華中科技大學
网络空间安全学院
School of Cyber Science and Engineering, HUST

CHINA LINUX KERNEL
中国Linux内核开发者大会

vivo

PORTGPT：基于大语言模型的自动化后向移植研究

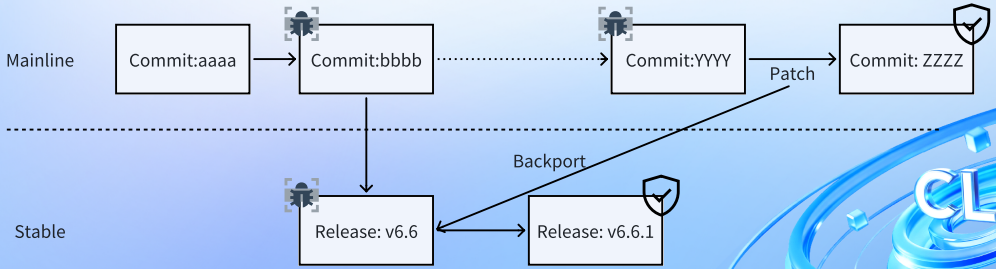
李朝阳

Open Source Operating System Optimal Security Lab (OS^3 Lab)

November 1, 2025

研究动机

- 开源项目（如 Linux Kernel、Node.js、Kubernetes）长期维护多个分支（mainline、stable、LTS）。
- 安全漏洞修复通常先在主线完成，随后需要“向后移植”（Backport）到旧版本。
- 现有维护模式依赖专家手动维护，耗时。
- 自动化研究**依赖语法/语义规则**，难以处理复杂结构变化。
- 该研究已被 **IEEE Symposium on Security and Privacy 2026 (S&P 26')** 接收。



问题定义

补丁后向迁移（Backporting）定义：
将主线版本中的补丁 P_n 迁移到旧版本 P_o，保持漏洞修复与原本功能的正确性。

核心挑战：

1. **定位（Localization）**：确定旧版本中对应的修改位置；
2. **变换（Transformation）**：调整补丁以适配旧版本上下文；
3. 二者都受到版本差异、符号变更、结构调整等因素影响。

传统方法的局限

- **文本匹配：**依赖上下文完全一致 (patch utility)；
- **语法匹配：**使用 AST（如 FIXMORPH），但难以处理语义变化；
- **语义模板：**使用 PDG 试图实现语义（如 TSBPORT），但扩展性有限，且仅部分定位；
- **基于 LLM 的方法：**通过微调和 in-context learning，只针对转化阶段 (PPathF,Mystique)；
- **核心问题：**缺乏灵活推理能力，依赖现有模式规则。

为什么使用大语言模型

- LLM 具备强大的代码理解与生成能力；
- 具备隐式上下文匹配与代码转化潜力；
- 可结合外部工具（Git、编译器）实现归因推理；
- **想法：**是否可以让大模型模拟专家工作模式实现 Backporting?

启发性示例：CVE-2022-32250

- 漏洞类型：Use-After-Free。
- 人类开发者通常：
  1. 查找函数定义与上下文；
  2. 分析 Git 历史追踪函数迁移；
  3. 手动调整变量名、修复编译错误；
  4. 最后运行测试验证。
- **启发：** Backporting 需要动态检索、追踪历史与基于反馈的修正。

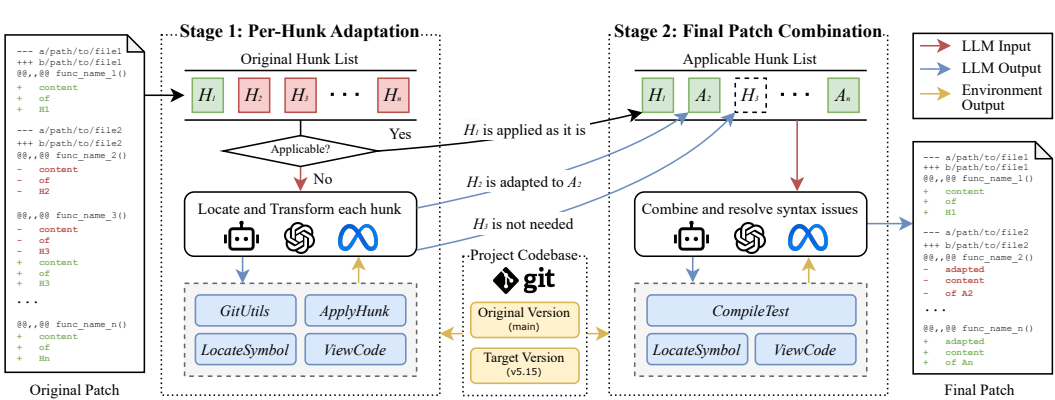
```
--- a/net/netfilter/nf_tables_api.c
+++ b/net/netfilter/nf_tables_api.c
@@ -2873,27 +2873,31 @@ *nft_expr_init(
  err = nf_tables_expr_parse(ctx, nla, ...);
  if (err < 0) goto err1;

+ err = -EOPNOTSUPP;
+ if (!(expr_info.ops...flags & NFT_EXPR_STATE))
+    goto err_expr_stateful;

  err = -ENOMEM;
  expr = kzalloc(expr_info.ops->size, ...);
@@ -5413,9 +5417,6 @@ *nft_set_elem_expr_alloc(
    return expr;

  err = -EOPNOTSUPP;
- if(!(expr->ops->type->flags & NFT_EXPR_STATE))
-    goto err_set_elem_expr;
-
  if (expr->ops->type->flags & NFT_EXPR_GC) {
    if (set->flags & NFT_SET_TIMEOUT)
      goto err_set_elem_expr;
```

Listing 8: Original Patch for CVE-2022-32250

关键观察

- 无法一次性提供全部信息，需按需查询；
- 信息应简洁、可由模型自行调用；
- 需要试错式（trial-and-error）过程；（验证与反馈）
- 因此——**应采用 Agent 式设计，而非单轮提示。**
- **关键思想：**将 LLM 设计为具备“工具调用”和“自我反馈”的智能体。

系统架构概览



- 阶段一：每个补丁块（hunk）的定位与转换；
- 阶段二：补丁组合、编译与自我修正；

设计理念

- 模拟人工迁移行为；
- 为模型提供“工具”与“推理链”：
  - **代码访问：** ViewCode；
  - **符号定位：** LocateSymbol；
  - **历史追踪：** GitUtils；
  - **补丁应用与修复：** ApplyHunk；
  - **编译验证：** CompileTest。
- 构成端到端自动化迁移流程。

ApplyHunk 机制流程

- 协助大模型处理补丁格式并反馈失败原因；
  1. 补丁格式错误：行号，‘+‘，‘-‘
  2. 上下文错误：编辑距离，反馈相似代码块
  3. 文件不存在：符号、相似文件名定位
- 连续出错后强制匹配相似代码块替换；
- 生成反馈并驱动自修复循环；
- 是系统中实现“验证”的核心组件之一。

```
--- a/foo.c
+++ b/foo.c
@@ -11,7 +11,9 @@
 }}

 int check (char *string) {{
+    if (string == NULL) {{
+        return 0;
+    }}
-    return !strcmp(string, "hello");
+    return !strcmp(string, "hello world");
 }}
 int main() {{
```

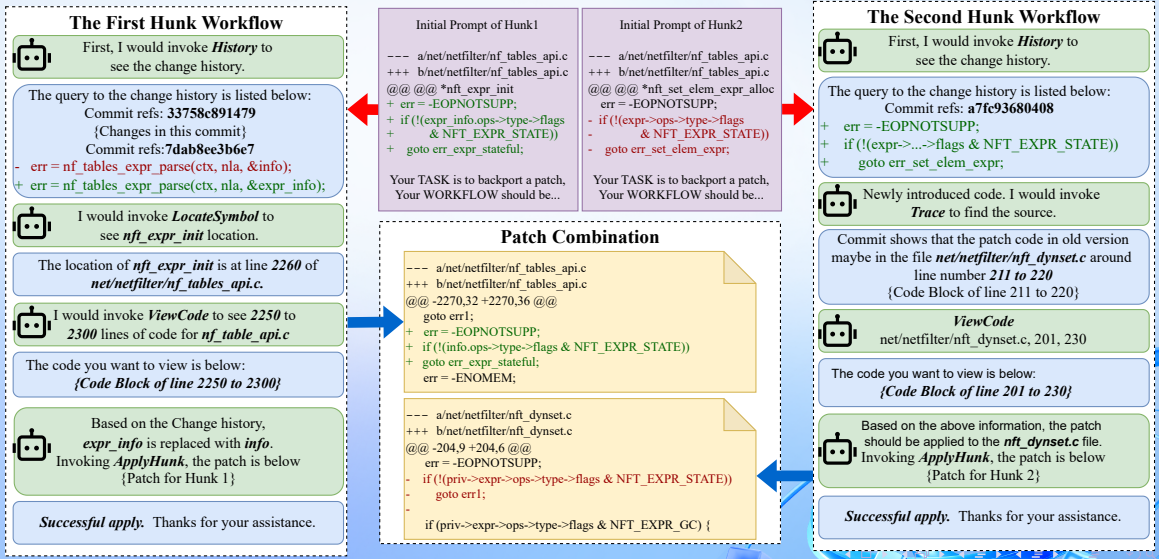
GitUtils 设计

- 利用 Git 辅助大模型分析代码演进历史。
- 包含两个协同工作的组件：
  - **History 组件：追踪局部演变**
    - **功能**：展示代码片段从“分叉点”（fork point）到新版本的所有相关提交历史。
    - **设计**：仅显示影响当前代码块（hunk）内代码行的提交。
  - **Trace 组件：识别全局变更**
    - **功能**：专门用于识别并追踪关键的全局性代码变更，特别是代码块的重定位（移动）。
    - **设计**：利用最小编辑距离算法，在 commit 的代码修改中检查是否存在相似的代码在不同文件中增删。
- 两者结合提供完整修改视角，帮助处理因代码重构或迁移导致的复杂情况。

阶段一：逐块适配

- 判断补丁块是否需要回传；
- 定位对应位置（符号匹配、历史追踪）；
- 获取相关代码片段（ViewCode）；
- 转换补丁并应用（ApplyHunk）；
- 若失败则触发自动上下文修正机制。

阶段一：逐块适配


The First Hunk Workflow
First, I would invoke History to see the change history.
The query to the change history is listed below:
Commit refs: 33758c891479
{Changes in this commit}
Commit refs:7dab8ec3b6c7
- err = nf_tables_expr_parse(ctx, nla, &info);
+ err = nf_tables_expr_parse(ctx, nla, &expr_info);
I would invoke LocateSymbol to see nft_expr_init location.
The location of nft_expr_init is at line 2260 of net/netfilter/nf_tables_api.c.
I would invoke ViewCode to see 2250 to 2300 lines of code for nf_table_api.c
The code you want to view is below:
{Code Block of line 2250 to 2300}
Based on the Change history, expr_info is replaced with info. Invoking ApplyHunk, the patch is below
{Patch for Hunk 1}
Successful apply. Thanks for your assistance.
Initial Prompt of Hunk1
--- a/net/netfilter/nf_tables_api.c
+++ b/net/netfilter/nf_tables_api.c
@@ @@ *nft_expr_init
+ err = -EOPNOTSUPP;
+ if (!(expr_info.ops->type->flags
+ & NFT_EXPR_STATE))
+ goto err_expr_stateful;
Your TASK is to backport a patch, Your WORKFLOW should be...
Initial Prompt of Hunk2
--- a/net/netfilter/nf_tables_api.c
+++ b/net/netfilter/nf_tables_api.c
@@ @@ *nft_set_elem_expr_alloc
err = -EOPNOTSUPP;
- if (!(expr->ops->type->flags
- & NFT_EXPR_STATE))
- goto err_set_elem_expr;
Your TASK is to backport a patch, Your WORKFLOW should be...
Patch Combination
--- a/net/netfilter/nf_tables_api.c
+++ b/net/netfilter/nf_tables_api.c
@@ -2270,32 +2270,36 @@
goto err1;
+ err = -EOPNOTSUPP;
+ if (!(info.ops->type->flags & NFT_EXPR_STATE))
+ goto err_expr_stateful;
err = -ENOMEM;
--- a/net/netfilter/nft_dynset.c
+++ b/net/netfilter/nft_dynset.c
@@ -204,9 +204,6 @@
err = -EOPNOTSUPP;
- if (!(priv->expr->ops->type->flags & NFT_EXPR_STATE))
- goto err1;
-
if (priv->expr->ops->type->flags & NFT_EXPR_GC) {
The Second Hunk Workflow
First, I would invoke History to see the change history.
The query to the change history is listed below:
Commit refs: a7fc93680408
+ err = -EOPNOTSUPP;
+ if (!(expr->...->flags & NFT_EXPR_STATE))
+ goto err_set_elem_expr;
Newly introduced code. I would invoke Trace to find the source.
Commit shows that the patch code in old version maybe in the file net/netfilter/nft_dynset.c around line number 211 to 220
{Code Block of line 211 to 220}
ViewCode
net/netfilter/nft_dynset.c, 201, 230
The code you want to view is below:
{Code Block of line 201 to 230}
Based on the above information, the patch should be applied to the nft_dynset.c file. Invoking ApplyHunk, the patch is below
{Patch for Hunk 2}
Successful apply. Thanks for your assistance.

阶段二：补丁组合与验证

- 将所有补丁块合并；
- 应用后执行编译；
- 若失败：
  1. 定位错误符号；
  2. 调用 ViewCode/LocateSymbol；
  3. 修复不兼容项；
- 输出可成功编译的最终补丁。

案例分析：CVE-2023-52752

```
--- a/fs/cifs/cifs_debug.c
+++ b/fs/cifs/cifs_debug.c
@@ -332,6 +332,11 @@
 struct cifs_ses *ses;
 list_for_each(...) {
+    spin_lock(&ses->ses_lock);
+        if (ses->ses_status == CifsExiting) {
+                spin_unlock(&ses->ses_lock);
+                continue;
+        }
         if ((ses->serverDomain == NULL) ||
```

(a) Patch Generated by Stage-1

```
--- a/fs/cifs/cifs_debug.c
+++ b/fs/cifs/cifs_debug.c
@@ -332,6 +332,11 @@
 struct cifs_ses *ses;
 list_for_each(...) {
+    spin_lock(&GlobalMid_Lock);
+        if (ses->status == CifsExiting) {
+                spin_unlock(&GlobalMid_Lock);
+                continue;
+        }
         if ((ses->serverDomain == NULL) ||
```

(b) Patch Generated by Stage-2

Figure 3: Patches Generated for CVE-2023-52752.

- 初始补丁因结构体字段差异导致编译错误；
- PORTGPT 自动：
  1. 定位结构体定义；
  2. 查找等价字段；
  3. 修改补丁、重新编译；
- 最终生成可合入版本。

提示词设计

Stage-1 Prompt

Your **TASK** is to backport a patch fixing a vuln from a original version of the software to an target version step by step. The project is {project_url}. For the ref {original_patch_parent}, the patch below is merged to fix a security issue:
{original_patch}
I want to backport it to ref {target_patch_parent}. To assist you in reviewing the relevant code, I have provided the following code blocks from the target version that closely match the current patch location:
{similar_block}
Your **WORKFLOW** should be: .....

Stage-2 Prompt

Your **TASK** is to validate and revise the patch until it is successfully backported to the target version and really fixes the vulnerability.
Below is the patch you need to backport:
{new_patch}
According to the patch above, I have formed a patch that can be applied to the target version:
{complete_patch}
Now, I have tried to compiled the patched code, the result is: {compile_ret}
You can validate the patch with provided tool *validate*. There are some processes to validate if the patch can fix the vulnerability: .....
If the patch can not pass above validation, you need to revise the patch with the help of provided tools. The patch revision **WORKFLOW** should be: .....

- 阶段一与阶段二使用不同任务提示；
- 包含：
  - 任务描述；
  - 工具使用说明；
  - 完整工作流示例；
- 强调逐步执行，工具调用与调用原因；
- **核心思想：**完整工作流约束 Agent 任务执行。

实现细节

- 框架：LangChain 实现 LLM Agent；
- 符号解析：ctags 提供符号表；
- 上下文匹配：编辑距离识别最相似块；
- 验证链：结合 PoC、测试套件、定向模糊测试；

补丁类型

- **Type-I (无变化)**
  - 回溯后的补丁在代码和应用位置上与原始补丁完全相同。
  - 可以通过 ‘git cherry-pick‘ 等工具直接应用，无需任何修改。
- **Type-II (仅位置变化)**
  - 补丁代码本身无需转换。
  - 仅应用的位置发生了变化（例如行号或文件名不同）。
- **Type-III (语法变化)**
  - 需要对补丁进行语法层面的修改以兼容目标版本。
  - 常见修改包括函数名、变量名的重命名等。
- **Type-IV (逻辑与结构变化)**
  - 需要进行更深层次的修改，例如在补丁中增删代码行。
  - 最终补丁在语法上与原始补丁有显著差异，但保持其核心功能。

实验设置

- 选型：GPT-4o
- 数据集：
  - FIXMORPH、TSBPORT 共 1815 个补丁 (纯 Linux，覆盖 13 到 22 年)；
  - 自建数据集 146 个（C/C++/Go)，跨 34 个 projects，不包含 Type-I
- 评价维度：
  1. 性能；
  2. 效率；
  3. 消融；
  4. 实用性。

结果：性能比较

Dataset	System	Type-I	Type-II	Type-III	Type-IV	Total
Prior works	FIXMORPH	20/170 (11.67%)	374/1208 (30.96%)	23/92 (25.00%)	30/345 (8.70%)	447/1815 (24.63%)
	ChatGPT†	67/170 (39.41%)	451/1208 (37.30%)	16/92 (17.58%)	22/345 (6.38%)	556/1815 (30.63%)
	TSBPORT	170/170 (100.00%)	1190/1208 (98.51%)	69/92 (75.00%)	160/345 (46.38%)	1589/1815 (87.59%)
	TSBPORT*	162/170 (95.29%)	919/1208 (76.08%)	61/92 (66.30%)	150/345 (43.48%)	1292/1815 (71.18%)
	PORTGPT	170/170 (100.00%)	1186/1208 (98.18%)	74/92 (80.43%)	188/345 (54.49%)	1618/1815 (89.15%)
Ours (C)	FIXMORPH	N/A	0/6 (0.00%)	3/29 (10.34%)	0/34 (0.00%)	3/69 (4.34%)
	TSBPORT	N/A	5/6 (83.33%)	14/29 (48.28%)	5/34 (14.71%)	24/69 (34.78%)
	TSBPORT*	N/A	2/6 (33.33%)	14/29 (48.28%)	4/34 (11.76%)	20/69 (28.99%)
	PORTGPT	N/A	6/6 (100%)	21/29 (72.41%)	15/34 (44.12%)	42/69 (60.87%)
Ours (C++)	PORTGPT	N/A	N/A	10/11 (90.91%)	5/17 (29.41%)	15/28 (53.57%)
Ours (Go)	PORTGPT	N/A	13/13 (100%)	7/9 (77.78%)	14/27 (51.85%)	34/49 (69.39%)

通用性

- **模型通用性**：
  - PORTGPT 的工作流可以无感迁移至任意模型。
  - GPT 系列模型表现最佳，而 Llama 3.3 由于在工具调用能力上的限制，性能较差。

Model	Type-II	Type-III	Type-IV	Total
GPT-5	19/19	42/49	31/78	92/146
GPT-4o	19/19	38/49	34/78	91/146
Gemini-2.5-Flash	19/19	37/49	27/78	82/146
DeepSeek-v3	18/19	33/49	25/78	76/146
Llama-3.3	17/19	7/49	0/78	24/146

消融实验 & 效率

- 各工具协同提升 Agent 智能性
  - 移除 GitUtils：性能下降 10%；
  - 移除 LocateSymbol：定位准确率显著下降；
  - 无自动修复或编译验证时整体性能下降约 15%；
- 整体 Cost 与效率
  - 平均单次迁移过程时间开销：166s
  - 平均单次迁移 api 调用开销：0.19$

Real-World Applicability

- 研究选取了模型知识截止日期（2023 年 10 月）之后的新补丁进行测试。
- **场景一：从主线 (Mainline) 到长期支持版 (LTS)**
  - **任务**：18 个从 Linux 主线到 ‘6.1-stable‘ 版本未迁移的补丁。
  - **结果**：PORTGPT 成功处理了其中的 9 个（50%）。
  - **验证**：经团队验证后提交到社区，均被维护者合入。
  - **对比**：在相同任务下，TSBPORT 仅成功 2 个（11.1%）。
- **场景二：从长期支持版 (LTS) 到下游发行版**
  - **任务**：针对 Ubuntu 的不同版本（Jammy, Focal, Bionic），测试了与 10 个 CVE 相关的 16 对补丁。
  - **结果**：PORTGPT 成功完成了 16 个任务中的 10 个（62.5%）。

讨论：失败案例分析

FIXMORPH (基于语法)

- **范围受限**：不支持头文件，难以处理复杂补丁。
- **转换失败**：自身 AST 处理方式脆弱。

TSBPORT (基于语义模板)

- **模板局限**：仅能处理预定模式补丁。
- **符号处理错误**：难以正确迁移头文件中的符号。
- **缺乏验证**：无编译验证环节。

PORTGPT (基于 LLM Agent)

- **忽略块间依赖**
- **遗漏前置依赖**
- **补丁不完整**：旧版本存在多处漏洞点。
- **代码结构剧变**

结论与展望

- PORTGPT 模拟人类专家迁移行为，实现端到端自动化；
- 结合多工具与验证链，有效提高智能化迁移效果；
- 实际补丁已被社区接受，具实用价值，对 AI 与安全的探索；
- **未来工作：**
  - 上下文限制 vs 补丁分块；
  - 语言/项目定制 vs 可扩展性；
  - 高可靠性验证链；

Q&A