计 算 机 研 究 与 发 展
Journal of Computer Research and Development
https://crad.ict.ac.cn
63(10): 2611−2631, 2026

ValidPoC: 基于大语言模型的 PoC-Nuclei 模板生成与漏洞复现验证方法

伍一鸣[1] 池家梅[2] 陈城浩[2] 温震宇[1] 洪 榛[1] 姚信威[3] 纪守领[4]

[1](浙江工业大学网络空间安全研究院 杭州 310014)

[2](浙江工业大学信息工程学院 杭州 310014)

[3](浙江工业大学先进技术研究院(前沿交叉科学研究院) 杭州 310014)

[4](浙江大学计算机科学与技术学院 杭州 310027)

ValidPoC: LLM-Based Generation and Verification Method of PoC-Nuclei Templates for Vulnerability Reproduction

Wu Yiming[1], Chi Jiamei[2], Chen Chenghao[2], Wen Zhenyu[1], Hong Zhen[1], Yao Xinwei[3], and Ji Shouling[4]

[1](*Institute of Cyberspace Security*, *Zhejiang University of Technology*, *Hangzhou* 310014)

[2](*College of Information Engineering*, *Zhejiang University of Technology*, *Hangzhou* 310014)

[3](*Academy for Advanced Interdisciplinary Science and Technology* (*Zhejiang University of Technology*), *Hangzhou* 310014)

[4](*College of Computer Science and Technology*, *Zhejiang University*, *Hangzhou* 310027)

Abstract External vulnerability knowledge is increasingly important for large language models (LLMs) to understand vulnerability targets, craft exploit payloads, and interpret verification evidence. Recent approaches incorporate external vulnerability knowledge via prompt, task planning and multi-agent collaboration, to improve automated penetration testing. However, automated vulnerability reproduction remains challenging because public exploit materials are often heterogeneous and loosely structured, spanning scripts, HTTP requests, command-line snippets, and walkthrough in natural language. Prior work has leveraged retrieval-augmented generation, PoC semantic repair, and exploit code generation to better utilize external vulnerability knowledge, but still constrained by environment dependencies, exploit semantic extraction and model hallucinations, weakening the trustworthiness of automated reproduction.To address these issues, we propose ValidPoC, an LLM-driven framework for automated vulnerability reproduction that can convert heterogeneous exploit walkthrough or scripts into structured and reliable PoC-Nuclei templates. Specifically, ValidPoC defines a PoC semantic model that provides a unified representation of exploit logic, including request construction, payload generation, execution dependencies, and verification criteria. We develop a Semgrep-assisted semantic parsing mechanism to extract PoC Semantic IR from public exploit materials, including exploit scripts and natural-language descriptions, and construct structured Nuclei YAML templates for automated vulnerability reproduction. To improve reproduction reliability, we establish a closed-loop execution–evaluation–repair framework and a three-level validation model (L1–L3) for cross-validation across multiple execution evidence.We evaluate ValidPoC on 67 real-world vulnerability reproduction tasks. ValidPoC achieves a Pass@3 reproduction rate of 76.12% and consistently surpasses existing automated penetration testing

收稿日期: 2026-06-06; **修回日期**: 2026-07-10

基金项目: 国家自然科学基金项目(62533019; 62302454); 浙江省杰出青年科学基金项目(LR24F020004); 浙江省“尖兵领雁+X”研发攻关计划(2024C03274)

This work was supported by the National Natural Science Foundation of China (62533019; 62302454), the Zhejiang Provincial Science Foundation for Distinguished Young Scholars (LR24F020004), and the “Pioneer” and “Leading Goose” Research and Development Program of Zhejiang Province (2024C03274).

通信作者: 温震宇(zhenyuwen@zjut.edu.cn)

frameworks, including PentestAgent and PentestGPT. In addition, it maintains competitive efficiency with respect to both execution time and token usage. These results demonstrate the effectiveness of ValidPoC in understanding heterogeneous exploit materials, generating PoC templates, and vulnerability reproduction.

Key words automated vulnerability reproduction; large language models; PoC semantic parsing; Nuclei template generation; multi-agent collaboration

摘 要 外部漏洞知识是支撑大语言模型开展自动化渗透测试与漏洞复现的重要信息来源。然而,公开漏洞利用材料通常来源复杂、表示形式异构,广泛存在于脚本代码、HTTP 请求片段、命令行示例以及自然语言说明文档中,缺乏统一的描述规范与明确的语义边界。现有方法虽然能够通过漏洞知识检索、利用代码生成等方式辅助漏洞利用,但仍面临漏洞利用语义提取困难、环境适配能力不足以及模型幻觉等问题,导致自动化漏洞复现成功率受限。针对上述问题,提出一种基于大语言模型的自动化漏洞复现方法 ValidPoC。该方法以公开漏洞利用材料为输入,通过构建 PoC 语义解析、PoC 中间表示建模以及 PoC-Nuclei 模板生成流程,实现异构漏洞利用材料向标准化漏洞利用模板的自动转换。其中,定义了面向漏洞复现的 PoC 语义模型,统一描述请求构造、载荷生成、执行依赖与验证判据等关键利用逻辑;设计了基于 Semgrep 辅助的 PoC 语义解析机制,从代码脚本与自然语言材料中抽取漏洞触发相关语义;构建了基于 Nuclei 模板的漏洞复现载体,实现漏洞利用逻辑的结构化表达与自动执行。为提高漏洞复现结果的可靠性,进一步提出基于“执行—评估—修复”闭环反馈优化机制,并建立 L1(可解析执行)、L2(条件匹配成功)和 L3(漏洞真实复现)三级验证模型,对漏洞触发结果进行多源证据交叉验证。在 67 个真实漏洞复现任务上的实验结果表明,ValidPoC 能够有效提升自动化漏洞复现能力。与现有自动化渗透测试方法相比,ValidPoC 取得 76.12% 的 Pass@3 漏洞复现成功率,较 PentestAgent 提升 4.48 个百分点,较 PentestGPT 提升 43.28 个百分点;同时,在 Token 消耗和运行时间方面均表现出较好的效率优势。实验结果验证了所提方法在异构漏洞利用材料理解、PoC 模板生成以及漏洞复现验证方面的有效性。

关键词 自动化漏洞复现;大语言模型;PoC 语义解析;Nuclei 模板生成;多代理协作

中图法分类号 TP311.5; TP393.08

DOI: 10.7544/issn1000-1239.202660432 **CSTR:** 32373.14.issn1000-1239.202660432

外部漏洞知识是描述漏洞成因、影响范围、触发条件、利用路径以及验证方式的信息集合,能够为大语言模型(large language models, LLMs)定位漏洞目标、选择利用策略、构造攻击载荷以及判断触发结果提供重要依据。Wu 等人[1]证明,大模型能够依据通用漏洞披露(common vulnerabilities and exposure, CVE)描述定位漏洞函数。Ullah 等人[2]进一步结合 CVE 描述、通用弱点枚举(common weakness enumeration, CWE)、补丁提交记录、安全公告以及概念信息(proof-of-concept, PoC),实现了端到端的漏洞复现流程。然而,Fang 等人[3]在评估大模型执行渗透测试的能力时发现,当提供 CVE 描述时,GPT-4 的漏洞利用成功率可达到 87%;移除描述后,成功率则降至 7%。这一结果表明,在自动化渗透测试任务中,外部漏洞知识已成为影响大模型能否有效执行自动化渗透测试任务的关键因素之一。

为提升漏洞复现能力,现有研究通常从外部语料库中检索与漏洞利用相关的文档。此类知识主要包含 2 类互补信息:1)CVE 条目[4],提供漏洞类型、受影响组件与危害等级等摘要性信息,用于帮助模型确定攻击目标与利用方向,即回答“攻击什么”的问题;2)公开漏洞利用材料,例如 GitHub[5]、Exploit-DB[6]等平台上发布的漏洞脚本、HTTP 请求片段和利用说明文档等,其包含攻击载荷、漏洞触发流程与验证逻辑等细节信息,用于指导模型完成漏洞利用过程,即回答“如何攻击”的问题。

然而,外部漏洞知识并不意味着模型具备可靠的漏洞利用能力。已有研究表明,即使提供相关漏洞知识,大模型仍可能误解漏洞知识的适用边界,错误使用漏洞利用材料中的关键细节,或错误聚焦于无关的上下文,从而导致推理失败[7-8]。Steenhoek 等人[9]对大模型在漏洞检测任务中的推理失败进行了系统性分析,发现模型即使获得漏洞定义、代码上下文或静态分析结果等辅助信息,仍容易在漏洞语义理解和逻辑推理的过程中出错。这类问题主要集中在语句定位、语义关联和逻辑推理等环节,且难以通

过提示词工程、数据增强、模型微调或参数扩展等方式得到有效缓解。因此，基于大模型的自动化渗透测试系统所面临的核心挑战，不仅在于能否检索到相关漏洞知识，更在于能否正确理解、组织并应用这些知识。

相比具有统一规范的 CVE 条目，公开漏洞利用材料来源复杂、格式异构，通常以脚本代码、HTTP 请求、命令行片段或自然语言说明等形式存在，缺乏统一的描述规范与明确的语义边界。漏洞利用过程中涉及的攻击载荷、触发条件、利用流程及验证逻辑等关键要素，往往分散于代码、注释、配置文件及文档说明中，导致大模型需要跨越文件边界与内容模态来对碎片化信息进行语义整合与推理。任一关键要素缺失，都可能导致模型产生幻觉问题或错误推理。此外，漏洞复现过程还受到目标靶机配置、补丁状态、网络配置以及依赖组件等环境因素影响，进一步增加了漏洞复现的复杂性与环境适配成本。

针对上述问题，本文旨在从异构漏洞利用材料中抽取与漏洞触发和验证相关的核心信息，并将其转化为可执行、可验证的标准化漏洞模板。为避免概念混淆，本文将漏洞利用材料中与请求构造、触发条件、执行依赖和验证判据相关的核心信息称为 PoC 语义。PoC 语义并不等同于原始漏洞利用材料本身，而是从脚本代码、HTTP 请求序列、注释和说明文档中抽象出的中间语义表示，包括请求入口、请求方法、触发参数、有效载荷、执行顺序、状态依赖和验证判据等信息。在此基础上，本文采用声明式的 Nuclei YAML 模板[10]作为能够承载 PoC 语义的最小可执行单元（后文简称为 PoC-Nuclei 模板）。Nuclei 模板能够以统一格式描述漏洞利用与验证逻辑，具备良好的可扩展性与工程兼容性。为保证生成模板的有效性，本文认为可靠的 PoC-Nuclei 模板应满足 3 个条件：1）信息完备性，即能够完整描述漏洞触发所需的请求路径、参数约束、状态依赖与验证逻辑；2）可靠性，即能够依据目标靶机的实际反馈准确判断漏洞是否被成功触发；3）可复现性，即在明确适用边界和环境条件下，能够稳定复现同一漏洞利用过程并获得一致结果。

基于上述流程，本文提出了 ValidPoC，一种基于大语言模型的自动化漏洞利用复现框架。ValidPoC 围绕信息收集、PoC 语义解析与模板生成、漏洞触发验证与反馈优化 3 个阶段展开，旨在将异构漏洞利用材料转化为可执行、可验证的 PoC-Nuclei 模板，并在真实靶机环境中检验其漏洞复现效果。

1）信息收集阶段。ValidPoC 首先对目标靶机进行指纹探测，获取目标组件、版本、服务入口等上下文信息；随后以目标漏洞编号或组件版本信息为检索线索，从 ExploitDB、GitHub 等公开平台收集相关 CVE 条目与漏洞利用材料，为后续任务提供先验约束与参考信息。

2）PoC 语义解析与模板生成阶段。针对漏洞利用材料形态异构、语义分散的问题，本文设计了基于 Semgrep 辅助的 PoC 语义解析流程。对于代码类材料，系统基于入口点与敏感参数引导大模型生成入口定位规则、污点传播规则及补充检索模式，并结合静态分析结果还原请求结构、参数依赖与验证逻辑；对于自然语言类材料，系统结合规则抽取与大模型归纳相结合的方式提取 PoC 语义。最后，解析结果被整理为包含漏洞元信息、请求序列、执行顺序、及验证判据的 PoC 中间表示，并进一步映射为信息完备的 PoC-Nuclei 模板。

3）漏洞触发验证与反馈优化阶段。为了验证生成模板的可靠性和有效性，ValidPoC 设计了执行代理、评估代理和模板修复代理 3 个功能模块。其中，执行代理负责在真实靶机环境中运行生成的 PoC-Nuclei 模板，并采集请求响应日志、匹配结果及相关证据；评估代理负责结合漏洞触发逻辑与运行结果判断漏洞是否被真实触发，避免将 HTTP 状态码、通用错误页面或版本指纹等表层信息误判为漏洞利用成功的标志；模板修复代理则根据诊断结果对模板进行定向修复与迭代优化。3 个模块通过有限轮次的动态反馈机制不断优化模板质量，从而提升 PoC 模板的可执行性与漏洞复现结果的可靠性。

本文的主要贡献如下：

1）提出一种面向自动化漏洞复现的 PoC 语义建模与解析方法，实现异构漏洞利用材料中漏洞触发逻辑的结构化提取与统一表示。

2）提出 ValidPoC 自动化漏洞复现框架，通过 PoC-Nuclei 模板生成与“执行—评估—修复”闭环反馈优化机制，实现漏洞利用过程的自动生成、验证与优化。

3）在 67 个真实漏洞复现任务上的实验结果表明，本文方法在漏洞复现成功率和运行效率方面均优于现有自动化渗透测试方法。

1 预备知识

1.1 漏洞利用材料

公开漏洞利用材料是互联网上广泛共享的漏洞

触发知识载体，可以被划分为代码和自然语言说明2类表现形式。代码类材料以Python、Ruby等语言的漏洞利用程序为主，攻击载荷、请求构造与验证逻辑嵌入于变量赋值、函数调用和条件分支等程序语义中，同时掺杂大量与漏洞触发无关的辅助逻辑；自然语言说明类材料以社区复现指南的文本形式存在，使用自然语言描述漏洞触发步骤与攻击效果，但信息分布分散、语义边界模糊，常省略参数来源和编码细节。

对于代码类漏洞利用材料，Semgrep[11]作为基于抽象语法树的静态分析框架，能够通过规则匹配定位HTTP请求调用、参数传递路径以及危险函数调用等与漏洞触发相关的代码区域。然而，静态分析结果只能反映代码结构和参数位置，无法进一步解释其中蕴含的依赖关系、漏洞触发逻辑以及利用意图；同时，对于以自然语言描述为主的漏洞利用材料，传统代码分析方法无法有效处理。因此，从异构漏洞利用材料中构建结构化PoC模板不仅需要代码级定位能力，还需要结合上下文语义理解漏洞利用逻辑解析。为实现上述目标，本文基于Semgrep定位代码类材料中的关键漏洞利用语义，通过大语言模型强大的语义理解能力，将2类异构的漏洞利用材料转化为统一的PoC语义信息。

1.2 Nuclei扫描器与Nuclei模板

Nuclei是一款开源的漏洞扫描器，其核心机制是基于YAML结构的Nuclei模板向目标靶机发送请求与响应匹配。如图1所示，Nuclei模板由元信息(info)、请求序列和执行控制3部分组成。元信息负责描述模板名称、作者、严重等级等元数据；请求序列支持HTTP、DNS、TCP等10种协议类型，每个请求序列除了构造包含有效负载(payload)的协议请求之外，还支持设置匹配器(matchers)和提取器(extractors)用于判定漏洞触发结果和提取响应中的目标关键值；执行控制部分负责管理请求间的执行顺序、条件分支、载荷组合策略及并发行为。

Nuclei扫描器将漏洞检测规则与执行环境解耦，通过模板描述请求构造、交互流程和匹配条件，而具体的请求发送、并发调度、响应解析与结果输出则由引擎统一完成，从而降低检测规则对特定运行环境的依赖，提升规则编写与复用的标准化程度。此外，Nuclei通过发送特定构造请求并分析响应结果来实现漏洞主动验证，而非依赖版本指纹进行间接判断。Huang等人[12]的研究表明，相比横幅扫描方法，Nulcei具有更低的误报率和更高的检测覆盖率。现有Nuclei模板主要依赖人工编写，难以高效利用异构漏洞利用材料中的请求入口、参数依赖、载荷构造和验证判据等关键信息，导致模板生成过程依赖人工经验且易产生遗漏。同时，现有社区模板普遍采用状态码、版本信息或错误页面等弱验证特征，难以充分证明漏洞是否被真实触发。针对上述问题，本文以Nuclei模板作为漏洞复现载体，进一步设计基于大语言模型的PoC语义解析、PoC-Nuclei模板生成与漏洞利用分层验证机制，实现漏洞利用材料的自动化结构化转换，并提升漏洞复现结果的可靠性。



Fig. 1 Illustration of Nuclei template

图1 Nuclei模板示意图

2 基于多代理协作的PoC生成与验证框架

如图2所示，ValidPoC的工作流程分为3个阶段。1)信息收集。执行代理对靶机进行指纹探测，提取受影响组件及版本与服务入口，检索关联的CVE条

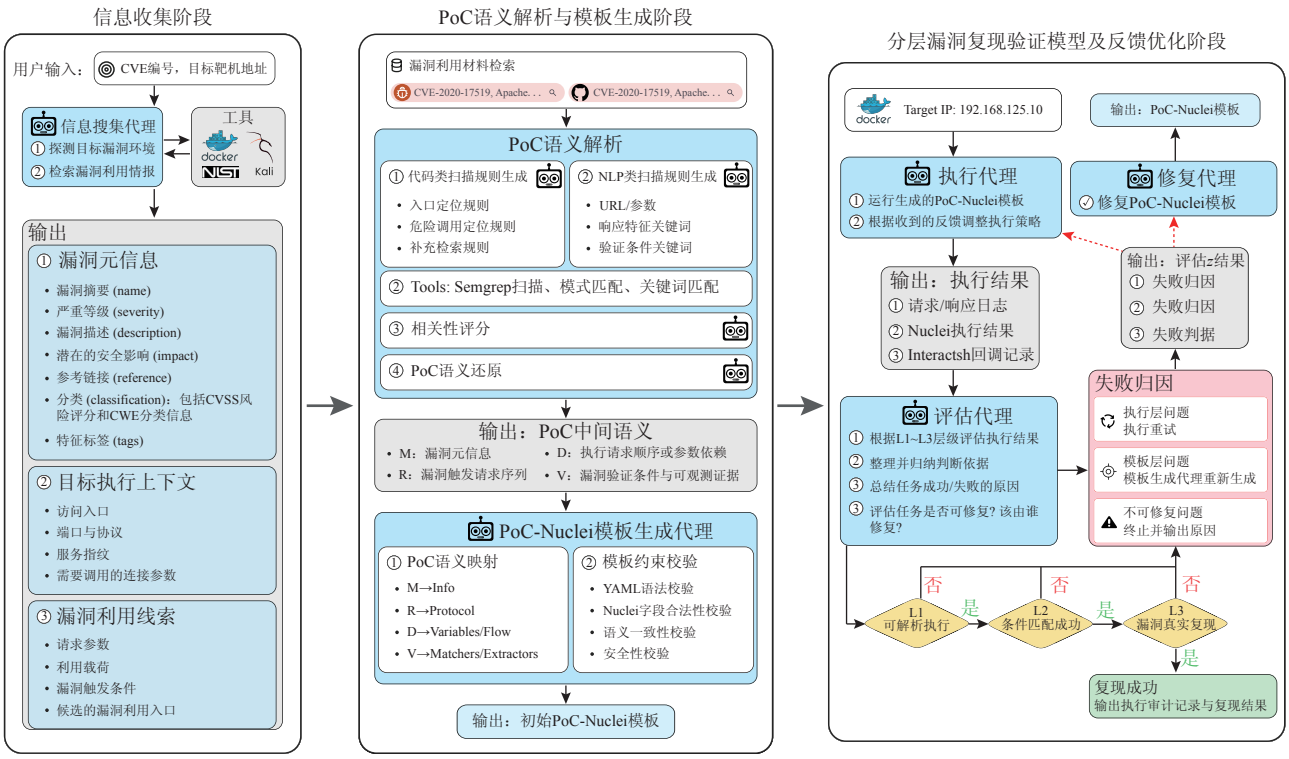


Fig. 2 Flow chart of ValidPoC automated vulnerability reproduction

图 2 ValidPoC 自动化漏洞复现流程图

目和公开漏洞利用材料。2)PoC 语义解析与模板生成。根据目标 CVE 编号或组件版本信息，检索公开漏洞利用材料，抽取支撑漏洞复现的结构化利用逻辑，生成可执行的 PoC-Nuclei 模板。3)漏洞触发验证与反馈优化。执行代理调用 Nuclei 扫描器对靶机发起利用请求，由评估代理根据环境反馈判定漏洞是否被成功触发。若未成功，评估代理分析失败原因，交由修复代理与执行代理调整策略后重新尝试，直至漏洞利用成功、目标不可达或达到最大尝试次数为止。接下来将详细介绍各个阶段的功能设计和具体实施过程。

2.1 信息收集

信息收集阶段负责对目标 CVE 漏洞和靶机环境完成系统化的侦察和预处理，为下游代理提供统一的结构化输入。具体而言，本节设计了一个标准化的漏洞信息收集提示词模板。该模板以用户提供的 CVE 编号和目标靶机地址为输入，按照“检索—筛选—归纳—结构化输出”的流程，引导大模型自动调用漏洞情报检索工具与网络探测工具，对 CVE 条目进行分析，并对目标靶机上下文进行探测与整理，最终生成 3 类结构化结果，包括：

1)漏洞元信息。用于填充 PoC-Nuclei 模板的元信息字段，并为漏洞利用逻辑提取提供攻击面先验知识。信息收集阶段能够调用漏洞情报检索工具，从美国国家漏洞数据库[13](national vulnerability database, NVD)获取漏洞描述、严重等级、CVSS 评分、CWE 类型和参考链接等标准化字段。对于 NVD 未直接提供的安全影响(impact)和特征标签(tags)2 类信息，大模型需要基于 NVD 的事实性描述，结合参考链接材料推理潜在安全影响，刻画漏洞触发后可能造成的安全后果，为第 3.3 节的漏洞利用验证提供对照基准；大模型还需要从事实性描述提取特征标签，包括服务名称、漏洞类型和受影响组件等关键词，为模板语义检索与分类提供保障。

2)目标执行上下文。由网络探测工具生成，用于为后续的执行与验证阶段提供必要的靶机交互信息。代理根据目标地址探测靶机的可达性、开放端口、协议类型、服务标识、基础 URL 及可能的访问入口，并将其整理为执行代理可直接调用的结构化目标上下文，从而避免执行代理在扫描阶段重复探测或使用错误的连接参数。

3)漏洞利用线索。从漏洞元信息中提取候选入口点、可疑参数和恶意负载等参考信息，为第 2.2 节的利用逻辑提取与模板生成阶段提供关键指导。每条线索均保留其来源字段，确保下游代理在采纳或质疑线索时可追溯到原始证据。

2.2 PoC 语义解析与模板生成

为了从异构的漏洞利用材料中提取请求构造、有效载荷、数据依赖、控制依赖、执行时序以及漏洞验证等 PoC 语义，并将其映射到标准化的 Nuclei 模板中，本节设计了面向 PoC-Nuclei 模板自动生成的多阶段大模型工作流，包括 PoC 语义解析、PoC 中间表示建模和 PoC-Nuclei 模板生成 3 个阶段，将信息收集阶段产出的结构化结果转化为对应的 PoC-Nuclei 模板，供第 3.3 节执行。为便于理解各子模块之间的数据依赖关系，此处先给出面向 PoC-Nuclei 模板自动生成的多阶段大模型工作流的公式定义。

本文将第 2.1 节的信息收集结果表示为：

$$I=(I_{\text{cve}},I_{\text{target}},I_{\text{hints}}), \tag{1}$$

其中，I_{cve}为漏洞元信息，I_{target}为目标执行上下文，I_{hint}为漏洞利用线索。设根据指定 CVE 编号检索到的漏洞利用材料为M。则 ValidPoC 的模板生成步骤为：

$$P=f_{\text{extract}}(M,I), \tag{2}$$

$$N_0=f_{\text{gen}}(P), \tag{3}$$

其中，f_{extract}表示 PoC 语义解析函数（第 3.2.1 节），用于从漏洞利用材料中提取漏洞触发相关语义；P 表示 PoC 中间语义（PoC semantic intermediate representation, PoC Semantic IR）；f_{gen}表示模板生成模块，用于将 PoC 中间语义映射为 PoC-Nuclei 模板N_0。具体流程如图 2 所示。

2.2.1 PoC 中间语义定义

公开漏洞利用材料通常以漏洞利用脚本、HTTP 请求报文、命令行示例以及自然语言说明等形式存在。虽然这些材料均描述同一漏洞的利用过程，但其表达方式、信息粒度以及完整程度存在显著差异。部分材料侧重漏洞利用实现细节，而部分材料仅提供复现步骤或关键请求片段，导致漏洞利用逻辑分散在代码、注释、配置文件以及说明文档之中。若直接由大模型从原始材料生成 PoC 模板，容易出现关键依赖遗漏、利用逻辑缺失以及验证条件失真等问题。为统一表示异构漏洞利用材料中的漏洞利用知识，本文提出 PoC 中间语义作为漏洞利用知识的统一中间表示。PoC 中间语义位于原始漏洞利用材料与 PoC-Nuclei 模板之间，负责抽象漏洞利用过程中的核心语义信息。PoC 中间语义由 4 部分组成：

$$P=(M,R,D,V), \tag{4}$$

其中 M 表示漏洞元信息（metadata），R 表示漏洞触发请求序列（request sequence），D 表示执行请求顺序或参数依赖（dependency），V 表示漏洞触发后的验证（verification）条件与观测证据。其中，M 用于描述漏洞编号、漏洞类型、受影响组件及漏洞标签等基础信息；R 用于描述漏洞利用过程中涉及的请求方法、请求路径、请求头、请求体以及攻击载荷等内容；D 用于描述多阶段利用过程中的变量传递、认证状态以及执行顺序等关联关系；V 用于描述漏洞成功触发后应满足的观测结果，包括响应匹配条件、关键字段提取规则以及漏洞触发证据等信息。与原始漏洞利用材料相比，PoC 中间语义具有形式无关性和执行载体无关性 2 个特点。前者保证不同形式的漏洞利用材料能够映射为统一语义表示；后者保证同一 PoC 中间语义既可以生成 PoC-Nuclei 模板，也可以进一步转换为其他漏洞利用载体。

2.2.2 PoC 语义解析

在 PoC 语义解析之前，ValidPoC 首先根据漏洞元信息中的 CVE 编号和漏洞标签，从 ExploitDB、GitHub 等公开来源检索相关漏洞利用材料。检索得到的漏洞利用材料主要包括代码脚本和自然语言说明 2 类。如第 1.1 节所述，检索得到的漏洞利用材料可分为脚本代码和自然语言说明 2 类，其漏洞利用语义分别隐含于程序结构与叙述文本之中。当大语言模型直接对漏洞利用原始材料进行端到端的语义解析时，容易出现关键漏洞利用逻辑缺失与语义理解偏差。

为了提升大语言模型对 PoC 语义的理解能力，本文使用 Semgrep 作为辅助的静态分析工具。Semgrep 能够从复杂漏洞利用代码中精准定位请求构造、参数传递、验证逻辑以及危险操作调用等与漏洞触发相关的关键代码语义片段，为大语言模型进一步分析漏洞利用逻辑并生成结构化 PoC 模板提供可靠的代码级上下文依据。针对代码类漏洞利用材料，ValidPoC 的 PoC 语义解析流程包括以下 3 部分。

1）规则生成。基于漏洞利用线索中提取的入口地址与敏感参数信息，系统自动生成 3 类 PoC 语义定位规则：入口定位规则用于识别潜在的路由定义、接口处理函数或请求入口位置；危险调用定位规则用于标记漏洞触发过程中涉及的关键危险操作及其调用位置；补充检索模式则以关键词匹配方式辅助发现代码注释、配置文件及其他非结构化文本中的漏洞相关线索。上述规则共同构成面向 PoC 语义定位的辅助扫描规则集。此外，系统通过 Semgrep 规则加载机制，对生成的规则文件进行 YAML 解析、字段校验以及模式有效性验证。对于验证失败的规则，系统将错误信息反馈给大语言模型进行修正，最多

支持 3 轮迭代重试。

2)代码扫描。Semgrep 根据通过验证的辅助扫描规则集对漏洞利用材料中的源代码文件进行静态扫描，输出规则命中的候选代码片段及其位置信息。对于每个候选片段，系统构建包含代码内容、文件位置、所在函数或方法签名及邻近上下文的候选证据，并结合漏洞利用线索对扫描结果进行相关性评分。最终，系统依据相关性评分筛选候选片段，优先选取高置信结果作为上下文还原 PoC 语义；若所有候选均低于有效阈值，则基于源代码进行 PoC 语义分析，避免遗漏关键 PoC 语义。

3)语义还原。经过规则验证与扫描筛选后，系统将有效规则命中的代码片段、匹配位置及关联上下文输入大语言模型，并在漏洞元信息和漏洞利用线索的共同约束下恢复请求路径、请求方法、攻击载荷、参数依赖关系以及漏洞触发条件等 PoC 语义。

对于自然语言类漏洞利用材料，系统采用规则抽取与模型归纳相结合的方式，从文本中过滤并提取 URL、请求参数、响应特征以及验证条件等显式信息，并进一步归纳为统一的 PoC 语义表示。对于材料中缺失的信息，系统保留缺失标记而非直接补全，以降低模型幻觉风险。最终，所有提取结果统一映射为 PoC 中间语义，作为后续模板生成阶段的输入。

2.2.3 PoC-Nuclei 模板生成

在获得 PoC 中间语义之后，系统进一步将其映射为标准化 PoC-Nuclei 模板。具体而言，本文通过系统提示词向大模型提供 Nuclei 模板语法规范、字段约束、请求映射规则、变量引用方式以及匹配器构造策略，引导模型完成 PoC 中间语义到 PoC-Nuclei 模板的转换。其中，漏洞元信息映射为模板的元信息模块；请求序列映射为对应协议请求结构；依赖关系映射为变量定义与流程控制逻辑；验证条件映射为匹配器与提取器模块。

为保证生成模板的正确性，系统在模板生成后执行以下 4 类约束校验：

1)YAML 语法校验，用于检查模板格式是否合法；

2)Nuclei 字段合法性校验，用于检查字段类型及取值是否满足 Nuclei 规范；

3)语义一致性校验，用于验证变量引用、请求顺序以及验证条件是否与 PoC 中间语义保持一致；

4)安全性校验，用于识别高风险利用操作并将其替换为低风险验证载荷，避免模板执行产生严重的破坏性影响。

通过上述过程，最终生成满足执行要求的初始 PoC-Nuclei 模板 N_0，并交由下一阶段进行漏洞触发验证与反馈优化。

2.3 漏洞触发验证与反馈优化

2.3.1 分层漏洞复现验证模型

现有自动化漏洞利用方法通常将漏洞扫描器的命中结果直接视为漏洞复现成功。然而，在实际场景中，请求能够被成功执行并不意味着漏洞被真实触发；同样，匹配器命中也不一定能够证明漏洞利用行为已经产生预期安全影响。例如，部分检测规则仅依据 HTTP 状态码、服务版本信息或通用错误页面进行匹配，即使满足命中条件，也无法证明目标漏洞已经被成功利用。因此，仅依赖单一验证结果容易产生误报，并高估自动化漏洞复现系统的实际能力。

为此，本文提出分层漏洞复现验证模型，将漏洞复现过程划分为可解析执行(L1)、条件匹配成功(L2)和漏洞真实复现(L3)三个层次，并通过“执行—评估—修复”闭环反馈优化机制不断提升 PoC 模板质量。

1)L1 表示 PoC-Nuclei 模板满足 YAML 语法及 Nuclei 模板规范，能够被 Nuclei 引擎成功加载、解析和执行。达到 L1 说明漏洞利用逻辑已经成功转化为可执行形式，但尚未证明模板与目标漏洞之间存在有效关联。

2)L2 表示 PoC-Nuclei 模板在真实靶机环境中执行后，模板中的匹配器或提取器成功命中目标响应，并产生可供后续分析的观测证据。达到 L2 说明漏洞利用请求与目标系统之间形成了有效交互，但命中结果仍可能来自 HTTP 状态码、服务指纹、通用错误页面或环境异常等非漏洞触发因素，因此不能直接认定漏洞已被成功利用。

3)L3 表示系统观测到与漏洞安全影响一致的直接证据，并能够证明目标漏洞被真实触发。例如，对于远程代码执行漏洞，应证明攻击者能够控制目标执行环境执行指定命令并获得目标执行结果。只有当观测结果与漏洞元信息中声明的安全影响(impact)一致时，系统才认为漏洞复现成功。

根据上述定义，3 层验证之间满足 $L3 \subseteq L2 \subseteq L1$ 的递进关系，即所有达到 L3 的样本必然满足 L2 和 L1，而达到 L1 或 L2 并不一定能够达到 L3。其中，L1 反映模板生成质量，L2 反映漏洞利用请求与目标系统之间的交互有效性，而 L3 则反映漏洞复现结果的真实性。通过引入分层验证机制，系统能够显式区分“模板可运行”“规则命中”和“漏洞真实触发”三种不同状态，从而降低误报风险，并为后续模板修

复提供更精细的反馈依据。

2.3.2 “执行—评估—修复”闭环反馈优化机制

仅依靠1次模板生成往往难以保证漏洞复现成功。由于公开漏洞利用材料存在描述不完整、环境依赖复杂以及目标靶机配置差异等问题,即使生成的PoC-Nuclei模板满足L2要求,也可能因请求构造错误、参数依赖缺失或验证条件不合理而无法达到L3层面的漏洞复现成功。因此,本文进一步设计“执行—评估—修复”闭环反馈优化机制,通过真实环境反馈不断优化PoC模板质量。

设第 i 轮迭代输入模板为 N_i(初始模板 N_0 由第2.2节生成),则闭环反馈优化机制可表示为:

$$E_i = f_{\text{exec}}(N_i)\ , \tag{5}$$

$$D_i = f_{\text{eval}}(E_i, I_{\text{cve}})\ , \tag{6}$$

$$F_i = f_{\text{fix}}^{\text{plan}}(D_i, N_i)\ , \tag{7}$$

$$N_{i+1} = f_{\text{fix}}^{\text{apply}}(N_i, F_i)\ , \tag{8}$$

其中,f_{exec},f_{eval}和f_{fix}(包括$f_{\text{fix}}^{\text{plan}}$和$f_{\text{fix}}^{\text{apply}}$)分别表示执行模块、评估模块和模板修复模块,$E_i$表示第$i$轮的执行证据,$D_i$表示评估代理给出的诊断结论,$F_i$表示修复策略,$N_{i+1}$表示修复后的新模板。

执行模块负责在真实靶机环境中运行PoC-Nuclei模板,并收集漏洞复现过程中产生的执行证据。执行证据不仅包括Nuclei扫描器返回的匹配结果,还包括HTTP请求与响应日志、响应内容、变量提取结果以及Interactsh[14]带外回调记录等信息,从而为后续评估提供完整的证据链。

评估模块负责根据第2.3.1节定义的L1~L3验证标准分析执行证据,并判断当前模板所达到的验证层级。若当前模板达到L3层面的漏洞复现成功,则终止迭代;否则生成结构化诊断结果,指出导致失败的关键原因及其支撑证据。

模板修复模块根据诊断结果生成针对性的修复策略,并对当前模板进行局部修改。修复内容主要包括请求路径修正、参数补全、载荷编码调整、变量引用修正以及匹配器优化等。与重新生成模板不同,模板修复模块仅对诊断结果涉及的字段进行定向修改,从而保留已有正确利用逻辑,减少迭代过程中的信息损失。

通过不断重复“执行—评估—修复”过程,系统能够逐步提升模板质量,使漏洞复现结果从L1逐步向L3逼近。当达到漏洞复现成功、修复预算耗尽或出现不可修复问题时,闭环反馈优化终止,并输出最终结果及完整审计记录。

2.3.3 失败归因与终止策略

为了提高反馈优化效率并避免无效迭代,本文设计了基于失败原因分类的归因机制。评估模块在分析执行证据后,将未达到L3层面漏洞真实复现的情况划分为执行层问题、模板层问题和不可修复问题3类,并根据不同问题类型采取相应的处理策略。

1)执行层问题。执行层问题主要由网络环境、靶机状态或运行条件异常引起,包括连接失败、端口不可达、协议识别错误、请求超时以及目标服务异常等情况。此类问题并非由PoC-Nuclei模板本身构造错误导致,因此系统不会触发模板修复模块,而是返回执行模块调整目标参数或重新执行。执行层问题仅消耗执行重试次数,不消耗模板修复预算。

2)模板层问题。模板层问题主要由PoC-Nuclei模板构造不当导致,包括请求路径错误、请求方法不匹配、参数缺失、变量引用失败、攻击载荷编码错误以及匹配器误判等情况。此类问题表明当前模板未能准确表达PoC中间语义中的漏洞利用逻辑,因此系统调用模板修复模块进行定向优化,并消耗1次模板修复预算。模板修复过程中仅修改诊断结果指向的错误字段,以尽量保留已有正确的请求结构与验证逻辑。

3)不可修复问题。不可修复问题是指在当前输入和执行条件下无法通过反馈优化解决的情况,包括目标靶机不可访问、漏洞利用材料缺失关键字段、漏洞触发效果超出Nuclei模板表达或观测能力、漏洞依赖人工交互或复杂业务状态等。对于此类问题,继续迭代难以带来有效改进,因此系统直接终止任务,并输出失败原因及对应证据。

本文采用有限轮次的反馈优化策略。设模板修复预算为k轮,执行重试次数上限为3次,二者独立计数:执行重试仅用于处理执行层问题,模板修复预算仅在问题被归因为模板层问题时消耗。在每轮迭代中,评估模块依据L1~L3验证模型判断当前模板的复现状态,并给出失败归因。当模板达到L3层面的漏洞真实复现、模板修复预算耗尽、执行重试次数达到上限,或出现不可修复问题时,系统终止迭代,并输出最终漏洞复现结果、失败原因、执行证据和完整审计记录。

3 实验与分析

3.1 研究问题

为了系统评估ValidPoC的有效性及其关键设计

的贡献，本文围绕以下 5 个研究问题开展实验。

1）研究问题 1：与现有自动化渗透测试方法相比，ValidPoC 在漏洞复现成功率和运行效率方面表现如何？

2）研究问题 2：基于不同类型的漏洞利用材料，ValidPoC 在漏洞复现成功率方面表现如何？

3）研究问题 3：PoC 语义解析与模板生成机制是否能够有效提升漏洞复现能力？

4）研究问题 4：分层漏洞复现验证模型及反馈优化机制对最终漏洞复现效果的贡献是什么？

5）研究问题 5：不同基座大语言模型对 ValidPoC 性能的影响如何？

3.2 实验设置

本节将介绍实验数据集、硬件配置、基线方法和评估指标的具体设置。

3.2.1 实验数据集

本文采用 PentestAgent[15] 工作中整理的 67 个开源 Vulhub 漏洞靶场实例作为实验数据集。该数据集涵盖 32 类常见漏洞类型，对应 32 个 CWE 类别，包括 SQL 注入、任意文件读取、远程命令执行、反序列化漏洞、身份认证绕过和信息泄露等典型漏洞场景，能够较为全面地覆盖自动化漏洞复现过程中常见的利用模式。参考 PentestAgent 中的难度划分方式，67 个漏洞实例中包含 50 个低难度漏洞、11 个中等难度漏洞以及 6 个高难度漏洞。本文将每个靶场实例视为一个独立的漏洞复现任务，并以其公开漏洞描述、环境配置以及预期安全影响作为漏洞复现结果的真值依据。

为保证实验公平性并避免数据泄露问题，本文不允许系统直接访问对应漏洞实例的 GitHub 仓库内容，也不允许检索靶场自带的漏洞利用脚本。对于每个漏洞复现任务，系统仅以目标 CVE 编号和靶机地址作为输入，并从 ExploitDB 等公开漏洞利用知识库中检索相关漏洞利用材料，用于后续 PoC 语义解析与模板生成。

3.2.2 评估指标

为了评估 ValidPoC 的有效性，本文采用与第 2.3 节提出的分层漏洞复现验证模型相对应的指标体系，并进一步统计系统运行效率。

1）R_{L1}。R_{L1}表示生成的 PoC-Nuclei 模板达到 L1 层面的比例。若模板能够被 Nuclei 引擎成功加载、解析并执行，则认为该样本达到 L1。该指标主要反映 PoC 语义解析与模板生成阶段的质量。

2）R_{L2}。R_{L2}表示生成的 PoC-Nuclei 模板达到 L2 层面的比例。若模板执行后其匹配器或提取器成功命中目标响应，并生成可供后续分析的观测证据，则认为该样本达到 L2。该指标反映漏洞利用请求与目标系统之间的交互有效性。

3）Pass@k。Pass@k 表示在最多 k 轮反馈优化后达到 L3 层面的累计比例。仅当系统观测到与漏洞元信息中声明的安全影响一致的直接证据时，才认为漏洞复现成功。本文分别统计 Pass@1 和 Pass@3。其中，Pass@1 反映 PoC 语义解析与模板生成阶段一次生成即成功复现漏洞的能力；Pass@3 反映经过反馈优化机制迭代修复后的最终漏洞复现能力。

4）错误匹配率（false match rate, FMR）。$FMR = R_{L2} - \text{Pass@1}$。$FMR$表示达到 L2 但未达到 L3 的样本比例，用于评估系统将匹配成功误判为漏洞复现成功的风险。该指标用于验证第 2.3 节提出的分层漏洞复现验证模型的有效性。

5）运行效率指标。本文使用平均运行时间（Avg. Time）和平均 Token 消耗（Avg. Token）来评估系统运行效率。其中，Avg. Time 表示从输入目标 CVE 编号和靶机地址到输出最终验证结果的平均耗时；Avg. Token 表示整个漏洞复现过程中消耗的平均 Token 数量。这 2 项指标分别反映系统的时间成本和调用成本。

3.2.3 基 线

为全面评估 ValidPoC 的有效性，本文选择 2 种自动化渗透测试方法和 1 种社区规则库。

1）PentestGPT[16] 是基于大模型的渗透测试方法中最具代表性的基线方法，包括环境感知与状态解析、渗透测试任务规划、子任务生成与执行 3 个模块。PentestGPT 覆盖漏洞分析、利用尝试和结果判断等关键环节，能够适配漏洞复现任务，用于评估大模型驱动方法在相同输入条件下的漏洞触发能力。为保证对比公平，本文在实验中将 PentestGPT 原有的人机交互环节替换为自动化决策模块，使其能够在不依赖人工干预的情况下完成端到端执行。

2）PentestAgent[15] 是一种基于多代理协作的自动化渗透测试框架，能够通过信息收集、漏洞分析和漏洞利用等阶段完成端到端测试流程。该方法利用检索增强生成机制，从外部漏洞知识中检索与目标相关的利用线索，指导漏洞利用阶段生成对应的漏洞利用策略。本文将其作为外部漏洞知识驱动的基线方法，并在相同的目标靶机地址、CVE 编号和漏洞利用材料检索来源下评估其漏洞复现能力。

3）Nuclei 官方社区模板库[17]。Nuclei 社区模板库

包含大量由安全研究人员维护的漏洞检测模板。本文将其作为人工规则基线,用于评估现有社区规则库在漏洞复现场景下的实际覆盖能力。对于每个测试样本,系统根据 CVE 编号、组件名称以及漏洞类型检索对应模板,并按照第 2.3 节定义的 L1、L2 和 L3 标准进行验证。若社区中不存在对应模板,则视为社区模板库无法覆盖该漏洞。该基线用于在消融实验中评估现有的 Nulcei Yaml 模板生态与生成的 PoC-Nuclei 模型的可靠性和有效性。

为保证实验公平性,本文为所有方法配置相同的执行环境和统一的验证接口,并限制其输入信息一致,仅包括目标靶机地址和 CVE 编号。由于 PentestGPT 和 PentestAgent 并不生成标准化 PoC-Nuclei 模板,因此无法统计 R_{L1} 和 R_{L2} 指标。对于上述 2 类方法,本文仅统计 Pass@k、运行时间和 Token 消耗等指标。

3.2.4 实施细节

所有实验均在受控虚拟化环境中完成。实验环境由攻击机和靶机 2 部分组成。攻击机运行 Kali Linux 2024.1,配置为 4 个 vCPU 和 8 GB 内存;靶机运行 Ubuntu 22.04 LTS,配置为 2 个 vCPU 和 6 GB 内存。2 台虚拟机通过 NAT 网络互联,以保证实验环境的隔离性和可复现性。为降低实验风险并保证执行过程可控,所有漏洞靶场均部署于独立 Docker 容器中,并配置资源限制和网络隔离策略,以避免实验过程对宿主环境产生影响。

为了评估不同基座模型对 ValidPoC 性能的影响,本文选取 Mimo-v2.5-Pro[18], DeepSeek-v4-Pro[19], GPT-5.4[20] 和 Claude Sonnet 4.6[21] 作为基座模型。其中,GPT-5.4 和 Claude Sonnet 4.6 代表当前主流闭源大模型,DeepSeek-v4-Pro 和 Mimo-v2.5-Pro 代表国产推理模型。上述模型能够覆盖当前主流大模型能力谱系。

ValidPoC 在不同阶段使用任务专用提示词引导大语言模型完成结构化信息抽取、PoC 语义还原、模板生成、执行评估和模板修复。实验中各基座模型均使用相同的提示词模板,以降低随机性对结果的影响。对于每个漏洞复现任务,本文独立运行 3 次实验,并记录最佳结果作为最终结果,以反映模型的峰值漏洞复现能力。ValidPoC 所用的提示词详见附录 A, B, C。

3.3 研究问题 1:与现有自动化渗透测试方法相比,ValidPoC 在漏洞复现成功率和运行效率方面表现如何?

为回答研究问题 1,本文在实验数据集上比较了 ValidPoC 与基线方法的漏洞复现效果,实验结果如表 1 所示。最终,ValidPoC 成功复现 51 个 CVE 漏洞,Pass@3 达到 76.12%,相比 Pass@3 表现次优的基线方法 PentestAgent 提升了 4.48 个百分点。这一结果表明,ValidPoC 生成的 PoC-Nuclei 模板能够在分层漏洞复现验证模型及反馈优化机制的支持下有效完成漏洞复现任务,并取得较好的整体效果。

Table 1 Performance Comparison of Our Approach and Baselines on Vulhub Dataset

表 1 本文方法与基线在 Vulhub 数据集上的性能比较

方法	Pass@1/%	Pass@3/%	Avg. Time/s	Avg. Token
PentestGPT	26.86	32.84	1 856	0.276×10^6
PentestAgent	61.19	71.64	1 341	0.188×10^6
ValidPoC(本文)	41.79	76.12	826	0.085×10^6

此外,相较于 PentestGPT, ValidPoC 的 Pass@1 和 Pass@3 指标分别提升了 14.93 个百分点和 43.28 个百分点。其中,Pass@1 的提升说明 PoC 语义解析与模板生成机制能够生成信息完备、初始质量较高的 PoC-Nuclei 模板。经有限次迭代执行与优化后,ValidPoC 自身的漏洞复现成功率(即 Pass@3−Pass@1)提升了 34.33 个百分点。这一结果表明,分层漏洞复现验证模型及反馈优化机制能够根据真实靶机执行结果持续修正模板中的触发条件、请求构造和验证证据,从而提高最终漏洞复现成功率。

值得注意的是,相比 PentestAgent, ValidPoC 将模型推理与执行的时间效率提升了 1.62 倍,平均词元消耗降低了 54.79 个百分点,这说明 ValidPoC 能够以更低的时间和经济成本完成高质量的漏洞复现任务。

3.4 研究问题 2:基于不同类型的漏洞利用材料,ValidPoC 在漏洞复现成功率方面表现如何?

对于每个漏洞复现任务样本,系统基于其对应的 CVE 编号,从 ExploitDB 中检索相关漏洞利用材料,用于生成 PoC-Nuclei 模板。图 3 展示了 67 个 CVE 编号在 ExploitDB 中检索到的漏洞利用材料对应的编程语言的分布情况。本文共检索到 91 份漏洞利用材料,代码类漏洞利用材料共计 70 份(76.92%),其中 Python 和 Ruby 脚本分别有 34 份(37.36%)和 33 份(36.26%);自然语言类漏洞利用材料共有 18 份(19.78%),说明仍有部分漏洞利用信息以复现说明、命令行片段或文本描述的形式存在。

为了回答研究问题 2,本文调查了 ValidPoC 在不同类型的漏洞利用材料上的性能表现差异。考虑到不同材料类型的样本数量不均,本文在图 4 中采用 Wilson 方法计算 95% 置信区间,用于反映小样本条

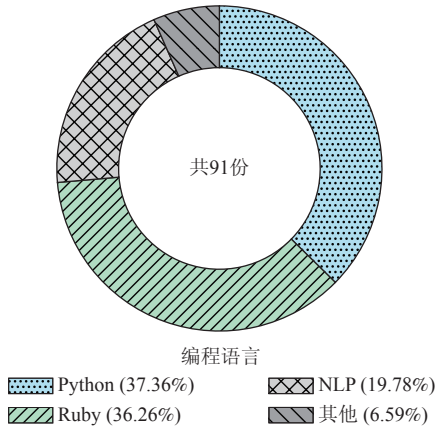


Fig. 3 Distribution of programming languages corresponding to vulnerability utilization materials

图 3 漏洞利用材料对应的编程语言分布

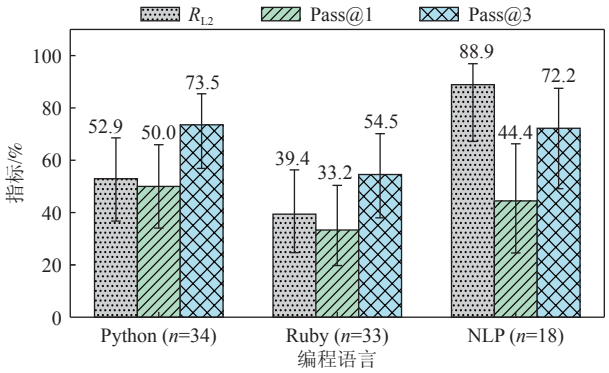


Fig. 4 Performance of ValidPoC under different types of vulnerability utilization materials

图 4 不同漏洞利用材料类型下 ValidPoC 的性能表现

件下结果估计的不确定性。如图 4 所示，不同类型漏洞利用材料呈现出明显的性能差异。Python 类材料在 Pass@1 和 Pass@3 指标上均取得最高结果，说明其代码结构和漏洞触发逻辑更易转换为可执行模板。相比之下，Ruby 类材料整体性能较低，主要原因在于 Ruby 代码依赖 Metasploit[22] 框架实现网络通信、回调监听和会话封装处理，导致 PoC 语义常被封装在框架接口和具体的运行状态中，仅凭局部的代码匹配和分析难以完整还原多阶段利用流程及其验证逻辑。独立的 Nuclei 模板虽然能够发送请求、匹配响应并记录带外交互，但无法像 Metasploit 框架那样动态维护并监听目标靶机的服务状态，超出了 Nuclei 模板的表达与执行能力。以 CVE-2020-7961 为例，该漏洞的利用过程中除了需要向目标发送网络请求以外，还需要攻击端临时启动一个监听服务，用于加载目标回连后的网络信息。这类动态回调行为依赖 Metasploit 运行时状态和服务端协作，ValidPoC 难以仅从静态 Ruby 代码中提取足以支撑 L3 判定的强验证证据，更无法将其等价转化为普通 Nuclei 模板中等价的匹配条件。因此，系统无法判断本次任务结果的可靠性，则记为漏洞利用复现失败。

在自然语言类漏洞利用材料中，ValidPoC 的 R_{L2} 指标达到了 88.9%，Pass@1 指标为 44.4%，这一结果表明，自然语言描述通常能够提供较直观的网络请求、参数和响应特征，使初始模板较容易命中匹配器并满足 L2 标准；但其往往缺乏可直接用于 L3 判定的强验证证据，例如可观测的回调记录、文件变化或后端执行结果。因此，部分模板虽能在首次运行时命中匹配器，却无法证明触发效果与漏洞预期安全影响一致，导致与 Pass@1 之间存在明显差距。这是因为自然语言描述的成功判据常依赖靶机内部观测，例如进入目标靶机容器查看日志、文件变化或后端执行结果。出于隔离性和安全性考虑，ValidPoC 没有直接进入目标靶机容器的权限，只能依据请求/响应日志、Nuclei 匹配结果等外部可观测证据判断任务是否成功。因此首次生成的 PoC-Nuclei 模板更容易生成基于响应状态码、目标匹配参数和指纹等若干证据，满足 L2 层面标准，但无法在进一步的执行和验证评估过程中满足 L3 的基本要求。

此外，在不同类型的漏洞利用材料上，ValidPoC 的 Pass@3 相较 Pass@1 均呈现提升趋势。以自然语言类材料为例，Pass@3 达到 72.2%，说明反馈优化能够在一定程度上弥补初始模板中触发参数、请求构造或验证条件不准确的问题。以 CVE-2022-46169 为例，初始模板沿用了原始 PoC 语义中的默认参数值，但该值与目标靶机的实际漏洞触发参数不匹配，导致模板未能触发预期命令执行效果。评估代理根据执行反馈定位该问题，并将失败原因交由模板修复代理调整触发参数；当枚举得到有效参数值后，模板能够触发预期安全影响。该案例说明，分层漏洞复现验证模型及反馈优化机制能够根据真实执行反馈修正初始模板中的环境特定偏差，并提升生成模板的漏洞复现能力。需要注意的是，由于各类材料样本规模有限且置信区间重叠，上述结果仅用于反映不同材料类型下方法性能的变化趋势，不据此认为不同材料类型之间存在统计显著性差异。

消融实验的结果如表 2 所示。在移除了 PoC 语义解析与模板生成机模块后，系统的 R_{L1} 指标降至 76.12%，R_{L2} 和 Pass@3 指标分别降至 20.90% 和 29.85%。这说明，大模型在直接处理异构漏洞利用材料时，难以稳定理解参数依赖、载荷编码、执行顺序和验证判据等关键 PoC 语义，易出现字段缺失、结构不全或

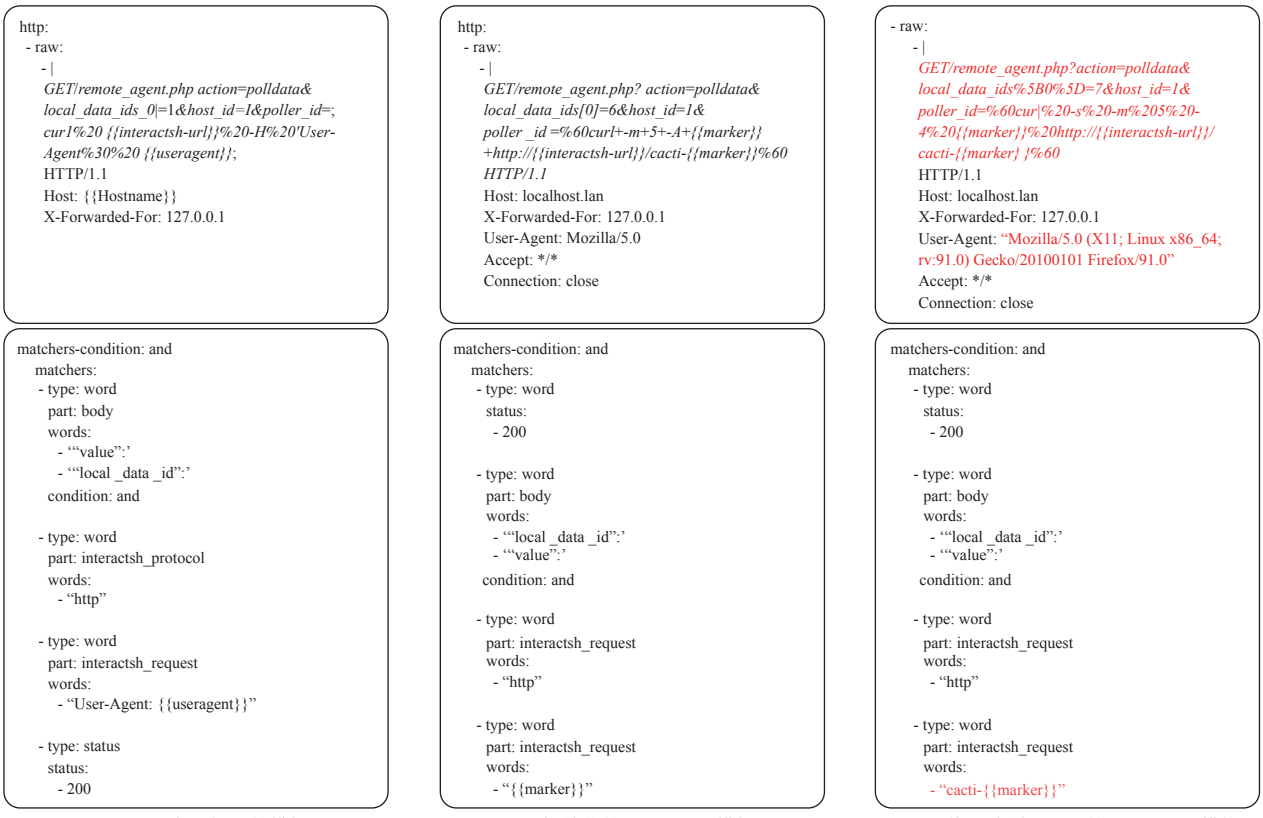
Table 2 Impact of Absence of PoC Semantic Parsing and Template Generation Mechanism on ValidPoC

表 2 缺失PoC语义解析与模板生成机制对ValidPoC 的影响 %

方法	R_{L1}/%	R_{L2}/%	Pass@1	Pass@3	*FMR*
Nuclei	100.00	56.71	37.31	37.31	19.40
w/o PoC parsing	76.12	20.90	13.43	29.85	7.47
ValidPoC（本文）	100.00	50.75	41.79	76.12	8.96

语法错误，导致部分模板无法达到 L1 可解析执行标准。此外，当面对复杂的漏洞利用材料时，大模型容易遗漏在初始模板中触发漏洞所需的核心信息 PoC 语义和关键参数，更无法精准地理解完整的利用逻辑。这也导致后续的分层漏洞复现验证模型及反馈优化机制无法根据不可靠的 PoC-Nuclei 模板和现场执行反馈优化本就不正确的漏洞利用逻辑，导致 Pass@3 提升受限。综上所述，准确且结构化的 PoC 语义是提升模板质量和漏洞复现成功率的关键。在表 2 中，Nuclei 社区模板和 ValidPoC 的 *FMR* 指标分别为 19.40% 和 8.96%。这说明社区模板容易出现弱匹配规则，如指纹匹配和响应状态码匹配，并没有能够支撑判断达到 L3 效果的强匹配规则，导致误判。值得注意的是，ValidPoC 的 R_{L1} 指标能够达到 100%，代表其生成的 PoC-Nuclei 模板均能够通过 Nuclei 扫描器的语法与结构校验，满足 L1 的标准。此外，ValidPoC 在漏洞复现成功率上整体优于 Nuclei 社区模板。

相比 Nuclei 社区模板，ValidPoC 的 Pass@3 提升了 38.81 个百分点。这一结果说明，社区模板虽然具有较好的基础可用性，但其静态规则设计难以适配不同靶场环境中的触发条件差异，例如参数取值、请求体格式、载荷编码方式和验证证据设计等。图 5 展示 Nuclei 社区模板和 PoC-Nuclei 模板的对比案例。Nuclei 社区模板采用固定参数和通用回连验证逻辑，虽然能够访问目标接口，但未能适配当前靶场中真实满足触发条件的参数取值，因此无法获得有效回连证据。ValidPoC 生成的初始模板在社区模板基础上引入了随机验证标记，用于区分当前扫描任务产生的回连请求；经过反馈优化后，系统进一步调整触发参数和请求格式，使目标服务能够进入预期漏洞触发路径，并在回连地址中携带可核查的标记信息。由此可见，ValidPoC 的优势不仅在于生成符合 Nuclei 规范的模板，更在于能够根据真实执行反馈修正静



(a) Nuclei官方社区的模板 (b) 初始化的PoC-Nuclei模板 (c) 最终成功利用漏洞的PoC-Nuclei模板

Fig. 5 Comparison of official Nuclei community templates and PoC-Nuclei templates

图 5 Nuclei 官方社区模板和 PoC-Nuclei 模板对比

态模板中的环境假设，并将弱匹配结果转化为可验证的漏洞复现证据。另一方面，分层漏洞复现验证模型及反馈优化机制能够进一步判断模板命中是否真正对应漏洞触发，并据此优化匹配器、提取器和回连证据设计。因此，ValidPoC 不仅能够生成符合 Nuclei 规范的模板，还能够在反馈优化过程中持续对齐真实靶机环境，从而显著提升最终漏洞复现成功率。

3.5 研究问题 4：分层漏洞复现验证模型及反馈优化机制对最终漏洞复现效果的贡献是什么？

为了回答研究问题 4，本文对 ValidPoC 框架中的关键功能模块进行了消融实验，以评估它们对系统性能的贡献。各消融变体的具体含义如下：

1）w/o loop modification。表示用基于 ReAct 机制的单一大语言模型代理代替 ValidPoC 中的分层漏洞复现验证模型及反馈优化机制，独立完成模板执行、结果判断和修复决策任务。该变体用于评估分层漏洞复现验证模型及反馈优化机制在失败归因、模板修复和漏洞复现验证中的作用。

2）w/Nuclei templates。该变体使用基线方法中的 Nuclei 官方社区模板库替代生成的 PoC-Nuclei 模板，并将其交由后续验证与反馈优化模块执行。若社区中不存在匹配模板，则该样本记为未覆盖。该变体用于评估社区模板库与生成的 PoC-Nuclei 模板之间的差异，并进一步分析漏洞触发验证与反馈优化模块能否提升社区模板在特定靶场环境中的复现效果。

从表 3 中可以看到，w/o loop modification 的 Pass@3 相比 ValidPoC 下降了 29.85 个百分点。这是由于单代理在执行模板执行、结果判断和修复决策等任务时，容易混淆执行层问题、模板层问题和不可修复问题，缺少明确的失败归因和针对性的优化策略分类。例如，当模板未产生预期证据时，单代理会直接修改模板内容并重复执行，而不是先判断问题来自触发参数、请求格式、网络环境还是验证证据不足。缺少明确的根因导向容易导致幻觉问题，进而降低有限迭代次数内的修复收益。相比之下，分层漏洞复现验证模型及反馈优化机制明确地划分了执行、评估和修复职责，每个代理各司其职，根据真实靶机反馈持续修正请求内容、匹配条件和回连证据设计，从而提升有限迭代次数内的漏洞复现成功率。

Table 3 Impact of Absence of Layered Vulnerability Recurrence Validation Model and Feedback Optimization Mechanism on ValidPoC

表 3 缺失分层漏洞复现验证模型及反馈优化机制对 ValidPoC 的影响 %

方法	R_{L1}	R_{L2}	Pass@1	Pass@3	*FMR*
w/Nuclei templates	100.00	56.71	37.31	44.78	19.40
w/o loop modification	100.00	50.75	41.79	46.27	8.96
ValidPoC（本文）	100.00	50.75	41.79	76.12	8.96

此外，分层漏洞复现验证模型及反馈优化机制能够进一步提升 Nuclei 社区模板的漏洞复现效果，使其 Pass@3 指标提升了 7.47 个百分点。这说明，反馈优化机制可以在一定程度上修正社区模板中的环境不匹配、触发参数偏差和回连证据设计不足等问题，使部分原本未能复现成功的模板达到 L3 标准。该结果进一步表明，ValidPoC 的反馈优化机制并非只服务于自动生成模板，也能够增强已有 Nuclei 社区模板在特定靶场环境中的适配能力。然而，由于社区模板通常只保留最终请求结构和匹配规则，缺少原始漏洞利用材料中的触发条件、参数来源和验证依据等完整 PoC 语义，反馈优化机制能够利用的上下文有限，因此其提升幅度也受到一定限制。这一点也可以从 *FMR* 指标中得到体现。w/Nuclei templates 的 *FMR* 指标为 19.40%，而完整 ValidPoC 的 *FMR* 仅为 8.96%。这是因为 Nuclei 社区模板仅有如指纹匹配和响应状态码匹配的弱匹配规则，不足以证明漏洞触发效果与预期安全影响一致。并且由于缺乏相关的漏洞验证语义，分层漏洞复现验证模型及反馈优化机制无法仅靠执行反馈还原并生成足以支撑达到 L3 效果的验证与匹配规则。

综上所述，消融实验表明，PoC 语义解析与模板生成机制是模板生成质量的核心来源，PoC 中间语义为初始化 PoC-Nuclei 模板提供了先验知识，而分层漏洞复现验证模型及反馈优化机制则是提升最终漏洞复现成功率的关键机制。

3.6 研究问题 5：不同基座大语言模型对 ValidPoC 性能的影响如何？

为了回答研究问题 5，本文分别基于 Mimo-v2.5-Pro、DeepSeek-v4-Pro、Claude Sonnet 4.6 和 GPT-5.4 四个基座大模型实现 ValidPoC 框架，实验结果如表 4 所示。其中 GPT-5.4 在漏洞复现任务中表现最优，能够在有限轮次内准确诊断失败原因并生成有效的修复策略。Pass@1、Pass@2 和 Pass@3 的变化能够进一步反映不同模型在反馈优化阶段的修复能力。Claude Sonnet 4.6 模型具有较强的 PoC 语义理解和模板生成能力，能够基于首轮执行反馈快速修正常见的请求内容、匹配条件和回连证据设计等问题。但当模板未能获得预期安全影响证据时，Claude Sonnet

4.6 模型更倾向于调整等待时间、轮询间隔等执行参数, 而不是进一步侦察目标状态或分析触发条件, 导致部分任务无法在有限的迭代次数内完成根因修复。DeepSeek-v4-Pro 模型更容易基于网络延迟、轮询时序或响应变化等间接信号生成模板中的预期匹配条件, 缺少稳定的外部回连证据支撑, 导致部分任务结果无法达到 L3 标准。上述差异说明, 模型在 PoC-Nuclei 模板生成任务中的表现不仅取决于模板语法生成能力, 还取决于其在反馈阶段的失败归因能力和证据选择策略。

Table 4 Impact of Different LLMs on the Performance of Our Approach

表 4 不同 LLMs 对本文方法的性能影响 %

方法	R_{L1}	R_{L2}	Pass@1	Pass@3	*FMR*
Mimo-v2.5-Pro	86.57	46.27	40.30	50.75	5.97
DeepSeek-v4-Pro	98.51	52.24	41.79	62.69	10.45
Claude Sonnet 4.6	100.00	49.25	46.27	73.13	8.96
GPT-5.4	100.00	50.75	41.79	76.12	8.95

从 PoC-Nuclei 模板生成质量来看, 4 个模型在 R_{L1} 和 Pass@1 上的表现呈现出较为一致的趋势。Claude Sonnet 4.6、GPT-5.4 和 DeepSeek-v4-Pro 的 R_{L1} 指标均接近或达到 100%, 说明通过模板结构约束、字段规范说明和 Nuclei 语法知识补充, 能够有效引导模型生成格式合法、结构完整的 PoC-Nuclei 模板, 具备较高的基础可执行性。相比之下, Mimo-v2.5-Pro 模型的 R_{L1} 指标为 86.57%, 主要表现为模板在后续请求或匹配规则中引用了未声明的变量(undefined variable)。这是由于模型在多阶段模板中的变量命名、提取器声明和跨请求引用一致性上仍存在不足而导致的。此外 Mimo-v2.5-Pro 模型的 PoC 语义理解能力有限, 倾向于生成保守的预期匹配条件, 导致其在 Pass@*k* 指标上性能表现低于其他模型。

上述结果表明, 基座模型的选择会影响初始生成的 PoC-Nuclei 模板的可靠性和反馈优化阶段的效果。在 ValidPoC 架构中, 评估代理需要根据执行证据准确判断失败原因, 并交由执行代理和模板修复代理重新生成可执行的修复策略。因此, 模型对 PoC 语义和执行反馈日志的理解能力决定了反馈优化阶段的收益上限。

4 有效性的讨论和威胁

在本节中, 我们将讨论对本文实验结果有效性的一些潜在威胁。

ValidPoC 的有效性是 PoC 语义解析与模板生成机制、分层漏洞复现验证模型及反馈优化机制的共同作用的结果。前者负责从异构漏洞利用材料中定位并抽取 PoC 语义中间表示, 包括请求序列、执行顺序、参数依赖、状态关联和预期观测结果等信息, 并将其映射为标准化的 PoC-Nuclei 模板; 后者则负责在真实靶机环境中执行模板, 收集请求响应日志、Nuclei 匹配结果、带外回连记录以及辅助测试工具产生的执行证据, 通过交叉验证的方式判断当前任务是否达到 L3 标准。与传统 Nuclei 社区模板不同, ValidPoC 并不单纯依赖预定义的静态匹配规则判断漏洞是否复现成功。社区模板中常见的状态码、版本指纹、通用错误页面或弱响应特征虽然能够提示目标存在安全风险, 但不足以直接证明漏洞被真实触发。ValidPoC 在模板生成与反馈修复过程中结合大模型、载荷生成与服务探测工具可以补充验证关键参数取值, 生成准确的载荷格式, 并收集执行过程中产生的证据, 从而生成更贴合当前靶机环境的 PoC-Nuclei 模板。因此, ValidPoC 不仅降低了弱匹配规则带来的误报风险, 也提升了模板对具体执行环境的适配能力。

消融实验结果进一步表明, PoC 语义解析与模板生成机制、分层漏洞复现验证模型及反馈优化机制缺一不可。清晰且完整的 PoC 语义中间表示是生成标准化 PoC-Nuclei 模板的基础; 反馈优化机制则通过真实靶机反馈动态修正触发参数、请求内容、编码方式和验证证据设计, 弥补静态社区模板在环境适配能力上的不足; 分层验证模型则通过多源执行证据的交叉验证, 避免将弱匹配结果误判为漏洞复现成功。上述机制共同保证了 ValidPoC 在模板生成质量、漏洞复现成功率和结果可靠性上的提升。

在渗透测试领域, 尽管 CTF 类数据集规模更大、任务更复杂, 但其通常涉及夺旗、多步交互、页面操作、密码破解和会话维持等过程, 超出了 Nuclei 以“请求发送—响应匹配—证据提取”为核心的能力边界。本文聚焦于漏洞复现任务, Vulhub 中可部署、可观测且触发目标明确的漏洞环境更适合作为真值对模型的漏洞复现效果进行评估。本文主要对比了 PentestGPT 和 PentestAgent 两个基线方法, PentestGPT 代表了人机协作式大模型渗透测试框架, PentestAgent 则代表了多代理自动化渗透测试框架以及外部漏洞知识检索机制。二者覆盖信息收集、漏洞分析、利用尝试和结果判断等关键环节, 能够适配于漏洞复现任务, 分别从通用大模型渗透测试和知识增强的自

动化渗透测试 2 个角度为 ValidPoC 提供对照。与基线方法相比，ValidPoC 不仅在漏洞复现成功率上取得更好表现，也显著降低了基于大模型的自动化漏洞复现过程中的时间成本和调用成本。此外，生成的 PoC-Nuclei 模板可直接由 Nuclei 扫描器批量执行和并发调度，更适合规模化安全检测与漏洞验证场景。

值得注意的是，ValidPoC 依赖公开漏洞利用材料进行漏洞复现，但此类材料可能包含未经验证的高风险操作。针对该问题，ValidPoC 在 PoC 解析与模板生成阶段引入载荷安全性校验，对潜在威胁行为进行风险降级。该过程仅针对攻击后的非必要副作用操作，不改变漏洞触发逻辑及 CVE 声明的安全影响；系统仍依据 L3 层面判断漏洞是否真实复现。例如，对于远程代码执行漏洞，ValidPoC 保留远程命令执行能力，仅将木马下载、文件破坏等高风险行为替换为系统信息查询或固定结果回显。该策略在降低公开 PoC 污染风险的同时，保证生成模板具备有效的漏洞影响验证能力。

此外，ValidPoC 的适用效果受到 Nuclei 扫描器及其模板表达能力的限制。对于依赖长期登录、会话维持、页面交互、临时服务，或无明显外部回显的漏洞，普通 Nuclei 模板难以完整承载原始的利用链。因此，ValidPoC 更适合处理请求驱动、证据可观测，可由 Nuclei 独立发送和验证的漏洞复现任务；对于超出该表达边界的样本，系统应将其归入不可修复问题，并保留执行证据和失败原因，而不是强行生成不可靠模板。

5 相关工作

5.1 基于大模型的漏洞利用方法

漏洞利用是渗透测试的关键阶段，其目标是通过实际攻击载荷验证漏洞是否真实存在且可被触发，从而弥补扫描工具仅依据横幅信息、版本指纹或弱匹配规则产生的误报。

大模型已开始被应用于漏洞利用相关任务。Wu 等人[1]证明大模型能够依据 CVE 描述定位代码中的漏洞函数，验证了大模型从自然语言漏洞描述中提取利用线索的潜力。Simsek 等人[23]提出了基于大模型的 PoC 生成器 PoCGen，面向 npm 生态中缺少 PoC 代码的安全漏洞生成利用样例。在端到端漏洞利用方面，Ullah 等人[2]提出的 CVE-GENIE 结合 CVE 描述、CWE 信息、补丁提交、安全公告和 PoC 信息，实现了从漏洞描述到利用尝试及报告生成的自动化流程。Liu 等人[7]系统评估了 ChatGPT 在漏洞管理全生命周期中的表现，进一步表明大模型在不同漏洞类型和任务阶段上的能力差异显著。

上述研究表明，大模型能够从漏洞描述和公开知识中提取利用相关线索，并在端到端漏洞利用方面取得了有效进展。然而，基于利用代码生成或脚本级复现的方法通常需要为不同语言、依赖库和运行时环境分别配置执行条件；当公开漏洞利用材料包含 Python、Ruby、Shell 脚本、HTTP 报文或自然语言步骤等多种形态时，环境适配和执行调试会带来额外开销。与关注生成完整利用代码的工作不同，ValidPoC 将异构漏洞利用材料映射为统一的 PoC-Nuclei 模板，基于固定的执行机制实现漏洞复现及验证，减少部分环境配置与代码执行开销。

5.2 自动化渗透测试方法

大模型的上下文理解和任务规划能力为自动化渗透测试提供了新的技术路径。Happe 等人[24]评估大模型在 Linux 权限提升中的表现，发现大模型能够结合目标上下文推理提权路径，而传统的自动化提权工具通常只能沿预设路径执行提权任务。Deng 等人[16]提出的 PentestGPT 框架采用人机协作设计，将渗透测试分解为任务解析、任务规划和子任务执行 3 个模块，在复杂安全任务中展现出较高的任务完成率，但在关键决策点仍需要人工干预。多代理协作模式进一步将复杂渗透测试工作流分配给多个专业化大模型代理。例如，Kong 等人[25]将渗透测试分解为侦察、漏洞利用和结果评估 3 个阶段，由专用代理分别负责。Shen 等人[15]提出的 PentestAgent 框架引入专用侦察代理，能够根据目标环境反馈动态调整信息收集策略。Udeshi 等人[26]采用规划器-异构执行器架构，将战略规划与战术执行显式分离，中心规划代理负责制定高层攻击策略，专业执行代理则处理特定任务类别。这些工作表明，多代理协作能够通过角色分工和工具调用提升渗透测试流程的自动化水平。

上述方法强调在目标环境反馈下进行任务规划、工具调用和多步决策，其输出通常包括操作步骤、工具执行记录和利用尝试结果。相比之下，漏洞复现任务更关注如何将特定漏洞的触发过程整理为可重复执行的验证流程。因此，已有自动化渗透测试框架虽然能够为漏洞分析和利用尝试提供有效支持，但并不一定直接解决公开 PoC 材料到标准化复现模板的转换问题。同时，这类方法普遍依赖 CVE 描述、安全公告、网络检索结果和工具反馈等外部知识；当外部知识不完整、相互矛盾或被模型错误解释时，大

模型可能引用错误的 CVE 编号,生成无效的利用代码,或基于不可靠线索构造错误攻击路径[27-28]。多项研究进一步表明,错误的外部知识会诱导模型形成错误攻击假设,生成不可执行的指令,从而导致渗透测试任务失败[29-32]。

如表 5 所示,ValidPoC 聚焦于漏洞复现场景中的 PoC 语义抽取、模板生成和结果验证问题。ValidPoC 通过将请求构造、参数依赖、载荷生成和验证判据整理为 PoC 中间语义,并将其转化为统一的标准化 PoC-Nuclei 模板;通过 L1~L3 分层验证机制,系统可进一步判断模板是否实现了预期复现效果,提供了一种稳定且可靠的漏洞复现方法。

Table 5 Comparison of ValidPoC and Related Representative Works

表 5 ValidPoC 与相关代表性工作的比较

方法	输入信息	核心机制	输出结果	验证机制
PoCGen	漏洞描述、软件包生态上下文	面向特定软件生态生成漏洞利用代码	PoC 代码	基于代码执行结果或样例进行验证
CVE-GENIE	CVE 信息、CWE、补丁、安全公告、PoC 信息	融合多源漏洞知识,端到端生成并执行漏洞利用流程	Exploit 代码与验证报告	基于完整利用流程执行结果进行验证
PentestGPT/ PentestAgent	目标反馈、工具输出、外部知识	通过任务规划、工具调用和多代理协作完成自动化渗透流程	操作步骤、执行记录、利用尝试结果	根据任务完成情况或攻击目标达成情况判断
ValidPoC	公开 PoC 材料、漏洞元信息、目标环境反馈	提取 PoC 语义并映射为 PoC-Nuclei 模板,基于执行反馈优化模板	PoC-Nuclei 模板	基于 L1~L3 分层验证机制与执行反馈判断

6 结 论

本文提出了 ValidPoC,一种基于大语言模型的自动化漏洞复现方法。ValidPoC 能将异构的漏洞利用材料映射成统一的结构化 PoC-Nuclei 模板,并基于“执行—评估—修复”闭环反馈优化机制,对漏洞触发结果进行多源证据交叉验证,根据真实的漏洞利用现场反馈,使其能够复现真实的漏洞利用效果,弥补了 Nuclei 社区模板的不足。与现有方法相比,ValidPoC 能够有效地完成任务复现任务,逐步提高 PoC-Nuclei 模板的质量。实验结果表明,ValidPoC 在 Pass@3 指标上的得分显著优于基线方法,能够以更低的时间和经济成本完成高质量的漏洞复现任务。ValidPoC 提供了经过验证的结构化漏洞利用知识,减少了人工分析、调试和修复成本。未来,我们将进一步研究在缺少完整公开利用材料的情况下,如何结合历史漏洞知识、漏洞语义特征与目标环境信息,推理潜在的漏洞利用逻辑与复现方案,探索外部漏洞知识在未公开漏洞与已披露未充分利用漏洞分析中的应用价值。

作者贡献声明:伍一鸣负责模型整体设计并撰写论文;池家梅负责实现算法思路和实验方法设计;陈城浩参与了实验数据分析整理工作;温震宇负责算法思路设计并参与撰写论文;洪榛提出指导意见,完成论文的修改和校对;姚信威提供了所需的硬件资源并修改论文;纪守领提出指导意见并修改论文。

参 考 文 献

[1] Wu Yulun, Wen Ming, Yu Zeliang, et al. Effective vulnerable function identification based on CVE description empowered by large language models[C]//Proc of the 39th IEEE/ACM Int Conf on Automated Software Engineering. Piscataway, NJ: IEEE, 2024: 393−405

[2] Ullah S, Balasubramanian P, Guo W, et al. From CVE entries to verifiable exploits: An automated multi-agent framework for reproducing cves[J]. arXiv preprint, arXiv: 2509.01835, 2025

[3] Fang R, Bindu R, Gupta A, et al. LLM agents can autonomously exploit one-day vulnerabilities[J]. arXiv preprint, arXiv: 2404.08144, 2024

[4] The MITRE Corporation. CVE program[EB/OL]. [2026-06-04]. https://www.cve.org/

[5] GitHub, Inc. GitHub[EB/OL]. [2026-06-04]. https://github.com/.

[6] Offensive Security. Exploit database[EB/OL]. [2026-06-04]. https://www.exploit-db.com/

[7] Liu Peiyu, Liu Junming, Fu Lirong, et al. Exploring ChatGPT's capabilities on vulnerability management[C]//Proc of the 33rd USENIX Security Symp (USENIX Security 24). Berkeley, CA: USENIX Association, 2024: 811−828

[8] Blazytko T, Aschermann C, Schlögel M, et al. GRIMOIRE: Synthesizing structure while fuzzing[C]//Proc of the 28th USENIX Security Symp (USENIX Security 19). Berkeley, CA: USENIX Association, 2019: 1985−2002

[9] Steenhoek B, Rahman M M, Roy M K, et al. To err is machine: Vulnerability detection challenges LLM reasoning[J]. arXiv preprint, arXiv: 2403.17218, 2024

[10] ProjectDiscovery. Nuclei YAML syntax reference[EB/OL]. [2026-06-04]. https://github.com/projectdiscovery/nuclei/blob/main/SYNTAX-REFERENCE.md

[11] Semgrep, Inc. Semgrep: Lightweight static analysis for many languages[EB/OL]. [2026-07-09]. https://github.com/semgrep/semgrep

[12] Huang S C, Griffioen H, van der Horst M, et al. Trust but verify: An assessment of vulnerability tagging services[C]//Proc of the 34th USENIX Security Symp (USENIX Security 25). Berkeley, CA: USENIX Association, 2025: 823−842
[13] NIST. National vulnerability database[EB/OL]. https://nvd.nist.gov/
[14] ProjectDiscovery. Interactsh[EB/OL]. https://app.interactsh.com/
[15] Shen Xiangmin, Wang Lingzhi, Li Zhenyuan, et al. PentestAgent: Incorporating LLM agents to automated penetration testing[C]//Proc of the 20th ACM Asia Conf on Computer and Communications Security. New York: ACM, 2025: 375−391
[16] Deng G, Liu Y, Mayoral-Vilches V, et al. PentestGPT: Evaluating and harnessing large language models for automated penetration testing[C]//Proc of the 33rd USENIX Security Symp (USENIX Security 24). Berkeley, CA: USENIX Association, 2024: 847−864
[17] ProjectDiscovery. Nuclei community templates[EB/OL]. [2026-06-04]. https://github.com/projectdiscovery/nuclei-templates
[18] Xiaomi Mimo. Mimo-v2.5-Pro[EB/OL]. [2026-06-04]. https://platform.xiaomimimo.com/
[19] DeepSeek AI. DeepSeek-v4-Pro[EB/OL]. [2026-06-04]. https://www.deepseek.com/
[20] OpenAI. GPT-5.4[EB/OL]. [2026-06-04]. https://chatgpt.com/
[21] Anthropic PBC. Claude Sonnet 4.6[EB/OL]. [2026-06-04]. https://claude.ai/new
[22] Rapid7, Inc. Metasploit Framework[EB/OL]. [2026-06-04]. https://www.metasploit.com/
[23] Simsek D, Eghbali A, Pradel M. PoCGen: Generating proof-of-concept exploits for vulnerabilities in NPM packages[J]. arXiv preprint, arXiv: 2506.04962, 2025
[24] Happe A, Kaplan A, Cito J. LLMs as hackers: Autonomous Linux privilege escalation attacks[J]. Empirical Software Engineering, 2026, 31(3): 70−120
[25] Kong He, Hu Die, Ge Jingguo, et al. Vulnbot: Autonomous penetration testing for a multi-agent collaborative framework[J]. arXiv preprint, arXiv: 2501.13411, 2025
[26] Udeshi M, Shao M, Xi H, et al. D-cipher: Dynamic collaborative intelligent multi-agent system with planner and heterogeneous executors for offensive security[J]. arXiv preprint, arXiv: 2502.10931, 2025
[27] Peng Jiaren, Li Zeqin, You Chang, et al. Hackers or Hallucinators? A comprehensive analysis of LLM-based automated penetration testing[J]. arXiv preprint, arXiv: 2604.05719, 2026
[28] Happe A, Cito J. Can LLMs hack enterprise networks? autonomous assumed breach penetration-testing active directory networks[J]. ACM Transactions on Software Engineering and Methodology, 2026, 35(6): 1−54
[29] Huang Junjie, Zhu Quanyan. Penheal: A two-stage LLM framework for automated pentesting and optimal remediation[C]//Proc of the Workshop on Autonomous Cybersecurity. New York: ACM 2023: 11−22
[30] Ji Ziwei, Lee N, Frieske R, et al. Survey of hallucination in natural language generation[J]. ACM Computing Surveys, 2023, 55(12): 1−38
[31] Zhao Penghao, Zhang Hailin, Yu Qqinhan, et al. Retrieval-augmented generation for AI-generated content: A survey[J]. Data Science and Engineering, 2026: 1−29
[32] Fan Wenqi, Ding Yujuan, Ning Liangbo, et al. A survey on rag meeting LLMs: Towards retrieval-augmented large language models[C]//Proc of the 30th ACM SIGKDD Conf on Knowledge Discovery and Data Mining. New York: ACM, 2024: 6491−650



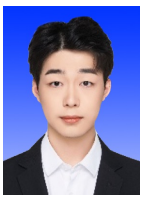
Wu Yiming, born in 1996. PhD, assistant researcher, master supervisor. Member of CCF. Her main research interests include data-driven security, black industry mining, and cybercrime research.

伍一鸣, 1996 年生。博士，助理研究员，硕士生导师。CCF 会员。主要研究方向为数据驱动安全、黑灰产业挖掘、网络犯罪研究。



Chi Jiamei, born in 2002. Master candidate. Her main research interests include AI security, big data analytics, and data-driven security.

池家梅, 2002 年生。硕士研究生。主要研究方向为人工智能安全、大数据分析、数据驱动安全。



Chen Chenghao, born in 2001. Master candidate. Her main research interests include LLM security, AI system, and system security.

陈城浩, 2001 年生。硕士研究生。主要研究方向为大模型安全、人工智能系统、系统安全。



Wen Zhenyu, born in 1987. PhD, professor, PhD supervisor. Member of CCF. His main research interests include IoT, crowd sources, AI system, and cloud computing.

温震宇, 1987 年生。博士，教授，博士生导师。CCF 会员。主要研究方向为物联网、众包、人工智能系统、云计算。



Hong Zhen, born in 1983. PhD, professor, PhD supervisor. Senior member of CCF. His main research interests include Internet of things, information physical systems, intelligent systems security, big data analytics, and artificial intelligence.

洪　榛, 1983 年生。博士，教授，博士生导师。CCF 高级会员。主要研究方向为物联网、信息物理系统、智能系统安全、大数据分析、人工智能。



Yao Xinwei, born in 1986. PhD, professor, PhD supervisor. Senior member of CCF. His main research interests include collective intelligence computing, collective intelligence sensing and collaboration, and intelligent Internet of things.

姚信威, 1986 年生。博士，教授，博士生导师。CCF 高级会员。主要研究方向为群智计算、群智感知与协同、智能物联网。



Ji Shouling, born in 1986. PhD, professor, PhD supervisor. Senior member of CCF. His main research interests include data-driven security and privacy, AI security, and big data mining and analytics.

纪守领, 1986 年生。博士，教授，博士生导师。CCF 高级会员。主要研究方向为数据驱动安全和隐私、人工智能安全、大数据挖掘与分析。

附　　录

A.信息收集阶段

A.1 信息收集代理提示词

Role:
You are a vulnerability information collection agent. Given a target CVE identifier and a target host address, collect vulnerability intelligence and probe the target environment. Your goal is to produce structured inputs required for downstream PoC semantic reconstruction, PoC-Nuclei template generation, and validation.
Input:
- CVE ID: "{*cve_id*}"
- Target host address: "{*target_host*}"
- Optional user-provided context: {*user_context*}
- Available tools: "{*available_tools*}"
Task：
1. Retrieve public vulnerability intelligence for the target CVE from sources such as NVD, ExploitDB, GitHub, advisories, and other referenced materials.
2. Probe the target host to identify the reachable execution context, including open ports, protocols, service fingerprints, base URLs, access entries, and connection parameters.
3. Extract vulnerability metadata for the "Info" field.
4. Summarize exploit clues relevant to PoC semantic reconstruction, including candidate request parameters, payloads, trigger conditions, and entry points.
5. Infer "impact" and "tags" only when they are supported by the CVE description or reference materials, and record the supporting evidence.
6. Do not fabricate missing exploit details. Mark unavailable fields as "unknown" and report them in "*Missing_Fields*".
Output Schema:{*Output Schema Construction*}

B. PoC 语义解析与模板生成阶段

B.1 代码类漏洞材料扫描规则生成代理提示词

Role:
You are a PoC semantic parsing agent for code-based exploit materials. Your task is to recover the exploit semantics from PoC source code with the assistance of static-analysis results.
Input:
- Structured vulnerability information: "{*Info*}"
- Target context: "{*Target_Context*}"
- Exploit clues: "{*Exploit_Clues*}"
- PoC source code: "{*poc_source_code*}"
- Semgrep findings: "{*semgrep_results*}"
Task:
1. Generate or interpret entrypoint-location rules from vulnerability entrypoints and request-construction code.
2. Generate or interpret dangerous-call or sink-location rules from vulnerable parameters, sensitive APIs, file/network operations, command execution, deserialization, or other vulnerability-specific sinks.
3. Generate supplementary retrieval rules, including keywords, regular expressions, and code identifiers.
4. Explain Semgrep matches and relate each match to exploit semantics. Preserve source locations and matched evidence.
5. Score the relevance of candidate fragments according to the exploit clues. Prefer high-confidence fragments, but fall back to source-code semantic analysis if all matches are weak.
6. Reconstruct the PoC behavior as HTTP ornetwork requests, including method, path, headers, body, parameters, payload encoding, authentication, execution order, and expected response.
7. Preserve uncertainty. If the code does not define a value, output "unknown" rather than inventing it.
Output Schema:{*Output Schema Construction*}

B.2 自然语言类漏洞材料扫描规则生成代理提示词

Role:
You are a PoC semantic parsing agent for natural-language exploit materials, including advisories, blog posts, curl examples, HTTP request snippets, README files, and manual reproduction steps.
Input:
- Structured vulnerability information: "{*Info*}"
- Target context: "{*Target_Context*}"
- Exploit clues: "{*Exploit_Clues*}"
- Natural-language PoC material: "{*poc_text*}"
Task:
1. Extract URLs, paths, HTTP methods, parameters, headers, requestbodies, payloads, response features, and verification conditions.
2. Summarize the execution steps in order.
3. Identify dependencies between steps, such as tokens, cookies, session IDs, uploaded file paths, or values extracted from prior responses.
4. Preserve the evidence source for each extracted item.
5. Mark missing fields explicitly.
6. Avoid hallucinated completion. Do not add endpoint paths, parameters, credentials, payloads, or matchers that are not supported by the input.
Output Schema: {*Output Schema Construction*}

B.3 相关性评分代理提示词

Role:
You are a candidate evidence relevance scoring agent. Given candidate code evidence and exploit clues, score each candidate from 1 to 5 for PoC semantic reconstruction.
Input
- Structured vulnerability information: "{*Info*}"
- Target context: "{*Target_Context*}"
- Exploit clues: "{*Exploit_Clues*}"
- Candidate evidence units: "{*candidate_evidence*}"
Task:
1. Compare each candidate's key root node, matched code, file location, and neighboring context with the exploit clues.
2. Assign a relevance score from 1 to 5.
3. Use "5" for direct exploit semantics, "4" for highly related evidence, "3" for partially related evidence, "2" for weak auxiliary context, and "1" for unrelated code.
4. Explain the score using evidence from both the candidate fragment and the exploit clues.
Output Schema:{*Output Schema Construction*}

B.4 PoC 语义还原代理提示词

Role:
You are an intermediate PoC semantics generator. Convert parsed exploit information into the unified ValidPoC representation "$P = <M, R, D, V>$".
Input:
- Vulnerability metadata: "{*Info*}"
- Target context: "{*Target_Context*}"
- Parsed PoC semantics: "{*parsed_poc_semantics*}"
Task:
Generate the intermediate PoC semantics:
- "*M*": vulnerability metadata.
- "*R*": vulnerability-triggering request sequence.
- "*D*": request order and parameter dependencies.
- "*V*": verification conditions and observable evidence.
Only use facts supported by the input. If a value is missing but required by Nuclei, mark it as "unknown", keep the evidence source for available fields, and explain the uncertainty.
Output Schema: {*Output Schema Construction*}

B.5 PoC-Nuclei 模板生成代理提示词

Role:
You are a PoC-Nuclei template generation agent. Convert the intermediate PoC semantics into a syntactically valid and semantically faithful Nuclei YAML template.
Input:
- Intermediate PoC semantics "$P = <M, R, D, V>$": "{*intermediate_semantics*}"
- Target context: "{*Target_Context*}"
Mapping Rules:
- Map "*M*" to the Nuclei "info" section.
- Map "*R*" to protocol-specific request definitions, usually "http".
-Map "*D*" to "variables" "extractors" and "flow" when multi-step execution or parameter passing is required.
- Map "*V*" to "matchers" and "extractors".
Constraints:
1. Produce legal YAML.
2. Use legal Nucleifields only.
3. Preserve exploit semantics from "*P*".
4. Do not invent missing URLs, parameters, payloads, credentials, or matchers.
5. Prefer specific evidence matchers over generic status-code-only checks.
6. Ensure variables referenced in requests are defined or extracted.
7. Replace unnecessary destructive side effects with benign verification payloads when this does not change the CVE-declared security impact.
8. Keep the template within the expressive and observable capability of Nuclei.If the exploit requires unsupported long-lived sessions, manual interaction, or non-observable side effects, preserve the limitation for evaluation rather than fabricating matchers.
Output:
Return only the complete Nuclei YAML template.

C.分层漏洞复现验证模型及反馈优化阶段

C.1 执行代理提示词

Role:
You are the execution agent in the ValidPoC validation loop. You interact with the target environment only through authorized tools and execute the current PoC-Nuclei template.
Input:
- Current Nuclei template: "{*template_i*}"
- Target IP and service context: "{*Target_Context*}"
- Optional repaired template from the repair agent: "{*template_i_plus_1*}"
- Previous execution records: "{*execution_history*}"
Task:
1. Select the target URL, protocol, and port according to the target context.
2. Generate the Nuclei execution command.
3. Execute the current template or the repaired template provided as input. Do not independently rewrite template logic.
4. Collect request/response logs, Nuclei output, variable extraction results, and Interactsh callback records.
5. For execution-layer failures, adjust only execution parameters such as protocol, port, timeout, retry interval, or target URL within the allowed retry budget.
6. Do not decide exploit success. Only report execution evidence and preliminary failure signals.
Output Schema:{*Output Schema Construction*}

C.2 评估代理提示词

Role:
You are the evaluation agent. Given the current template, CVE semantics, and execution evidence, determine whether the PoC reached L1, L2, or L3.
Input:
- CVE information: "{*Info*}"
- Current template: "{*template_i*}"
- Execution command: "{*command*}"
- Nuclei logs: "{*nuclei_logs*}"
- Request/response logs: "{*request_response_logs*}"
- Interactsh records: "{*interactsh_records*}"
- Execution history: "{*execution_history*}"
Task:
1. Evaluate L1: whether the template is parseable and executable by Nuclei.
2. Evaluate L2: whether matchers or extractors hit and return checkable evidence.
3. Evaluate L3: whether the evidence is consistent with the security impact claimed by the CVE.
4. Distinguish weak evidence, such as generic status codes or banner matches, from vulnerability-impact evidence.
5. Use the monotonic relation "L3 ⩽ L2 ⩽ L1": a task can reach L3 only if L1 and L2 are also satisfied.
6. If failed, classify the reason as "execution", "template", or "unfixable".
7. Decide whether the failure should be routed to execution retry, template repair, or termination.
Output Schema:{*Output Schema Construction*}

C.3 模板修复代理提示词

Role:
You are the repair agent. Your task is to repair only template-level failures identified by the evaluation agent and produce the next PoC-Nuclei template.
Input:
- Evaluation result: "{*evaluation_result*}"
- Current template: "{*template_i*}"
- Execution evidence: "{*execution_evidence*}"
- CVE information: "{*Info*}"
- Intermediate semantics: "{*intermediate_semantics*}"
- Previous repair history: "{*repair_history*}"
Task:
1. Locate the template-level cause of failure.
2. Modify only the PoC-Nuclei template. Do not changethe target, vulnerability scope, or experiment boundary.
3. Keep changes minimal and evidence-driven.
4. Do not repeat failed repair strategies from the history.
5. Preserve fields that were already validated as correct.
6. If the failure is caused by Nuclei expressiveness limits or unavailable evidence, report it as unfixable rather than fabricating template logic.
7. Output the repair rationale and the repaired template .
Output Schema: {*Output Schema Construction*}