计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
[doi: 10.15888/j.cnki.csa.010286] [CSTR: 32024.14.csa.010286]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

# 漏洞报告自动化复现智能体技术①

袁乐天, 于正民, 聂翌楠, 张　源

(复旦大学 计算与智能创新学院, 上海 200433)
通信作者: 张源, E-mail: yuanxzhang@fudan.edu.cn

**摘　要**: 现有大语言模型智能体工作在自动化复现漏洞报告过程中, 面临着缺少信息源、对应用运行状态关注不足且难以适应复杂环境的技术挑战. 鉴于此, 本文提出一种漏洞报告自动化复现智能体技术 VulVerifier. 该技术运用了 3 项关键技术: 首先, 通过综合分析漏洞描述、源码仓库及相关文档等多信息源内容来对关键信息的补全与完善; 其次, 通过自动化交互来引导目标应用逐步进入满足漏洞触发的前置状态, 从而提升复现漏洞的可行性; 最后, 在复现执行过程中动态感知异常并进行实时纠错, 从而提升对复杂环境的适应能力与复现漏洞的健壮性. 在 86 份真实漏洞报告上的实验结果表明, VulVerifier 的复现成功率达到 44.7%, 接近人类专家 66.3% 的水平; 相较于现有方法 CVE-Genie, 额外成功复现 34 份漏洞报告, 并降低平均复现成本 3.19 美元.

**关键词**: 自动化; 漏洞复现; 大语言模型智能体

引用格式: 袁乐天,于正民,聂翌楠,张源.漏洞报告自动化复现智能体技术.计算机系统应用. http://www.c-s-a.org.cn/1003-3254/10286.html

## Vulnerability Report Automated Reproduction Agent Technology

YUAN Le-Tian, YU Zheng-Min, NIE Yi-Nan, ZHANG Yuan

(College of Computer Science and Artificial Intelligence, Fudan University, Shanghai 200433, China)

**Abstract**: Existing large language model (LLM) agent based approaches to automated vulnerability report reproduction face several technical challenges, including limited information sources, insufficient attention to application runtime states, and difficulty in adapting to complex environments. To address these issues, this study proposes VulVerifier, an agent-based technique for automated vulnerability report reproduction. The proposed technique uses three key components. First, it integrates multiple information sources, such as vulnerability descriptions, source code repositories, and related documentation, to fill in and refine missing key information. Second, it uses automated interactions to guide the target application step by step into a prerequisite state for vulnerability triggering, thus improving the feasibility of vulnerability reproduction. Finally, during the reproduction execution, anomalies are dynamically detected and corrected in real time, thus enhancing adaptability to complex environments and improving the robustness of vulnerability reproduction. Experimental results on 86 real-world vulnerability reports show that VulVerifier achieves a reproduction success rate of 44.7%, which is close to the 66.3% success rate of human experts. Compared with the existing method, CVE-Genie, the proposed system successfully reproduces 34 additional vulnerability reports and reduces the average reproduction cost by 3.19 dollars.

**Key words**: automation; vulnerability reproduction; large language model (LLM) agent

① 基金项目: 国家自然科学基金联合基金重点项目 (U2436207)
收稿时间: 2026-03-11; 修改时间: 2026-04-03; 采用时间: 2026-04-24; csa 在线出版时间: 2026-07-17

国家级漏洞数据库、行业漏洞平台及企业安全响应中心 (security response center, SRC) 等平台每年从厂商、安全研究人员、漏洞赏金计划及安全机构等渠道接收大量报告. 平台需对报告进行初步核查、漏洞验证、影响评估、编号分配及公开披露等操作. 其中, 漏洞验证最为关键且耗时, 通常包括信息收集、环境搭建与漏洞复现等步骤[1,2]. 虽然漏洞验证在漏洞数据库及整个安全生态系统中具有重要作用[2,3], 但在实际情况中, 每份报告需要人工分析描述、阅读补丁、搭建环境并尝试复现, 验证周期通常为数小时甚至数天, 耗时耗力. 由于漏洞报告数量快速增长且验证资源有限, 漏洞验证往往难以充分进行. CVE (common vulnerabilities and exposures) 平台甚至并未将漏洞验证作为发布前的必备步骤[4]. 因此, 漏洞验证已成为制约漏洞处理效率与安全生态运行能力的关键瓶颈. 低效的漏洞验证不仅延长处置周期、增加已知漏洞长期暴露的风险, 还限制了安全团队对大规模漏洞报告的及时响应能力, 进而导致漏洞管理与防御决策滞后, 削弱整体信息安全防护水平, 并对业务连续性和用户信任造成不利影响.

在现实中, 漏洞报告来源多样, 涉及不同软件类型、漏洞类别、漏洞库及报告者, 内容和格式差异明显, 而 Web 应用漏洞在其中占据较大比例[5]. 因此, 本文以 Web 应用漏洞报告为主要研究对象, 同时兼顾其他类型漏洞, 期望设计并实现一种基于漏洞报告的自动化漏洞复现智能体技术, 以提升漏洞平台的处置效率, 进而促进网络安全生态的整体防护水平.

现有工作尝试通过自动化概念验证 (proof of concept, PoC) 生成[6–8]、自动化环境搭建[9,10]以及基于模糊测试的测试用例生成[11–16]等技术, 来缩短漏洞验证流程. 然而, 这些方法通常针对特定技术栈或应用场景, 扩展性和通用性有限. 部分工作尝试利用大语言模型 (large language model, LLM, 简称大模型) 智能体实现漏洞报告的端到端自动化复现[17–19], 但目前尚未形成成熟稳健的方法体系, 复现成功率仍不理想.

具体来说, 现有研究中, 相关智能体在自动化漏洞复现方面成功率不理想的原因主要包括以下 3 大挑战: 一是漏洞报告可能存在着内容形式复杂、信息不充分和描述不准确的问题, 多数报告未详细说明目标应用的安装与配置, 也缺少复现所需关键字段[5,20], 导致信息不足, 影响后续自动化复现流程; 二是目标漏洞应用在初始部署状态下往往不满足漏洞触发的前置条件, 报告往往未明确说明如何调整应用以触发漏洞, 使智能体在复现过程中尝试大量无效操作, 例如某些博客类 Web 应用初始状态下没有业务数据, 因此无法满足漏洞触发的前置条件; 三是漏洞复现操作所需的环境复杂多样且复现步骤异常频发, 漏洞复现通常依赖特定操作系统、语言版本及第三方库, 而报告无法覆盖所有细节, 智能体在环境搭建和漏洞触发中易出现执行错误, 最终导致复现失败.

为解决上述挑战, 本文提出了名为 VulVerifier 的漏洞报告自动化复现智能体. 该智能体运用了 3 项关键技术, 包括漏洞信息补全技术、前置状态构建技术与纠错执行复现技术. 漏洞信息补全技术综合利用源代码、漏洞报告及应用文档等多模态信息, 对缺失或不完整的关键信息进行补全, 提升漏洞复现的可靠性; 漏洞前置状态构建技术则基于漏洞报告、应用界面及使用文档, 推断漏洞触发所需的应用状态, 引导应用达到必要前置条件, 提高漏洞复现的可行性; 纠错执行复现技术通过“感知、推理和执行”的反馈闭环, 实现执行异常的自动感知、分析与纠正, 增强复现过程的稳定性与成功率.

综上所述, 本文的主要贡献如下.

(1) 面向自动化漏洞复现中的关键挑战, 以 Web 应用漏洞报告为核心研究对象, 并兼顾其他类型漏洞, 提出漏洞信息补全、前置状态构建与纠错执行复现这 3 项技术, 实现了 VulVerifier 的漏洞报告自动化复现智能体, 有效缓解信息缺失、应用状态依赖及缺乏动态纠错等问题, 从而提升复现智能体的整体性能.

(2) 构建包含 86 份真实漏洞报告的数据集, 并对 VulVerifier 进行系统评估, 实验结果表明其在数据集中达到 47.7% 的复现成功率, 在复现效果与效率上均优于现有 SOTA 工具 CVE-Genie.

## 1 相关工作

已有的与漏洞报告复现相关研究工作包括漏洞复现相关实证研究、漏洞复现辅助技术以及基于智能体的漏洞自动化复现技术.

### 1.1 漏洞复现相关实证研究

Mu 等人[21]花费了 1 600 工时, 对 368 个真实世界的安全漏洞进行了首次人工的实证分析, 成功复现了 368 个漏洞中的 202 个 (54.9%), 平均每个漏洞花费

4.3 个工时. 他们发现, 有 95.1% 的漏洞往往会缺少一些复现漏洞所需的必要信息字段, 87% 的报告不会阐述软件安装与配置.

漏洞报告中缺少了复现漏洞的必要信息在实践中是十分常见的情况. Bettenburg 等人[22]基于开发人员问卷研究指出, 漏洞报告中常缺乏复现所需的关键信息. 同时, Rahman 等人[23]对 576 份不可复现的报告进行了系统实证分析, 归纳出 11 类不可复现的主要成因, 包括误报、关键细节缺失以及描述模糊或存在歧义等问题. 此外, Johnson 等人[24]对 180 份安卓应用报告的深入研究表明, 报告中运行环境与配置细节往往提供不完整, 实际案例中普遍存在信息缺失现象, 进一步降低了漏洞报告在复现阶段的有效性与可靠性.

### 1.2 漏洞复现辅助技术

漏洞复现通常包含多个环节, 如阅读与理解报告、环境搭建、PoC 脚本编写、执行验证及结果确认等. 该过程步骤复杂且高度依赖人工判断, 往往耗时耗力. 因此, 研究者尝试将脚本编写与环境配置等关键环节自动化, 以缩短整体复现时间并提升验证效率.

Huynh 等人[5]提出一种自动化方法生成 Web 应用事件序列的可执行测试脚本. Simsek 等人[7]提出 PoCGen, 用于自动生成并验证 npm 漏洞的 PoC. Kang 等人[8]提出 Libro 框架, 利用大语言模型处理代码任务并生成测试用例. Nitin 等人[25]提出一种基于大模型智能体的自动化流程 FaultLine, 用于生成漏洞触发样例. 尽管上述方法取得一定效果, 但普遍受限于特定编程语言、框架及漏洞类型. 例如, 文献[5]的工作主要面向 Web 应用, 难以适配其他系统; PoCGen[7]依赖 JavaScript 及 npm 生态; Libro[8]局限于 Java 与 JUnit 框架. 这类方法对特定技术栈依赖较强, 通用性与可扩展性有限.

模糊测试 (fuzz) 是生成测试用例的有效手段之一. 现有研究主要包括黑盒模糊测试 (black-box fuzzing)[11]、导向式模糊测试 (target guided fuzzing)[12,13]以及混合式模糊测试 (hybrid fuzzing)[14,15]等. 上述方法在漏洞发现与 PoC 生成方面已取得显著成效. 然而, 模糊测试对漏洞类型和应用形态依赖较强, 不同应用类别需要差异化的模糊机制与工程实现, 例如 Web 应用通常通过构造并发送网络请求驱动执行[16], 而二进制程序多依赖标准输入或文件输入并持续变异以探索路径. 这使得模糊测试工具高度专用化, 难以以统一方式适配多样化漏洞类型与应用场景.

与此同时, Ruan 等人[9]重点关注了复现所需环境的构建, 并提出一种自动生成内核漏洞易受攻击环境的方法. 类似地, Chen 等人[10]基于二进制相似性实现漏洞环境的自动化配置. 另有工作通过分析 Web 界面并选择合适的交互事件, 引导对应用的自动探索[26]. 然而, 这类方法通常面向特定漏洞类型或场景, 通用性有限.

总体来看, 已有方法虽在各自场景取得进展, 但普遍依赖特定编程语言或软件框架, 复现流程受领域约束明显, 难以扩展到不同编程语言及软件形态的通用复现任务. 因此, 在自动化漏洞复现智能体的更广泛应用中, 仍需一种融合漏洞语义理解、环境构建与执行验证的统一框架, 以提升通用性与可扩展性.

### 1.3 基于智能体的漏洞自动化复现研究

随着大语言模型技术的发展, 它们在语言推理、代码生成和任务执行等方面展现出显著潜力. 在漏洞复现这一长期依赖人工经验和时间的任务中, 利用大模型智能体进行漏洞总结、PoC 生成甚至自动复现, 已成为研究热点.

Fang 等人[17]简单研究了 LLM 在复现 CVE 漏洞中的能力. 作者选取 15 个现实 CVE 样本, 并为模型提供执行命令、网络搜索及文件编辑等工具. 实验结果显示, 当提供漏洞描述时, GPT-4 可自主成功利用 87% 的漏洞, 但在缺少描述时成功率仅为 7%.

Ullah 等人[18]提出了一个基于 LLM 的多智能体自动化漏洞复现框架 CVE-Genie. 该框架以 CVE 条目为输入, 自动收集资源、重建漏洞环境并生成可验证 PoC. 作者构建了包含 841 个 2024–2025 年 CVE 条目的数据集, 并声称能复现约 51% 条目. 然而, 本文对比评估显示, 其中存在大量虚假成功 (false success), 即模型认为漏洞被复现, 但实际未成功. CVE-Genie 在端到端复现过程中, 模型幻觉与错误在各阶段累积, 导致虚假成功频发.

Liu 等人[19]比较了不同大语言模型与智能体的漏洞复现能力. 他们收集了 80 个多类型漏洞样本, 其中 Claude-Sonnet-4 与智能体 OpenHands 组合的端到端复现成功率最高, 为 22.5%.

总体来看, 大模型智能体在漏洞复现自动化领域仍处于早期阶段, 尚未形成成熟和稳健的方法体系. 现有研究面临模型幻觉、虚假成功及智能体能力不足等问题, 这限制了其应用与推广. 因此, 迫切需要开发一

种兼具可靠性、稳定性和高效率的漏洞复现智能体, 以减少幻觉和虚假成功, 并在复杂安全环境中保持高水平能力, 从而提升自动化漏洞验证的效率和可靠性.

## 2 系统实现

VulVerifier 的整体架构如图 1 所示. 系统以漏洞报告为输入, 以是否成功复现漏洞为输出, 构建了一个端到端的漏洞复现自动化大模型智能体. 其工作流程主要包括以下几个步骤: 首先, 通过漏洞信息补全技术对报告中可能缺失的关键信息进行搜集与补全; 随后, 利用纠错执行复现技术对报告中描述的目标应用进行搭建; 接着, 借助漏洞前置状态构建技术, 逐步引导目标应用进入漏洞复现所需的前置状态; 然后, 再次采用纠错执行复现技术对报告中的操作步骤进行执行, 从而尝试完成漏洞复现; 最后, 为避免大模型的局限性对复现结果的准确性产生影响, 系统还对复现结果进行独立验证, 以降低智能体出现虚假成功或虚假失败的风险.

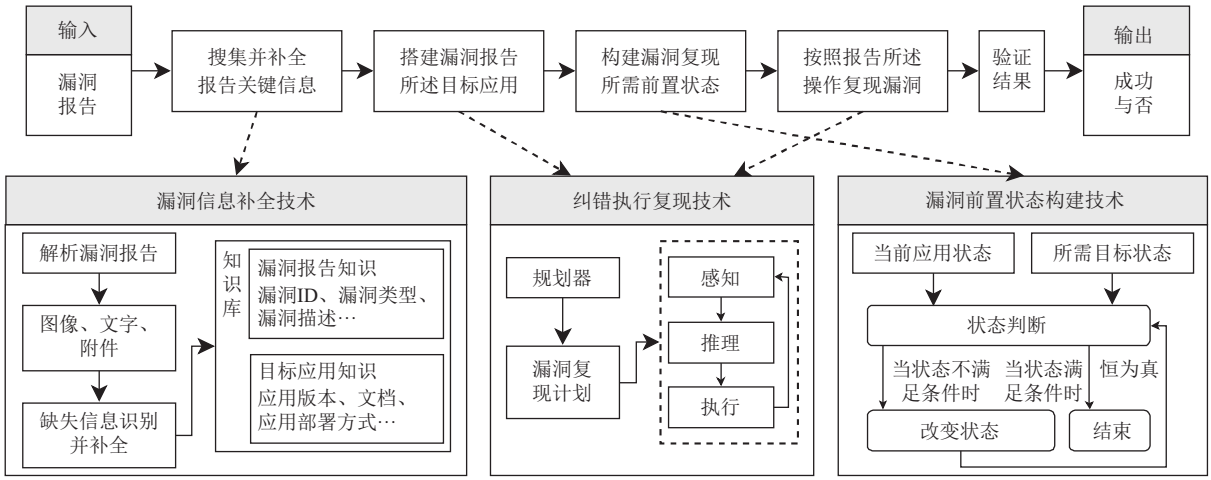


图 1 VulVerifier 整体架构图

此外, 本文为智能体设计了丰富的工具集, 共包含 8 个工具: 切换工作目录、执行命令、读取文本文件、读取多模态文件、写入文件、读取网页内容、等待指定时间以及操作浏览器. 通过提供多样化工具, 增强了智能体与外部环境的交互能力. 同时, 针对不同功能模块分配差异化工具权限, 以实现对智能体行为的有效约束.

接下来将详细介绍该智能体的 3 个关键模块所用到的技术.

### 2.1 漏洞信息补全

图 2 给出了一个真实漏洞报告示例. 报告中依次描述了受影响应用、漏洞概述以及复现步骤. 然而, 其中仍缺少若干关键细节, 如目标应用的获取与部署方式以及漏洞根因信息. 在现实漏洞报告中, 此类信息缺失的情况十分常见[21], 其原因包括报告者写作习惯差异以及漏洞平台要求不统一等. 为解决漏洞报告关键信息缺失导致漏洞难以复现的问题, 本文设计了漏洞信息补全模块.

首先, 模块需要对漏洞报告进行解析. VulVerifier 支持解析多种漏洞报告格式, 包括 PDF、Word、Markdown、HTML、在线网页、图片和纯文本. 考虑到漏洞报告通常包含文本、图片等多模态信息, 为统一处理不同格式文档, 本文将报告映射为三元组形式:

$$Report \mapsto \langle T, I, S \rangle \tag{1}$$

其中, $Report$ 表示原始漏洞报告; $T$ (Text) 表示文本内容; $I$ (Image) 表示报告中的图片; $S$ (Screenshot) 表示对报告生成的屏幕截图.

**Stored XSS in PDF renderer**

**Summary**

A stored XSS is present in Gogs which allows client-side Javascript code execution.

**Application version**

0.14.0+dev

**PoC**

1. Upload the Proof of Concept file hosted at https://codeanlabs.com/wp-content/uploads/2024/05/poc_ge neralized_CVE-2024-4367.pdf in a repository.
2. Click on the file to be previewed.

图 2 CVE-2024-4367 的漏洞报告

在解析过程中, 系统同时保留文本、图像及生成的截图. 这是因为部分格式 (如 PDF) 中的文本与图片按坐标组织, 直接排序可能与实际阅读顺序不一致, 截图可缓解该问题. 同时, 截图可能因缩放导致文字过小或图像模糊, 因此仍保留原始文本与图片以避免信息丢失. 对于附件, 系统将其本地路径或 URL 加入 $T$, 供后续模块下载使用. 视频内容则不做处理, 因为此类情况较少且视频解析对大模型要求较高.

知识库是漏洞信息补全模块的核心输出. 完整的漏洞报告知识库主要包含两类信息: 目标应用信息与漏洞报告信息. 前者包括应用名称、版本、所需编程语言及版本、部署方式和使用文档等; 后者包括漏洞类型、漏洞成因、受影响版本及复现步骤等. VulVerifier 会对各信息条目逐项检查以识别缺失内容, 并调用智能体工具进行搜集与补全, 最终构建完整知识库. 在实现上, 知识库被表示为信息条目与其取值之间的映射关系:

$$\{E \mapsto V\} \tag{2}$$

其中, $E$ (Entry) 表示知识库条目, 如“目标应用名称”或“受影响版本”; $V$ (Value) 表示对应取值, 例如图 2 中的“Gogs”与“0.14.0+dev”.

漏洞信息补全流程如图 3 所示. 该阶段以目标应用源码目录结构及解析得到的$\langle T, I, S\rangle$为输入. 利用切换工作目录、执行命令、读取文本文件、读取多模态文件和读取网页内容这 6 个工具, 分析与漏洞报告或应用部署相关的源码与文档, 并结合已有信息, 对关键条目进行检查与补全, 最终得到一份包含完整信息内容的知识库.

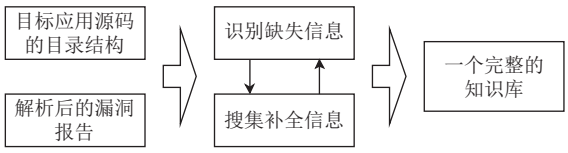


图 3 漏洞信息补全的具体流程

需要说明的是, 本文的智能体面向漏洞平台的漏洞验证人员设计, 而平台收到的报告通常为尚未公开的零日漏洞. 为贴近该场景, 实验中将公开漏洞报告视为“未公开零日漏洞”. 因此, 在本模块中尽量避免搜索与漏洞报告直接相关的互联网信息. 具体通过两种方式实现.

(1) 在提示词中明确禁止搜索与漏洞报告相关的互联网信息.

(2) 若实验中发现智能体尝试检索相关信息, 则人工终止该次运行并重新实验.

上述限制仅针对漏洞报告信息, 由于目标应用本身通常公开, 其相关信息的获取不受该限制.

### 2.2 漏洞前置状态构建

在许多漏洞复现流程中, 漏洞通常无法在目标应用初始安装状态下触发, 因应用刚部署时缺少必要业务数据或用户身份, 这在 Web 漏洞报告中尤为常见.

然而, 自动化复现中报告往往未指明如何补充这些前置条件, 增加了复现难度. 为解决该问题, 本文设计了漏洞前置状态构建模块, 引导目标应用逐步达到复现所需状态. 模块可使用切换目录、执行命令、读写文件、等待和浏览器操作等 6 类工具.

以图 2 中的漏洞报告为例, 其中缺失的关键步骤包括应用安装、用户注册及仓库创建. 应用部署完成后系统中既没有已注册用户, 也没有任何 Git 仓库, 而这两者是触发漏洞的必要条件. 然而, 这些步骤没有写入漏洞报告, 因为报告假设读者已经拥有可用的应用环境和必要数据. 如果智能体无法识别并补充这些前置条件, 漏洞触发操作将必然失败.

为此, 漏洞前置状态构建技术如算法 1 所示, 以已部署应用、知识库和工具集为输入. 算法首先分析应用当前状态, 并结合知识库推断漏洞触发所需目标状态, 同时引入最大执行步数避免无限循环, 本文根据经验将其设为 10.

算法 1. 漏洞前置状态构建技术算法

1) 基于目标应用和知识库信息, 利用大模型推断当前状态;
2) 基于目标应用和知识库信息, 利用大模型推断目标状态;
3) 若无需状态变更, 则算法终止并返回“成功”;
4) 利用大模型选择并调用工具集中的工具, 尝试修改应用当前状态, 同时记录操作步数 *steps*;
5) 若当前状态已达到目标状态, 则算法终止并返回“成功”;
6) 若操作步数 *steps* 超过阈值 10, 则算法终止并返回“失败”;
7) 回到第 4) 步.

每轮迭代中, 若当前状态未满足前置条件, 智能体利用工具集进行交互操作, 使应用状态向目标状态演进; 若条件已满足, 则返回“成功”.

图 4 展示了漏洞前置状态构建模块在复现 CVE-2024-4367 时的一轮流程. 模块融合界面截图、知识库与漏洞报告等多源信息, 推断目标应用所需前置状态, 并进入“判断状态和改变状态”的循环. 在判断阶段, 基

于当前环境评估是否满足条件; 在改变阶段, 推断并执行相应操作以逐步逼近目标状态. 该过程持续迭代, 直至满足前置条件并终止, 从而在复杂交互环境中构建可复现漏洞的运行状态.

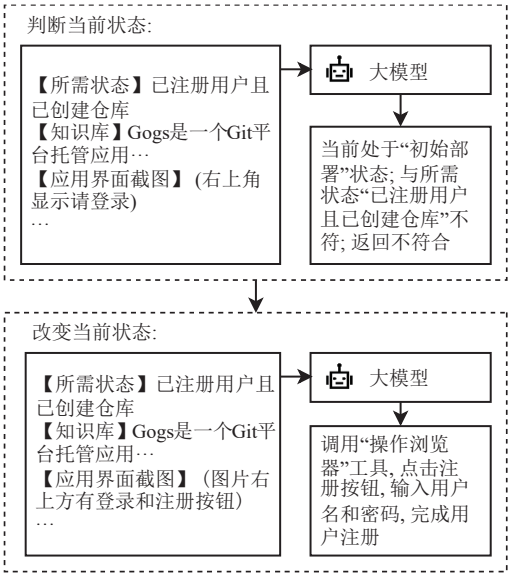


图 4 漏洞前置状态构建模块在复现 CVE-2024-4367 过程中的一次迭代流程示例

### 2.3 纠错执行复现

漏洞复现操作所需的环境复杂多样, 漏洞复现依赖特定操作系统、语言版本及第三方库, 而报告无法覆盖所有细节, 智能体在环境搭建和漏洞触发中易出现执行错误, 最终导致复现失败. 在图 1 的漏洞报告中, 文件上传方式并未说明, 既可能通过 Web 界面, 也可能通过 Git 命令实现. 实际执行时可能遇到按钮缺失、命令报错或网络超时等问题. 这些异常往往在不同阶段反复出现, 对自动化流程造成持续干扰. 若缺乏异常感知与处理能力, 智能体容易陷入重复尝试或流程停滞.

为此, 本文设计了具备异常感知与纠错能力的纠错执行复现模块, 以保障智能体在搭建与复现阶段的稳定运行. 模块在两阶段流程一致, 本文以漏洞复现阶段为例进行说明. 本文为纠错执行复现模块配备了工具集中的全部工具, 以赋予其最大操作权限.

规划器是模块核心组成部分. 漏洞复现任务包含大量步骤, 大模型在处理时容易出现幻觉、错误累积及指令遗忘, 直接端到端执行难以保证稳定性. 因此, 将任务分解为可执行子任务是提升成功率与稳定性的有效方法. 规划器会分析解析后的漏洞报告$\langle T, I, S\rangle$与知识库$\{E \mapsto V\}$, 结合其中漏洞复现知识, 对各子任务进行规划, 并生成完整复现计划, 整个过程如式 (3) 所示:

$$P = f_{\text{plan}}(\langle T, I, S\rangle \oplus \{E \mapsto V\}) \tag{3}$$

其中, $\oplus$表示将两侧内容字符串化并进行字符串拼接, $f_{\text{plan}}$是指将拼接后的字符串内容输入给大模型以生成复现计划 $P$. 在得到了复现计划 $P$ 之后, 接下来将会去执行该复现计划. 然而, 在漏洞复现任务中, 目标应用、环境配置及交互结果通常存在较强不确定性, 使一次性规划难以保证顺利完成. 为提升复现稳定性, VulVerifier 在纠错执行复现模块中引入异常感知与纠错机制, 使智能体在每步操作后感知执行结果, 并动态调整后续策略, 从而及时发现并修正偏差. 该机制借鉴了 ReAct (reasoning and acting) 算法思想[27], 在执行过程中交替进行推理与行动, 已广泛应用于大模型智能体系统.

算法 2. 纠错执行复现技术中感知、推理与执行的算法

1) 根据执行计划、知识库、当前环境状态、工具集和目标任务推理下一步操作;
2) 从工具集中选择执行所需的工具;
3) 使用选定工具执行操作, 记录已进行的操作次数 *steps*;
4) 记录操作输出及其对环境的影响, 感知当前状态;
5) 若操作已完成目标任务, 则算法终止并返回“成功”;
6) 若操作步数 *steps* 超过阈值 50, 则算法终止并返回“失败”;
7) 回到第 1) 步.

算法 2 展示了模块的整体流程. 算法以执行计划 $P$、知识库、当前环境状态、工具集和目标任务为输入, 驱动智能体完成目标任务的执行. 每轮循环中, 智能体结合当前状态和各种信息进行推理, 生成待执行动作并选择工具执行, 同时获取执行反馈. 反馈用于更新当前状态并辅助后续推理, 形成推理、执行与感知闭环. 为避免无限循环, 算法设最大执行步数为 50, 超出阈值则返回失败. 该算法允许智能体根据执行反馈动态调整行为, 而非严格遵循计划 $P$, 这在信息可能不完整或含错误的情况下尤为必要. 通过这种自适应执行方式, 模块显著提升了复杂漏洞复现任务的稳定性与成功率.

最后, 为了避免验证结果受到错误累计、大模型幻觉或遗忘等因素的影响, 本文设计并实现了结果验证环节, 用于对自动化复现结果进行独立评估. 该环节接受漏洞报告、知识库和前述模块的输出作为输入, 不会直接采信前述任何模块的结论, 最终对漏洞是否被实际触发进行独立分析与判定, 从而减少大模型的固有缺陷造成的误判.

## 3 实验

### 3.1 实验设置

本文基于 Python 3.12.9 和 LangGraph 1.0.4, 实现了共 4016 行代码的 VulVerifier 系统. 系统调用的大模型通过 OpenAI API[28]远程访问, 未在本地部署. 为提升漏洞复现的稳定性, 模型的随机性系数 (temperature) 设为 0.1.

所有实验在单台虚拟机中完成, 该虚拟机配备 8 GB 内存和 16 个 AMD Ryzen 9 7945HX 处理器内核, 操作系统为 Ubuntu 22.04.5, 具备图形界面支持, 以保证对 Web 应用的自动化操作顺利执行. 实验环境独立运行于虚拟机内, 确保安全性与可复现性. 每完成一份报告即重置虚拟机, 避免任务间干扰.

VulVerifier 主要以命令行形式运行, 本身不需要图形界面, 但其运行环境仍建议具备图形界面支持, 以保证对涉及浏览器操作的 Web 应用漏洞的有效复现. 系统通过接收漏洞报告, 并在终端环境中调用各个模块完成任务, 输出相应的复现结果与日志信息. 系统采用模块化设计, 依次完成漏洞报告解析、目标应用部署、前置状态构建和漏洞复现及结果验证.

在进行实验时, 每份漏洞报告均重复运行 3 次, 若任意一次复现成功, 则判定 VulVerifier 具备复现该报告的能力.

在实验设计方面, 本文主要设计了实验以评估各个大模型的表现、本文工具的性能、各个模块的贡献、与现有 SOTA 工具 CVE-Genie[18]的对比以及与人类专家的对比.

### 3.2 数据集构建

本文使用的数据集由两部分组成. 一部分来源于合作方国家信息安全漏洞库 (China National Vulnerability Database of Information Security, CNNVD) 为我们提供的 40 份未公开漏洞报告, 另一部分是从公开数据集 Exploit-DB (https://www.exploit-db.com/) 收集的 105 份漏洞报告, 总计 145 份. 其中公开数据集的收集流程如下: 人工查阅截至实验前 Exploit-DB 的最新的包含漏洞报告的 105 个条目.

随后, 剔除了 145 份报告中因条件受限无法复现或存在伦理问题的条目, 最终得到 86 份可用于实验的报告. 这 59 份漏洞报告被剔除的原因, 主要分为 5 类.

(1) 无法获取指定版本的应用 (34 份). 主要出现在闭源软件中, 有两种情况: 一是开发者关闭了历史版本下载渠道, 部分开发者修复漏洞后仅提供最新版本; 二是应用获取依赖申请或付费机制, 即便申请或支付费用, 也无法获得包含漏洞的历史版本.

(2) 不支持的安装方式 (8 份). 包括操作系统安装、应用框架级安装及仅支持 Windows 平台的应用. 操作系统安装要求独立环境和完整工具链; 应用框架的安装则要求智能体首先基于目标框架编写并部署可执行程序, 这不在本文的研究范围中; Windows 应用需不同技术栈. 为降低技术成本且考虑到这类报告数量较少, 本文剔除此类报告.

(3) 未知应用 (7 份). 漏洞报告未明确受影响应用或无法定位目标应用, 导致无法复现.

(4) 需要特定硬件 (5 份). 部分漏洞复现依赖特定硬件, 如特定厂商或型号的路由器. 由于获取和部署成本高, 本文剔除此类报告.

(5) 存在伦理问题 (5 份). 个别漏洞需在线系统复现, 可能破坏真实环境. 基于研究伦理, 本文拒绝对这些漏洞报告进行实验.

### 3.3 大模型选择

本文选取了 10 份流程简短且易复现的漏洞报告, 用以评估不同大语言模型对工具性能的影响. 同时, 选择若干常见大模型, 对上述报告进行复现实验.

实验结果如表 1 所示. DeepSeek-chat 因非多模态模型, 无法处理包含图片信息的漏洞报告, 导致部分依赖界面或截图的漏洞未能复现. Qwen-vl-max 整体表现稳定, 但指令遵循能力弱于 GPT-4.1-mini, 执行过程中易偏离提示词, 导致复现失败率较高. Qwen2-vl-72b-instruct 受限于模型规模, 处理复杂漏洞报告时能力不足, 失败率较高. 相比之下, GPT-4.1-mini 在所选 10 份报告中均成功复现, 体现出较强综合能力和稳定性. GPT-5-mini 存在过度帮助倾向、指令遵循不足及工具调用静默问题, 常对提示词过度推断, 导致复现流程偏离初始指令且忽略可用工具, 失败率极高.

表 1 不同大语言模型在简单数据集上的复现结果

| 大模型 | 复现成功 | 成功率 (%) |
|---|---|---|
| DeepSeek-chat | 4/10 | 40 |
| Qwen-vl-max | 4/10 | 40 |
| Qwen2-vl-72b-instruct | 1/10 | 10 |
| GPT-4.1-mini | 10/10 | 100 |
| GPT-5-mini | 1/10 | 10 |

因此, 本文最终选择 GPT-4.1-mini 作为 VulVerifier

的大模型底座, 并在此基础上开展后续实验与评估.

### 3.4 漏洞自动化复现性能评估

为评估 VulVerifier 在自动化漏洞复现任务中的效果, 本研究对前述 86 份漏洞报告进行了系统实验, 并从成功率、Token 消耗及时间开销等维度分析其表现.

图 5 显示了智能体在 86 份漏洞报告上的整体结果, 其中 65 份报告完成了目标应用的安装与部署, 进一步有 41 份报告成功复现漏洞, 整体复现成功率为 47.7% (=41/86).

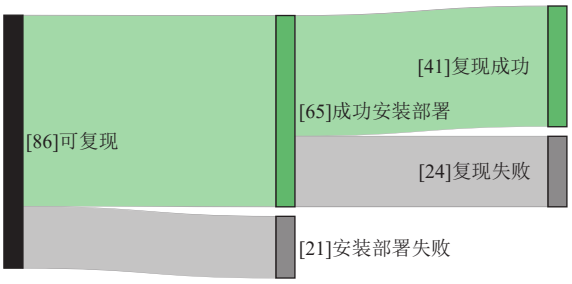


图 5 86 份漏洞报告的复现结果

表 2 显示了 VulVerifier 各阶段的平均大模型成本与时间消耗. 在统计阶段成本时, 仅考虑实际执行该阶段的漏洞报告, 例如应用搭建失败的报告不会纳入后续状态构建统计. 实验结果显示, 应用搭建阶段的成本和时间均最高, 这主要源于智能体需读取大量在线文档、配置文件和源码, 同时执行网络请求和耗时命令, 如安装包下载与应用启动. 纠错机制和技术容错设计使智能体可能多次尝试长耗时命令, 进一步延长平均耗时. 状态构建阶段消耗较低, 因为多数应用初始部署状态已满足前置条件, 无需复杂交互. 验证阶段消耗最低, 智能体仅进行固定一轮对话判断漏洞复现情况.

表 2 VulVerifier 在各个阶段的平均大模型成本与时间消耗

| 阶段 | 平均输入Token | 平均输出Token | 平均耗时 (s) |
|---|---|---|---|
| 漏洞信息补全 | 143 538 | 1 862 | 85 |
| 应用搭建 | 398 959 | 21 541 | 4 696 |
| 漏洞前置状态构建 | 25 465 | 1 340 | 18 |
| 漏洞复现 | 88 326 | 3 866 | 358 |
| 结果验证 | 10 521 | 70 | 5 |
| 总计 | 666 809 | 28 679 | 5 162 |
| 总计金额 (美元) | 0.528 | 0.092 | — |

对 VulVerifier 的实验结果分析显示, 其自动化复现失败的原因主要包括两类: 21 份报告因目标应用安装或部署失败而未能复现, 24 份报告在漏洞复现过程中失败. 主要原因归纳如下.

(1) 应用搭建过于复杂. 该问题集中于 WordPress 及其插件, 安装流程涉及 PHP 环境、数据库部署与配置、填写安装页面、管理员账户设置及插件安装激活等多个相互依赖步骤. 一旦某环节版本错误, 整个搭建流程即失败. 智能体虽可多次尝试完成部署, 但在每样本仅允许 3 次尝试的限制下, 复杂应用的成功率仍较低.

(2) 搭建错误版本. 智能体可能未能识别所部署版本与漏洞版本不一致, 例如部分流程使用 docker pull xxx:latest 拉取最新镜像, 而最新版本通常已修复漏洞.

(3) 配置应用失败. 当目标应用缺乏详细文档, 智能体无法获取必要配置信息, 如数据库连接参数, 导致应用无法正常运行.

(4) 按报告操作无法复现. 原因包括报告自身存在隐蔽错误 (如恶意载荷缺失空格或引号、端口号或命令参数错误) 或环境条件描述不完整.

(5) 智能体工具不支持的操作. 包括鼠标悬停、浏览器报文篡改、多命令行窗口并行操作等, 这类操作实现复杂且在报告中出现频率低, 因此未被支持.

(6) 智能体无法推断操作流程. 在部分场景中, 即便结合图形界面、漏洞报告及官方文档, 智能体仍无法判断正确操作路径, 导致无法完成复现流程.

### 3.5 漏洞自动化复现消融实验评估

为评估 VulVerifier 中漏洞信息补全模块与漏洞前置状态构建模块对系统性能的影响, 本文对两模块分别进行了消融实验.

图 6 显示移除漏洞信息补全模块后的结果, 应用安装部署成功率明显下降, 整体漏洞复现率由 47.7% 降至 2.3% (=2/86), 几乎无法完成任务. 主要原因在于多数漏洞报告未提供完整安装部署信息. 唯一成功的两份报告中, 一份描述较完整, 另一份部署过程简单, 仅需一条 pip 安装命令即可完成. 实验结果表明, 当报告缺少关键部署信息时, 智能体难以仅凭先验知识完成安装部署, 验证了漏洞信息补全模块的重要性.

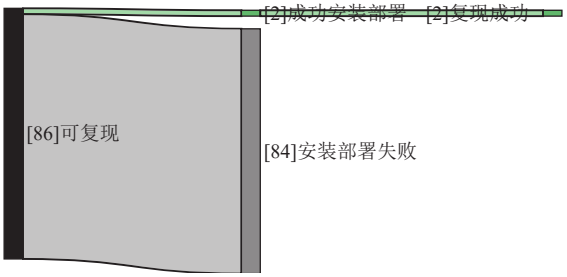


图 6 VulVerifier 在去除了漏洞信息补全模块后的实验结果

图 7 展示去除漏洞前置状态构建模块后的实验结果, 漏洞复现率从 47.7% 降至 41.9% (=36/86). 尽管多数应用无需额外前置操作, 但依赖特定业务数据或用户身份的场景中, 智能体无法凭漏洞报告完成隐含前置条件. 例如博客应用需先发布文章, Git 托管平台需提前建仓库, 这些操作通常未在报告中说明. 实验验证了漏洞前置状态构建模块在复杂漏洞复现场景中的有效性和必要性.

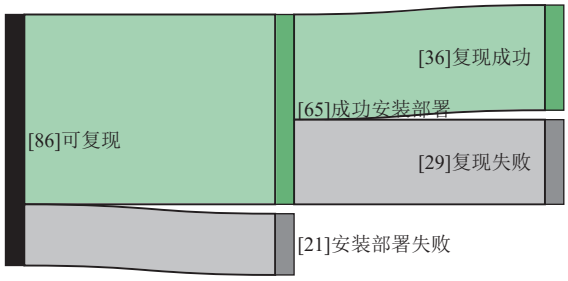


图 7 VulVerifier 在去除了漏洞前置状态构建模块后的实验结果

### 3.6 漏洞自动化复现对比评估

CVE-Genie[18]是目前在漏洞报告端到端自动化复现智能体研究中的 SOTA 工具. 因此, 本节评估了 VulVerifier 与现有工作 CVE-Genie 在漏洞报告自动化复现任务中的表现. 实验基于本文构建的数据集, 对比分析两者复现成功率, 并探究 CVE-Genie 的失败原因. CVE-Genie 源代码来自其作者在 GitHub 上公开版本 (https://github.com/saadullah01/submission/), 虽然 CVE-Genie 针对 CVE 条目设计, 但其也可接受漏洞报告作为输入. 为保证公平性, 实验为两工具提供相同漏洞报告、大模型配置与硬件环境.

图 8 显示 CVE-Genie 在 86 份报告中成功复现 7 份, 复现率 8.1%. 而 VulVerifier 能够额外成功复现 34 份报告, 复现率明显优于 CVE-Genie. 同时, CVE-Genie 的平均 Token 成本为 3.81 美元, 比 VulVerifier 平均每份漏洞报告额外多出 3.19 美元.

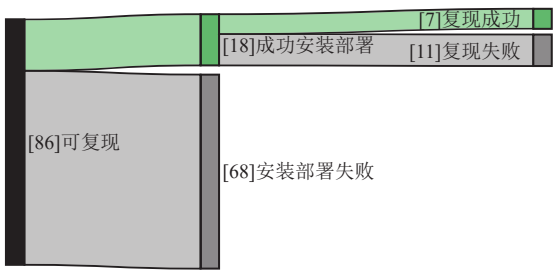


图 8 CVE-Genie 在本文数据集上的实验结果

本文深入分析了 CVE-Genie 的失败原因, 可归纳为 3 类.

(1) 执行模块设计缺陷. CVE-Genie 采用 Plan、Execute 和 Critic 三阶段架构, 其中 Plan 生成执行计划, Execute 执行操作, Critic 判断是否成功. Critic 判断失败时会重复执行, 假设 Plan 完全正确且必须严格遵循, 缺乏动态调整能力. 遇到非预期环境或异常情况时, 容易陷入无效迭代, 直至时间或 Token 预算耗尽.

(2) 工具能力不足. 工具集有限, 无法操作浏览器或 Web 界面, 导致依赖交互的漏洞复现失败.

(3) 虚假成功. 复杂场景中, CVE-Genie 倾向通过重复尝试模拟成功, 错误判定漏洞已复现. 例如在复现漏洞 CVE-2025-31131 时, 目标应用未成功搭建, CVE-Genie 临时生成脚本模拟漏洞行为, 大模型解释为“模拟已成功部署的目标系统”. 这种行为维持流程连续性, 但掩盖了应用搭建失败, 最终导致虚假成功判定, 并未真正复现漏洞.

### 3.7 漏洞自动化复现人类专家对照评估

为评估 VulVerifier 是否接近人类专家水平, 本文设计了人工复现实验, 对数据集中的漏洞报告进行复现统计. 实验为专家提供与 VulVerifier 相同的虚拟机环境. 每个任务限制 2 h, 与 VulVerifier 平均执行时间 5 162 s 接近. 专家可查阅文档、搭建环境、执行复现及回溯修正错误, 未完成复现则判定失败. 为保证公平性, 不允许直接网络搜索其他现成 PoC, 实验仅提供漏洞报告本身.

图 9 展示了实验结果, 86 份报告中 57 份被成功复现, 成功率 66.3% (=57/86), 略高于 VulVerifier 的 47.7%. 人类专家额外复现 16 份报告 (18.6%), 表明 VulVerifier 已接近专家水平, 但在复杂场景仍存在差距, 主要体现在异常处理和错误纠正能力上.

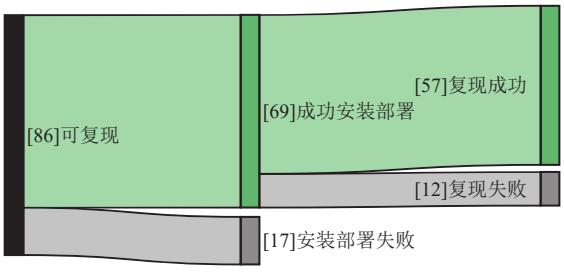


图 9 人类专家复现漏洞报告的实验结果

本文深入分析了“人类专家复现而智能体未能复现”的案例, 实验结果表明, 智能体在报告存在错误时

通常无法识别不一致或缺陷, 倾向判定操作无法触发漏洞. 相比之下, 人类专家可凭经验直觉发现异常并修正错误完成复现. 例如 CVE-2025-2011 提供一键复现脚本, 智能体执行后判定失败, 但无法定位原因; 人类专家发现脚本中 docker-compose 与当前 Docker 不兼容, 修改为 docker compose 后成功复现漏洞.

## 4 结论与展望

漏洞验证是漏洞报告处理中的关键环节. 然而, 当前漏洞验证高度依赖人工操作, 效率低下, 成为制约整体审核流程的瓶颈. 为提升效率, 本文设计并实现了基于漏洞报告的自动化复现智能体 VulVerifier. 针对现有自动化复现的挑战, 本文提出 3 项核心技术. 首先, 漏洞信息补全技术通过整合多源信息, 对漏洞相关内容进行统一补充与分析. 其次, 漏洞前置状态构建技术以应用状态为核心, 对目标应用进行有目的探索与操作, 使其逐步满足漏洞复现前置条件. 最后, 纠错执行复现技术使智能体在操作过程中能够感知、分析并修正行为, 以应对复现中出现的复杂情况.

尽管 VulVerifier 在适用性和可靠性上取得了一定成果, 其技术实现仍有改进空间. 首先, VulVerifier 在处理复杂图形界面操作时仍存在稳定性问题. 当前智能体能在结构相对简单的 Web 界面完成操作, 但当界面元素复杂、布局混乱或缺乏语义标注时, 操作容易失败. 多次重试虽可能获得成功, 但不能根本解决不稳定性问题. 未来研究可探索更鲁棒的界面感知与交互机制, 以提升智能体在复杂图形界面下的可靠性. 其次, 大语言模型固有缺陷仍是复现过程的重要瓶颈. 模型存在幻觉与遗忘问题, 可能忽略提示词中的关键指令. 未来可考虑更强大模型或更健壮的智能体架构, 以降低此类问题对复现结果的影响. 最后, 数据集中 Web 应用漏洞占比高, 二进制程序等类型数量较少. 因此, 智能体设计更多聚焦 Web 漏洞复现, 对其他类型应用复现中涉及的环境配置、调试与交互关注不足. 后续研究可扩展智能体支持更多类型漏洞, 以提升通用性.

## 参考文献


1 陈伟雄, 杨晓晨, 春增军, 等. 电力企业网络安全威胁情报管理体系的研究与实践. 电信科学, 2022, 38(7): 184–189. [doi: 10.11959/j.issn.1000−0801.2022133]

2 王红伟, 陈勋, 殷颂棋, 等. 铁路网络安全漏洞管理平台建设与关键技术研究. 铁路计算机应用, 2023, 32(10): 59–62. [doi: 10.3969/j.issn.1005-8451.2023.10.12]

3 Su HJ, Pan JY. Crowdsourcing platform for collaboration management in vulnerability verification. Proceedings of the 18th Asia-Pacific Network Operations and Management Symposium. Kanazawa: IEEE, 2016. 1–4. [doi: 10.1109/APNOMS.2016.7737235]

4 Schloegel M, Klischies D, Koch S, *et al*. Confusing Value with Enumeration: Studying the use of CVEs in academia. Proceedings of the 34th USENIX Security Symposium. Seattle: USENIX, 2025. 2887–2906.

5 Huynh T, Miller J. An empirical investigation into open source Web applications' implementation vulnerabilities. Empirical Software Engineering, 2010, 15(5): 556–576. [doi: 10.1007/s10664-010-9131-y]

6 Wang WW, Li ZD, You F, *et al*. Vulnerability report analysis and vulnerability reproduction for Web applications. Proceedings of the 9th International Symposium on Dependable Software Engineering: Theories, Tools, and Applications. Nanjing: Springer, 2023. 279–297. [doi: 10.1007/978-981-99-8664-4_16]

7 Simsek D, Eghbali A, Pradel M. PoCGen: Generating proof-of-concept exploits for vulnerabilities in npm packages. arXiv:2506.04962, 2025.

8 Kang S, Yoon J, Yoo S. Large language models are few-shot testers: Exploring LLM-based general bug reproduction. Proceedings of the 45th IEEE/ACM International Conference on Software Engineering. Melbourne: IEEE, 2023. 2312–2323. [doi: 10.1109/ICSE48619.2023.00194]

9 Ruan BN, Liu JH, Zhang CQ, *et al*. KernJC: Automated vulnerable environment generation for Linux kernel vulnerabilities. Proceedings of the 27th International Symposium on Research in Attacks, Intrusions and Defenses. Padua: ACM, 2024. 384–402. [doi: 10.1145/3678890.3678891]

10 Chen LG, Guo J, He ZL, *et al*. RoBin: Facilitating the reproduction of configuration-related vulnerability. Proceedings of the 20th IEEE International Conference on Trust, Security and Privacy in Computing and Communications. Shenyang: IEEE, 2021. 91–98. [doi: 10.1109/TrustCom53373.2021.00030]

11 Fioraldi A, Maier D, Eißfeldt H, *et al*. AFL++: Combining incremental steps of fuzzing research. Proceedings of the 14th USENIX Conference on Offensive Technologies. USENIX Association, 2020. 10.

12 Böhme M, Pham VT, Nguyen MD, *et al*. Directed greybox

fuzzing. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. Dallas: ACM, 2017. 2329–2344. [doi: 10.1145/3133956.3134020]

13 Huang HQ, Guo YY, Shi QK, *et al*. BEACON: Directed grey-box fuzzing with provable path pruning. Proceedings of the 2022 IEEE Symposium on Security and Privacy. San Francisco: IEEE, 2022. 36–50. [doi: 10.1109/SP46214.2022.9833751]

14 Stephens N, Grosen J, Salls C, *et al*. Driller: Augmenting fuzzing through selective symbolic execution. Proceedings of the 24th Network and Distributed System Security Symposium. San Diego, 2016. 1–16. [doi: 10.14722/ndss.2016.23368]

15 Yun I, Lee S, Xu M, *et al*. QSYM: A practical concolic execution engine tailored for hybrid fuzzing. Proceedings of the 27th USENIX Security Symposium. Baltimore: USENIX Association, 2018. 745–761.

16 Trickel E, Pagani F, Zhu C, *et al*. Toss a fault to your witcher: Applying grey-box coverage-guided mutational fuzzing to detect SQL and command injection vulnerabilities. Proceedings of the 2023 IEEE Symposium on Security and Privacy. San Francisco: IEEE, 2023. 2658–2675. [doi: 10.1109/SP46215.2023.10179317]

17 Fang R, Bindu R, Gupta A, *et al*. LLM agents can autonomously exploit one-day vulnerabilities. arXiv:2404.08144, 2024.

18 Ullah S, Balasubramanian P, Guo WB, *et al*. From CVE entries to verifiable exploits: An automated multi-agent framework for reproducing CVEs. arXiv:2509.01835, 2025.

19 Liu B, Zhao YJ, Xu GA, *et al*. LLM agents for automated Web vulnerability reproduction: Are we there yet?. arXiv:2510.14700, 2025.

20 Dong Y, Guo WB, Chen YQ, *et al*. Towards the detection of inconsistencies in public security vulnerability reports. Proceedings of the 28th USENIX Security Symposium. Santa Clara: USENIX Association, 2019. 869–885.

21 Mu DL, Cuevas A, Yang LM, *et al*. Understanding the reproducibility of crowd-reported security vulnerabilities. Proceedings of the 27th USENIX Security Symposium. Baltimore: USENIX Association, 2018. 919–936.

22 Bettenburg N, Just S, Schröter A, *et al*. What makes a good bug report?. Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering. Atlanta: ACM, 2008. 308–318. [doi: 10.1145/1453101.1453146]

23 Rahman MM, Khomh F, Castelluccio M. Works for me! Cannot reproduce—A large scale empirical study of non-reproducible bugs. Empirical Software Engineering, 2022, 27(5): 111. [doi: 10.1007/s10664-022-10153-2]

24 Johnson J, Mahmud J, Wendland T, *et al*. An empirical investigation into the reproduction of bug reports for Android apps. Proceedings of the 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering. Honolulu: IEEE, 2022. 321–322. [doi: 10.1109/SANER53432.2022.00048]

25 Nitin V, Ray B, Moghaddam RZ. FaultLine: Automated proof-of-vulnerability generation using LLM agents. arXiv:2507.15241, 2025.

26 李子东. 基于漏洞报告的 Web 应用漏洞自动复现方法研究 [硕士学位论文]. 北京: 北京化工大学, 2023.

27 Yao SY, Zhao J, Yu D, *et al*. ReAct: Synergizing reasoning and acting in language models. Proceedings of the 11th International Conference on Learning Representations. OpenReview.net, 2023.

28 Achiam J, Adler S, Agarwal S, *et al*. GPT-4 technical report. arXiv:2303.08774, 2023.

(校对责编: 李慧鑫)