我们需要更多样化的、更即时的基于多元数据的检测更好地感知开源漏洞.

在过去的 10 年中, 检测漏洞的方法与手段层出不穷. 因为已有前人[4,5]分别综述过 2007–2019 年间基于机器学习的软件漏洞检测技术和 2018 年之前的基于源代码的漏洞静态检测技术, 本文对其中已经出现过的论文不再赘述. 图 2 展示了本文选取的和已有综述的每年论文发表数量. 由于 2022 年的论文还没有发表完全, 所以数量较少. 从图中可以发现, 漏洞检测相关的国际论文数量总体呈逐年递增趋势, 反映出漏洞感知这一研究问题逐渐受到学术界关注. 另一方面, 图 3 展示了基于开源社区讨论和代码补丁的论文发表情况. 可以发现, 相关领域的研究论文较少且年份较晚, 说明研究者直到近些年对于该类方法才逐渐重视.

本综述按照下列步骤在文献库中检索和选取相关文献.

(1) 本文首先基于 IEEE 电子文献数据库, ACM 电子文献数据库以及 Google 学术搜索等数据库和搜索引擎进行搜索, 检测的关键词包括 vulnerability、detection、discovery、perception, 搜索时间从 2011 年开始.

(2) 本文根据中国计算机学会 (CCF) 在软件工程和网络与信息安全领域推荐的国际学术会议和期刊列表进行文献检索, 例如 FSE/ESEC, ICSE, CCS, IEEE S&P 等.

(3) 除此之外, 本文还对初选中论文的参考文献列表进行检索, 查找筛选与漏洞感知相关的文献.

本文最终选取 2011–2022 年漏洞感知相关论文 45 篇, 所选论文大部分来自软件工程领域的顶级会议或期刊, 如 ICSE (7 篇), ASE (4 篇), TSE (4 篇), 或信息安全领域的顶级会议, 如 CCS (3 篇), IEEE S&P (3 篇) 等. 本文根据不同方法所对应的开源社区漏洞生命阶段, 将这些论文分成基于代码、基于开源社区讨论和基于代码补丁 3 大类别, 如图 4 所示. 可以发现, 一半以上的方法还是集中在代码级别检测漏洞, 其进一步分类可以分为基于机器学习、基于静态分析、基于动态分析和混合方法. 其次是基于开源社区讨论的漏洞感知方法, 而基于代码补丁的漏洞感知方法数量最少.

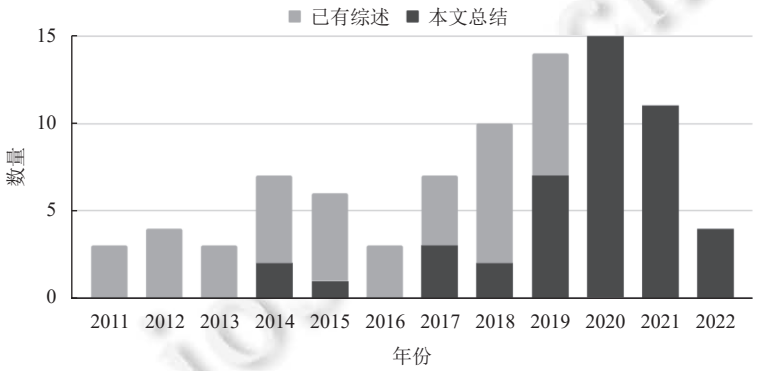

图 2 不同年份论文发表

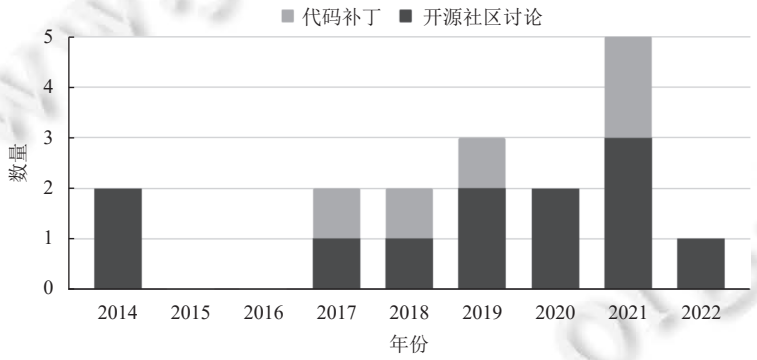

图 3 不同年份基于开源社区讨论和代码补丁的论文发表

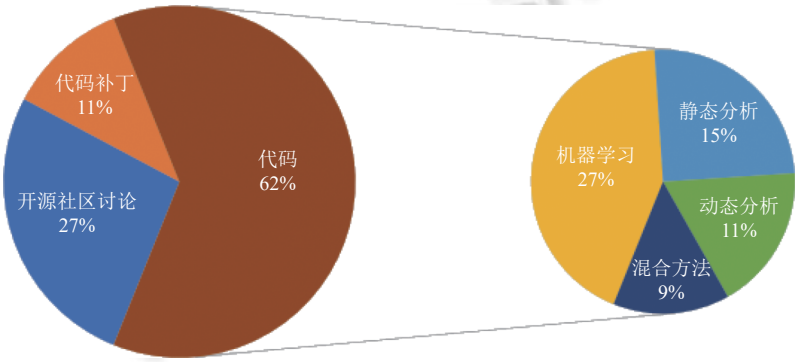

图 4 所选论文各类别数量

3 基于代码的漏洞感知技术

基于代码的漏洞感知技术利用静态分析、动态分析和基于机器学习与深度神经网络的方法直接检测开源软件中的漏洞. 该技术也是目前使用最为广泛的漏洞感知技术. 事实上, 在 2018 年, 李珍等人[5]对面向源代码的软件漏洞静态检测技术进行了综述. 在 2020 年, 李韵等人[4]对 2007–2019 年间基于机器学习的软件漏洞检测技术进行了综述. 而近两年随着人工智能领域和开源软件的迅速发展, 很多新的漏洞检测技术被用于漏洞感知. 因此, 本文对已有综述没有覆盖到的新技术进行更加全面地分析和总结.

3.1 基于机器学习的代码漏洞感知技术

李韵等人[4]总结了 2007–2019 年国内外将机器学习与深度学习技术应用到漏洞挖掘领域的相关研究, 归纳了技术特征与工作流程, 接着按照软件度量、抽象语法树、图、token 作为代码的表征形式对研究进行分类, 并进行系统性的对比分析. 随着最近几年人工智能领域的迅速发展, 许多新的机器学习技术尤其是深度学习技术被用于漏洞的感知. 本文对 2020 年以来的基于机器学习的软件漏洞感知技术进行相关总结.

针对不同类型的漏洞, 研究者提出了不同的漏洞感知技术. 对于通用的漏洞, 很多研究者采用图模型和预训练模型来识别软件中的漏洞. 图的表征形式能反映代码中更丰富的语法和语义特征, 有很好的感知能力, 因此在软件的漏洞感知技术中, 许多研究者基于图来构建深度学习模型. 2020 年 Wang 等人[12]提出一种新的基于图的学习方法 FUNDED. 通过分析程序的语法语义和控制流, FUNDED 可以为下游漏洞感知任务找到更好的代码表示. Wang 等人在 C、Java、Swift 和 PHP 编写的开源数据集进行了评估, 并将其与 6 种其他代码漏洞感知方法进行比较, 评估结果明显优于其他方法. 2021 年 Chakraborty 等人[13]从 Chromium (Chrome 和 Linux Debian Kernel) 项目开发者和用户的漏洞报告中收集了新的数据集, 对 Vulseeker[14]等深度学习方法能否在这些真实世界的开源项目中应用进行了测试, 发现它们的性能至少下降了 50%. Chakraborty 等人认为这是因为这些方法采用数据集和模型过于简单而不能对真实世界漏洞建模. 基于以上研究, Chakraborty 等人针对模型设计和数据集选取的几个问题——数据重复、数据集漏洞比例与真实场景不平衡等问题提出了 ReVeal, 基于 Yamaguchi 等人[15]提出的代码属性图和门图神经网络 (gated graph neural networks) 训练模型, 在该数据集和其他数据集中取得了优于其他模型的效果. 纵向来看, 从简单的软件度量到语法树, 再到复杂的图表示, 研究者从代码中提取愈加细致的特征, 使用愈加复杂的模型进行检测. 但正如 Chakraborty 等人[13]的研究所示, 真实世界漏洞的复杂性还是使得基于深度学习的方法在数据集和在真实情况下的表现有较大的差距. 这也说明了一个真正大规模的, 能够反映真实世界的漏洞数据集实际上还是稀缺的.

2021 年以来, 随着预训练模型的兴起, 如 CodeBERT[16]和 GraphCodeBERT[17]的发布, 许多研究者开始使用预训练模型来感知开源代码中的漏洞. 2022 年, Nguyen 等人[18]提出基于图神经网络的语言无关的漏洞感知工具 ReGVD 在函数级别感知漏洞. ReGVD 根据预训练编程语言模型提供的结点特征, 将每个输入的源代码视为一列 token 建图, 然后利用 GNN 层之间的残差连接和图级别的最大与求和池化生成源代码的图表示, 最后将图表示输入一个全连接层来预测源代码中是否存在漏洞. 在 CodeXGLUE[19]的漏洞相关数据集测试中, ReGVD 精度达到了 63.69%, 较 CodeBERT[16]与 GraphCodeBERT[17]分别提升了 1.61% 和 1.39%. 2022 年, Hin 等人[20]提出语句级别漏洞感知工具 LineVD. LineVD 利用图神经网络和 CodeBERT 来编码源代码, 从而更好地利用语句之间的控制和数据依赖关系. 而每一个语句也就转化为图上的节点, 这将漏洞感知任务转化为节点分类任务. 在对大量真实 C/C++ 漏洞的实验中 LineVD 比 IVDetect[21]的 $F1$ 值提高了 105%. 和 Hin 等人的工作类似, Cao 等人[22]提出 MVD, 利用流敏感的图神经网络来感知语句级别的内存相关漏洞.

针对 Web 应用的漏洞感知一直以来都备受关注. 2020 年, Calzavara 等人[23]提出 Mitch 来感知 Web 应用跨域请求伪造漏洞. Mitch 利用基于机器学习的分类算法将跨域请求分为安全信息敏感和安全信息不敏感两类来有效减少误报. 对于安全信息敏感的请求, Mitch 扩展请求生成新的 HTML 元素. 然后利用新的 HTML 元素自动重播敏感请求来模拟 CSRF 攻击, 从而感知漏洞. 2021 年, Amouei 等人[24]提出 RAT, 利用强化学习和自适应测试的黑

盒测试策略来感知 Web 应用防火墙漏洞. RAT 使用 n-gram 从攻击样本中自动提取模式, 并将类似的样本聚集在一起. 然后, 它利用强化学习技术结合自适应搜索算法来搜索簇, 并有效地发现旁路攻击模式. 实验表明 RAT 可以高效地发现旁路攻击并优于最先进的其他技术.

近年来, 区块链技术受到了广泛的关注, 2020 年, Chen 等人[25]则首先提出了一种通过比较历史版本来感知智能合约安全问题的方法, 该方法利用合同交易信息对合约进行聚类, 再根据时间排序和文本相似度比较来找到自毁合约的升级版本, 最后比较自毁合约与其升级版本总结得到智能合约的安全问题.

此外, 还有一些工作关注基于代码变更来感知安全漏洞. 2015 年 Perl 等人[26]提出 VCCFinder, 结合代码度量分析 (code-metric analysis) 与从代码存储库中收集的元数据发现潜在漏洞代码. VCCFinder 首先进行了一次大规模映射 CVE 到 GitHub 提交, 从而创建了一个漏洞相关的提交数据库. 基于这个数据库, VCCFinder 训练了一个 SVM 分类器来标记可疑的提交. 类似的, 2017 年 Zhou 等人[27]结合自然语言处理技术和机器学习技术提出了 K-fold Stacking 模型, 与上文基于 SVM 的分类器 VCCFinder 相比在保持相同的召回率基础上准确率提高了 54.55%. Chen 等人[28]设计了一个基于机器学习的漏洞管理系统来自动预测每个数据项的漏洞相关性. 该系统支持完整的流程——数据收集, 模型训练和预测, 以及在部署之前对新模型进行验证. 该系统的全流程可以迭代执行以不断地改进模型. 在分类阶段, Chen 等人对 Zhou 等人提出的 K-fold Stacking 模型[27]进行了改进, 并且设计了新的部署稳定性指标来评估新模型的质量, 从而协助感知漏洞. Perl 等人[26]与 Zhou 等人[27]关注的都是如何在用户提交的代码变更中感知安全风险, 但在感知到存在漏洞风险的提交后, 还应该立即对其进行评估与管理. 2021 年 Le 等人[29]基于通用漏洞评分系统 (common vulnerability scoring system, CVSS), 提出了一种新的多任务深度学习模型 DeepCVA. DeepCVA 首先使用基于注意力机制的卷积门控循环单元, 从含有漏洞的提交中提取代码变更和上下文的特征, 并进一步根据这些特征使用多任务学习范式来预测 7 个 CVSS 评估指标, 即机密性 (confidentiality)、完整性 (integrity)、可用性 (availability)、访问向量 (access vector)、访问复杂性 (access complexity)、认证 (authentication) 和严重性 (authentication), 并利用这些指标来指导软件漏洞的管理和修复.

总的来说, 随着神经网络技术的不断发展和算力的不断提升, 尤其是大规模语言模型在编程语言的应用, 研究者逐渐利用越来越大规模的预训练模型、图神经网络来进行漏洞感知, 在语句级别的漏洞感知上取得了更好的效果. 但另一方面, 神经网络技术在漏洞感知上的可解释性和有效性还存在着更大的进步空间. 与此同时, 研究者容易陷入在一个固定的数据集上采用愈加先进的技术提高评价指标的局限, 而往往忽略了收集采用的数据集是否反映真实世界情况等问题.

3.2 基于静态检测的漏洞感知技术

2018 年, 李珍等人[5]总结了 2018 年之前的基于静态检测技术进行漏洞检测的研究工作. 李珍等人从基于代码相似性的漏洞检测、基于符号执行的漏洞检测、基于规则的漏洞检测以及基于机器学习的漏洞检测 4 个方面进行了总结. 因此, 本文对 2019–2022 年出现的新技术进行总结与分析.

2019 年, Rahaman 等人[30]提出了工具 CryptoGuard, 对大规模开源 Java 项目代码中密码相关 API 的误用漏洞进行静态感知, 使得误报较少从而开发人员可以有效使用它来感知漏洞. CryptoGuard 特定语言上下文细化、按需流敏感、上下文敏感和字段敏感的程序切片 (program slicing) 等一系列特定于加密的技术, 在实验中可以将误报减少 76%–80%. Rahaman 等人在 46 个 Apache 开源项目中运行 CryptoGuard, 发现 39 个项目至少有一种加密误用, 33 个项目至少有两种加密误用.

2020 年, Cui 等人[31]提出了 VulDetector, 基于图的相似性度量来感知 C/C++漏洞. VulDetector 的关键是加权特征图模型. VulDetector 首先根据漏洞敏感关键字对函数的控制流图进行切片, 从而在不影响安全相关语义的前提下压缩图. 然后, 使用加权特征图表征每个切片子图, 最后与已知漏洞的图表征进行比较以实现漏洞的感知.

近年来, 一些工作尝试通过引入补丁信息来提升传统基于相似性度量 (如克隆检测) 的漏洞感知的性能[32,33]. 对于含有漏洞的函数, 其修复前后的版本是高度相似的, 因此传统的基于相似性度量的检测方案存在较高的误报率 (即易将已被修复的版本同样判定为含有漏洞). 2020 年, Xu 等人[32]提出二进制代码层面的漏洞感知工具 BinXray,

包含两阶段的特征匹配过程: 阶段 1, 利用与已知漏洞函数的特征进行匹配, 寻找潜在的对象; 阶段 2, 通过比较已知漏洞函数修复前后的差异生成补丁特征, 从阶段一的潜在对象中筛除被错误识别的、已被修复的函数. 相似地, Xiao 等人[33]针对源码层面的漏洞感知提出了名为 MVP 的工具, 如果目标函数与漏洞特征匹配, 并且与相应的补丁特征不匹配, 才会被识别为包含漏洞的函数.

针对智能合约中的漏洞感知, 2020 年, Xue 等人[34]提出了 Clairvoyance, 利用静态污点分析技术来感知可交互的智能合约中的可重入漏洞. 为了减少漏报, Clairvoyance 通过跟踪可能有污点的路径来实现跨智能合约的调用链分析. 为了减少误报, 该文总结了 5 种主要的路径保护技术从而支持快速而精确的路径可行性检查. 2021 年, Ma 等人[35]提出支持跨合约场景下漏洞感知工具 Pluto. Pluto 首先构建一个合约间控制流图来提取合约调用之间的语义信息. 然后采用符号执行探索控制流图, 并推导求解合约间路径约束, 以更准确地检查执行路径的可达性. 最后, Pluto 根据一些预定义的规则判断合约中是否存在整型溢出、时间独立和重入漏洞.

Belleville 等人[36]于 2021 年提出了 KALD 来感知操作系统内核代码中指针泄露的漏洞. KALD 将指向内核代码和全局变量的指针视为敏感指针, 并编译了一系列可以向用户空间写入信息的标准函数, 然后通过指针分析判断敏感指针会不会通过这些函数流向用户可见区域, 从而感知指针泄露漏洞. KALD 在 Linux 4.14 版本中发现 73 个相关漏洞.

3.3 基于动态分析的漏洞感知技术

基于动态分析的漏洞感知技术是指通过运行程序, 观察程序在给定输入下的行为是否与预期不同来感知漏洞的技术.

2020 年 Hough 等人[37]提出了 Rivulet 来感知代码注入漏洞. Rivulet 在软件发布前通过动态污点分析强化开发者自己编写的测试数据. 另外, Rivulet 使用白盒且上下文敏感的输入生成元来生成有效的测试注入漏洞.

2019 年 Fu 等人[38]提出了第 1 个使用差分模糊技术感知以太坊虚拟机 (Ethereum virtual machine, EVM) 漏洞的工具, Evmfuzzer. Evmfuzzer 采用动态优先级调度算法不断生成种子智能合约, 并将它们作为输入提供给待检测的 EVM 和作为基准的 EVM, 尽可能多地发现两者执行结果间的不一致, 最终通过输出交叉引用发现漏洞. 2020 年 Nguyen 等人[39]提出了 sFuzz, 结合 AFL[40]策略和针对难以高效覆盖分支的多目标自适应策略更好感知智能合约中的漏洞.

2019 年 Li 等人[41]针对灰盒模型的两个基本问题: 如何选择下一个种子进行模糊测试 (种子优先级), 和所选种子要生成多少的新输入提出了 Cerebro. Li 等人提出了输入势 (input potential) 的概念来考虑在输入的执行轨迹附近尚未覆盖的代码的复杂性, 并提出算法来快速计算输入势来优化种子生成新输入的过程. 其次, Li 等人还提出了一个基于多目标的模型和种子排序算法来优化种子优先级. 实验表明相较于 AFL[40]等工具, Cerebro 能感知到更多漏洞、达到更高覆盖率.

2020 年 Kim 等人[42]研究了客户端业务流篡改漏洞的特性, 提出一套感知相关漏洞流程.

(1) 收集与业务逻辑相关的函数.

(2) 分析每个收集到的函数, 寻找那些如果修改则可能干预业务流预期行为的位置.

(3) 利用分析技术选择更容易攻击的函数并生成对应的攻击方式.

(4) 根据篡改方式重新访问网站实施篡改, 并确认感知结果是否确实造成了业务流篡改漏洞.

该方法可以成功感知 23 个网站的 27 个漏洞从而实现绕过网站付费墙, 禁用广告拦截器检测, 获得非法奖励点等操作.

3.4 基于混合方法的漏洞感知技术

除上述漏洞感知技术之外, 许多研究者结合静态分析、动态分析技术进行漏洞感知.

2020 年 Wang 等人[43]基于类型状态的静态分析方法, 提出 UAFL 来动态感知释放后用 (use after-free) 漏洞. 该文认为仅覆盖控制流图的单条边不能有效地发现释放后用漏洞, 而需要以特定的顺序遍历某些边. 为此, 该论文将释放后用漏洞建模为类型状态属性, 首先进行静态类型状态分析, 获取可能违反该属性的操作序列. 然后, 由操

作序列指导模糊测试过程, 以便逐步生成触发属性违反的测试用例. 此外, Wang 等人还结合信息流分析提高模糊测试的效率.

2020 年 Chen 等人[44]提出基于符号执行和模糊测试的缺陷驱动 (bug-driven) 的混合测试新框架 SAVIOR. 与过去的混合测试方法不同, SAVIOR 优先符号执行那些可能暴露更多漏洞的种子, 然后对于执行路径上验证所有可能有漏洞的位置. 通过使用约束求解对漏洞进行建模, SAVIOR 能自动对漏洞的可行性进行推理, 并生成具体的测试用例作为证据. 实验表明, 缺陷驱动的方法优于其他主流的自动化测试技术.

并发程序中许多漏洞是在特定的执行下发生的, 而探索多线程程序的所有可能在执行实际使用中又很困难, 所以感知并发程序的漏洞一直以来都是一个挑战. 2021 年 Yu 等人[45]提出一种基于偏序事件 (partial orders of events) 感知并发内存漏洞的新方法 ConVulPOE, 感知多线程并发程序的内存漏洞. ConVulPOE 允许通过交换可能含有漏洞事件的执行顺序来构建暴露特定漏洞的可行跟踪, 来减少可能执行的搜索空间, 从而提高计算效率.

2021 年 Liu 等人[46]提出了 REVEALER 来感知 ReDoS (regular expression denial-of-service), 一类由正则匹配引起的算法复杂度漏洞. 该文对流行的正则表达式引擎中易受攻击的正则表达式模式进行建模, 并设计对应攻击字符串. REVEALER 采取了一种混合的方法来感知该类漏洞, 它首先静态地定位正则表达式的潜在漏洞结构, 然后动态地验证该漏洞是否可以被触发, 最后生成可以触发的攻击字符串.

以上工作总结对比结果如表 1 所示.

表 1　基于代码的漏洞感知技术现有工作总结

方法类型	文献	方法概述	漏洞类型
机器学习	[23]	基于机器学习分类器	Web 应用跨域请求伪造
	[24]	强化学习、黑盒测试	Web 应用防火墙漏洞
	[25]	智能合约历史版本	智能合约漏洞
	[12]	图神经网络	多种漏洞
	[13]	代码属性图、门图神经网络	多种漏洞
	[18]	图神经网络	多种漏洞
	[20]	图神经网络、CodeBERT预训练模型	语句级别漏洞
	[22]	流敏感图神经网络	内存相关漏洞
	[26]	SVM	多种漏洞
	[27]	K-fold Stacking	多种漏洞
	[28]	全周期漏洞管理系统	多种漏洞
	[29]	注意力机制、多任务学习	多种漏洞
静态检测	[30]	流敏感、上下文敏感的程序切片	Java密码学API误用漏洞
	[31]	加权特征图	多种漏洞
	[34]	静态污点分析	智能合约可重入漏洞
	[35]	合约间控制流图	智能合约漏洞
	[36]	指针分析	Linux内核指针泄露
	[34]	基本块映射与追溯	多种漏洞
	[33]	程序切片、特征签名	多种漏洞
动态分析	[37]	动态污点分析	代码注入漏洞
	[38]	差分模糊	智能合约漏洞
	[39]	AFL 模糊器	智能合约漏洞
	[41]	多目标模型、种子排序算法	多种漏洞
	[42]	业务控制流图	客户端业务流篡改漏洞
混合方法	[43]	静态类型分析指导模糊测试	释放后用漏洞
	[44]	符号执行、模糊测试	多种漏洞
	[45]	偏序事件	并发程序内存漏洞
	[46]	静态定位、动态验证	ReDoS

通过对这些文献的整理归纳分析, 我们发现, 纵向来看基于代码的漏洞感知方法有着更细粒度、更精准的趋势. 研究者对于漏洞感知的粒度要求越来越高, 以前的方法大多集中在函数级别和代码片段级别, 而现在已经有越来越多语句级别的漏洞感知方法; 研究对漏洞感知的精准度也越来越高, 传统的静态分析等方法往往有着很高的误报率, 而研究者通过对程序更加敏感的特征提取和更精细的建模来降低误报率. 而横向来看, 各种漏洞感知技术可以相互辅助、相互融合来增强工具感知漏洞的能力. 例如静态分析研究中利用的分析代码性质的图、树等结构与更细粒度的程序切割方法可以处理后作为神经网络的输入, 而动态测试下黑盒测试又可以和强化学习、神经网络等技术自然的结合来更自动高效的生成输入. 总的来看, 基于代码的漏洞感知技术在过去的研究中逐渐成熟, 能够一定程度上用于实际漏洞的检测. 但另一方面, 我们要探索更加实时的漏洞感知技术和更加实用的漏洞感知工具, 以降低安全专家的门槛, 提高漏洞感知技术的应用价值. 而如何在一个统一的框架下更高效的结合现有的各种技术, 是一个亟待解决的挑战.

4 基于开源社区讨论的漏洞感知技术

当开源软件的用户在实际开发过中发现新的漏洞时, 若严格依据协同披露机制, 应该通过特定的渠道向项目管理团队私下报告, 以确保在采取有效的消减措施前漏洞信息不会提早泄露. 然而, 在实际开发中, 由于复杂的因素, 如开发者缺乏安全领域知识 (错将安全漏洞当作普通漏洞)、缺乏风险意识 (不清楚相应的标准或不遵照执行)[6], 项目管理团队缺乏作为而导致用户选择直接曝光漏洞[7], 漏洞相关的信息在正式披露前就已经出现在公开的频道中 (如缺陷报告、问答网站等). 泄露的漏洞信息给了潜在攻击者提前发动攻击的可能性, 同时也为漏洞提前感知提供一种渠道. 为了避免陷入被动之中, 一系列研究工作旨在开发自动化的工具帮助开源软件的用户提前感知漏洞.

4.1 基于缺陷报告的漏洞感知技术

基于缺陷报告的漏洞感知技术是指根据开发者提交的公开的缺陷报告, 从中识别出与安全缺陷相关的内容并感知其中潜在的安全漏洞技术.

2014 年 Behl 等人[47]以缺陷报告为研究对象, 提出了一种基于 TF-IDF (term frequency-inverse document frequency) 并使用朴素贝叶斯网络的漏洞挖掘方法, 用于识别涉及安全漏洞的缺陷报告. 该方法在包含 10 000 个缺陷报告的数据集上取得了 93.99% 的成功率和 92.56% 的准确率. Pereira 等人[48]则表明即使只有标题可用于分类, 同样可以对缺陷报告进行相当准确的预测. 其模型训练过程包括两个主要步骤: 使用 TF-IDF 将数据编码为特征向量和训练监督学习的分类器. Pereira 等人训练和比较了 3 种分类技术: 朴素贝叶斯, AdaBoost 和逻辑回归.

2017 年 Peters 等人[49]指出现有的预测模型容易错误识别安全缺陷报告, 可能的原因包括数据不平衡, 以及具有误导性的同时存在于正例和负例中的交叉安全关键词. 基于这一发现, Peters 等人进一步提出名为 FARSEC 的框架, 该框架在模型训练前会首先移除含交叉安全关键词的负例, 以避免对模型训练造成干扰. 通过在来自 Chromium 和 4 个 Apache 项目的 45 940 个缺陷报告上开展评估, FARSEC 被证明可以缓解类别不平衡问题, 并将被错误标记的安全缺陷报告数量减少 38%.

在 FARSEC 的基础上, Shu 等人[50]提出名为 SWIFT 的方法, 使用超参数优化来提升现有的自动化方法 (通常使用默认的参数设定) 检测安全缺陷报告的性能. Shu 等人探究了 3 种不同的优化策略: 仅对模型进行超参数优化、仅对预处理器进行超参数优化以及同时优化二者, 并发现同时优化二者能获得最佳性能. 他们还使用 ϵ-dominance 技术来避免同时进行超参数优化时可能导致的指数级别的开销.

2019 年 Kudjo 等人[51]首次提出用 TF-IGM (term frequency-inverse gravity moment) 代替普遍使用的 TF-IDF 来将文本编码为特征向量. Kudjo 等人首先应用标准的预处理步骤对缺陷报告的自然语言描述进行处理, 然后使用 TF-IGM 对词项进行加权, 最后分别使用决策树、随机森林和 K-邻近分类 3 种机器学习算法作为分类器. 在基于这 3 种机器学习算法在 10 个公开漏洞数据集上的评估结果表明基于 TF-IGM 方法在安全缺陷报告检测任务中整体优于基准方法.

在上述挖掘软件仓库的研究中, 都需要使用大量的数据标注结果来构建预测模型, 而安全缺陷报告的标注需要一定的安全领域知识和大量的人力成本. 基于此, Goseva-Popstojanova 等人[52]针对安全缺陷报告检测首次探究了无监督方法. 该方法基于异常检测, 将与 CWE-888 中描述相近的缺陷报告归类成安全相关. 他们在 3 个 NASA 数据集上评估了提出的无监督算法, 其性能仅略差于监督算法.

此外, 标签的正确性也会极大地影响分类模型的性能. 2021 年 Wu 等人[53]对已有工作[49,50]使用的数据集质量进行系统探究, 他们发现这些工作中所使用 5 个公开的安全缺陷报告预测数据集中均包含大量的错误标记实例 (安全缺陷报告被错误标记为非安全缺陷报告), 这导致 Peters 等人[49]和 Shu 等人[50]的预测模型性能不佳. Wu 等人手工修正了数据集中的错误标签, 并比较模型在有噪声 (即校正前) 和干净 (即校正后) 的数据集上的性能, 发现使用校正后的数据集能够显著提升模型的性能. 对于校正后的数据集, 简单的文本分类模型能获得优于 Peters 等人[49]和 Shu 等人[50]特别设计的分类算法的性能. 这再次说明了漏洞感知领域优质的数据集较为匮乏, 明显影响了漏洞感知工具的测试与评价.

在最近的一项研究工作中, Pan 等人[7]首次指出在漏洞报告的生命周期早期对其进行检测, 能够帮助开源软件托管平台 (如 GitHub) 缓解由于不规范的安全实践而导致的漏洞信息泄露, 此外还可以帮助开源软件用户及时感知潜在漏洞以采取消减措施. Pan 等人以被 CVE 关联的、真实的漏洞报告作为研究对象, 并构建了大规模的数据集. 他们提出了一个名为 MemVul 的深度学习模型, MemVul 中包含一个外部存储模块用于引入来自 CWE 的外部漏洞知识. 在跨项目场景下的评估表明, MemVul 相较于已有方法能取得更好的分辨能力.

综合来看, 目前基于缺陷报告的漏洞感知技术实质上就是将缺陷报告视作自然语言进行分类处理, 利用自然语言处理的相应技术可以在一定程度上感知到潜在的漏洞, 但另一方面, 缺陷报告本身所蕴含的代码信息却没有得到足够的利用, 这导致了现有技术只是结合漏洞知识的文本分类模型. 如何将缺陷报告中的自然语言与程序语言的语义相结合, 对缺陷报告中可能存在的漏洞进行程序层面的验证和分析, 是未来一个重要的研究方向.

4.2 基于邮件列表与公开讨论的漏洞感知技术

2014 年, Pletea 等人[54]从 GitHub 上提交和拉取请求的讨论中挖掘了与安全问题相关的内容, 他们发现与安全相关的内容约占所有讨论的 10%. 其次, 在与安全相关的讨论中用户更容易表达出负面情绪. 这些发现证实了需要适当引导开发人员以帮助他们更好地解决安全问题, 并对应用程序进行测试以减少潜在的安全漏洞可能带来的负面情绪.

在软件开发期间, 通过检查开发人员之间的通信可以挖掘出软件潜在的安全问题. 文献 [55,56] 探究了将项目通信消息 (如问题报告和提交) 分类为安全相关或不相关, 但在近期的一项工作中, Oyetoyan 等人[57]指出这些已有的方法通常是项目特定的, 从而限制了其被其他项目或组织使用. 基于此, Oyetoyan 等人[57]提出了一个适用于不同项目的通用分类模型. 他们从安全相关的信息源 (如 NVD) 中提取一组关键词, 将其分为 4 类: 资产、攻击/威胁、控制/消减和隐含 (不属于前 3 种类别但通常与安全有隐式关联的词汇), 并进一步基于这些关键词构建了分类模型. Oyetoyan 等人在来自工业界、学术界以及开源社区的包含超过 45 个项目的数据集上对所提出的方法进行了评估. 结果显示所提出的方法取得的平均召回率区间为 55%–86%, 平均 $F1$ 分数的区间为 3.4%–88%, 平均 AUC 的区间为 69%–89%. 实验结果还表明基于来自外部信息源的特征构建的模型性能优于使用项目特定的特征.

2020 年, Le 等人[58]提出了一个名为 PUMiner 的学习框架来自动挖掘问答网站中安全相关帖子. 不同于常规的机器学习方法, PUMiner 的构建仅依赖正样本 (positive, 即安全相关的样本) 和无标签 (unlabel) 的样本. 动机在于无法依赖手工标注构建大规模、高质量的数据集, 但其中与安全相关的帖子可从 Security StackExchange 自动获得. PUMiner 首先结合上下文语境来提取帖子的特征, 然后利用正样本和无标签的样本建立两阶段 PU 学习模型. PUMiner 在来自 StackOverflow 的超过 1 720 万个帖子和来自 Security StackExchange 的 52 611 个帖子进行了评估. 实验表明, PUMiner 在所有模型配置中 $F1$ 分数均超过 0.85, 且即使在样本标签高度不平衡的情况下仍能取得理想性能.

上述工作试图从开源社区公开的报告和讨论中, 正向检测出与漏洞相关的内容, 而 Ramsauer 等人[59]则提出

一种逆向的思维, 即常规的开发活动会在公共频道留下痕迹 (如讨论, 拉取请求等), 而安全相关的漏洞修复遵循严格、标准的流程则不会. 因此他们将软件仓库中的代码变更提交记录与公开频道中的开发信息相对应, 若某个提交记录无法找到与之对应的开发信息, 则认为其是潜在的安全补丁. Ramsauer 等人以 Linux 内核等项目为研究对象, 寻找未出现在公共邮件列表内的提交, 最终发现了对应 12 个漏洞修复的 29 个安全补丁, 且相较于漏洞正式公开披露提早 2–179 天.

通过对缺陷报告, 邮件列表与公开讨论这类基于开源社区漏洞感知的文献综述, 我们可以发现其核心目的就是利用各种技术在庞大的开源社区中进行分类, 找到与安全漏洞相关的内容, 如涉及安全问题的缺陷报告等. 一般来说开源社区讨论中与安全相关的内容只占少部分, 在训练的过程中必然面临正负样本的不平衡性问题, 即真正涉及漏洞的缺陷报告在所有缺陷报告中只占据很小的一部分. 如何处理正负样本的高度不平衡, 是这类技术所面临的一个挑战. 以上工作总结如表 2 所示.

表 2　基于开源社区讨论的漏洞感知技术总结

文本形式	文献	方法概述
缺陷报告	[47]	TF-IDF、朴素贝叶斯
	[49]	TF-IDF、贝叶斯滤波、集成学习
	[50]	对偶优化、ϵ-dominance
	[52]	基于文本挖掘、有监督学习 (朴素贝叶斯、SVM)和无监督学习方法 (K-邻近)
	[48]	TF-IDF、朴素贝叶斯、AdaBoost、逻辑回归
	[51]	TF-IGM
	[7]	含有外部漏洞知识的深度学习模型
	[53]	矫正数据集、简单文本分类模型
GitHub 讨论	[54]	基于情感分析
邮件列表	[59]	数据挖掘
问答网站	[58]	PU 学习
开发人员通信	[57]	利用安全关键词分类

5　基于代码补丁的漏洞感知技术

根据协同披露政策, 开源软件的维护团队在知晓漏洞后, 需对其进行“隐秘”地修复, 即在正式披露之前不公开漏洞和补丁的相关信息 (见图 1). 然而, 漏洞被正式披露的延迟可能从几天到几个月不等甚至若干年[8,9]. 此外, 还有部分开源项目可能由于考虑声誉 (用户可能认为披露漏洞较少的软件更安全) 等因素选择不公开披露漏洞[60]. 在披露延迟的时间段内, 潜在的攻击者可能发现漏洞并加以利用, 尽管此时项目主仓库已经打上补丁, 但是用户由于未得到披露信息而未及时更新. 由于开源软件的透明性, 可以通过自动化检测代码变更以及时识别潜在的漏洞补丁.

2017 年, Xu 等人[61]提出了一种二进制补丁分析框架 SPAIN. SPAIN 能够自动识别安全补丁, 并总结补丁模式及其对应的漏洞模式. 给定二进制程序的原始和已更改版本, SPAIN 能够定位出其中修改过 (即打过补丁) 的函数, 并进一步定位出与该函数相关的基本块, 基于此来判定该更改是否为安全补丁. 通过对基本块之间可能的执行顺序进行语义分析, 对函数进行污点分析, 可以总结模式用于寻找相似的补丁或漏洞. Xu 等人在真实世界的项目中验证了 SPAIN 的准确率与可扩展性, 并总结了 5 种补丁及其对应的漏洞模式.

不同于 Xu 等人[61]在二进制代码层面检测安全补丁, 近年来更多的工作聚焦于源码层面, 即基于开源软件仓库的代码变更提交记录识别潜在的安全补丁. 2018 年 Sabetta 等人[62]提出了一种基于机器学习自动识别安全相关的代码提交的工具, 他们将代码变更视作用自然语言编写的文档, 并使用标准文档的分类方法进行分类. 该方法分别基于提交日志和源码变更构建了两个独立的分类器, 并以高准确率作为模型调优的首要目标. Sabetta 等人发现使用其中任意一个模型都只能取得极低的召回率, 因此他们实现了一个投票机制将两个模型进行整合. 具体地, 当

其中任意一个模型预测为安全相关, 该提交即被分类为正例. 最终, 联合模型在实验中取得较高的准确率 (80%) 和相对可以接受的召回率 (43%). 与 Sabetta 等人的工作类似, Zhou 等人[8]同样以提交日志和源码变更作为输入来自动检测安全补丁. 不同之处在于他们使用了深度学习技术, 具体地, 构建了两个神经网络用以分别提取日志描述和语句级别的代码变更的语义信息. 此外, Zhou 等人还构建了一个大规模的安全补丁数据集, 包含来自 4 个开源 C 语言项目的共计 38 291 个安全补丁, 并进行了人工验证.

不同于 Sabetta 等人[62]和 Zhou 等人[8]同时考虑提交日志和源码变更的信息, Zhou 等人[9]认为在隐秘修复的场景下提交日志不应该提及任何安全相关的信息, 所以仅基于代码变更构建分类模型. 他们基于 Transformer 提出了一个名为 VulFixMiner 的跨语言跨项目的安全补丁识别工具, 使用预训练语言模型 CodeBERT 对提交的代码变更提取高阶语义信息, 以用于识别潜在的隐秘漏洞修复补丁. VulFixMiner 首先通过微调 (finetune) CodeBERT 来学习文件粒度的代码变更的语义信息, 然后将文件粒度的代码变成信息进行聚合 (一个提交可能包含多个文件的更改) 并用于最终的提交粒度的分类.

2019 年, Wang 等人[60]对开源软件中的隐秘补丁开展了大规模的实证研究. 他们指出隐秘补丁可以被攻击者利用以发起零日攻击 (zero-day attack), 补丁不仅为攻击者指明具体存在漏洞的代码行, 而且还可被进一步用于寻找相似软件中的相似漏洞. 为构建自动化的方法以检测安全补丁, Wang 等人首先构建了一个含 4 700 多个安全补丁的数据库, 然后选取了一组特征并构建基于机器学习的分类器. 此外, Wang 等人还探究了利用代码克隆技术发现统一类型开源软件中的相似的补丁或漏洞, 他们在 OpenSSL、LibreSSL 和 BoringSSL 的案例研究中发现了 12 个隐秘的安全补丁.

对于隐秘补丁的检测, 其核心是如何构建一个分类器将修复漏洞的隐秘补丁和普通的变更区分开来. 现有技术与基于开源社区的漏洞感知技术类似, 主要围绕对变更特征的选取和模型的设计展开, 并结合外部知识构造数据库, 最终识别其他项目软件中的类似补丁. 以上工作总结如表 3 所示.

表 3　基于代码补丁的漏洞感知技术现有工作总结

类型	文献	方法概述
二进制补丁	[61]	语义分析、污点分析
识别隐秘补丁	[60]	代码克隆
识别隐秘补丁	[9]	CodeBERT预训练模型
辨别修复提交	[62]	自然语言处理、SVM
辨别代码变更提交	[8]	深度神经网络

6　公开数据集与工具

6.1　公开漏洞感知工具

工具是验证模型有效性的重要途径, 也是研究者成果的集中体现. 开源工具使得后来研究者容易对比, 方便用户使用, 可以促进该领域的发展. 后文表 4 列出了 2018 年以来的部分漏洞感知领域的公开工具, 列出了其名称, 发布年份, 针对的漏洞感知主题和对应的下载地址, 方便后来的研究者获取. 由于我们并未找到高质量且实用性强的基于开源社区与代码补丁的工具, 所以表 4 所涉及的工具集中在代码级别的漏洞感知上. 其中除了 sFuzz 以外的工具都由 GitHub 托管, 而 sFuzz 可以直接在网页上使用以判断给定智能合约是否含有安全漏洞.

6.2　公开数据集

高质量的数据集对于漏洞感知模型的检验与优化尤为重要, 表 5 列出了漏洞感知领域的一些数据集, 标明了其发布年份、数据来源和下载地址, 方便后来的研究者参考验证.

VPD、VCID 是由 Li 等人[63]收集的数据集. 其中, VPD 包含来自 19 个 C/C++开源软件的 1 761 个漏洞, 涉及 3 454 个不同的块; 而 VCID 则包含 455 个未打补丁的函数实例. 每个块包含更改的文件名, 以及添加和删除的序列. 添加的源代码行前面有“+”符号, 而删除的代码行前面有“–”符号.

表 4 漏洞感知工具

名称	年份	主题	下载地址
VulSeeker	2018	给定漏洞	https://github.com/buptsseGJ/VulSeeker
CryptoGuard	2019	Java密码API误用	https://github.com/CryptoGuardOSS/cryptoguard
SlowFuzz	2019	算法复杂度漏洞	https://github.com/nettrino/slowfuzz
JSFlowTamper	2020	客户端篡改漏洞	https://github.com/yirugi/JSFlowTamper
Rivulet	2020	Java代码注入漏洞	https://github.com/gmu-swe/rivulet
SAVIOR	2020	C/C++漏洞	https://github.com/evanmak/savior-source
FUNDED	2020	多种漏洞	https://github.com/HuantWang/FUNDED_NISL
ReVeal	2020	C语言漏洞	https://github.com/VulDetProject/ReVeal
VulDetector	2021	C/C++漏洞	https://github.com/leontsui1987/VulDetector
ContractFuzzer	2018	智能合约漏洞	https://github.com/gongbell/ContractFuzzer
sFuzz	2021	智能合约漏洞	https://sfuzz.github.io
Pluto	2021	智能合约漏洞	https://github.com/PlutoAnalyzer/pluto/tree/main/PlutoTool

表 5 漏洞感知数据集

名称	年份	数据来源	程序语言	下载地址
VPD, VCID	2016	开源项目	C/C++	https://github.com/vulpecker/Vulpecker
SARD	2017	真实软件等	C/C++, Java等	https://samate.nist.gov/SARD/
CGD	2018	开源项目	C/C++	https://github.com/CGCL-codes/VulDeePecker
KB	2019	开源项目	Java	https://github.com/SAP/project-kb/tree/master/MSR2019
VDISC	2021	开源项目	C/C++	https://osf.io/d45bw/
Chromium/Debian	2020	开源项目	C	https://bit.ly/3bX30ai
CryptoAPI	2021	人工制造	Java	https://github.com/CryptoAPI-Bench
OWASP	2022	安全供应商等	Java	https://owasp.org/www-project-benchmark
CodeXGLUE/Defect-detection	2022	开源项目	C	https://github.com/microsoft/CodeXGLUE/Defect-detection

SARD (software assurance reference dataset) 是由美国国家标准与技术研究院 (NIST) 自 2006 年开始更新的包含大量样例的漏洞数据集. 数据集面向 C, C++, Java, PHP 和 C#编程语言, 包括软件生命周期的所有阶段, 如源代码, 二进制文件等. 数据集不仅有来自工业界的真实软件程序漏洞, 也有人造的测试程序和学术界的研究用例. 前文介绍的论文如文献 [37] 就采用了该大型数据集中的部分样例.

CGD 是由 Li 等人[64]基于 SARD 和 NVD 收集的, 主要包含缓冲区错误漏洞 (CWE-119) 和资源管理错误漏洞 (CWE-399) 的数据集. CGD 数据集包含 61 638 个代码块, 其中 17 725 个是含有漏洞的代码块、43 913 个是不含漏洞的代码块. 在 17 725 个含有漏洞的代码块中, 10 440 个对应缓冲区错误漏洞, 而其余的 7 285 个对应资源管理错误漏洞.

KD 是由 Ponta 等人[65]收集的来自开源项目的数据集. 该数据集包含 1 282 个提交, 用以修复来自 205 个开源 Java 项目的 624 个公开披露的漏洞. 前文介绍的文献 [62] 就使用了该数据集.

VDISC 是由 Russel 等人[66]收集的数据集, 包含了从开源软件中挖掘的 127 万个函数的源代码, 并用静态分析标记了潜在漏洞.

由 Chakraborty 等人[13]收集的 Chromium 和 Debian 数据集由含有漏洞代码片段的 2 240 个样本和不含漏洞代码片段的 20 494 个样本构成. 每一个样本由源码、所属项目、哈希值和大小构成.

CryptoAPI 是由 Rahaman 等人[30]构建的数据集, 其中包含 171 个样例, 对应 16 种 Java 加密漏洞.

OWASP 基准 (benchmark) 项目是一个迄今为止仍在更新的 Java 测试套件, 它的 v1.1 版本共包含 21 041 个可利用的测试用例, 包含 SQL 注入、跨域访问等常见漏洞. 且每个样本都映射到对应的 CWE, 以方便研究者使用.

CodeXGLUE[19]是微软发布的代码智能领域的大规模多任务基准, 其中包括了与漏洞检测相关的数据集. 该数

据集最初由 Zhou 等人[67]收集, 来源于 Linux Kernel、QEMU、Wireshark 和 FFmpegTrain 这 4 个开源项目, 其中训练集、验证集、测试集的大小分别为 21 854, 2 732, 2 732.

7 挑战与展望

7.1 挑　战

通过归纳分析已有的研究成果, 本文发现已有的漏洞感知技术虽然在很多方面已经十分成熟, 但仍然面临以下问题挑战.

• 开源软件生态往往涉及诸多开源软件, 因此整个系统生态的安全保障与每一环都密不可分, 攻击者不再等待公开的漏洞披露而是主动将新漏洞注入开源软件并利用它攻击下游用户. 而传统的漏洞感知技术难以及时发现这种漏洞.

• 在基于机器学习尤其是深度神经网络的研究工作中, 主流做法都是在某些数据集上进行验证, 少有学者探讨其模型检测出漏洞的可解释性. 而在漏洞感知的过程中, 神经网络本身对于使用者来说与黑盒类似, 无法提供更多的解释信息和判断原理, 而数据集与实际场景也不尽相同, 这不免导致使用者对模型实际效用产生怀疑, 这将极大限制漏洞感知模型的有效性和实用性.

• 高质量真实数据集的获取. 很多研究者都是直接在少量实际使用的开源软件进行试验以验证方法的有效性, 少有研究者将其作为一个数据集提出, 不方便后来的研究者复现研究工作和进一步研究. 开源软件的多样性也导致难以在少量的开源软件上验证感知技术的泛化能力. 同时, 数据集标注的正确性对于训练模型至关重要[53], 真正高质量且规模大的数据集十分稀缺.

• 误报率更低且更细粒度的漏洞感知. 当前多数漏洞检测工具还是集中在函数级别的漏洞感知, 而在实际使用中开发者很可能需要更细粒度的如语句级别的漏洞感知以及误报率更低的工具.

7.2 展　望

结合面临的挑战与当前的研究进展, 我们对未来的研究方向做以下展望.

• 构建高质量大规模的开源数据集. 可以基于各种开源项目的代码段, 结合自动标注技术构建出真实、庞大且准确的包含各类漏洞的数据集.

• 结合人工智能领域最新研究进展. 关注人工智能、深度神经网络研究, 将其最新进展与漏洞感知领域的问题结合设计出更加高效且通用性更强的模型. 另一方面, 研究者也可以进一步将动静态的程序分析技术与机器学习技术相结合, 从而更精准有效的感知漏洞. 从论文梳理我们可以发现越来越多的技术不只采用一种方法而是混合多种方法, 如何取各种漏洞感知技术组成一个更大的漏洞感知系统将会是未来一个重要的发展方向.

• 由于生产实际的需要和编程语言的特性, 从表 4 与表 5 可以发现, 现有的数据集和工具大多针对 C/C++, Java 这些编程语言或智能合约场景. 而近几年人工智能、物联网技术的不断发展, 对安全问题提出了新的挑战. 如何结合经典方法与新问题的特性有效地解决这些新生问题, 具有重要的研究意义.

• 开源软件漏洞生命全周期检测与感知. 研究者可以综合漏洞生命周期的 3 个感知阶段感知漏洞技术完成对开源软件漏洞的全周期的感知. 而在开源软件的迭代中, 3 个阶段的技术可以同时发挥作用, 而如何综合全周期漏洞感知技术从而更快捕捉漏洞具有重要的研究意义.

• 基于公共平台的漏洞感知技术的落地. 从本文总结的漏洞感知公开工具可以看出, 主流工具还是由传统技术, 如静态分析和动态测试的方式开发而成, 而基于公共平台的漏洞感知技术因其需要多种实时元数据, 需要大量外部知识等原因难以单独作为一个工具落地. 如何将这种感知技术落地为实际工具, 将会有重要的实践意义.

8 结　语

随着开源软件的广泛使用, 其漏洞安全问题也随之而来. 有许多研究者提出各种技术来感知软件的潜在漏洞来降低漏洞所带来的损失. 本文梳理了近年来漏洞检测与感知领域的已有研究成果, 将其分为基于代码的漏洞感

知技术、基于开源社区讨论的漏洞感知技术和基于软件补丁的漏洞感知技术, 并进行系统的归纳和总结. 最后, 本文对该领域面临的挑战进行了分析和总结, 并对未来研究的方向进行了展望.

References:

[1] Liu BC, Shi L, Cai ZH, Li M. Software vulnerability discovery techniques: A survey. In: Proc. of the 4th Int'l Conf. on Multimedia Information Networking and Security. Nanjing: IEEE, 2012. 152–156. [doi: 10.1109/MINES.2012.202]

[2] Synopsys. [2023] open source security and risk analysis report. 2023 (in Chinese). https://www.synopsys.com/zh-cn/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html

[3] Ami Luttwak AS. Log4shell 10 days later: Enterprises halfway through patching. 2021. https://www.wiz.io/blog/10-days-later-enterprises-halfway-through-patching-log4shell/

[4] Li Y, Huang CL, Wang ZF, Yuan L, Wang XC. Survey of software vulnerability mining methods based on machine learning. Ruan Jian Xue Bao/Journal of Software, 2020, 31(7): 2040–2061 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6055.htm [doi: 10.13328/j.cnki.jos.006055]

[5] Li Z, Zou DQ, Wang ZL, Jin H. Survey on static software vulnerability detection for source code. Chinese Journal of Network and Information Security, 2019, 5(1): 1–14 (in Chinese with English abstract). [doi: 10.11959/j.issn.2096-109x.2019001]

[6] Gegick M, Rotella P, Xie T. Identifying security bug reports via text mining: An industrial case study. In: Proc. of the 7th IEEE Working Conf. on Mining Software Repositories. Cape Town: IEEE, 2010. 11–20. [doi: 10.1109/MSR.2010.5463340]

[7] Pan SY, Zhou JY, Cogo FR, Xia X, Bao LF, Hu X, Li SP, Hassan AE. Automated unearthing of dangerous issue reports. In: Proc. of the 30th ACM Joint European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. Singapore: ACM, 2022. 834–846. [doi: 10.1145/3540250.3549156]

[8] Zhou YQ, Siow JK, Wang CY, Liu SQ, Liu Y. SPI: Automated identification of security patches via commits. ACM Trans. on Software Engineering and Methodology, 2021, 31(1): 13. [doi: 10.1145/3468854]

[9] Zhou JY, Pacheco M, Wan ZY, Xia X, Lo D, Wang Y, Hassan AE. Finding a needle in a haystack: Automated mining of silent vulnerability fixes. In: Proc. of the 36th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Melbourne: IEEE, 2021. 705–716. [doi: 10.1109/ASE51524.2021.9678720]

[10] Shahriar H, Zulkernine M. Mitigating program security vulnerabilities: Approaches and challenges. ACM Computing Surveys, 2012, 44(3): 11. [doi: 10.1145/2187671.2187673]

[11] Lin GJ, Wen S, Han QL, Zhang J, Xiang Y. Software vulnerability detection using deep neural networks: A survey. Proc. of the IEEE, 2020, 108(10): 1825–1848. [doi: 10.1109/JPROC.2020.2993293]

[12] Wang HT, Ye GX, Tang ZY, Tan SH, Huang SF, Fang DY, Feng YS, Bian LZ, Wang Z. Combining graph-based learning with automated data collection for code vulnerability detection. IEEE Trans. on Information Forensics and Security, 2020, 16: 1943–1958. [doi: 10.1109/TIFS.2020.3044773]

[13] Chakraborty S, Krishna R, Ding YRB, Ray B. Deep learning based vulnerability detection: Are we there yet? IEEE Trans. on Software Engineering, 2022, 48(9): 3280–3296. [doi: 10.1109/TSE.2021.3087402]

[14] Gao J, Yang X, Fu Y, Jiang Y, Sun JG. VulSeeker: A semantic learning based vulnerability seeker for cross-platform binary. In: Proc. of the 33rd ACM/IEEE Int'l Conf. on Automated Software Engineering. Montpellier: ACM, 2018. 896–899. [doi: 10.1145/3238147.3240480]

[15] Yamaguchi F, Golde N, Arp D, Rieck K. Modeling and discovering vulnerabilities with code property graphs. In: Proc. of the 2014 IEEE Symp. on Security and Privacy. Berkeley: IEEE, 2014. 590–604. [doi: 10.1109/SP.2014.44]

[16] Feng ZY, Guo DY, Tang DY, Duan N, Feng XC, Gong M, Shou LJ, Qin B, Liu T, Jiang DX, Zhou M. CodeBERT: A pre-trained model for programming and natural languages. In: Proc. of the Findings of the 2020 Association for Computational Linguistics. Association for Computational Linguistics, 2020. 1536–1547. [doi: 10.18653/v1/2020.findings-emnlp.139]

[17] Guo DY, Ren S, Lu S, Feng ZY, Tang DY, Liu SJ, Zhou L, Duan N, Svyatkovskiy A, Fu SY, Tufano M, Deng SK, Clement CB, Drain D, Sundaresan N, Yin J, Jiang DX, Zhou M. GraphCodeBERT: Pre-training code representations with data flow. In: Proc. of the 9th Int'l Conf. on Learning Representations. OpenReview.net, 2021.

[18] Nguyen VA, Nguyen DQ, Nguyen V, Le T, Tran QH, Phung D. ReGVD: Revisiting graph neural networks for vulnerability detection. In: Proc. of the 44th IEEE/ACM Int'l Conf. on Software Engineering: Companion Proc. Pittsburgh: IEEE, 2022. 178–182. [doi: 10.1145/3510454.3516865]

[19] Lu S, Guo DY, Ren S, Huang JJ, Svyatkovskiy A, Blanco A, Clement CB, Drain D, Jiang DX, Tang DY, Ge Li, Zhou LD, Shou LJ, Zhou

L, Tufano M, Gong M, Zhou M, Duan N, Sundaresan N, Deng SK, Fu SY, Liu SJ. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. In: Proc. of the 35th Conf. on Neural Information Processing Systems Track on Datasets and Benchmarks. Openreview.net, 2021.

[20] Hin D, Kan A, Chen HM, Babar MA. LineVD: Statement-level vulnerability detection using graph neural networks. In: Proc. of the 19th Int'l Conf. on Mining Software Repositories. Pittsburgh: ACM, 2022. 596–607. [doi: 10.1145/3524842.3527949]

[21] Li Y, Wang SH, Nguyen TN. Vulnerability detection with fine-grained interpretations. In: Proc. of the 29th ACM Joint Meeting on European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. Athens: CAN, 2021. 292–303. [doi: 10.1145/3468264.3468597]

[22] Cao SC, Sun XB, Bo LL, Wu RX, Li B, Tao CQ. MVD: Memory-related vulnerability detection based on flow-sensitive graph neural networks. In: Proc. of the 44th IEEE/ACM Int'l Conf. on Software Engineering. Pittsburgh: IEEE, 2022. 1456–1468. [doi: 10.1145/3510003.3510219]

[23] Calzavara S, Conti M, Focardi R, Rabitti A, Tolomei G. Machine learning for Web vulnerability detection: The case of cross-site request forgery. IEEE Security & Privacy, 2020, 18(3): 8–16. [doi: 10.1109/MSEC.2019.2961649]

[24] Amouei M, Rezvani M, Fateh M. RAT: Reinforcement-learning-driven and adaptive testing for vulnerability discovery in Web application firewalls. IEEE Trans. on Dependable and Secure Computing, 2022, 19(5): 3371–3386. [doi: 10.1109/TDSC.2021.3095417]

[25] Chen JC. Finding ethereum smart contracts security issues by comparing history versions. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering. Melbourne: IEEE, 2020. 1382–1384.

[26] Perl H, Dechand S, Smith M, Arp D, Yamaguchi F, Rieck K, Fahl S, Acar Y. VCCFinder: Finding potential vulnerabilities in open-source projects to assist code audits. In: Proc. of the 22nd ACM SIGSAC Conf. on Computer and Communications Security. Denver: ACM, 2015. 426–437. [doi: 10.1145/2810103.2813604]

[27] Zhou YQ, Sharma A. Automated identification of security issues from commit messages and bug reports. In: Proc. of the 11th Joint Meeting on Foundations of Software Engineering. Paderborn: ACM, 2017. 914–919. [doi: 10.1145/3106237.3117771]

[28] Chen Y, Santosa AE, Yi AM, Sharma A, Sharma A, Lo D. A machine learning approach for vulnerability curation. In: Proc. of the 17th Int'l Conf. on Mining Software Repositories. Seoul: ACM, 2020. 32–42. [doi: 10.1145/3379597.3387461]

[29] Le THM, Hin D, Croft R, Babar MA. DeepCVA: Automated commit-level vulnerability assessment with deep multi-task learning. In: Proc. of the 36th IEEE/ACM Int'l Conf. on Automated Software Engineering. Melbourne: IEEE, 2021. 717–729. [doi: 10.1109/ASE51524.2021.9678622]

[30] Rahaman S, Xiao Y, Afrose S, Shaon F, Tian K, Frantz M, Kantarcioglu M, Yao DF. CryptoGuard: High precision detection of cryptographic vulnerabilities in massive-sized Java projects. In: Proc. of the 2019 ACM SIGSAC Conf. on Computer and Communications Security. London: ACM, 2019. 2455–2472. [doi: 10.1145/3319535.3345659]

[31] Cui L, Hao ZY, Jiao Y, Fei HQ, Yun XC. VulDetector: Detecting vulnerabilities using weighted feature graph comparison. IEEE Trans. on Information Forensics and Security, 2020, 16: 2004–2017. [doi: 10.1109/TIFS.2020.3047756]

[32] Xu YF, Xu ZZ, Chen BH, Song F, Liu Y, Liu T. Patch based vulnerability matching for binary programs. In: Proc. of the 29th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2020. 376–387. [doi: 10.1145/3395363.3397361]

[33] Xiao Y, Chen BH, Yu CD, Xu ZZ, Yuan ZM, Li F, Liu BH, Liu Y, Huo W, Zou W, Shi WC. MVP: Detecting vulnerabilities using patch-enhanced vulnerability signatures. In: Proc. of the 29th USENIX Security Symp. USENIX Association, 2020. 1165–1182.

[34] Xue YX, Ma ML, Lin Y, Sui YL, Ye JM, Peng TY. Cross-contract static analysis for detecting practical reentrancy vulnerabilities in smart contracts. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering. Melbourne: IEEE, 2020. 1029–1040.

[35] Ma FC, Xu ZY, Ren M, Yin ZJ, Chen YL, Qiao L, Gu B, Li HZ, Jiang Y, Sun JG. Pluto: Exposing vulnerabilities in inter-contract scenarios. IEEE Trans. on Software Engineering, 2022, 48(11): 4380–4396. [doi: 10.1109/TSE.2021.3117966]

[36] Belleville B, Shen WB, Volckaert S, Azab AM, Franz M. KALD: Detecting direct pointer disclosure vulnerabilities. IEEE Trans. on Dependable and Secure Computing, 2021, 18(3): 1369–1377. [doi: 10.1109/TDSC.2019.2915829]

[37] Hough K, Welearegai G, Hammer C, Bell J. Revealing injection vulnerabilities by leveraging existing tests. In: Proc. of the 42nd IEEE/ACM Int'l Conf. on Software Engineering. Seoul: IEEE, 2020. 284–296.

[38] Fu Y, Ren M, Ma FC, Shi HY, Yang X, Jiang Y, Li HZ, Shi X. EVMFuzzer: Detect EVM vulnerabilities via fuzz testing. In: Proc. of the 27th ACM Joint Meeting on European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. Tallinn: ACM, 2019. 1110–1114. [doi: 10.1145/3338906.3341175]

[39] Nguyen TD, Pham LH, Sun J, Lin Y, Minh QT. sFuzz: An efficient adaptive fuzzer for solidity smart contracts. In: Proc. of the ACM/IEEE 42nd Int'l Conf. on Software Engineering. Seoul: ACM, 2020. 778–788. [doi: 10.1145/3377811.3380334]

[40] Zalewski M. American fuzzy lop. 2020. https://lcamtuf.coredump.cx/afl/

[41] Li YK, Xue YX, Chen HX, Wu XH, Zhang C, Xie XF, Wang HJ. Cerebro: Context-aware adaptive fuzzing for effective vulnerability detection. In: Proc. of the 27th ACM Joint Meeting on European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. Tallinn: ACM, 2019. 533–544. [doi: 10.1145/3338906.3338975]

[42] Kim IL, Zheng YH, Park H, Wang WH, You W, Aafer Y, Zhang XY. Finding client-side business flow tampering vulnerabilities. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering. Seoul: ACM, 2020. 222–233. [doi: 10.1145/3377811.3380355]

[43] Wang HJ, Xie XF, Li Y, Wen C, Li YK, Liu Y, Qin SC, Chen HX, Sui YL. Typestate-guided fuzzer for discovering use-after-free vulnerabilities. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering. Seoul: ACM, 2020. 999–1010. [doi: 10.1145/3377811.3380386]

[44] Chen YH, Li P, Xu J, Guo SJ, Zhou RD, Zhang YL, Wei T, Lu L. SAVIOR: Towards bug-driven hybrid testing. In: Proc. of the 2020 IEEE Symp. on Security and Privacy. San Francisco: IEEE, 2020. 1580–1596. [doi: 10.1109/SP40000.2020.00002]

[45] Yu KP, Wang CX, Cai Y, Luo XP, Yang ZJ. Detecting concurrency vulnerabilities based on partial orders of memory and thread events. In: Proc. of the 29th ACM Joint Meeting on European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. Athens: ACM, 2021. 280–291. [doi: 10.1145/3468264.3468572]

[46] Liu YX, Zhang MX, Meng W. Revealer: Detecting and exploiting regular expression denial-of-service vulnerabilities. In: Proc. of the 2021 IEEE Symp. on Security and Privacy. San Francisco: IEEE, 2021. 1468–1484. [doi: 10.1109/SP40001.2021.00062]

[47] Behl D, Handa S, Arora A. A bug mining tool to identify and analyze security bugs using naive bayes and TF-IDF. In: Proc. of the 2014 Int'l Conf. on Reliability Optimization and Information Technology. Faridabad: IEEE, 2014. 294–299. [doi: 10.1109/ICROIT.2014.6798341]

[48] Pereira M, Kumar A, Cristiansen S. Identifying security bug reports based solely on report titles and noisy data. In: Proc. of the 2019 IEEE Int'l Conf. on Smart Computing. Washington: IEEE, 2019. 39–44. [doi: 10.1109/SMARTCOMP.2019.00026]

[49] Peters F, Tun TT, Yu YJ, Nuseibeh B. Text filtering and ranking for security bug report prediction. IEEE Trans. on Software Engineering, 2019, 45(6): 615–631. [doi: 10.1109/TSE.2017.2787653]

[50] Shu R, Xia TP, Chen JF, Williams L, Menzies T. How to better distinguish security bug reports (using dual hyperparameter optimization). Empirical Software Engineering, 2021, 26(3): 53. [doi: 10.1007/s10664-020-09906-8]

[51] Kudjo PK, Chen JF, Zhou MM, Mensah S, Huang R. Improving the accuracy of vulnerability report classification using term frequency-inverse gravity moment. In: Proc. of the 19th IEEE Int'l Conf. on Software Quality, Reliability and Security. Sofia: IEEE, 2019. 248–259. [doi: 10.1109/QRS.2019.00041]

[52] Goseva-Popstojanova K, Tyo J. Identification of security related bug reports via text mining using supervised and unsupervised classification. In: Proc. of the 2018 IEEE Int'l Conf. on Software Quality, Reliability and Security. Lisbon: IEEE, 2018. 344–355. [doi: 10.1109/QRS.2018.00047]

[53] Wu XX, Zheng W, Xia X, LO D. Data quality matters: A case study on data label correctness for security bug report prediction. IEEE Trans. on Software Engineering, 2022, 48(7): 2541–2556. [doi: 10.1109/TSE.2021.3063727]

[54] Pletea D, Vasilescu B, Serebrenik A. Security and emotion: Sentiment analysis of security discussions on GitHub. In: Proc. of the 11th Working Conf. on Mining Software Repositories. Hyderabad: ACM, 2014. 348–351. [doi: 10.1145/2597073.2597117]

[55] Cois CA, Kazman R. Natural language processing to quantify security effort in the software development lifecycle. In: Proc. of the 27th Int'l Conf. on Software Engineering and Knowledge Engineering. Pittsburgh: KSI Research Inc. and Knowledge Systems Institute Graduate School, 2015. 716–721.

[56] Hindle A, Ernst NA, Godfrey MW, Mylopoulos J. Automated topic naming to support cross-project analysis of software maintenance activities. In: Proc. of the 8th Working Conf. on Mining Software Repositories. Honolulu: ACM, 2011. 163–172. [doi: 10.1145/1985441.1985466]

[57] Oyetoyan TD, Morrison P. An improved text classification modelling approach to identify security messages in heterogeneous projects. Software Quality Journal, 2021, 29(2): 509–553. [doi: 10.1007/s11219-020-09546-7]

[58] Le THM, Hin D, Croft R, Babar MA. PUMiner: Mining security posts from developer question and answer websites with PU learning. In: Proc. of the 17th Int'l Conf. on Mining Software Repositories. Seoul: ACM, 2020. 350–361. [doi: 10.1145/3379597.3387443]

[59] Ramsauer R, Bulwahn L, Lohmann D, Mauerer W. The sound of silence: Mining security vulnerabilities from secret integration channels in open-source projects. In: Proc. of the 2020 ACM SIGSAC Conf. on Cloud Computing Security Workshop. New York: Association for Computing Machinery, 2020. 147–157 [doi: 10.1145/3411495.3421360]

[60] Wang XD, Sun K, Batcheller A, Jajodia S. Detecting "0-day" vulnerability: An empirical study of secret security patch in OSS. In: Proc. of the 49th Annual IEEE/IFIP Int'l Conf. on Dependable Systems and Networks. Portland: IEEE, 2019. 485–492. [doi: 10.1109/DSN.2019.00056]

[61] Xu ZZ, Chen BH, Chandramohan M, Liu Y, Song F. SPAIN: Security patch analysis for binaries towards understanding the pain and pills. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Software Engineering. Buenos Aires: IEEE, 2017. 462–472. [doi: 10.1109/ICSE.2017.49]

[62] Sabetta A, Bezzi M. A practical approach to the automatic classification of security-relevant commits. In: Proc. of the 2018 IEEE Int'l Conf. on Software Maintenance and Evolution. Madrid: IEEE, 2018. 579–582. [doi: 10.1109/ICSME.2018.00058]

[63] Li Z, Zou DQ, Xu SH, Jin H, Qi HC, He J. VulPecker: An automated vulnerability detection system based on code similarity analysis. In: Proc. of the 32nd Annual Conf. on Computer Security Applications. Los Angeles: ACM, 2016. 201–213. [doi: 10.1145/2991079.2991102]

[64] Li Z, Zou DQ, Xu SH, Ou XY, Jin H, Wang SJ, Deng ZJ, Zhong YY. VulDeePecker: A deep learning-based system for vulnerability detection. In: Proc. of the 25th Annual Network and Distributed System Security Symp. San Diego: The Internet Society, 2018.

[65] Ponta SE, Plate H, Sabetta A, Bezzi M, Dangremont C. A manually-curated dataset of fixes to vulnerabilities of open-source software. In: Proc. of the 16th IEEE/ACM Int'l Conf. on Mining Software Repositories. Montreal: IEEE, 2019. 383–387. [doi: 10.1109/MSR.2019.00064]

[66] Russell R, Kim L, Hamilton L, Lazovich T, Harer J, Ozdemir O, Ellingwood P, McConley M. Automated vulnerability detection in source code using deep representation learning. In: Proc. of the 17th IEEE Int'l Conf. on Machine Learning and Applications. Orlando: IEEE, 2018. 757–762. [doi: 10.1109/ICMLA.2018.00120]

[67] Zhou YQ, Liu SQ, Siow J, Du XN, Liu Y. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. In: Proc. of the 33rd Conf. on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2019. 10197–10207.

附中文参考文献:

[2] 新思科技. [2023]开源安全和风险分析报告. 2023. https://www.synopsys.com/zh-cn/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html

[4] 李韵, 黄辰林, 王中锋, 袁露, 王晓川. 基于机器学习的软件漏洞挖掘方法综述. 软件学报, 2020, 31(7): 2040–2061. http://www.jos.org.cn/1000-9825/6055.htm [doi: 10.13328/j.cnki.jos.006055]

[5] 李珍, 邹德清, 王泽丽, 金海. 面向源代码的软件漏洞静态检测综述. 网络与信息安全学报, 2019, 5(1): 1–14. [doi: 10.11959/j.issn.2096-109x.2019001]

詹奇(2001－), 男, 博士生, CCF 学生会员, 主要研究领域为智能化软件工程.

鲍凌峰(1988－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为软件工程, 区块链.

潘圣益(1999－), 男, 博士生, CCF 学生会员, 主要研究领域为软件安全, 智能化软件工程.

夏鑫(1986－), 男, 博士, CCF 专业会员, 主要研究领域为智能化软件工程, 软件仓库挖掘, 经验软件工程.

胡星(1993－), 女, 博士, 副教授, CCF 专业会员, 主要研究领域为智能化软件工程, 开源软件供应链安全.