Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

第三方库检测技术研究

硕士研究生 徐程柯

2025年07月13日

问题回溯

- 总结反思
  - 讲解语速较快，时间较短，语气词较多
  - 内容详略不当，算法部分讲解深度不足、浅尝辄止
  - 互动部分较少
- 相关内容
  - **2024.10.08** 高玺凯 《二进制代码相似性检测技术》
  - **2023.07.27** 陆永鑫 《准确高效地检测安卓**APP**中的第三方库》

内容提要

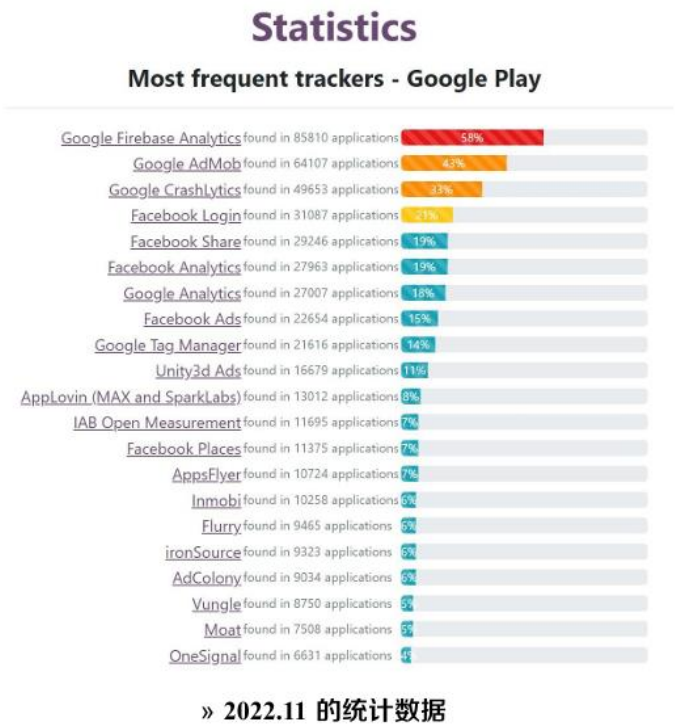
- 预期收获
- 内容引入
- 内涵解析与研究目标
- 研究历史
- 知识基础
- 算法原理
  - **LibAM**
- 特点总结与工作展望
- 参考文献

预期收获

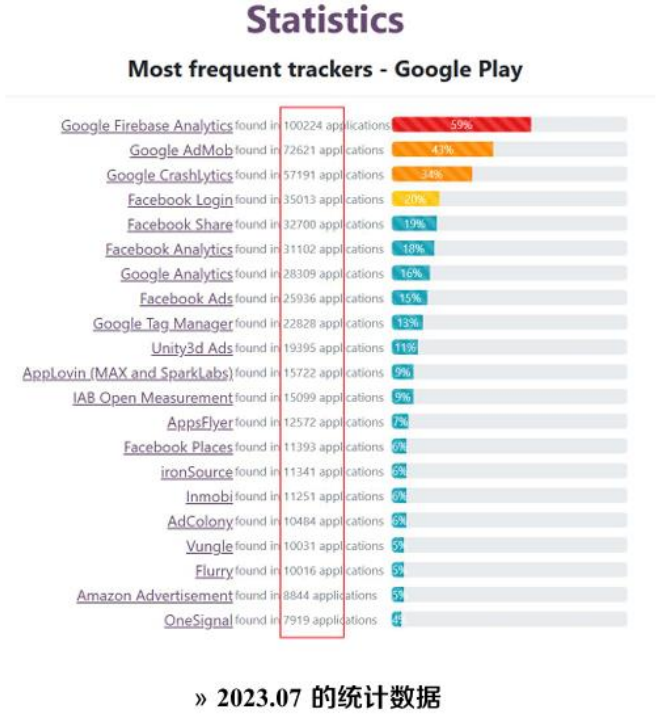
- 预期收获
  - 掌握第三方库检测的基本概念
  - 了解第三方库检测发展历程
  - 理解第三方库检测的基本原理
  - 明确第三方库检测的应用和前沿发展

内容引入

- 第三方库（**Third-Party Library，TPL**）
  - 由非应用开发者编写、打包的代码组件，可被直接集成到应用中
- 第三方库的普遍使用
  - **Exodus Privacy统计Google Play中大约 59% 的APP包含Google Firebase**
  - **APP中超过 60% 的代码属于TPL**



» 2022.11 的统计数据



» 2023.07 的统计数据

内容引入 安全隐患

- **TPL的广泛使用带来安全问题**
  - 供应链攻击风险：通过污染供应链，传播恶意代码
  - 已知漏洞传染风险：非安全审核库若被引入，可能危及整个应用的安全
  - 数据泄露风险：一些**TPL**具备数据收集、日志分析、用户追踪功能

	Version	Vulnerabilities	Repository	Usages	Date
1.16.x	1.16.1		Central	119	Apr 29, 2023
1.15.x	1.15.4		Central	173	Feb 18, 2023
	1.15.3		Central	417	Aug 24, 2022
	1.15.2	1 vulnerability	Central	97	Jul 04, 2022
	1.15.1	1 vulnerability	Central	102	May 15, 2022
1.14.x	1.14.3	1 vulnerability	Central	392	Sep 30, 2021
	1.14.2	1 vulnerability	Central	245	Aug 15, 2021
	1.14.1	2 vulnerabilities	Central	51	Jul 10, 2021
1.13.x	1.13.1	2 vulnerabilities	Central	575	Mar 01, 2020

Jsoup Java HTML Parser » 1.13.1

jsoup is a Java library for working with real-world HTML. It provides a very convenient API for fetching URLs and extracting and manipulating data, using the best of HTML5 DOM methods and CSS selectors. jsoup implements the WHATWG HTML5 specification, and parses HTML to the same DOM as modern browsers do.

License	MIT
Categories	HTML Parsers
Tags	html parser
Organization	Jonathan Hedley
HomePage	https://jsoup.org/
Date	Mar 01, 2020
Files	jar (384 KB) View All
Repositories	Central
Ranking	#133 in MvnRepository (See Top Artifacts) #1 in HTML Parsers
Used By	3,501 artifacts
Vulnerabilities	Direct vulnerabilities: CVE-2022-36033 CVE-2021-37714 Vulnerabilities from dependencies: CVE-2023-26049 CVE-2023-26048 CVE-2022-25647 CVE-2021-34428 CVE-2020-27223 CVE-2020-27218 CVE-2020-15250

内容引入 二进制 vs Android

- **DEX 字节码**
  - **DEX（Dalvik Executable）是 Android 应用的核心执行文件格式，保留了类、函数、包名、字段、控制流等结构化信息**
- 二进制机器码
  - **ELF 文件（Linux 程序、嵌入式固件）**
  - **PE 文件（Windows 可执行程序）**

特性	Android 第三方库检测	二进制第三方库检测
分析对象	Android APK 中的DEX 字节码	二进制文件（如 ELF、PE，汇编指令、机器码）
代码结构层次	类、包、函数、控制流	函数、指令、控制流
特征类型	类继承关系、函数原型、函数调用图、控制流图	指令序列、控制流图、符号特征、函数调用图
特征提取难度	字节码稳定，结构容易解析	需要反汇编、函数识别、符号恢复
抗混淆能力	抗包名/类名混淆强，控制流混淆可通过特征简化抵御	难以抵御混淆，混淆影响指令、控制流、函数识别

内涵解析与研究目标

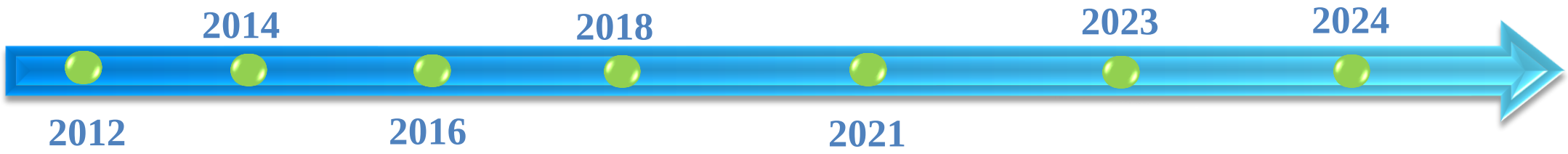
- 内涵解析
  - 第三方库检测（**Third-party Library Detection**）旨在自动识别应用程序或二进制程序中，所包含的第三方库
  - 对二进制文件或**APP**进行分析，主要用于安全性评估、软件合规性检查以及漏洞管理
- 研究目标
  - 支持软件应用的下游安全任务，如漏洞风险识别、供应链安全溯源等，利用自动化特征提取，高效识别第三方库，提升组件管理与风险治理能力
  - 优化第三方库的匹配机制，融合结构特征、调用关系等方法，增强在多平台、多编译环境下的库检测准确性与鲁棒性

研究历史 第三方库检测

Meng等人提出**LibRadar**方法，提出基于包特征的聚类检测方法，通过相似包聚类发现第三方库

Feng等人提出了**LIBPECKER**方法，提出自适应类相似度阈值机制，动态调整比对阈值；使用类内部依赖关系生成特征签名，提升细粒度区分能力

Xue等人提出了**FIRM**方法，融合文件结构特征、字符串特征、控制流特征，适合处理符号缺失、复杂架构环境的嵌入式固件库检测



Jiang 等人首个提出“基于函数常量特征”的二进制相似性检测方法,提取函数中的字符串常量、数值常量、控制流特征等简单特征

Backes等人提出**LibScout**，生成库的类结构签名，利用相似性进行匹配，可识别混淆后的库

Chen等人提出**ATVHunter**，首次提出两阶段控制流图匹配方法，粗匹配阶段：快速筛选相似控制流结构；精细匹配阶段：基本块操作码比对

Zhan等人提出**LibAM**，首次结合锚点函数对齐 + 区域结构相似性 + 函数相似度的三重约束方法，提升在跨优化、跨架构场景下的检测效果

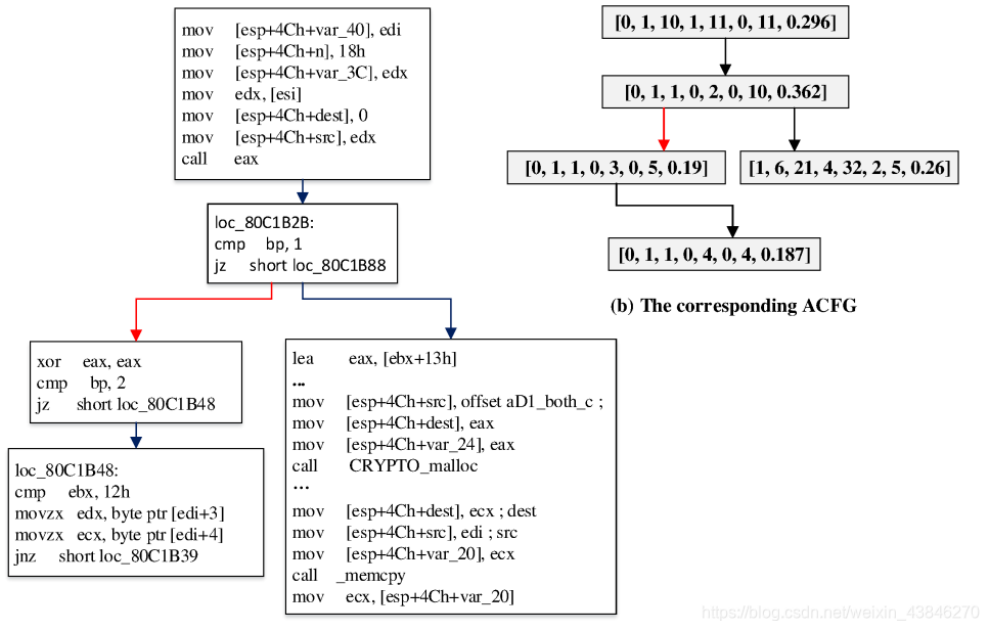
知识基础 混淆技术

- 混淆技术（**Obfuscation**）
  - 一种通过修改代码结构、控制流、名称等信息，使得程序难以被分析的技术
  - **Android** 混淆：主要破坏名字等语义信息，但控制流、字节码特征仍可部分提取
  - 二进制混淆：控制流、指令序列、函数边界都会改变

特性	Android 混淆	二进制混淆
对象	DEX 字节码（类、函数、变量、控制流）	二进制文件（汇编指令、控制流、机器码）
目的	防反编译、反逆向、节省体积	防破解、反逆向、隐藏功能
常见手段	命名混淆、类结构重构、控制流混淆、字符串加密、方法合并/拆分	指令替换、垃圾指令插入、控制流平坦化、寄存器重命名
抗分析策略	利用类关系、函数签名、控制流图可部分分析	符号执行、语义恢复、静态分析
典型工具	ProGuard、R8、DexGuard	Obfuscator-LLVM、VMProtect、Tigress

知识基础 带属性控制流图

- 带属性控制流图（**ACFG, Attributed Control Flow Graph**）
  - 在 **CFG** 的基础上，给每个节点加上属性特征（**Attributes**），让图结构更具可区分性

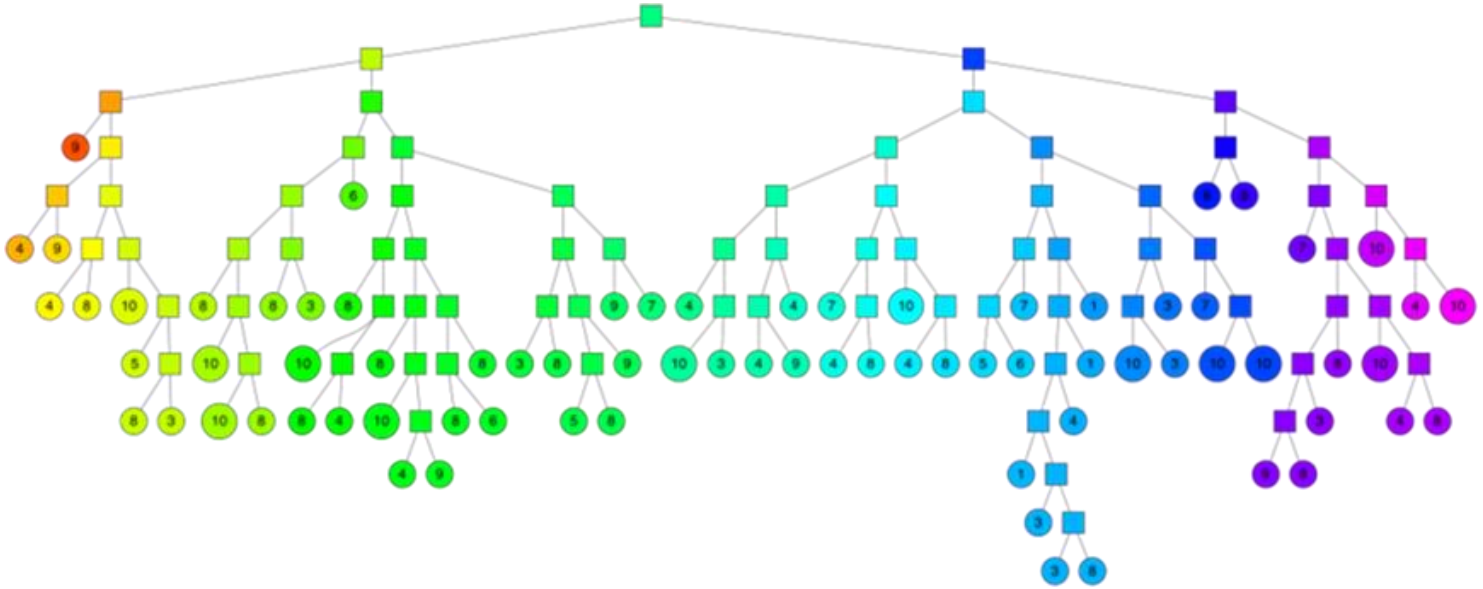


(b) The corresponding ACFG

属性特征	内涵
字符串常量	基本块中包含的字符串常量的数量
数值常量	基本块中包含的数值常量数量
传送指令数目	基本块中的跳转、分支、条件判断、返回等控制指令数量
调用指令数目	基本块中的函数调用指令数量
指令数目	基本块中所有指令的总数
算术指令数目	基本块中算术运算指令数量
子节点数目	当前基本块的直接后继基本块数目
介数中心性	基本块在整个 CFG 中的“中心程度”

知识基础 Annoy

- **Annoy（Approximate Nearest Neighbors Oh Yeah）**
  - 由 **Spotify** 开源的高效近似最近邻搜索库
  - 适用于推荐系统、向量匹配、相似性搜索等场景
  - 构建多棵树：每棵树是一个随机投影树（**Random Projection Tree**）
    - 构建方式：随机选取两个向量作为划分参考点 **A** 和 **B**
    - 将所有其他点投影到 **AB** 的超平面上，按中位数划分为左右子树
    - 重复构建多个不同的树（如**10**棵、**50**棵、**100**棵）以提升搜索精度



LibAM: An Area Matching Framework for Detecting Third-Party Libraries in Binaries

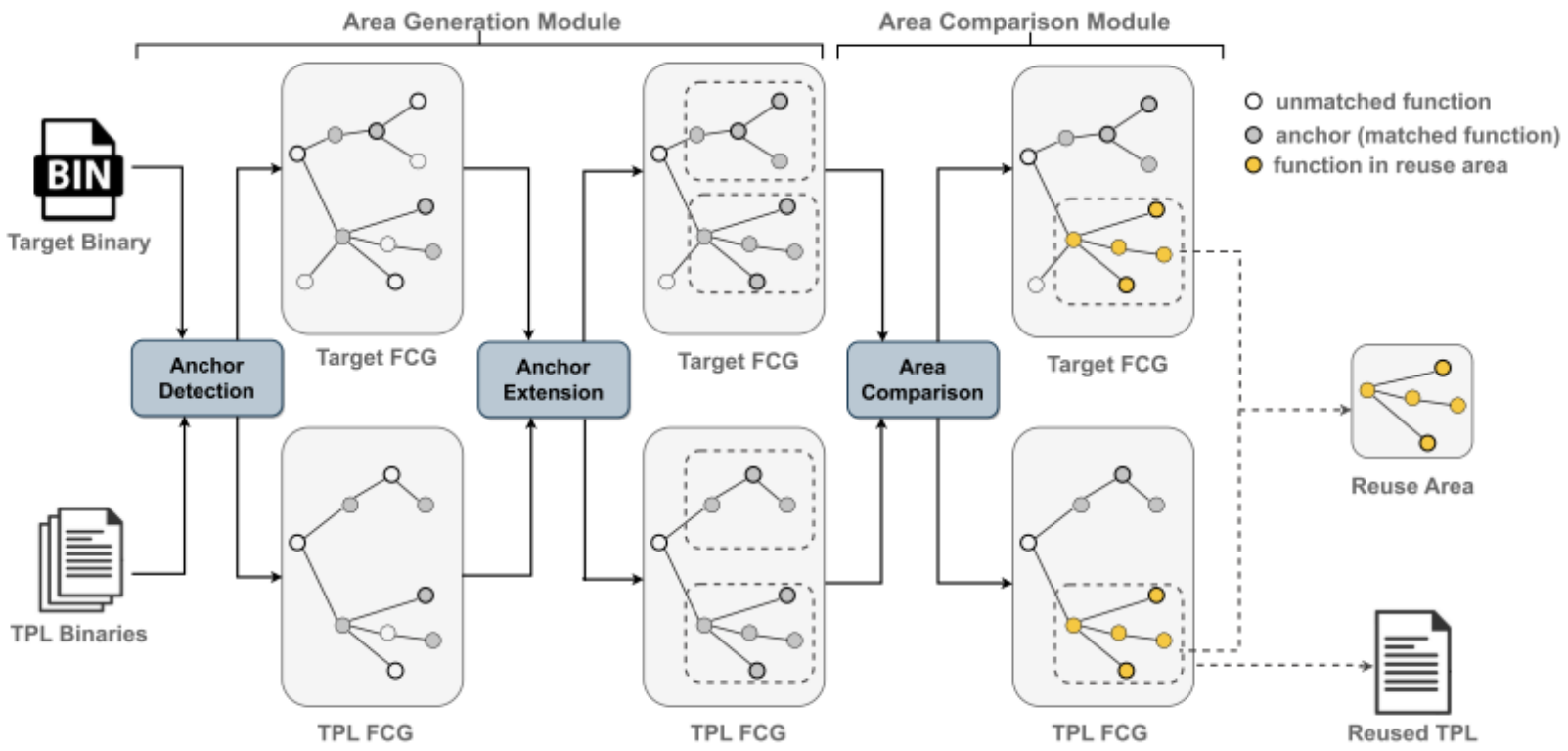
LibAM TIPO

T	目标	检测目标二进制代码所使用的第三方库
I	输入	目标二进制代码 第三方库函数数据库
P	处理	**1.**区域生成：将目标二进制和 **TPL** 二进制中的函数组合成“函数区域” **2.**区域比较：比较目标区域与**TPL**区域的相似性，判定是否为真实复用区域
O	输出	目标二进制代码所使用的第三方库列表

P	问题	细粒度识别第三库
C	条件	研究对象为二进制代码；需要第三方库函数数据库
D	难点	如何正确对齐、选取最优区域，避免区域重复检测
L	水平	**Transactions on Software Engineering and Methodology,2023（SCI一区）**

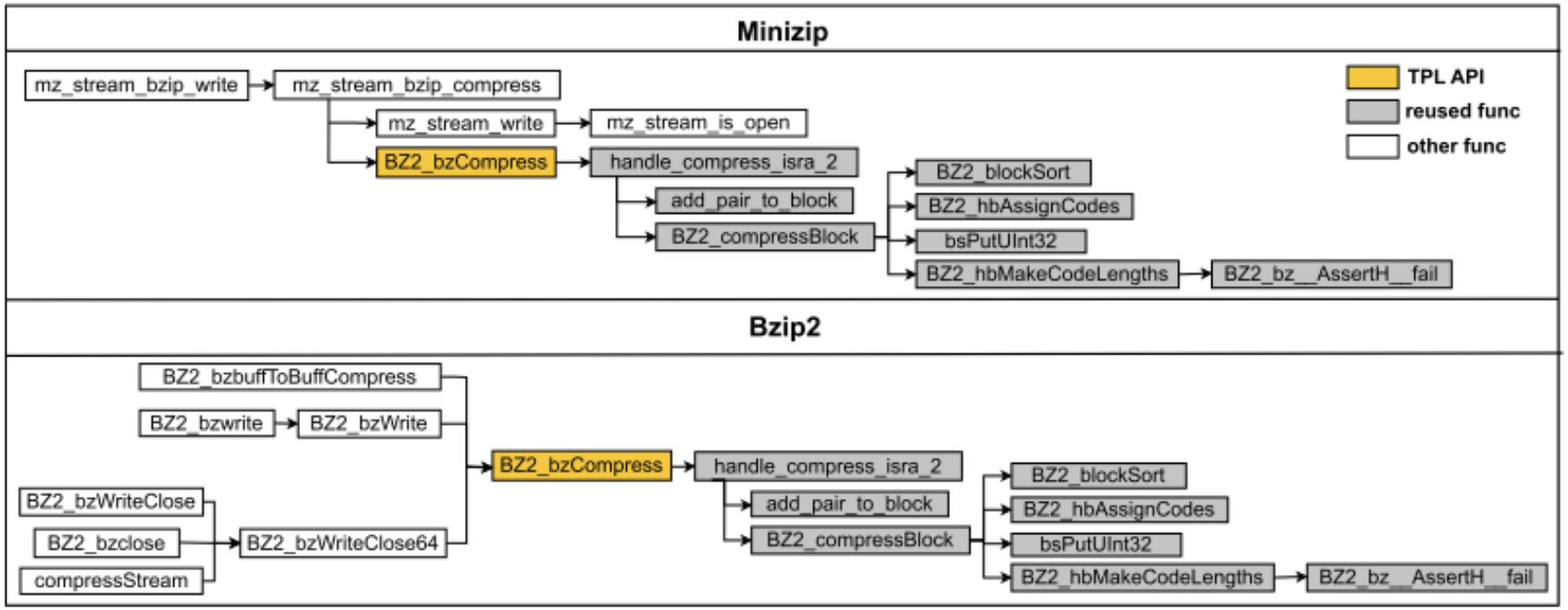
LibAM 算法原理

- 算法原理图
  - 区域生成：将目标二进制和 TPL 二进制中的函数组合成“函数区域”
  - 区域比较：比较目标区域与TPL区域的相似性，判定是否为真实复用区域



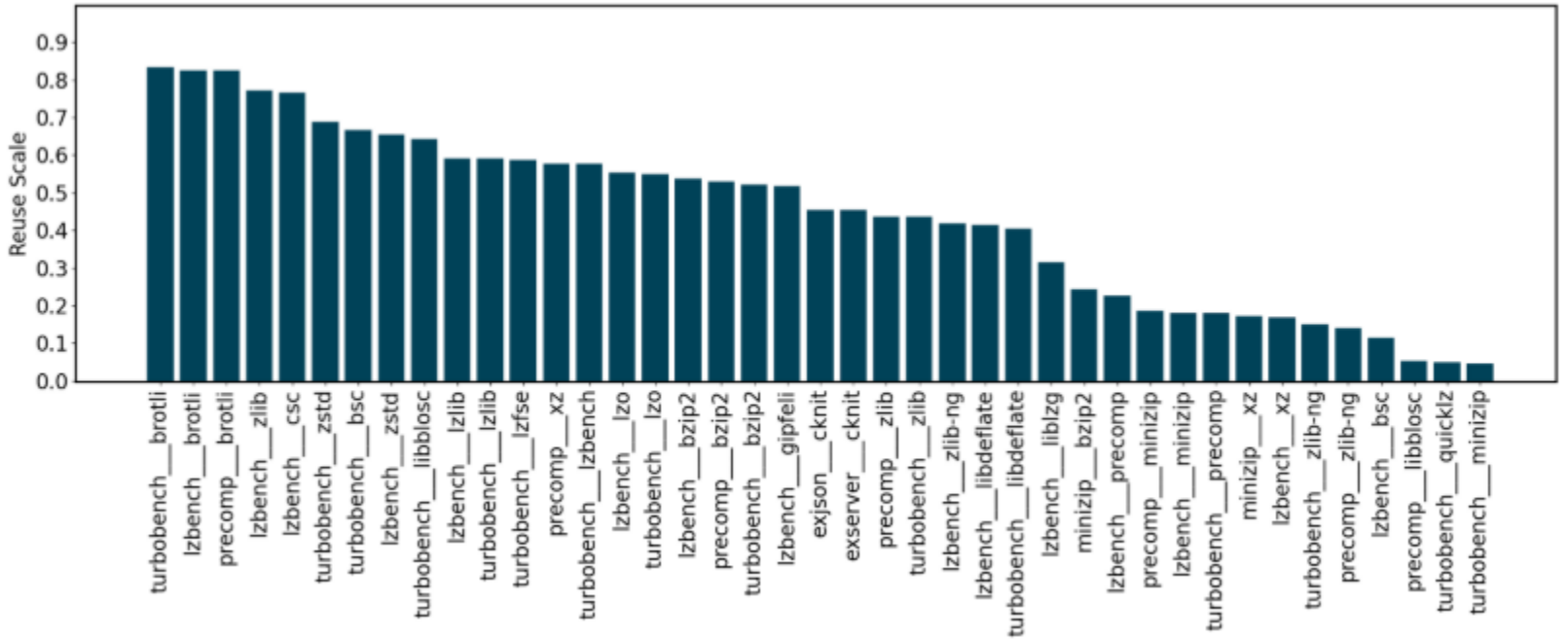
LibAM 动机

- 函数复用具有“结构传递性”
  - 在真实场景中，当一个函数被复用时，它的被调用函数（**callee functions**）往往也会一并复用
  - 比起“单函数匹配”，利用函数间的调用关系、检测“区域级别”复用，更稳健、准确



LibAM 动机

- 真实复用往往是“部分复用”，而非整库复用
  - 超过一半的复用关系，只复用了 TPL 中不到一半的函数
  - 文件级或整库级检测方法效果很差，容易漏报或误报



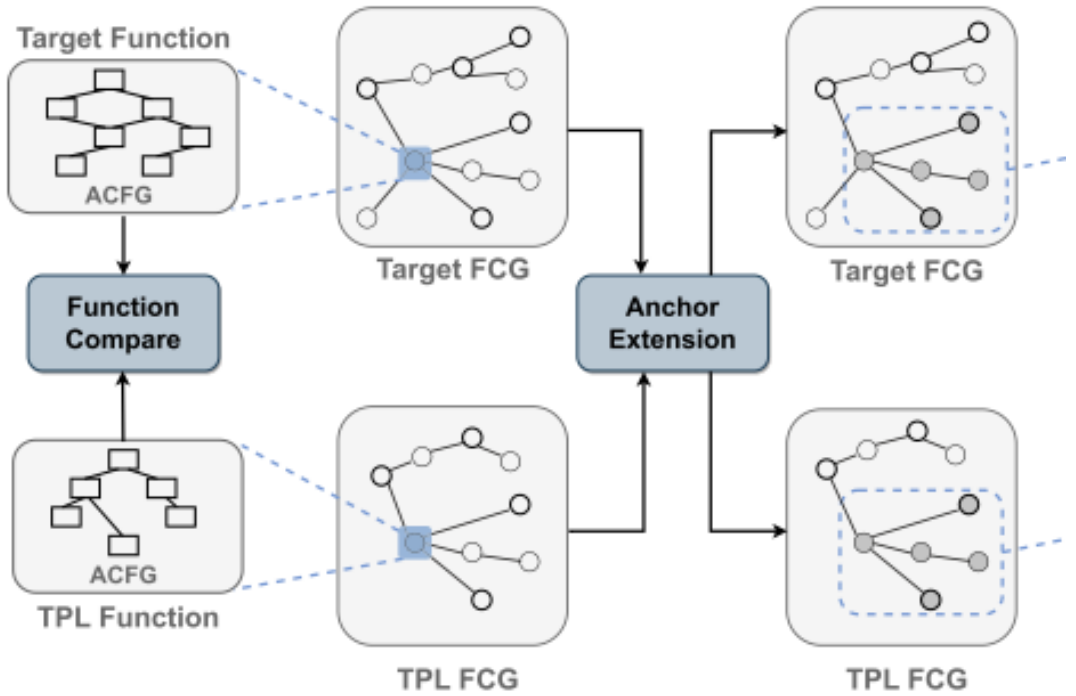
LibAM 创新分析

- 区域生成
  - 提取CFG和 FCG
    - 利用IDA Pro提取二进制代码的函数列表与调用关系
  - 锚点检测
    - 把CFG转化为Attributed Control Flow Graph（ACFG）
    - 基于Structure2Vec学习函数向量
    - 余弦相似度计算，阈值设置为0.72

属性特征	内涵
字符串常量	基本块中包含的字符串常量的数量
数值常量	基本块中包含的数值常量数量
传送指令数目	基本块中的跳转、分支、条件判断、返回等控制指令数量
调用指令数目	基本块中的函数调用指令数量
指令数目	基本块中所有指令的总数
算术指令数目	基本块中算术运算指令数量
子节点数目	当前基本块的直接后继基本块数目

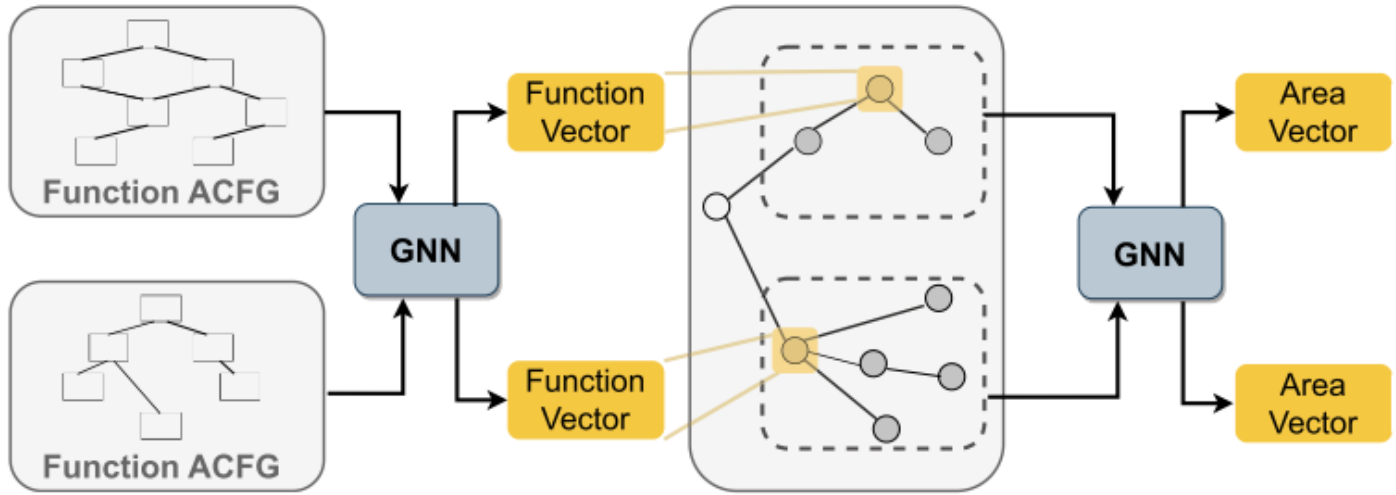
LibAM 创新分析

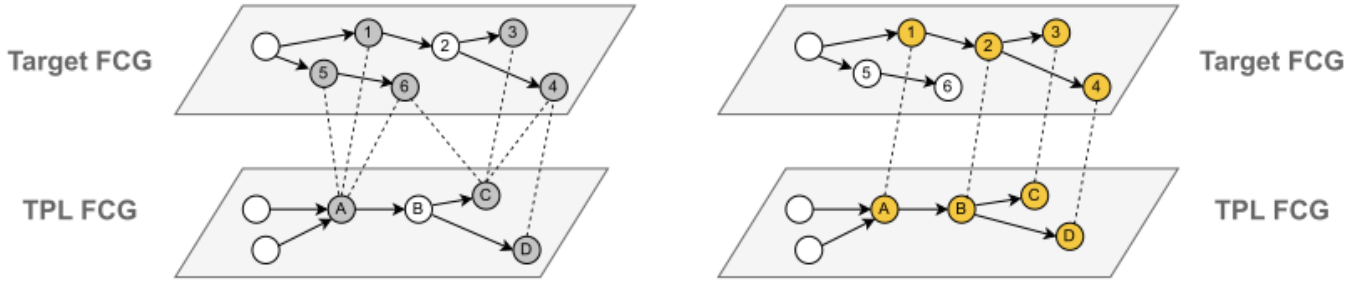
- 区域生成
  - 提取CFG和 FCG
  - 锚点检测
    - 采用Annoy加速策略
    - 每个目标二进制最多比对 200 个 TPL
    - 每个目标函数最多匹配 100 个 TPL 函数
  - 锚点扩展：以锚点函数为起点，在 FCG 中向下扩展所有被调用函数，形成区域
    - 编译优化、架构差异会让函数结构变化
    - 单个函数的匹配容易有噪声、误匹配



LibAM 创新分析

- 区域比较
  - 结构相似性
    - 利用 **Structure2Vec**， 将区域中的函数向量聚合成区域向量
  - 锚点对齐
    - 枚举起点锚点对
    - 通过迭代扩展找到最长一致锚点路径
  - 第三方库检测
    - 区域结构相似度阈值为**0.8**
    - 对齐锚点数阈值为**3**





LibAM 实验设计

- 数据资源
  - **Dataset_OSS**：爬行260个来自Github 和SourceForge 的常用开源项目
  - **Dataset_ISRD**：覆盖 24 个流行开源项目的85个二进制文件，并在X64中包含默认优化选项
  - **Dataset_ExtISRD**：在Dataset_ISRD基础上，手动编译三个架构（ARM，X86，X64）和四个优化选项（O0，O1，O2，O3）的二进制文件
  - **Dataset_FW**：来自 10 个厂商的 167 个固件镜像，包括 IP 摄像头、路由器、交换机等设备
  - 计算指标： **Precision (P)、Recall (R)、 F-Measure (F1)**

Dataset	Binaries	Target Binaries	TPLs	Contents	Applications
Dataset_OSS	22,100	–	–	cross arch and opti	software for traning
Dataset_ISRD	85	17	85	real-world	software for TPL detection
Dataset_ExtISRD	289	204	85	cross arch and opti	software for TPL detection
Dataset_FW	12,915	12,699	216	real-world	firmware for TPL detection

LibAM 实验设计

- 参数实验
  - 随机选取了 **400** 个函数对 和 **400** 个 **FCG** 区域对，作为实验样本
  - 正样本（**positive**） = 同源函数或同源区域（真正复用的）
  - 负样本（**negative**） = 非同源的、不相关的样本
  - 最终选出能最好平衡 **precision** 和 **recall**的最佳阈值
    - 函数相似度阈值：**0.72**
    - 区域结构相似度阈值：**0.8**

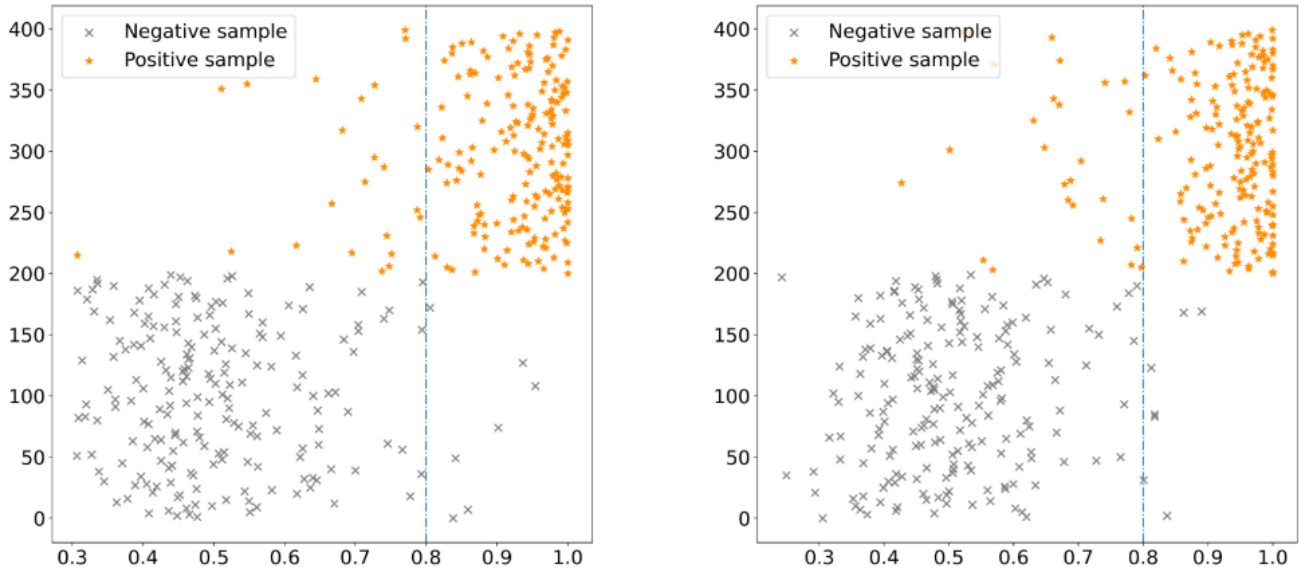


Fig. 9. The similarity score of 400 random function pairs (Left) and area pairs (Right).

LibAM 对比实验

- 实验结果
  - **LibDX(2020)**：提取目标二进制中的“十个连续字符串”
  - **B2SFinder(2019)**：使用更全面的常量特征集进行匹配（包括字符串、数值常量、switch/case结构、布尔值、函数参数等）
  - **ISRD Gemini(2021)**：若二进制中有一半以上函数能与 TPL 匹配，则判定为复用
  - **LibDB(2022)**：在函数匹配基础上，增加 FCG 中的 3 条边过滤规则
  - **LibDX 具有高精度和召回率，但依赖连续字符串作为特征，鲁棒性差**
  - **B2SFinder检测以整个文件为粒度设置阈值，无法细致识别函数或区域级别复用**

Model	LibDX	B2SFinder	$ISRD_{Gemini}$	LibDB	LibAM
P	0.920	0.692	0.304	0.534	**1.0**
R	0.809	0.644	0.518	0.920	**0.993**
F1	0.837	0.664	0.312	0.568	**0.996**

LibAM 对比实验

- 在区域检测能力上的实验结果
  - **LibDX(2020)**：提取目标二进制中的“十个连续字符串”
  - **B2SFinder(2019)**：使用更全面的常量特征集进行匹配（包括字符串、数值常量、switch/case结构、布尔值、函数参数等）
  - **ISRD Gemini(2021)**：若二进制中有一半以上函数能与 TPL 匹配，则判定为复用
  - **LibDB(2022)**：在函数匹配基础上，增加 FCG 中的 3 条边过滤规则
  - **LibDB虽然结构性比常量比较的方法更好，但规则简单，仍存在明显差距**
  - **LibDX匹配到的字符串大多不是由函数实际调用， 难以界定具体复用代码块**

Model	LibDX	B2SFinder	$ISRD_{Gemini}$	LibDB	LibAM
P	0.779	0.519	0.250	0.613	**0.985**
R	0.291	0.372	0.573	0.719	**0.847**
F1	0.379	0.393	0.311	0.619	**0.910**

LibAM 鲁棒性

- **x64-x64不同编译选项组合下的TPL检测效果**
  - **LibAM 在所有优化级别组合下，平均精度为 0.93，召回为 0.94，显示出对优化变动的良好适应性**
  - **LibDX 和 B2SFinder（基于常量）对优化级别变化不敏感，结果较稳定，但检测能力弱**

Model	O0-default			O1-default			O2-default			O3-default			Average		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
LibDX	0.55	0.44	0.43	0.80	0.61	0.65	0.74	0.61	0.63	0.74	0.61	0.63	0.71	0.56	0.58
B2SFinder	0.60	0.39	0.45	0.60	0.39	0.45	0.58	0.39	0.45	0.58	0.39	0.45	0.59	0.39	0.45
$ISRD_{Gemini}$	0.13	0.34	0.19	0.31	0.36	0.25	0.33	0.36	0.28	0.22	0.35	0.27	0.25	0.35	0.25
LibDB	0.38	0.47	0.32	0.44	0.83	0.50	0.47	0.85	0.54	0.45	0.81	0.51	0.44	0.74	0.46
$LibAM_{-align}$	0.16	1.0	0.23	0.17	1.0	0.24	0.19	1.0	0.26	0.19	1.0	0.28	0.18	1.0	0.25
$LibAM_{-gnn}$	0.47	0.86	0.58	0.52	0.96	0.62	0.54	0.98	0.63	0.58	0.96	0.67	0.53	0.94	0.62
LibAM	**0.90**	**0.86**	**0.88**	**0.94**	**0.97**	**0.95**	**0.94**	**0.97**	**0.95**	**0.93**	**0.96**	**0.94**	**0.93**	**0.94**	**0.93**

LibAM 鲁棒性

- **O2选项下不同架构下的TPL检测效果**
  - **LibAM的区域级别的结构匹配方法具备一定的跨架构鲁棒性**
  - 函数相似性方法（如 **ISRD Gemini**）在跨架构环境下表现极差，因函数间的表示差异被放大，导致匹配失败

Model	x86-x64			x64-x64			arm-x64			Average			Mix		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
LibDX	0.74	0.61	0.63	0.74	0.61	0.63	0.55	0.44	0.44	0.68	0.55	0.57	0.66	0.52	0.53
B2SFinder	0.60	0.39	0.45	0.58	0.39	0.45	0.60	0.38	0.44	0.59	0.39	0.44	0.60	0.39	0.44
$ISRD_{Gemini}$	0.33	0.36	0.26	0.33	0.36	0.28	0.26	0.33	0.28	0.31	0.35	0.28	0.24	0.34	0.25
LibDB	0.43	0.78	0.48	0.47	0.85	0.54	0.43	0.63	0.44	0.44	0.76	0.49	0.40	0.64	0.41
$LibAM_{-align}$	0.18	1.0	0.26	0.19	1.0	0.26	0.16	0.99	0.23	0.18	1.0	0.25	0.17	1.0	0.24
$LibAM_{-gnn}$	0.50	0.98	0.62	0.54	0.98	0.63	0.66	0.94	0.74	0.56	0.97	0.67	0.53	0.91	0.63
LibAM	**0.97**	**0.97**	**0.97**	**0.94**	**0.97**	**0.95**	**0.97**	**0.93**	**0.94**	**0.96**	**0.96**	**0.95**	**0.90**	**0.88**	**0.88**

LibAM 消融实验

- **LibAM−align**：移除锚点对齐过程，结构相似度来判断区域是否为复用区域
  - 仅依赖结构相似性，导致一些非同源区域也被错误识别为复用区域
- **LibAM−gnn**：保留锚点对齐，但不使用结构相似性评分，仅依赖对齐长度判断区域复用
  - **Precision 随 n 增大而上升**：更严格的对齐要求能减少误报
  - **Recall 随 n 增大而下降**：太严格会漏掉真实复用

Dataset	Metrics	$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 5$
Dataset_ISRD	P	0.604	0.794	1.0	1.0	1.0
	R	0.993	0.993	0.993	0.947	0.804
	F1	0.708	0.856	0.996	0.971	0.883
Dataset_ExtISRD	P	623	0.671	0.863	0.811	0.599
	R	0.905	0.896	0.848	0.452	0.253
	F1	0.704	0.734	0.842	0.533	0.322

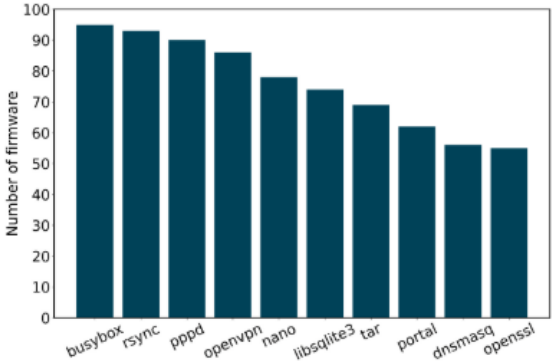
LibAM 效率实验

- **LibAM 与其他方法在 TPL 检测与 Area 检测中的时间效率对比，对其各阶段耗时进行详细分析区域**
  - **LibDX只使用字符串特征，并利用倒排索引加速，速度最快**
  - **B2SFinder使用更多种类的常量特征，但缺乏加速机制**
  - **LibAM是唯一能准确检测复用区域的方法**
  - **LibAM通过锚点筛选与候选上限控制，检测时间不随 TPL 数据库规模增加而增长，具有良好扩展性**

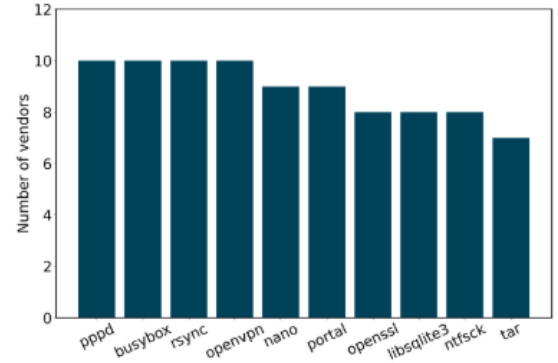
Model	Feature extraction	Embedding	TPL detection	Area detection	All_{TPL}	All_{area}
LibDX	7.5s	–	7.5s	–	15.1s	15.1s
B2SFinder	7.5s	–	70.4s	–	77.9s	77.9s
$ISRD_{Gemini}$	38.8s	1.8s	1.1s	–	41.8s	41.8s
LibDB	42.5s	1.8s	28.6s	–	72.8s	72.8s
LibAM	42.5s	3.4s	8.3s	109.9s	52.7s	162.6s

LibAM 大规模重用场景实验

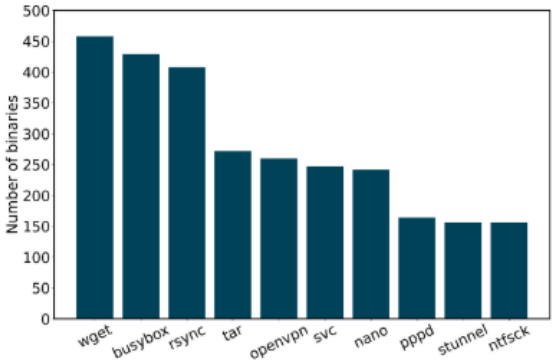
- 验证LibAM 在大规模真实世界中能够输出有价值的安全分析结果
  - **(a)： 被最多固件复用的前10个TPL库**
  - **(b)： 被最多厂商复用的前10个TPL库**
  - **(c)： 被最多二进制文件复用的前10个TPL库**
  - **(d)：CVE数量最高的前10种漏洞类型（共计216 个包含公开漏洞信息的 TPL 二进制作为比对候选）**



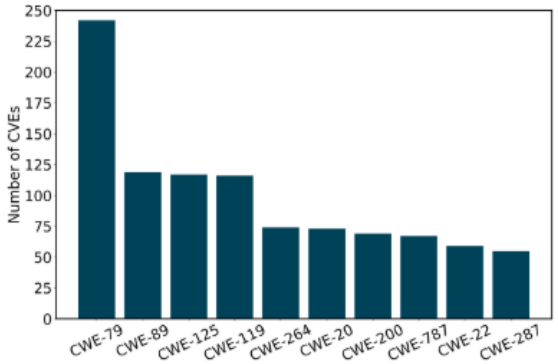
(a) No. of Firmware



(b) No. of Vendors



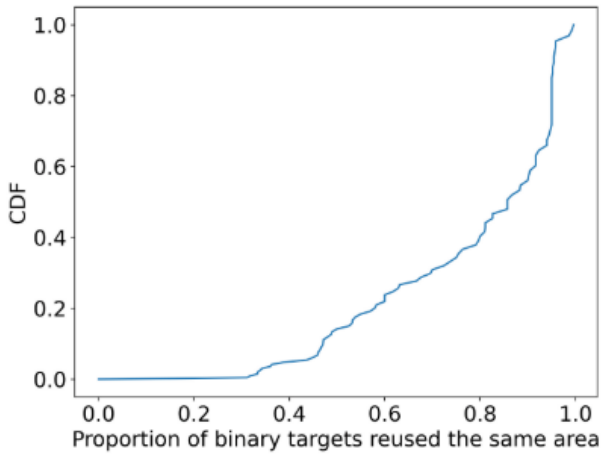
(c) No. of Binaries



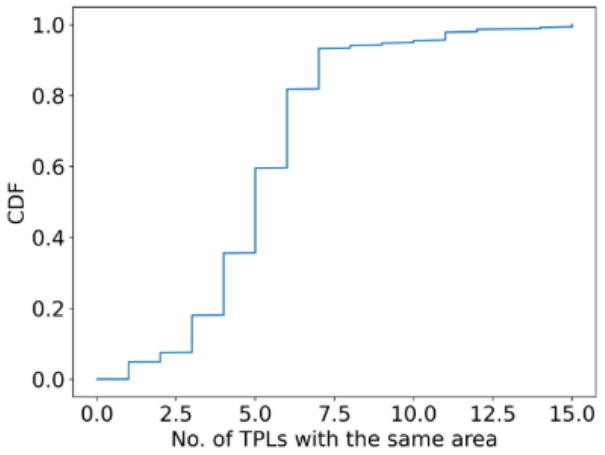
(d) No. of CVEs

LibAM 大规模重用场景实验

- 检测效果
  - 成功检测出 **3353** 个重用关系
  - 完成整个检测任务仅耗时 **30** 小时
- 漏洞识别与过滤
  - 通过版本识别 + 区域检测过滤后，识别出保 **14,863** 个有效漏洞
- 重用与漏洞传播规律挖掘
  - **>85.8%** 的 **TPL** 在多个固件中复用相同区域，部分代码高度复用
  - **>97.5%** 的固件中有复用多个 TPL 的相同区域，存在跨库共性代码



(a) CDF of the proportion of target binaries



(b) CDF of the number of TPLs

特点总结与未来展望

特点总结与未来展望

- 特点总结

优势	劣势
跨编译选项适应性好	时间开销较大
具有较强的解释性	功能调用图在不同优化/架构下差异过大
支持特定漏洞函数检测	-

- 未来展望
  - 提升区域匹配的鲁棒性
  - 构建更灵活的索引系统，提高检索效率

参考文献

[1] Li et al. LibAM: An Area Matching Framework for Detecting Third-Party Libraries in Binaries[J]. ACM Transactions on Software Engineering and Methodology, vol. 33, no. 2, pp. 1-35.

道德经

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！