Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

源代码安全补丁存在性测试

博士研究生 段学明

2025年07月20日

问题回溯

- 总结反思
  - 基础知识可适当增加

- 相关内容
  - 2023.07.16 张钊：《代码变更表示学习及其应用研究》
  - 2023.10.07 赵智洋：《代码变更表示学习技术》
  - 2025.03.02 徐程柯：《二进制代码补丁存在性测试》
  - 2025.03.09 段学明：《源代码补丁正确性测试》

内容提要

- 预期收获
- 内容引入
- 内涵解析与研究目标
- 研究背景与研究意义
- 研究历史与现状
- 知识基础
- 算法原理
  - RepoSPD
- 特点总结与工作展望
- 参考文献

预期收获

- 预期收获
  - 1. 了解源代码安全补丁存在性测试的基本概念和研究方向
  - 2. 理解安全补丁存在性测试的必要性
  - 3. 了解源代码补丁正确性测试的前沿方法和未来发展

内容引入

- 开源项目的数量激增使其成为攻击者的重要目标
- 为应对不断出现的安全漏洞，开发者发布安全补丁以修复项目中的漏洞
- 安全补丁通常与普通功能补丁混杂在一起，需要从中识别安全补丁
- 正向紧迫性：
  - 多数厂商采取静默发布方式，并不告知漏洞修复的具体补丁版本
  - 在海量补丁中寻找关键安全补丁成为挑战
- 反向紧迫性
  - 攻击者可通过分析补丁的差异（diff）来推断漏洞信息，并在补丁部署尚未完成之前开发出攻击手段
  - 因此，迫切需要一种方法，能够在安全补丁被公开前识别它们，从而为防御方争取时间

内容引入 补丁

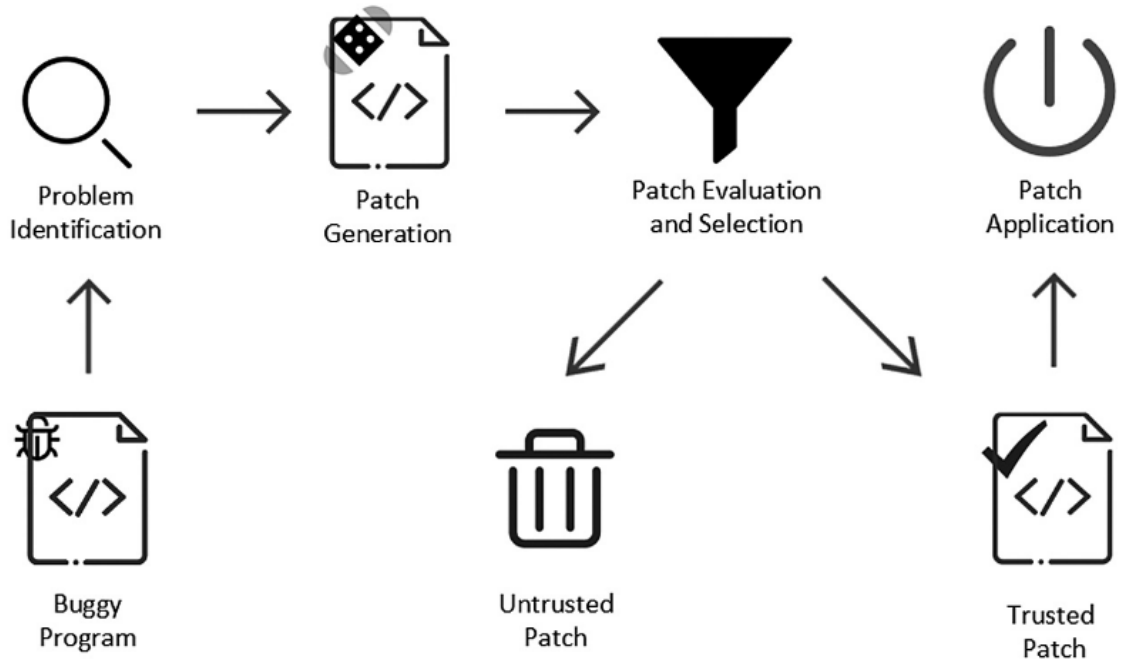
- 补丁（patch）
  - 定义：指对源代码进行修改以修复软件缺陷的代码片段，包括代码变更、修复描述
  - 代码变更：对源代码具体修改，如增加新代码行、删除错误代码行、修改现有代码
  - 安全补丁（Security Patch）
    - 修复缓冲区溢出漏洞
    - 关闭未授权访问的后门
  - Bug 修复补丁（Bug Fix Patch）
    - 解决软件崩溃问题
    - 修复内存泄漏
  - 功能增强补丁（Feature Patch）
    - 提高软件的运行效率
    - 增加新接口或API
    - 增强兼容性

```
@@ -421,7 +421,7 @@ public class TestFormAuthenticator
extends TomcatBaseTest {
-   private class FormAuthClientBase extends
SimpleHttpClient {
+   private abstract class FormAuthClientBase extends
SimpleHttpClient {
        protected static final String LOGIN_PARAM_TAG =
        "action=";
        protected static final String LOGIN_RESOURCE =
        "j_security_check";
```

Commit Message:
Make base class **abstract**

内涵解析与研究目标

- 研究目标
  - 检测当前补丁是否为安全补丁
- 内涵解析
  - 安全补丁：区别于Bug修复补丁和功能增强补丁，安全补丁主要修复危害性漏洞或关闭未授权访问的后门



背景意义

- 研究背景
  - 安全补丁是漏洞防御的关键手段
  - 攻击者可通过补丁反向推理漏洞，使得补丁本身成为一种“信息泄露”
  - 缺乏对安全补丁的自动识别手段
    - 当前大多数开源仓库不显式标记哪些补丁是安全相关的
- 研究意义
  - 构建自动化补丁识别机制是软件供应链安全的基础
    - 自动识别哪些补丁是“安全补丁”有助于及时响应风险、推动补丁应用、生成SBOM（软件物料清单）并实现补丁传播链的可追溯
  - 推动补丁级程序分析方法的发展
    - 安全补丁识别任务促进了“以变更为中心”的程序分析研究，推动传统静态分析、程序理解向补丁语义建模、变更上下文建模等方向演进
  - 提升防御响应速度，缩短攻击窗口

研究历史

基于规则

Wu等人认为补丁（提交或应用）和安全规则违规（例如，越界访问）的影响都可以建模为可以自动解决的约束，提出方法结合了规则比较，使用补丁的欠约束符号执行来确定未应用补丁的安全影响，方法可以根据漏洞的安全影响进一步自动对漏洞进行分类。

基于深度学习

Wang等人提出了一种基于深度学习的防御系统PatchRNN，用于自动识别OSS中的秘密安全补丁，除了在提交消息中考虑描述性关键字（即在文本级别）外，还利用了源代码级别的句法和语义特征。

基于深度学习

Zuo等人提出了一种基于Transformer的补丁分类器，它不使用任何代码更改作为输入，实验表明方法可以显著优于其他最先进的工作，精度高达93.0%，假阳性率低，进一步证实了精心设计的提交消息对于以后的软件维护至关重要。

2020 2020 2021 2023 2023 2023

Wang等人使用从国家漏洞数据库（NVD）收集的大规模数据集，按类型（即相应的漏洞类型）对安全补丁进行了实证研究。基于分析结果，开发了一个基于机器学习的系统，以帮助识别给定安全补丁的漏洞类型。

基于机器学习

Wang等人提出了一种基于图神经网络的安全补丁检测系统GraphSPD，该系统将补丁表示为语义更丰富的图，并利用补丁定制的图模型进行检测。首先开发了一种名为PatchCPG的新型图结构，通过合并补丁前和补丁后源代码的两个代码属性图（CPG）以及保留补丁的上下文、删除和添加的组件来表示软件补丁。

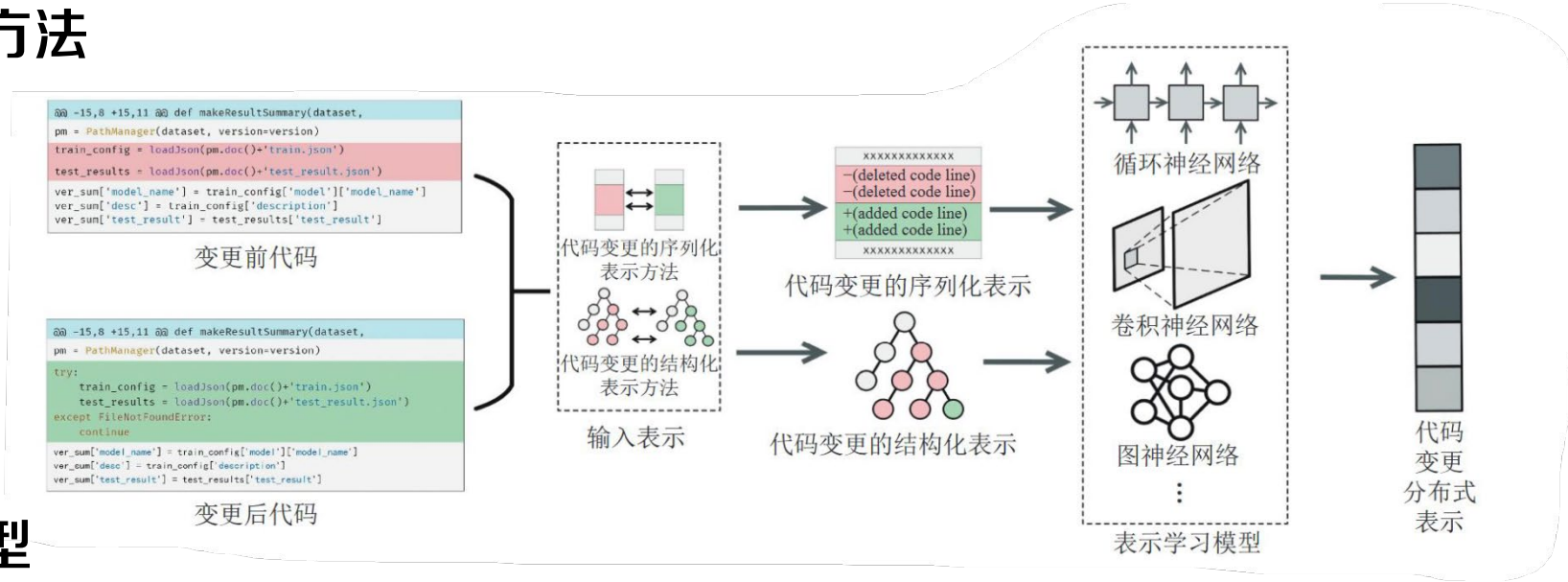
基于深度学习

Zhou等人提出一种基于Transformer的模型VulFixMiner，可自动识别开源项目中未公开披露的漏洞修复补丁，该方法通过学习提交级代码变更的语义，提升漏洞修复检测的效率与准确性。实验证明方法能在较少人工检查成本下发现大量真实漏洞修复，具备发现未披露漏洞的能力。

基于深度学习

知识基础 检测补丁存在性的一般方法

- 模型角度
  - 基于规则与静态分析的方法
  - 基于机器学习的方法
    - 提交日志内容
    - 代码变更位置
    - 添加/删除代码行数
    - 调用模式
  - 基于深度学习的序列模型
    - 使用代码与提交日志的序列输入
  - 基于图结构的表示学习方法
    - 能建模跨文件、多函数的复杂关系，但图构建开销大
- 建模角度：建模漏洞特征、建模代码特征、建模变更特征



知识基础 表示代码的一般方法

- 序列表示
  - 将源代码视为词或标记（Token）序列，常用于基于自然语言处理的模型，如RNN、Transformer
- 抽象语法树（AST, Abstract Syntax Tree）
- 控制流图（CFG, Control Flow Graph）
  - 表示程序执行流程中的控制结构（如条件分支、循环等）
- 数据流图（DFG, Data Flow Graph）
  - 描述变量在程序中的定义与使用关系
- 代码属性图（CPG, Code Property Graph）
  - 将AST、CFG和DFG三者统一为一个多视角图结构，适用于综合分析程序的语义与结构

Repository-Level Graph Representation Learning for Enhanced Security Patch Detection

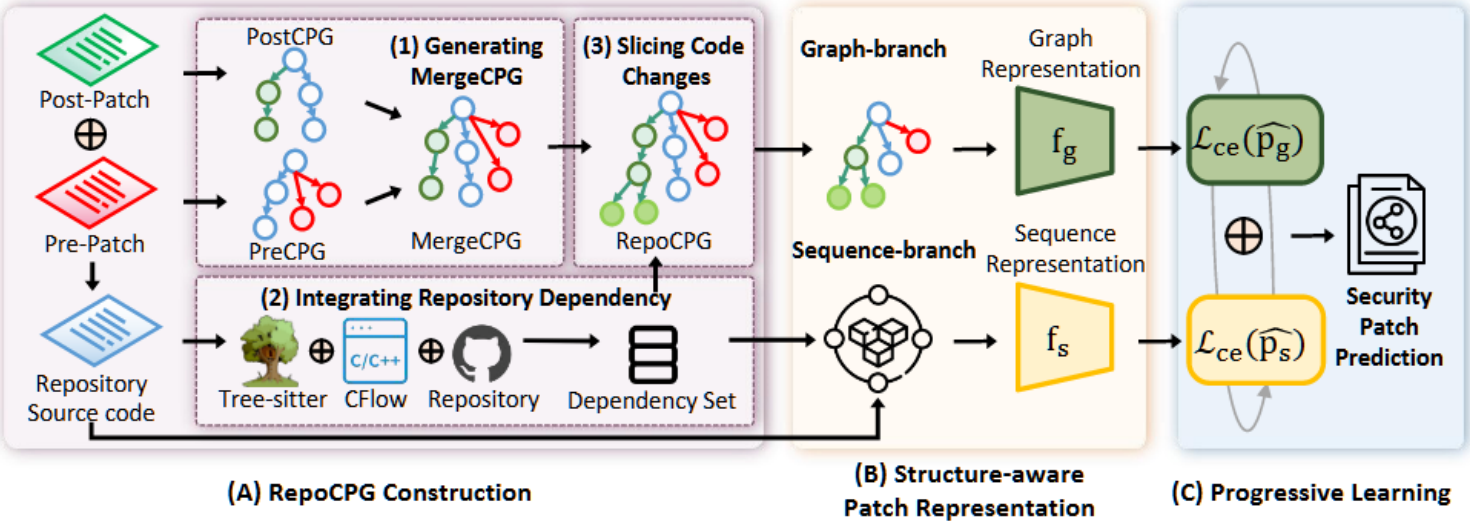
算法原理 TIPO

T	目标	检测安全补丁的存在性
I	输入	补丁、源代码仓库
P	处理	1. 提取仓库级别上下文 2. 结构感知的补丁表示 3. 渐进式学习
O	输出	当前提交补丁是否为安全补丁

P	问题	现有安全补丁检测方法在表征复杂补丁时存在较高的误报率和漏报率，无法充分利用仓库级别的上下文和跨文件、跨函数的依赖关系，影响了补丁检测的准确性
C	条件	补丁存在该版本提交下对应的完整仓库源代码
D	难点	如何有效结合代码仓库的上下文信息、补丁的语义与结构信息
L	水平	ICSE2025

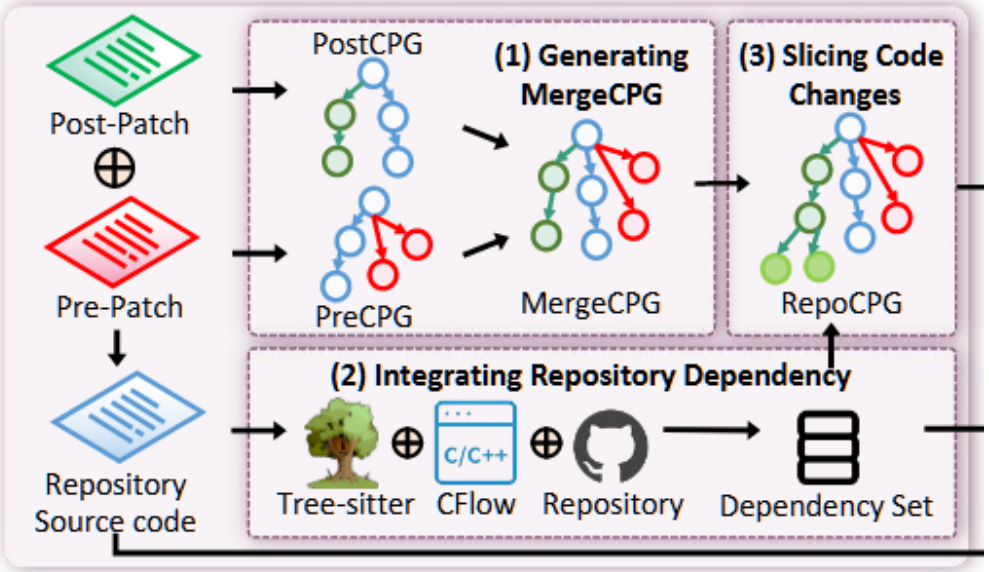
算法原理

- 算法原理图
  - RepoCPG构建
    - 提取仓库级别的全面上下文，并保留补丁内的代码更改语义
  - 结构感知的补丁表示
    - 结合图分支和序列分支，增强安全补丁的表示
  - 渐进式学习
    - 首先训练序列分支模型，再训练图分支模型，确保对不同数据表示的全面学习



算法原理　RepoCPG构建

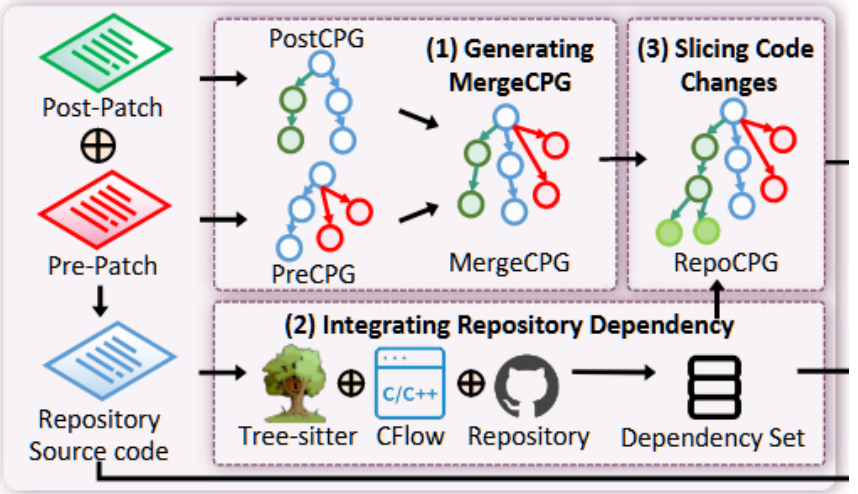
- 模块概述
  - 目标：提取仓库级别的全面上下文，并保留补丁内的代码更改语义
  - 输入：补丁前后的源代码、完整仓库
  - 操作：
    - 生成MergeCPG
      - 建立补丁前和补丁后的关系
    - 整合仓库级依赖
      - 将仓库级依赖整合到RepoCPG中，获取更多上下文信息
    - 仓库级代码切片
      - 只保留补丁相关依赖上下文的RepoCPG
  - 输出：仓库级代码属性图RepoCPG



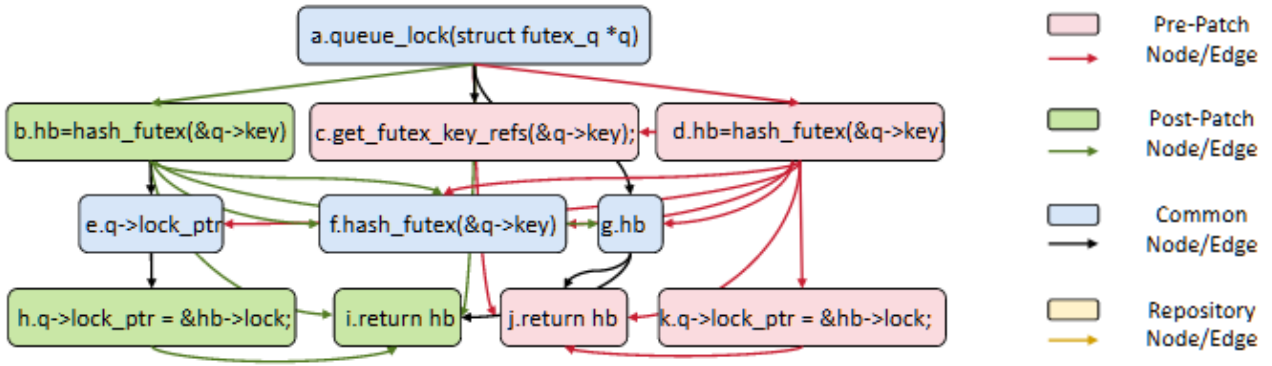
(A) RepoCPG Construction

算法原理　RepoCPG构建

- 生成MergeCPG
  - 动机：补丁只包含几行代码更改，当多个代码更改发生在同一个文件中时，它们往往缺乏结构性连接
  - 操作：
    - 借助Git命令找到补丁提交前后的仓库版本
    - 分别构建CPG，并整合
    - 整合过程中
      - 保留补丁前后版本中通用的节点
      - 合并补丁前后版本中独立存在的节点

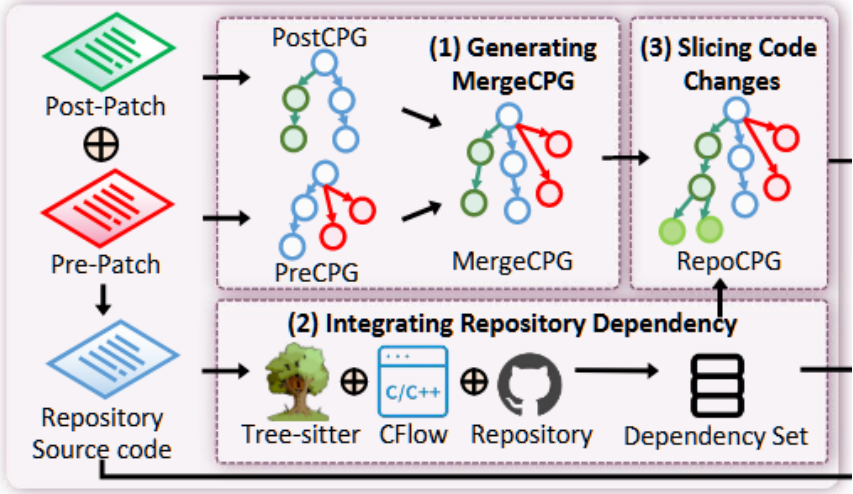


(A) RepoCPG Construction

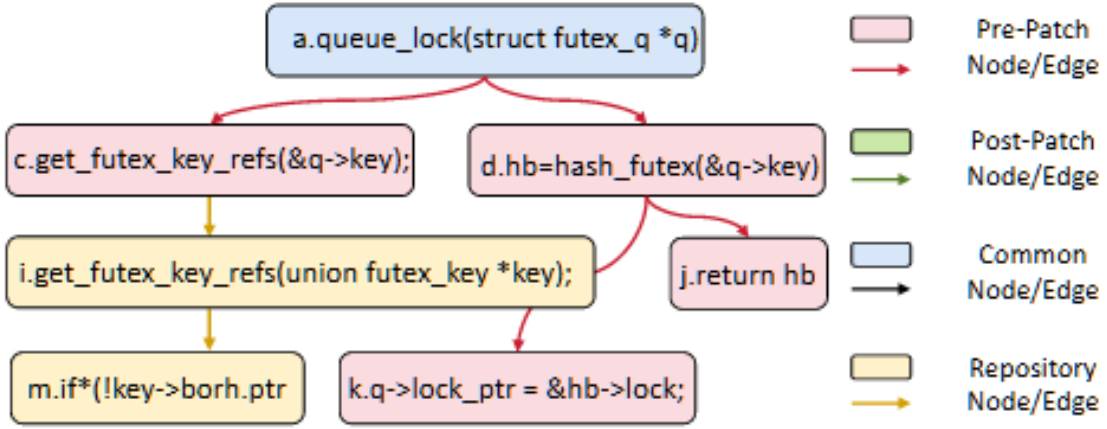


算法原理　整合仓库级依赖

- 整合仓库级依赖
  - 动机：获得补丁前后版本变化的更多上下文信息
  - 操作：
    - 定位仓库：借助Git命令找到补丁提交对应的仓库版本
    - 识别调用依赖：借助Cflow分析整个仓库，构建调用图
    - 从仓库中检索被调用函数定义：
      - 调用图中找到目标函数
      - 在整个仓库代码中查找该函数的源代码
        并提取其根节点（即函数体或函数入口）
    - 图中增加依赖信息
      - 从MergeCPG定位调用语句节点，
        添加一条边指向仓库中目标函数根节点

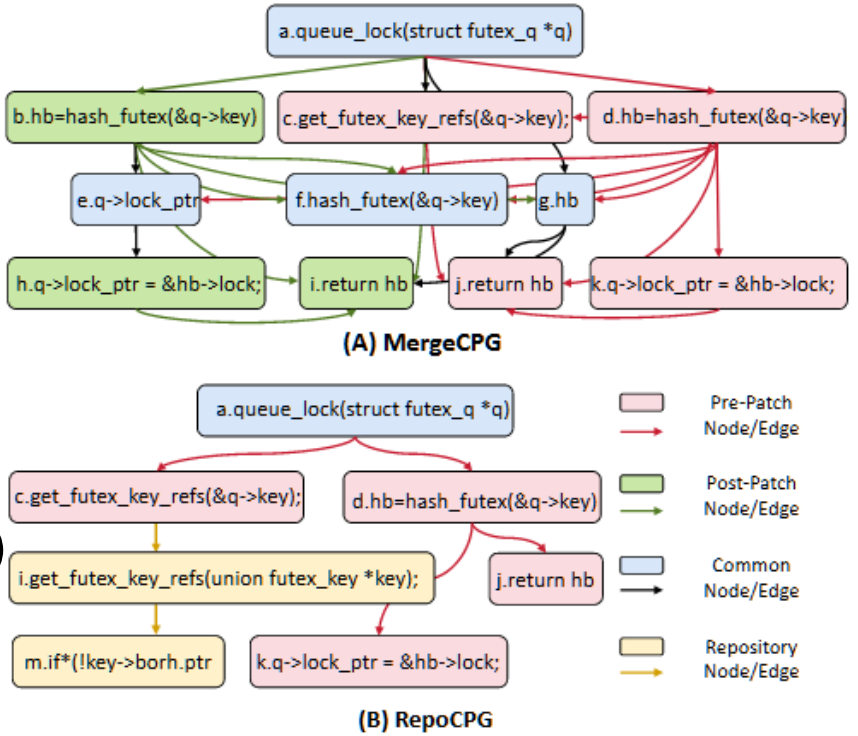


(A) RepoCPG Construction



算法原理　仓库级代码切片

- 仓库级代码切片
  - 动机：为了避免引入过多无关代码节点
    - 需要从MergeCPG出发，只保留那些与补丁变更语句存在直接依赖关系的仓库代码上下文
    - 包括：控制依赖、数据依赖、函数调用依赖
  - 操作：
    - 切片：识别MergeCPG中2类关键语句（删除、新增）
    - 对删除型切片：
      - 以被删除语句为起点，向上回溯依赖源头，保留所有对其产生直接影响的语句节点
    - 对新增型切片：
      - 以新增语句为起点，向前追踪其上下文条件或先决条件，保留所有该新增代码依赖的语句节点
    - 合并

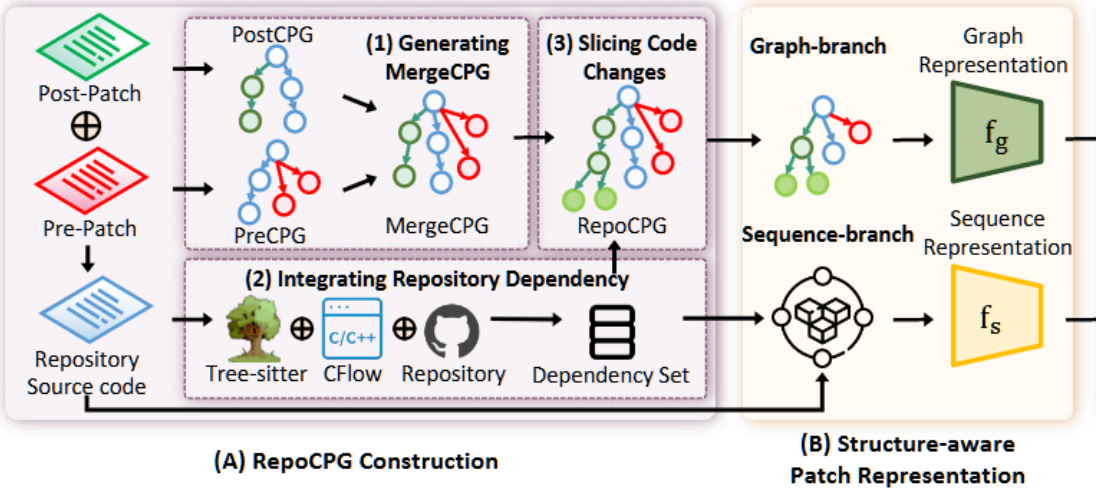


谁控制它的执行？（控制依赖）

谁为它提供了数据来源？（数据依赖）

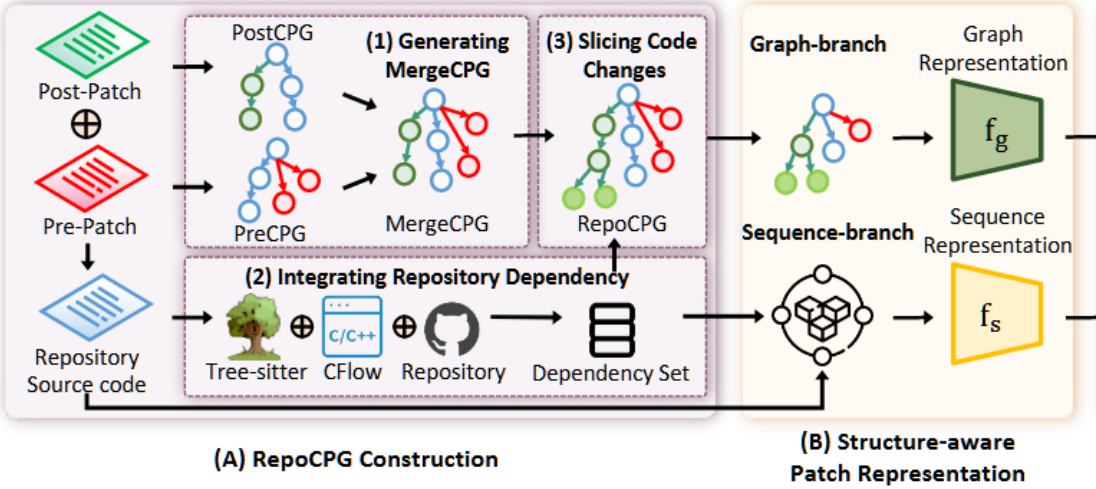
算法原理　结构感知的补丁表示

- 模块概述
  - 目标：将补丁范围内的代码变更及依赖结构转化为适合神经网络处理的表示形式
    - 如何捕捉补丁变更中涉及的跨文件调用、数据控制依赖等结构
    - 如何结合变更语句与仓库上下文，构造对模型友好的图与序列输入
    - 如何通过深层模型提取变更的表示，支持后续的分类或检测任务
  - 2条并行分支：图分支、序列分支
    - 图分支输入：RepoCPG
    - 序列分支输入：补丁中的源代码片段、仓库中提取出来的依赖函数源代码



算法原理　结构感知的补丁表示

- 图分支
  - 目标：从语义信息和结构信息中学习漏洞模式
  - 节点嵌入
    - 使用UniXcoder初始化向量的节点表示
    - $h_i^{(0)} = H^0(N_i) + H^L(N_i)$
  - 更新节点表示
    - 使用图注意力网络
    - 注意力计算：$\alpha_{ij}^{(l)}[k] = softmax(LeakyReLU(a^{(l)}[Wh_i^{(l-1)} || Wh_j^{(l-1)}]))$
      - $a^{(l)}$和W分别表示可学习向量和权重矩阵，k表示子图的索引
    - 节点嵌入计算：$h_i^{(l)}[k] = \sigma\left(\sum_{j \epsilon N(i)} \alpha_{ij}^{(l)}[k] W[k] h_j^{(l-1)}\right)$
      - $N(i)$和σ分别表示节点i的相邻节点集和激活函数
    - 图分支表示：池化层



算法原理　结构感知的补丁表示

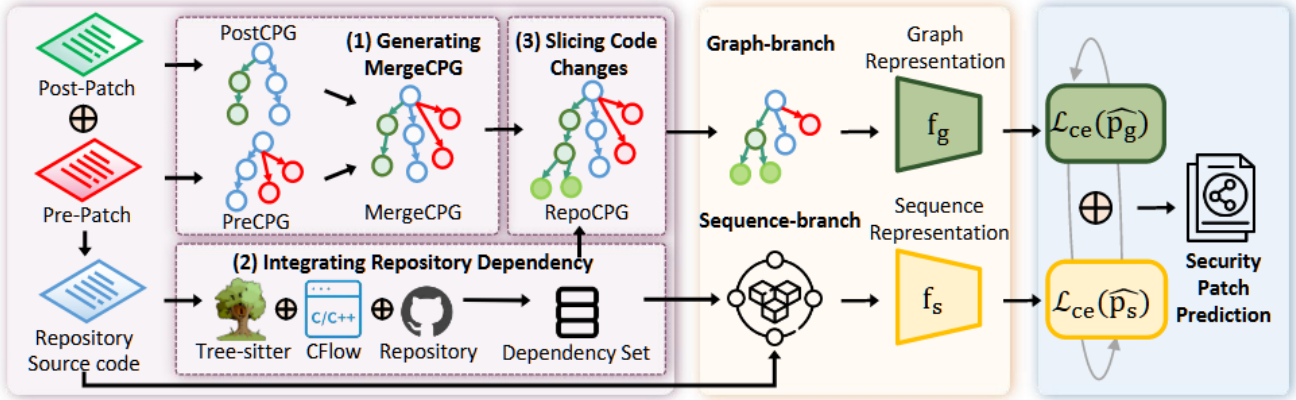
- 序列分支
  - 目标：分析补丁中代码的变更部分，特别是补丁中每一行代码的语义信息
  - 存在问题：典型的补丁经常包含大量无关内容，例如行号和差异标记（如“@@ string1 @@”），这些无关信息会在学习过程中干扰模型
  - 操作：
    - 保留重要信息：选择性地保留补丁代码中的实际变更行，并丢弃诸如行号、文件名和位置标识符等不重要的元素
    - 微调：对UniXcoder进行微调

```
@@ -1363,7 +1363,6 @@ static inline struct futex_hash_bucket
*queue_lock(struct futex_q *q)
 {
            struct futex_hash_bucket *hb;

-           get_futex_key_refs(&q->key);
            hb = hash_futex(&q->key);
            q->lock_ptr = &hb->lock;
```

```
@@ -1856,8 +1854,6 @@ static int futex_wait(u32 __user *uaddr,
int fshared,
            ret = -ERESTART_RESTARTBLOCK;

-out_put_key:
-           put_futex_key(fshared, &q.key);
 out:
            if (to) {
                        hrtimer_cancel(&to->timer)
```

算法原理　渐进式学习

- 渐进式学习
  - 存在问题：图分支和序列分支在输入特性和模式上存在差异，学习率和共享参数在各分支之间差异很大
  - 解决方案：



- 2个分支分别送入2个分类器，合并特征向量：$p = \frac{W_g^T f_g + W_s^T f_s}{2}$
- 首先冻结图分支，训练序列模型，优先使用预训练模型提供的领域知识
  - 序列分支可以较快地捕捉到补丁的语义信息
- 再训练更复杂的图结构
  - 侧重于捕捉代码的结构信息，尤其是代码行之间的依赖关系

实验设计

- 数据资源
  - 数据集：SPI-DB数据集（25,000个补丁，10,000个被标记为安全相关补丁）、PatchDB数据集（超过36,000个代码片段，约12,000个被标记为安全补丁）
  - 预处理：筛选出存在完整提交日志、补丁前后版本完整仓库的安全补丁
- 对比方法
  - 基于监督学习的方法：PatchRNN(2021)、GraphSPD(2023)
  - 基于预训练模型的方法： CodeBERT、CodeT5、 UniXcoder
  - 基于大模型的方法：Llama3-70b
  - 静态分析方法：Cppcheck、RATS、Semgrep、Flawfinder、VUDDY
    - 通过在补丁前后的代码中检测漏洞来识别安全补丁
  - 指标：Accuracy、F1-score、 FPR

实验设计　对比实验

- 实验结果

Dataset	Method	Accuracy↑	F1 Score↑	FP Rate↓
SPIDB*	GraphSPD	59.40	59.11	22.08
	PatchRNN	57.95	60.46	43.97
	CodeBERT	65.61	61.40	32.56
	CodeT5	66.62	61.87	30.31
	UniXCoder	66.27	60.26	28.27
	Llama3-70b	56.35	42.98	26.38
	RepoSPD	**74.55***	**68.98**	**14.67**
PatchDB*	GraphSPD	71.42	48.67	16.52
	PatchRNN	70.15	30.45	8.23
	CodeBERT	78.27	65.27	16.56
	CodeT5	80.84	58.52	9.28
	UniXCoder	80.70	64.34	8.68
	Llama3-70b	74.65	55.84	15.43
	RepoSPD	**83.35***	**69.13**	**6.65**

- 分析
  - 性能显著优于基于监督学习的方法
  - 预训练模型方法和大模型方法由于缺乏仓库级别的依赖关系和补丁的特定领域知识，存在性能瓶颈

实验设计　对比实验

- 实验结果

Method	#$Vul_{pre\text{-}patch}$	#$Vul_{post\text{-}patch}$	#Security Patch↑	Accuarcy(%)↑
Cppcheck	31	31	0	0.00
RATS	109	110	0	0.00
Semgrep	16	20	0	0.00
Flawfinder	148	150	5	2.60
VUDDY	131	59	110	57.29
RepoSPD	-	-	**151**	**78.65**

- 分析
  - 尽管VUDDY在漏洞检测方面具有较高的检测率，但它对补丁内容的理解较为有限，导致在处理补丁时，效果不如RepoSPD

实验设计 消融实验

- 消融实验

Dataset	Variant	Accuracy↑	F1 score↑	FP Rate↓
SPIDB*	w/o RepoCPG	64.75	57.66	24.82
	w/o sequence	69.60	62.33	17.99
	w/o graph	66.27	60.26	28.27
	w/o progressive	72.25	66.91	18.45
	w/o change order	67.45	62.65	25.92
	RepoSPD	74.55	68.98	14.67
PatchDB*	w/o RepoCPG	82.91	68.69	7.41
	w/o sequence	72.31	31.49	4.86
	w/o graph	80.70	64.34	8.68
	w/o progressive	80.94	65.55	9.38
	w/o change order	81.66	68.25	10.62
	RepoSPD	83.35	69.13	6.65

- 消融情况： 去除RepoCPG、去除序列分支、去除图分支、去除渐进学习时的三种指标的实验结果

特点总结与未来展望

算法总结

- 算法创新
  - 引入仓库级语义依赖，构建跨文件补丁上下文图
  - 结合结构图和代码序列的双分支表示机制
  - 设计渐进式学习机制，提升对真实漏洞补丁的检测能力
- 算法优劣
  - 优势：误报率低、速度快、成本低
  - 劣势：处理大型仓库和复杂补丁时，生成RepoCPG对计算资源要求高、性能在面对一些特定的漏洞类型时还有提升空间，如对于某些边界较模糊或上下文依赖性较弱的漏洞
- 未来工作
  - 优化计算效率；增强漏洞类型适应性；跨语言和跨平台适应性；增强模型解释性
  - 代码版本的变更表征

参考文献

[1] S. Wang, X. Wang, K. Sun, S. Jajodia, H. Wang and Q. Li. GraphSPD: Graph-Based Security Patch Detection with Enriched Code Semantics, 2023 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2023, pp. 2409-2426, doi: 10.1109/SP46215.2023.10179479.

[2] X. -C. Wen, Z. Lin, C. Gao, H. Zhang, Y. Wang and Q. Liao. Repository-Level Graph Representation Learning for Enhanced Security Patch Detection, 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE), Ottawa, ON, Canada, 2025, pp. 1-13, doi: 10.1109/ICSE55347.2025.00121.

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！