Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

基于智能体的自动化漏洞重现方法

硕士研究生 杨语航

2026年03月15日

问题回溯

- 存在的问题
  - 讲解语气较平淡，注意起伏
  - 讲解语速适当加快
  - 知识基础部分多增加一些内容
- 相关内容
  - 2026.03.09 王怡男《Agent or not？从程序自动修复评估智能体》
  - 2025.07.20 段学明《源代码安全补丁存在性测试》
  - 2025.06.15 王怡男《大模型支持的程序崩溃故障定位方法》

内容提要

- 预期收获
- 题目内涵解析
- 研究背景与意义
- 研究历史与现状
- 知识基础
- 算法原理
  - DrillAgent
  - K-REPRO
- 特点总结与工作展望
- 参考文献

预期收获

- 预期收获
  - 了解漏洞重现的基础知识和作用
  - 理解自动化漏洞重现中LLM的应用
  - 掌握两种基于智能体的漏洞重现技术

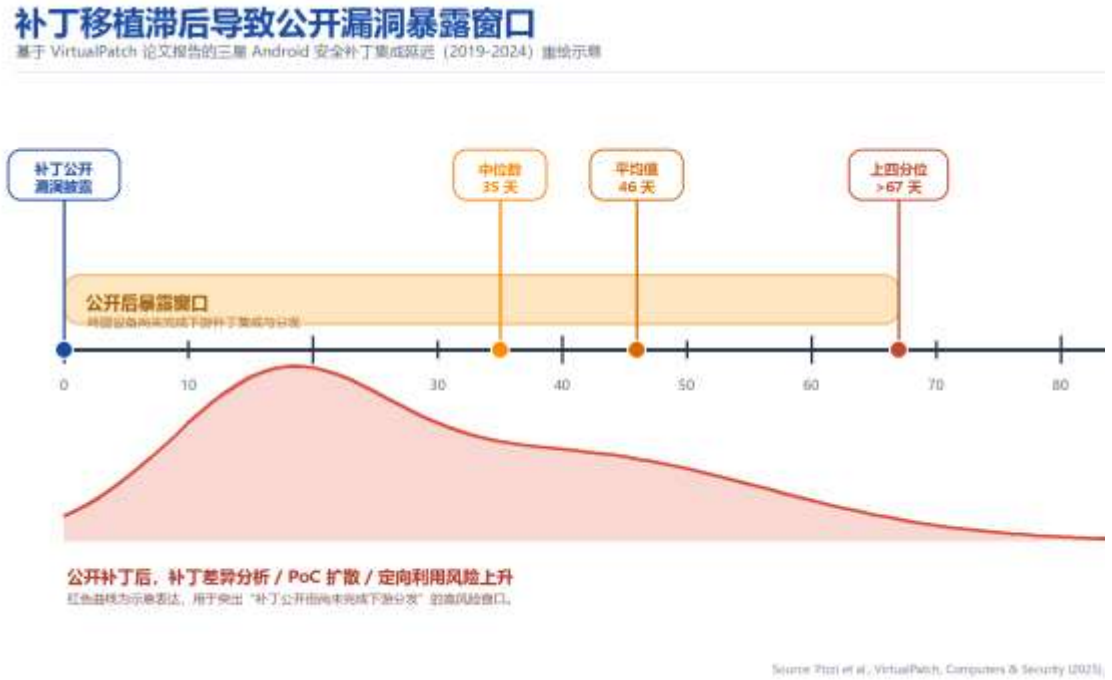
内涵解析与研究目标

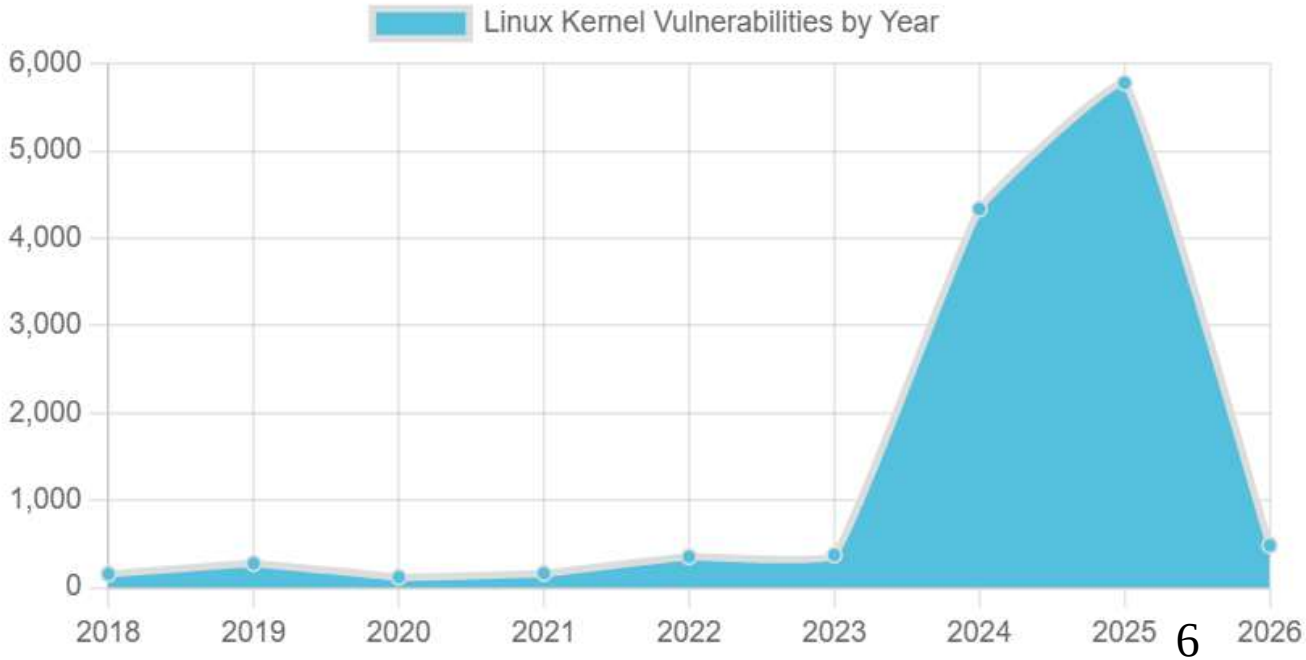
- 题目内涵解析（基于智能体的自动化漏洞重现）
  - 智能体：感知、决策、执行、验证、循环反馈
  - 漏洞重现：基于给定的漏洞信息，在受控环境中构造特定的输入来稳定触发目标程序的预期失效状态
- 研究目标
  - 漏洞重现常见以用户态程序、Web应用程序为研究对象
  - 基本流程一般包括原始漏洞信息分析 、环境构建 、生成与验证

自治闭环的通用智能体系统基本流程

研究背景与意义

- 研究背景
  - 基础设施：Linux内核被广泛使用，其攻击面庞大导致漏洞频发
  - 补丁移植滞后：安全补丁从上游同步到下游通常存在数十天甚至半年的延迟
- 研究意义：
  - 研究自动化的漏洞重现技术能极大缩短补丁验证周期、提高安全性





研究历史　漏洞重现

AFLGo 首次系统化提出定向灰盒模糊测试，把覆盖率导向的灰盒模糊测试转为距离导向的搜索问题，通过距离度量和模拟退火式能量分配，使模糊测试能够围绕补丁位置、崩溃栈或危险函数定向推进

KernelGPT 使用大模型自动推断和修复 Syzkaller 所需的 syscall specification，不完全依赖人工理解内核源码、维护规格，让 LLM 从源码、文档与验证反馈中补齐缺失的描述信息，从而显著提升 fuzzing 的可达性和有效性

DrillAgent 将 PoV 生成建模为“假设 - 执行 - 验证 - 修正”的闭环过程，与仅依赖静态分析方案不同，引入执行状态感知机制，把底层执行反馈转为源代码级语义约束并反馈给 LLM 持续修正输入

2017　2023　2025　2025　2026　2026

SyzDirect 将 directed fuzzing 真正迁移到 Linux 内核场景，针对内核中特有的syscall选择、参数入口条件等难点，结合静态分析识别可达目标位置的系统调用及参数约束，并用这些结果指导测试用例的生成、变异和调度

AVVV提出一套从 CVE/NVD 描述出发的端到端漏洞验证框架，利用 LLM 和 RAG 补全公开披露信息中缺失细节，自动完成容器化环境构建、exploit 生成、运行测试与结果验证

Patch-to-PoC提出K-Repro系统，从内核安全补丁出发，自动完成根因分析、代码浏览、虚拟机管理、交互、调试和 PoC 迭代生成，最终实现 Linux kernel N-day 漏洞的端到端重现

研究现状

- 传统漏洞重现方法
  - 定向搜索：设置目标区域，建立以距离为导向的定向灰盒模糊测试
  - 静态分析：利用程序依赖、路径约束、调用关系等信息缩小输入空间
  - 局限性：对复杂语义约束的深层逻辑漏洞处理有限
- LLM在漏洞重现中的应用
  - 常见应用：输入生成、种子选择、变异引导等
  - 主要局限：对于大模型的应用层次较浅，缺少执行反馈和闭环迭代的机制
- 前沿自动化漏洞重现方法
  - 基于智能体的自动化、端到端的漏洞验证流程

知识基础

- 漏洞证明（PoC/PoV）
  - 定义：证明漏洞存在的概念性代码、程序
- 检测器（Sanitizer）
  - 定义：编译器集成的运行时错误检测工具
  - 示例：常见有ASan、MSan、UBSan、KASan
- 代码插桩（Code Instrumentation)
  - 定义：不改变程序原逻辑情况下，插入额外的分析代码或指令，以此监控记录程序运行状态
  - 示例：常见插桩目的有覆盖率统计、漏洞检测等

Execution-State-Aware LLM Reasoning for Automated Proof-of-Vulnerability Generation

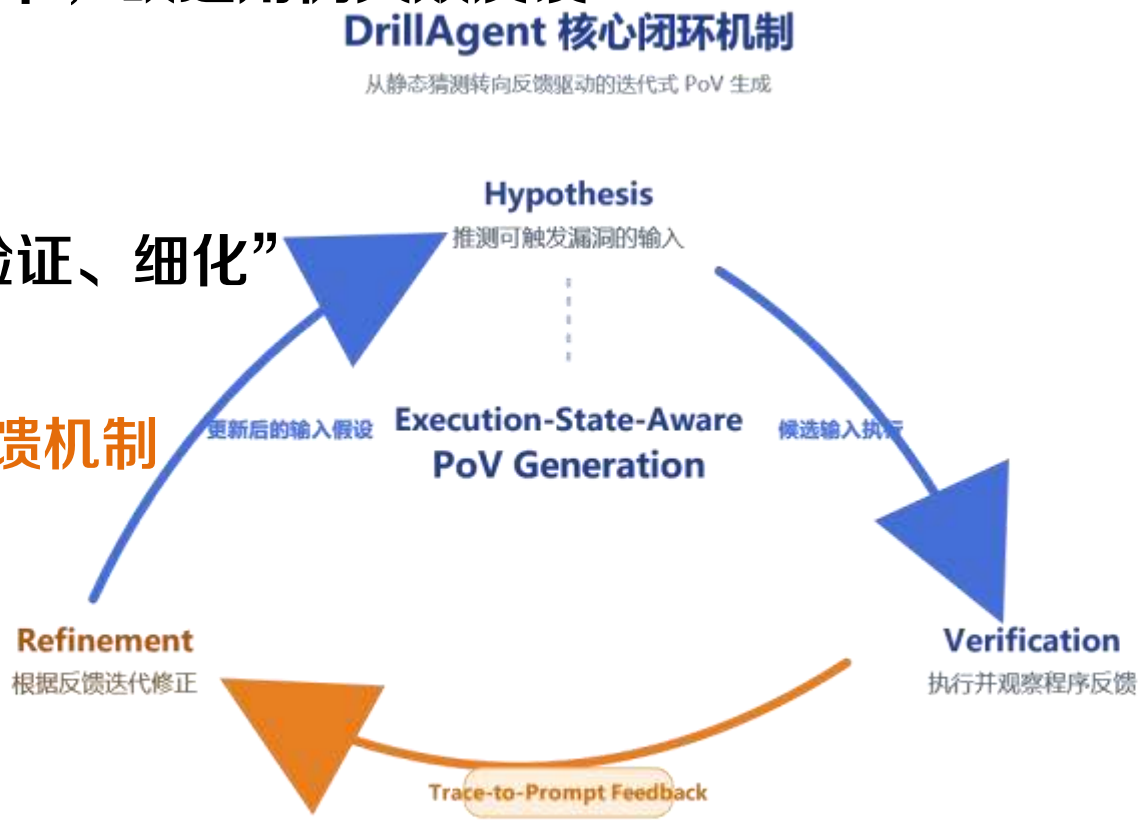
DrillAgent TIPO

T	目标	自动化PoV生成
I	输入	目标源码*1、原始漏洞信息*1
P	处理	1. 漏洞分析，将原始漏洞信息转化为结构化的推理依据 2. 代码插桩，对目标程序编译插桩 3. 路径探索，生成能到达漏洞函数附件的高质量种子 4. 崩溃触发，基于种子微调来生成PoV
O	输出	已验证PoV*1、漏洞分析报告*1
P	问题	DGF难以处理深层漏洞；基于LLM方法缺乏具体运行反馈
C	条件	高性能LLM的代码理解和生成能力；软件环境
D	难点	1. 如何处理非局部上下文依赖检索 2. 如何将底层二进制执行信号转化为可理解源码级约束
L	水平	arXiv 2026

DrillAgent 创新性引入

- 现有PoV生成方法的局限性
  - 定向灰盒模糊测试：缺乏语义理解，难以触摸深层逻辑漏洞
  - 基于LLM的方法：静态推理与执行态脱节，缺乏用例失败反馈

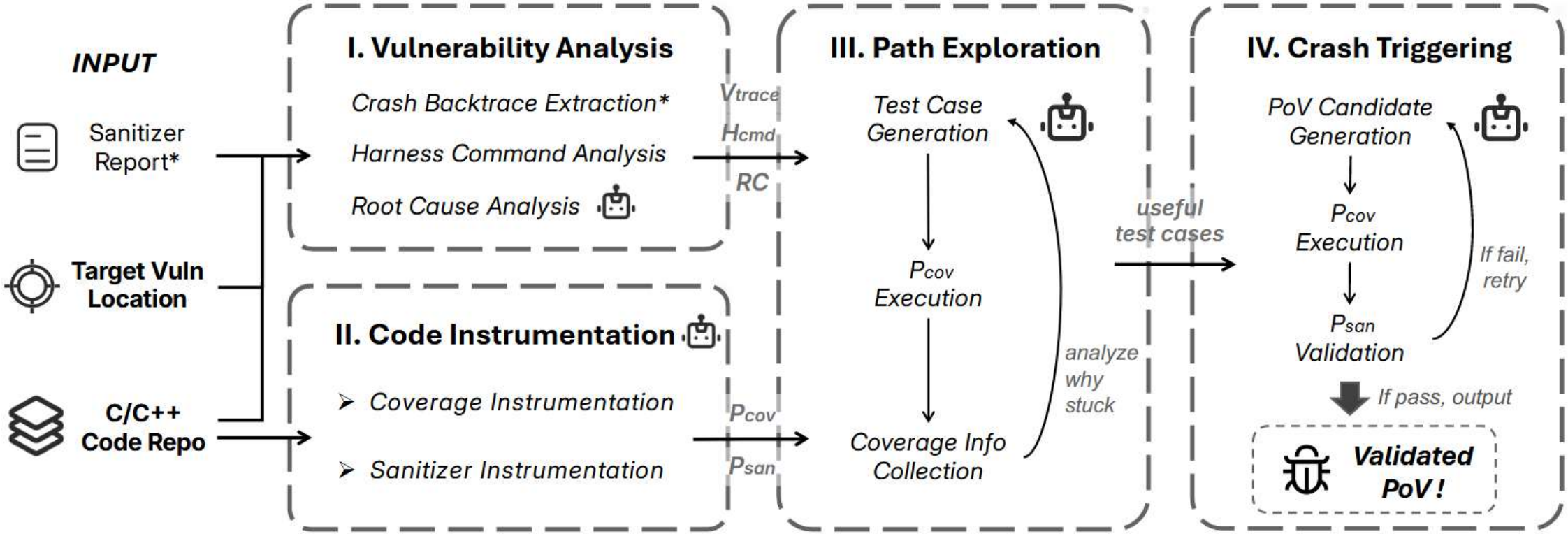
- DrillAgent创新点引入
  - 创新点1：设计一种迭代式的“假设、验证、细化”的闭环流程
  - 创新点2：语义化的Trace-to-Prompt 反馈机制



核心思想：在执行、验证与反馈中持续逼近有效 PoV，而不是一次性静态生成

DrillAgent 算法原理图

- 思路分析
  - 迭代闭环：模拟人类专家的“假设-验证-修复”迭代循环思路
  - 多Agent协同：设计四个Agent，将PoV生成任务结构化分解由LLM主导修复



算法原理 Vulnerability Analysis & Code Instrumentation

- 漏洞分析 - VAAgent
  - 目标：原始漏洞信息转化为LLM可理解的结构化信息
  - 崩溃回溯：得到函数调用栈，并对源码行号校准
  - Harness命令分析：确定程序入口、分析运行所需参数、环境等
  - 根因分析：正向推理+反向溯源+漏洞类型分析
- 自动化插桩 - ITAgent
  - 目标：对二进制程序自动编译插桩
  - P_{cov}：覆盖率插桩的二进制程序
  - P_{san}：Sanitizer插桩的二进制程序

- 原始Sanitizer报告

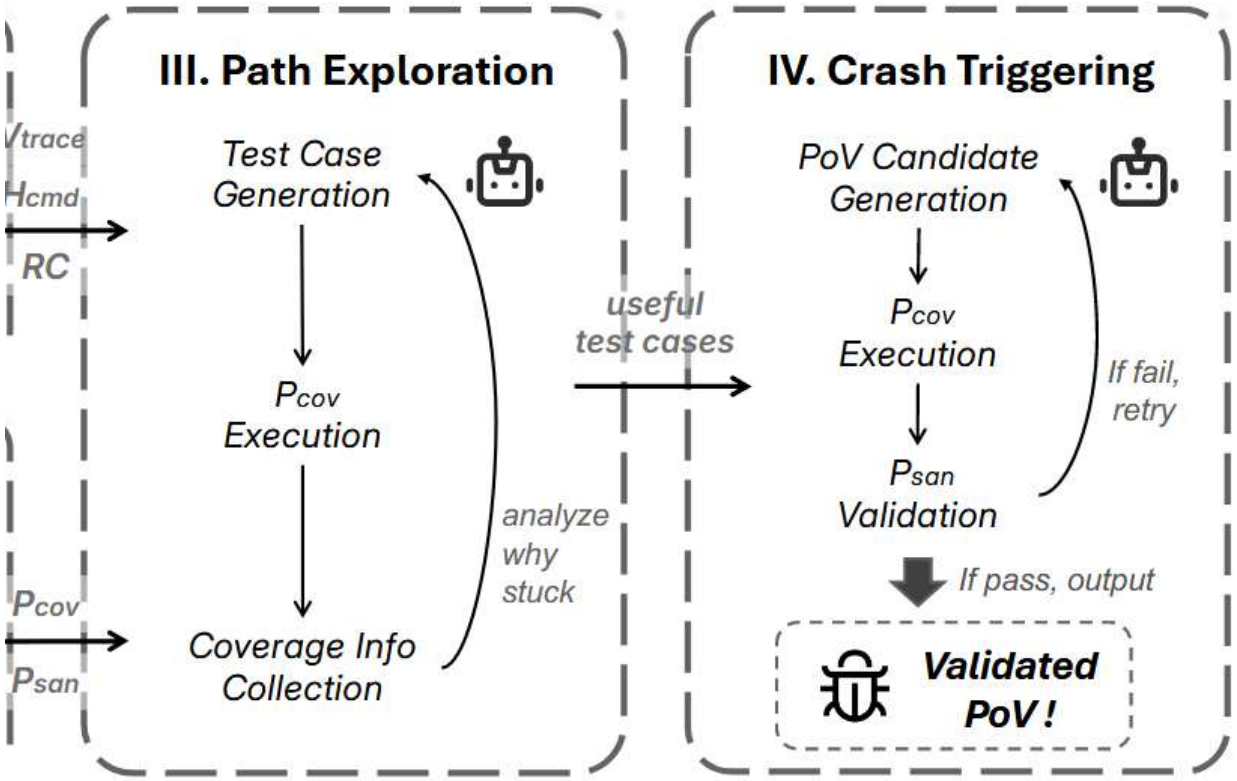
```
==================================================================
BUG: KASAN: null-ptr-deref in instrument_atomic_read_write include/linux/instrumented.h:96 [inline]
BUG: KASAN: null-ptr-deref in atomic_long_sub_and_test include/linux/atomic/atomic-instrumented.h:4521
BUG: KASAN: null-ptr-deref in put_cred_many include/linux/cred.h:284 [inline]
BUG: KASAN: null-ptr-deref in put_cred include/linux/cred.h:298 [inline]
BUG: KASAN: null-ptr-deref in io_unregister_personality io_uring/register.c:82 [inline]
BUG: KASAN: null-ptr-deref in __io_uring_register io_uring/register.c:703 [inline]
BUG: KASAN: null-ptr-deref in __do_sys_io_uring_register io_uring/register.c:907 [inline]
BUG: KASAN: null-ptr-deref in __se_sys_io_uring_register+0xcf8/0x3370 io_uring/register.c:884
Write of size 8 at addr 0000000000000406 by task syz.3.15/3114

CPU: 1 UID: 0 PID: 3114 Comm: syz.3.15 Not tainted 6.13.0-rc1-syzkaller #0
Hardware name: Google Google Compute Engine/Google Compute Engine, BIOS Google 09/13/2024
Call Trace:
 <TASK>
 __dump_stack lib/dump_stack.c:94 [inline]
 dump_stack_lvl+0x108/0x280 lib/dump_stack.c:120
 print_report+0xe8/0x550 mm/kasan/report.c:492
 kasan_report+0x143/0x180 mm/kasan/report.c:602
 kasan_check_range+0x282/0x290 mm/kasan/generic.c:189
 instrument_atomic_read_write include/linux/instrumented.h:96 [inline]
 atomic_long_sub_and_test include/linux/atomic/atomic-instrumented.h:4521 [inline]
 put_cred_many include/linux/cred.h:284 [inline]
 put_cred include/linux/cred.h:298 [inline]
 io_unregister_personality io_uring/register.c:82 [inline]
 __io_uring_register io_uring/register.c:703 [inline]
 __do_sys_io_uring_register io_uring/register.c:907 [inline]
 __se_sys_io_uring_register+0xcf8/0x3370 io_uring/register.c:884
 do_syscall_x64 arch/x86/entry/common.c:52 [inline]
 do_syscall_64+0x8d/0x170 arch/x86/entry/common.c:83
 entry_SYSCALL_64_after_hwframe+0x77/0x7f
RIP: 0033:0x7f09ef37ff19
Code: ff ff c3 66 2e 0f 1f 84 00 00 00 00 00 0f 1f 40 00 48 89 f8 48 89 f7 48 89 d6 48 89 ca 4d 89 c2 4
RSP: 002b:00007f09f0229058 EFLAGS: 00000246 ORIG_RAX: 00000000000001ab
RAX: ffffffffffffffda RBX: 00007f09ef545fa0 RCX: 00007f09ef37ff19
RDX: 0000000000000000 RSI: 000000000000000a RDI: 0000000000000003
RBP: 00007f09ef3f3986 R08: 0000000000000000 R09: 0000000000000000
R10: 0000000000000000 R11: 0000000000000246 R12: 0000000000000000
R13: 0000000000000000 R14: 00007f09ef545fa0 R15: 00007ffef35002e8
 </TASK>
==================================================================
```

- 简化后的崩溃函数调用链

```
"crash_trace": [
    {
      "sequence_num": 0, ! crash site
      "function_name": "gf_isom_box_size",
      "file_path": "src/isomedia/box_funcs.c",
      "line_number": 1997
    }, {
      "sequence_num": 1,
      "function_name": "gf_isom_is_identical_sgpd",
      "file_path": "src/isomedia/isom_read.c",
      "line_number": 5865
    },
    ...
]
```

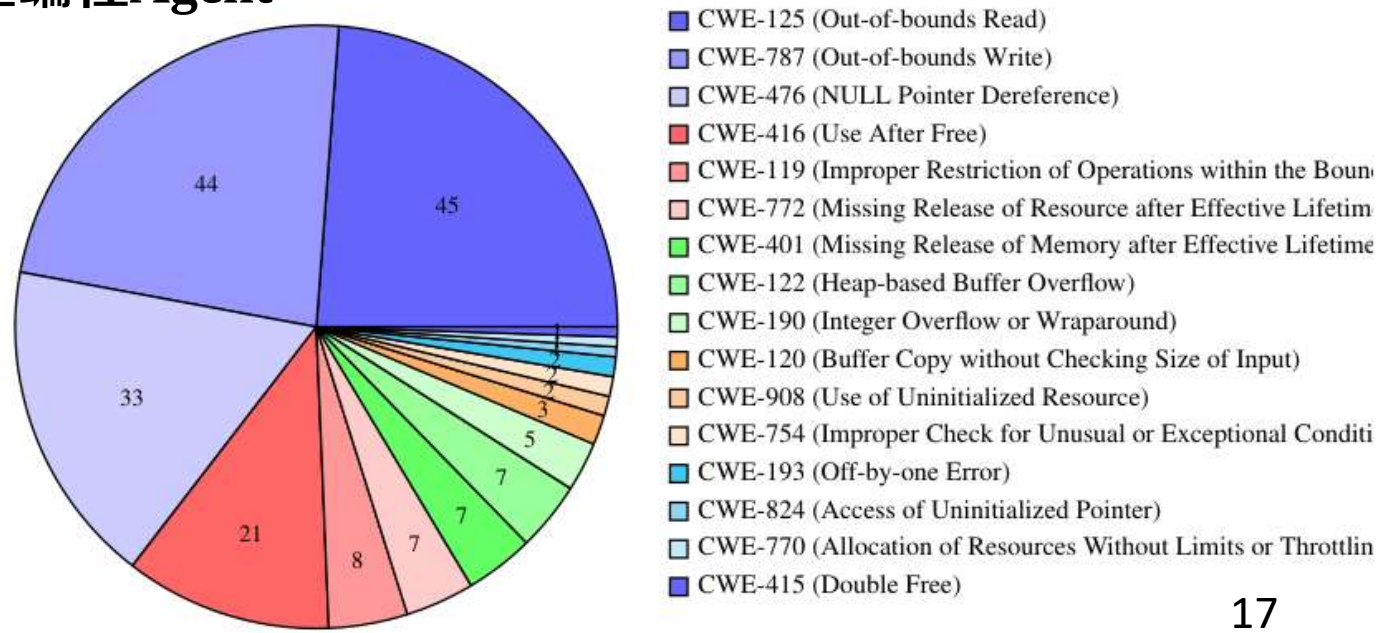
算法原理 Path Exploration & Crash Triggering

- 路径探索 - PEAgent
  - 目标：生成高质量的种子用例
  - Trace-to-Prompt
    - 混合反馈：粗粒度函数级覆盖和细粒度行级覆盖数据
    - 语义修复：底层失败信息转为可理解源码级语义约束，引导迭代修复输入脚本
- 崩溃触发 - CTAgent
  - 目标：微调种子用例，生成有效PoV
  - 漏洞引导：以VAAgent分析的漏洞类型，选择针对性prompt
  - 迭代细化：以PEAgent输出的种子用例，“微调、验证、反馈”循环去生成有效PoV



实验设计　数据资源

- 实验配置
  - 测试基准：SEC-bench-Full（190）、SEC-bench-60
  - 评价指标：有效PoV成功率、推理开销
  - 模型和参数设置：Claude Sonnet 4.5；Temperature=0.1(PEAgent和CTAgent设置0.7）
- 对比方法
  - OpenHands： SOTA级别的通用型编程Agent
- 研究问题
  - RQ1：算法有效性
  - RQ2：组件贡献情况
  - RQ3：生成效率与开销



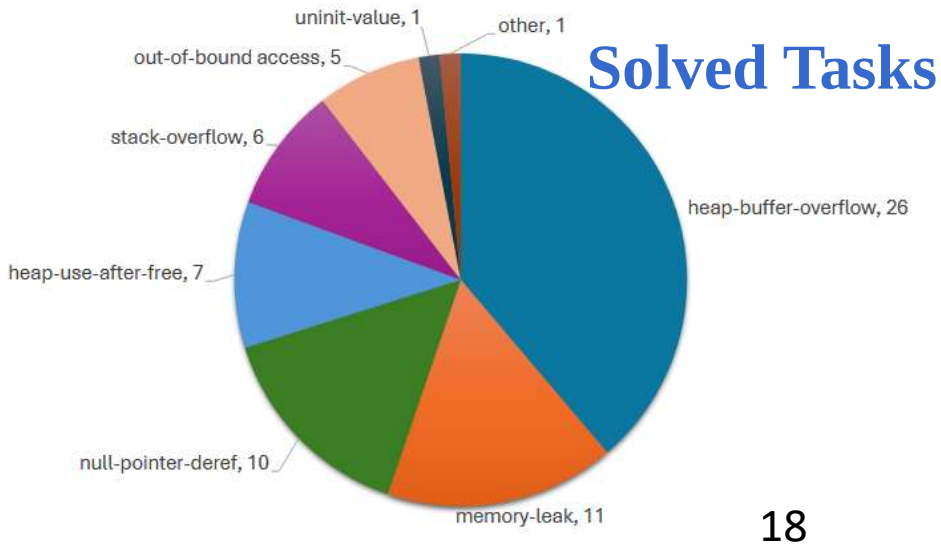
实验设计　对比实验

- **RQ1：算法有效性**

Method	Benchmark	Budget	Validated PoVs	Variant PoVs	Total Crashes	Resolved Rate†	Crash Rate†
DrillAgent	SEC-bench-Full	$1.5/task	**55**	12	**67**	28.9%	**35.3%**
DrillAgent	SEC-bench-60	$1.5/task	15	2	17	25.0%	28.3%
OpenHands	SEC-bench-60	$1.5/task	6	–	6	10.0%	10.0%
OpenHands	SEC-bench-60	unlimited	18	–	18	30.0%	30.0%

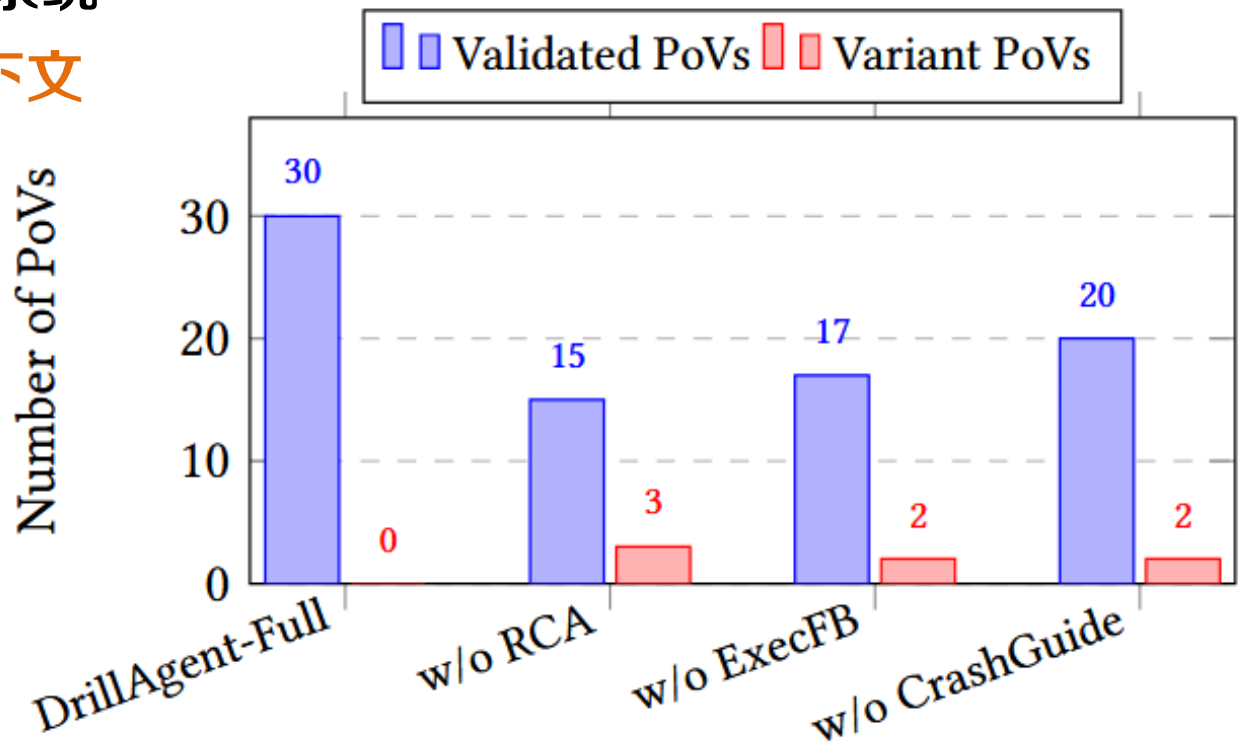
† Resolved Rate is computed as the number of Validated PoVs divided by the total number of tasks in the benchmark. Crash Rate is computed as the total number of crashes (i.e., Validated PoVs + Variant PoVs) divided by the total number of tasks.

- **结果分析**
  - **控制预算，DrillAgent效果确实强于目前最强的通用软件工程智能体OpenHands**
  - **DrillAgent更擅长构造内存破坏类的PoV**



实验设计　消融实验

- RQ2：各组件对有效PoV生成影响
  - 实验设置：在SEC-benchAblation-30上进行消融实验，仅包含DrillAgent成功解决的任务
  - DrillAgent-Full：完整的DrillAgent系统
  - w/o RCA：删除根因分析相关的上下文
  - w/o ExecFB：禁用覆盖率查询功能
  - w/o CrashGuide：禁用崩溃触发的漏洞引导策略，将部分功能集成到PEAgent



- **RQ3： PoV生成效率分析**
  - **实验设置：在SEC-benchAblation-30上进行消融实验，仅包含DrillAgent成功解决的任务**

Method	Benchmark	Budget	Cost/Task ($)	Cost/Success* ($)	Exec Time (min)
DrillAgent	SEC-bench-Full	$1.5/task	1.79	**6.18**	11.5
DrillAgent	SEC-bench-60	$1.5/task	1.93	**7.72**	11.8
OpenHands	SEC-bench-60	$1.5/task	1.53	15.30	3.2
OpenHands	SEC-bench-60	unlimited	6.04	20.13	7.7

*Cost / Success is computed as the total monetary cost divided by the number of validated PoVs. And the three above metrics are averaged over all tasks.

Phase	Avg. Time (min)	Input Tokens	Output Tokens
Vulnerability Analysis	2.5	*228,145*	*8,147*
Code Instrumentation	1.9	-*	-*
Path Exploration	3.7	*195,908*	*12,748*
Crash Triggering	3.3	*171,438*	*11,584*

*Token usage during the code instrumentation phase is omitted, as it involves few LLM interactions and no tool calls.

算法总结 DrillAgent

- 算法流程
  - 预处理：漏洞分析和自动化代码插桩
  - 路径探索：基于Trace-to-Prompt机制，有效引导生成高质量的种子
  - 崩溃触发：采用漏洞引导策略和验证反馈的迭代机制，生成有效PoV
- 算法优势
  - 提出“假设-验证-细化”的多智能体框架
  - 执行反馈语义化
- 算法不足
  - 主要验证C/C++程序的内存破坏类漏洞，对于其他语言或不产生崩溃的非内存逻辑漏洞验证没有进行验证

Patch-to-PoC: A Systematic Study of Agentic LLM Systems for Linux Kernel N-Day Reproduction

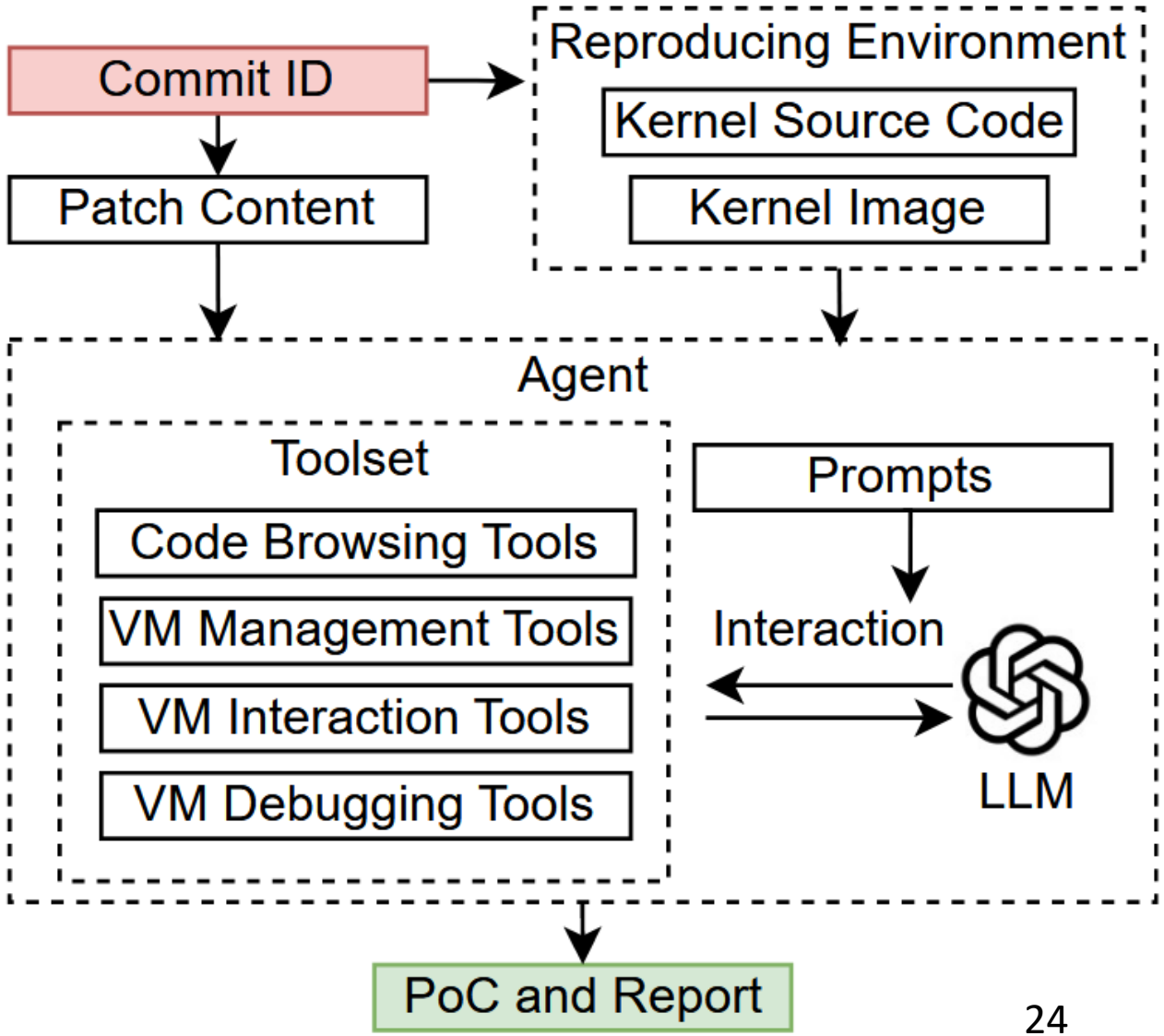
K-REPRO TIPO

T	目标	Linux内核N-day漏洞自动化重现
I	输入	安全补丁*1、内核源代码
P	处理	1. 补丁分析：Agent基于补丁进行根因识别，并配置复现环境 2. 代码搜索和分析：Agent利用工具在内核源码寻找所需定义、系统调用入口或相关函数逻辑等 3. 迭代式PoC生成：执行C程序，基于反馈验证和调试，循环修复PoC
O	输出	有效PoC*1、调试日志和报告*1

P	问题	1. Linux内核代码量大且复杂 2. 缺少反馈闭环
C	条件	具备稳定的内核漏洞复现环境和工具；依赖强推理模型
D	难点	1. 内核源码庞大且复杂，精准定位触发漏洞的系统调用序列困难 2. 调试反馈的语义理解难度远高于用户态程序
L	水平	arXiv 2026

K-REPRO 算法原理图

- 思路分析
  - 补丁分析
    - 基于Patch根因分析，判断漏洞类型、漏洞位置
  - 自动化环境构建
  - 受控代码导航与搜索
    - 自主设计浏览工具按需检查内核源码
  - 迭代式PoC生成
    - 模拟人类专家分析过程，Agent完成PoC生成、上传运行、收集反馈、调试修复循环



算法原理 Toolset & Prompt

- **Toolset**
  - **代码浏览工具**
    - **ls_dir(path)、grep_code(pattern, path)、cat_file(path, start_line, end_line)、find_definition(symbol)等**
  - **VM 管理工具**
    - **启动VM、重启VM、编译源码并上传**
  - **VM 交互工具**
    - **执行shell命令、获取输出、发送控制信号**
  - **VM 调试工具**
    - **检查内核寄存器和内存、设置管理断点、执行GDB命令**
- **Prompt**
  - **High-level Guidance： Agent行为上约束**
  - **Technical Guidance： 为四类工具提供具体的操作规范和调用示例**

实验设计　数据资源

- 实验配置
  - 测试基准：KernelCTF Dataset，筛选100真实世界可利用漏洞
  - 模型：GPT-5.1 Codex Max(Xhigh推理级别)、 GPT-5.1 Codex(medium推理级别)
  - 环境与工具
    - 采用模糊测试集群Syzbot的标准配置
    - 采用GCC进行内核构建
    - 基于QEMU/KVM的虚拟机环境
  - 任务约束
    - 不考虑成本开销、生成的PoC触发崩溃就是为重现成功
- 对比方法
  - SyzDirect (2023)：linux内核重现领域，SOTA非LLM定向模糊测试方法

实验设计　对比实验

- **RQ1：K-REPRO有效性**
  - **采用SyzDirect中的数据集，过滤无法成功构建环境的案例，剩下85**

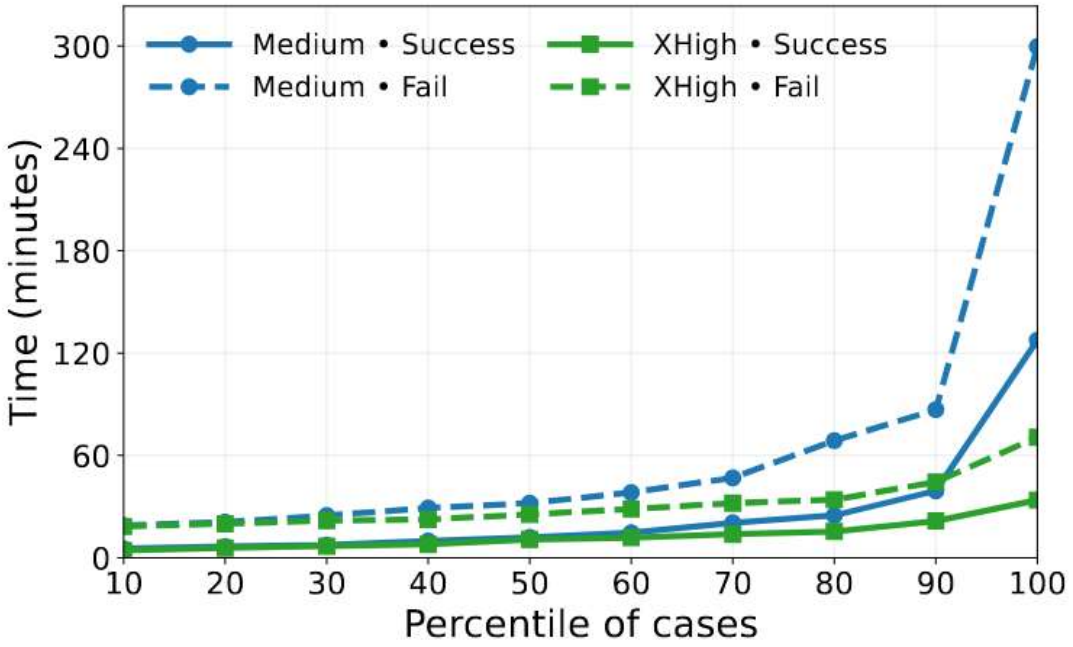
Method	Success Rate	Avg. Time (Success)	Avg. Time (Overall)
K-REPRO (XHigh)	57/85 (67.1%)	17.04 min	22.48 min
SyzDirect	42/100 (42.0%)	11.74 h	18.85 h

全部平均成本 2.19 $/task
成功平均成本 1.58 $/task

- **实验结论**
  - **K-REPRO的成功率远胜于SyzDirect，并保证可控的成本开销**

实验设计　对比实验

- **RQ2：模型性能差异**
  - **Medium： GPT-5.1 Codex；Xhigh： GPT-5.1 Codex Max**



Model	Result	counts	Avg. Cost (USD)	Avg. Time (min)
Medium	all	100	4.66	33.80
Medium	success	47	2.57	18.67
Medium	fail	53	6.52	47.22
XHigh	all	100	4.02	19.33
XHigh	success	56	2.49	11.89
XHigh	fail	44	5.96	28.81

- **实验结论**
  - **高性能不代表高消耗，Xhigh平均耗时和花费都低于Medium**
  - **复杂任务中，高性能模型相对稳定、及时止损能力更强**

实验设计　对比实验

- **RQ3：影响PoV生成的因素?**

Subsystem	Model	Success Rate	OR (p)
net	Medium	24/50 (48.0%)	1.08 (1.000)
	XHigh	28/50 (56.0%)	1.00 (1.000)
netfilter	Medium	14/34 (41.2%)	0.70 (0.526)
	XHigh	18/34 (52.9%)	0.83 (0.676)
bpf	Medium	4/7 (57.1%)	1.55 (0.703)
	XHigh	4/7 (57.1%)	1.05 (1.000)
others	Medium	5/9 (55.6%)	1.46 (0.731)
	XHigh	6/9 (66.7%)	1.64 (0.727)

漏洞发生的子系统?

Vulnerability Type	Model	Success Rate
Race	Medium	5/24 (20.8%)
漏洞类型?	XHigh	7/24 (29.2%)
Non-race	Medium	42/76 (55.3%)
	XHigh	49/76 (64.5%)

Cutoff Group	Model	Success Rate
Pre-cutoff	Medium	28/58 (48.3%)
模型知识?	XHigh	34/58 (58.6%)
Post-cutoff	Medium	19/42 (45.2%)
	XHigh	22/42 (52.4%)

- 实验结论
  - **K-REPRO具有一定的普适性和泛化能力，但对于竞态条件漏洞处理较差**

算法总结 K-REPRO

- 算法流程
  - 补丁分析与环境构建：基于补丁信息完成根因分析和复现环境初始化
  - 代码浏览与交互：通过API化的工具集按需检索内核源码、交互、调试等，进行静态分析和动态测试
  - 迭代式PoC生成：执行C程序，基于反馈验证和调试，循环修复PoC
- 算法优势
  - K-REPRO在真实世界KernelCTF数据集上超过50%的复现成功率并控制成本
  - K-REPRO泛化能力强，在不同子系统或模型知识截止日期上，性能无明显差异
- 算法不足
  - 竞态条件瓶颈：对精准的并发漏洞复现效果较差
  - 依赖补丁质量：当补丁信息缺失或不准确的时候，容易产生错误的根因分析，导致后续失败

特点总结与未来展望

特点总结与未来展望

- **DRILLAGENT**
  - 主要针对通用C/C++项目
  - 设计一套端到端的、多智能体、自动化POV生成框架
  - 通过Trace-to-Prompt弥合执行态与模型推理之间的语义差距
- **K-REPRO**
  - 主要针对linux内核
  - 首次设计一套内核研究的智能体系统
- 未来发展
  - 增强智能体对动态并发状态的控制能力，帮助更好处理竞争类型漏洞
  - 针对低质量补丁，引入额外分析，减少模型在初始RCA阶段错误

参考文献

[1] Li H, Che X, Wang Y, et al. Execution-State-Aware LLM Reasoning for Automated Proof-of-Vulnerability Generation[J]. arXiv preprint arXiv:2602.13574, 2026.

[2] Pu J, Li X, Li H, et al. Patch-to-PoC: A Systematic Study of Agentic LLM Systems for Linux Kernel N-Day Reproduction[J]. arXiv preprint arXiv:2602.07287, 2026.

知人者智，自知者明。胜人者有力，自胜者强。知足者富。强行者有志。不失其所者久。死而不亡者，寿。

谢谢！