

保留因果关系的溯源图压缩方法

张子龙, 董卫宇, 王奕森

(信息工程大学, 河南 郑州 450001)

摘要: 溯源图是检测与调查复杂网络攻击的一种有效工具, 但其存储和分析所需的高昂开销使得图压缩成为该领域不可或缺的环节。然而, 现有的大多数压缩方法会破坏溯源图中节点间的重要因果关联, 从而影响攻击调查的有效性。尤其是现有能够保留攻击调查能力的压缩算法, 通常难以有效处理节点间循环交互产生的冗余问题。针对这一挑战, 提出了一种图压缩算法CPC, 可在确保节点间的因果关系不受损害的同时, 实现对溯源图的有效压缩。该算法通过深入分析节点间的因果关系, 记录节点依赖等关键信息, 能够精准识别并剔除多余的依赖传递边, 在维护攻击调查结果的情况下, 显著提升了溯源图边数据的压缩效率。此外, 针对离线与在线溯源图构建场景, 分别实现了全依赖保留压缩CPC-A、部分依赖保留压缩CPC-P。在公开数据集上的实验表明, 与当前先进的压缩算法相比, 这两种算法在压缩性能上具有明显优势, 整体压缩率提升幅度为9.3%~86.4%。

关键词: 溯源图; 压缩算法; 因果关系; 攻击调查

中图分类号: TP393

文献标志码: A

DOI: 10.11959/j.issn.2096-109x.2025068

Causality-preserving compression method on provenance graph

ZHANG Zilong, DONG Weiyu, WANG Yisen

Information Engineering University, Zhengzhou 450001, China

Abstract: Provenance graphs were widely used for detecting and investigating sophisticated network attacks. However, their high storage and analysis overhead made graph compression indispensable. Most existing compression methods were found to disrupt critical causal dependencies among nodes, thereby compromising the effectiveness of attack investigations. In particular, current compression algorithms that preserved investigative capabilities struggled to handle redundancies arising from cyclic interactions between nodes. To address this challenge, a novel graph compression algorithm, CPC, was proposed. By thoroughly analyzing causal relationships and recording essential node dependency information, redundant transitive dependency edges were precisely identified and removed, significantly improving edge compression efficiency while preserving the integrity of attack investigation results. Two variants of the algorithm were developed: CPC-A, which retains full dependencies for offline provenance graph construction, and CPC-P, which preserves partial dependencies for online scenarios. Experimental evaluations on public datasets showed that, compared with current state-of-the-art compression algorithms, the proposed approaches achieved substantially higher compression performance, with overall compression ratios improved by 9.3%~86.4%.

Keywords: provenance graph, compression algorithm, causal relationship, causality association, attack investigation

0 引言

以高级持续性威胁 (advanced persistent threat, APT) 攻击为代表的复杂网络攻击事件频发, 对企业、组织甚至国家安全造成巨大威胁。例如, 2022 年, 中国西北工业大学遭境外网络攻击, 超过 140 GB 的高价值数据被窃取。又如 2024 年发生的 Change Healthcare 勒索事件, 导致约三分之一该服务覆盖的美国公民个人信息泄露, 给联合健康集团 (UnitedHealth Group) 造成超过 8.72 亿美元的损失。APT 等复杂攻击通常具有多阶段、长期潜伏和高度隐蔽等特征, 致使传统检测方法难以准确识别和分析此类威胁。当前主流的安全产品 (如 Splunk 和 LogRhythm), 通常通过对可疑告警进行关联分析来检测复杂攻击, 但这类方法普遍面临“告警疲劳”问题。溯源图是一种基于系统日志构建的表示方法, 其将日志中的系统实体建模为节点, 实体之间的交互行为建模为边。由于完整保留了系统实体及其行为之间的关联关系, 溯源图不仅在多阶段复杂攻击检测方面具有显著优势, 还常被用作取证的辅助工具^[1-2], 帮助安全人员还原攻击链路, 掌握攻击事件的全貌。因此, 溯源图被广泛研究并应用于复杂网络攻击的检测与调查。

溯源图的数据主要来源于主机系统审计日志。相较于应用程序日志或网络流量日志, 系统日志具有通用性强、记录完整、部署便捷等优点。然而, 由于记录了全部系统行为, 系统日志通常数据量极大。据统计^[3-5], 单台主机每日产生的日志可轻松达到 GB 级别, 而在面对长期、跨主机的复杂攻击时, 存储的数据规模可能高达 PB 级别。海量数据不仅带来存储压力, 其后续分析过程也对计算与内存资源提出了极高要求。因此, 溯源图压缩被视为溯源图威胁检测领域一项不可或缺的工作^[6-8], 能够有效缓解海量数据带来的存储与计算压力。

溯源图的数据压缩方法主要分为有损压缩和无损压缩两大类。无损压缩通常采用编码压缩技术减少数据的存储开销^[1,9]。Xie 等^[10]借鉴网页压缩思路, 设计了针对溯源图数据的编码压缩方案。有损压缩通过从多个角度识别并去除图结构中的冗余数据实现压缩^[11-12]。在实际应用中, 将

无损压缩与有损压缩结合使用, 可以进一步减少数据存储开销。尽管无损压缩具备“数据无失真”的独特优势, 但该技术往往需要额外设计配套的存储管理机制与解码流程, 这在一定程度上会增加系统的复杂度与查询时延。为此, Sardine^[13]和 LEONARD^[14]等方案分别通过差异化策略, 致力于缓解编码过程导致的查询性能下降问题。相较于无损压缩的“间接精简”, 有损压缩通过特定策略直接删除冗余数据, 从根源上缩减图数据规模, 因此本文将其定义为“冗余数据压缩”。在冗余数据压缩中, 信息的保真度与压缩效果之间存在互斥性。因此, 设计兼具高压缩比与高信息保真度的压缩算法, 成为该领域极具应用价值的研究方向^[15]。在溯源图中, 边的数量通常为节点数量的数十倍^[6], 而支持攻击调查 (即分析攻击的来源与影响) 是溯源图的核心功能之一。因此, 在保证攻击调查结果准确性不受影响的前提下对边数据进行高效压缩的算法, 更契合实际应用需求, 也因此受到研究者的广泛关注。

当前大多数溯源图压缩算法以特定攻击假设为前提进行设计, 通过合并或删除良性节点数据实现压缩目标。但这类操作往往会破坏原始图结构, 造成信息损失, 甚至可能因误删攻击链中的关键节点使攻击链还原的完整性受损。与之相比, 以保障攻击调查结果准确性为核心目标的边压缩方法, 在实际应用中展现出更突出的优势。该方法通过精准识别并删除节点间的冗余交互边, 既能实现数据规模的有效缩减, 又能确保系统实体之间的因果关系不被破坏。在该研究方向上, Xu 等^[3]首次引入“完全可达性”概念, 提出了一种不影响攻击调查结果的溯源边压缩算法。Hossain 等^[4]在其研究成果上放宽了算法的适用条件, 显著提升了压缩效率, 但该方案需引入版本图管理机制, 不可避免地带来了额外的版本节点与边数据开销。Zhu 等^[5]针对实时溯源图构建场景, 提出一种保留全局语义的压缩算法 GS。

然而, 现有这些以维持攻击调查结果准确性为目标的研究普遍遵循一个基本假设: 若节点出现新的输入边, 则表明其状态发生改变, 相应的后续边应予以保留。这一前提限制了算法的能

力,使其难以有效识别并删除节点之间因循环、重复交互而产生的冗余边。尽管 Hossain 等^[4]尝试通过后处理机制检测并处理两个节点间的循环读写冗余,但该方法仅适用于版本图结构,且会引入额外的存储与计算开销,实用性受限。针对上述问题,本文基于对节点间依赖关系的深入分析,提出一种冗余边识别与删除算法。该算法在压缩过程中能够始终保持因果结构完整,同时有效处理多节点之间的循环冗余问题,拓展了压缩方法的适用场景。为验证算法性能,在常用公开数据集上开展实验,结果显示:与当前先进的 GS 算法^[5]相比,本文方法的压缩效率提升了 9.3%~86.4%。

本文的主要贡献如下。

1) 从攻击调查视角出发,提出一种可解决循环冗余问题的溯源图压缩方法,扩展了溯源图压缩技术在处理多节点循环交互场景中的适用范围。

2) 设计了全依赖保留与部分依赖保留两种压缩算法。二者分别针对不同的应用场景,前者旨在满足离线溯源图构建对信息完整性的严苛要求;后者则面向在线场景,侧重于满足其对高压压缩效率与实时性的核心需求。

3) 在公开数据集上对所提方法进行了系统评估,结果表明,本文方法在压缩效率方面优于当前先进的压缩算法 GS。

1 相关工作

溯源图压缩的核心目标在于解决海量溯源数据引发的存储压力与分析开销问题。现有压缩方法在技术路径上存在一定的局限性。无损压缩能够完整保留原始信息,但其编码与解码过程常伴随较高的计算开销;基于冗余的压缩(即有损压缩)通过直接删除部分数据实现,却不可避免地造成信息损失。考虑到溯源图主要服务于攻击检测与调查取证应用,理想的冗余压缩方法需要保留关键攻击信息,尽可能降低对调查准确性的影响。

本文所提方法属于冗余压缩范畴。当前该领域的研究可以划分为3类,即冗余节点压缩、冗余边压缩和攻击保留压缩。本文将系统梳理这3类研究进展,以此明晰本文方法的技术优势与价值。

1.1 冗余节点压缩相关研究

冗余节点压缩以系统实体(节点)为压缩对象,通常根据特定场景制定相应策略,通过删除或合并节点实现压缩目标。Lee 等^[11]提出一种“垃圾回收”压缩方法,其核心思想在于识别并删除程序运行过程中产生的无后续影响的临时性文件。该方法需依赖细粒度的数据采集工具,而且在实际环境中大多数文件在生成后仍会与其他程序或进程发生交互,导致其适用范围受限,压缩效果也不理想。Tang 等^[16]发现进程初始化过程中的部分固定读取操作不会影响后续因果分析,据此提出一种基于模板的节点压缩算法。该算法通过识别并提取程序初始化阶段加载的库文件与只读资源构建“程序模板”,并基于模板对具有相同初始化行为的节点进行聚合,从而实现压缩。然而,该算法的有效性在一定程度上依赖于程序行为的规范性,若程序的实际执行逻辑偏离预设模板,则压缩效果与信息保真度可能下降。Hassan 等^[17]针对集群环境提出一种基于图语法学习的溯源数据压缩方法。该方法通过建模跨主机的相同应用程序行为模式,识别并消除冗余的溯源数据传输,在集群场景中展现出良好的压缩效果。Hossain 等^[4]提出一种冗余节点优化(RNO)策略,该策略基于节点版本更新机制识别并省略与依赖分析无关的版本更新操作,直接减少图中节点数量,从而有效简化溯源图结构。

1.2 冗余边压缩相关研究

边压缩在实现逻辑上与节点压缩具有相似性,核心在于识别并删除或合并图中的冗余边。算法的压缩效果取决于冗余边识别策略,若能根据后续溯源图分析需求准确判定并移除冗余边,即可在维持结果准确性的前提下实现压缩目标。

GrAALF^[18]通过聚合具有相同源节点、目的节点与边类型的多条边实现压缩。Xu 等^[3]提出因果保持缩减(causality preserved reduction, CPR)方法,将系统事件划分为关键事件与影子事件,通过将影子事件合并到关键事件中,在保持因果关系完整的前提下显著减少数据量。然而,CPR 所采用的“完全可追踪性等价”定义过于严格,使得应用场景受限。为此,Hossain 等^[4]指出,并非所有时间维度上的节点依赖关系都需严格保

留,因为在某些时段节点状态并未发生变化。基于此,该团队提出完全依赖保持(FD)压缩算法,扩展了CPR的适用场景,在不影响取证分析的前提下进一步提升压缩效果。此外,该团队还提出一种仅需保证溯源分析中源节点的依赖关系的源依赖保持(SD)算法,进一步提升了压缩效果。为实现在线实时压缩,KCAL^[19]修改审计系统,结合BEEP^[20],检测单元级别的冗余边。然而,该方法受限于操作系统且引入了额外的开销。Zhu等^[5]从节点语义角度出发,提出GS压缩算法,仅在保持源节点语义不变的前提下,保留第一条写入目的节点的边,从而在保持取证分析结果准确性的同时实现了较好的压缩效果。

上述的CPR、FD、GS算法未能有效解决常见的循环冗余(如多个节点之间的循环读写)问题。Hossain等^[4]提出的CCO算法虽能事后检测循环冗余,但仅支持两个节点的冗余检测,适用性有限。在此背景下,本文从节点间依赖关系出发,提出一种新型冗余边压缩方法。该方法能够在维持攻击调查结果的前提下,识别因重复信息传递而产生的冗余边,尤其适用于多个节点之间的循环冗余场景。在具体实现上,本文设计了两种算法:全依赖保留算法能够从根本上解决循环冗余问题,但计算开销较大,适用场景受限;部分依赖保留算法则在保持较高压缩率的同时,能够显著减少资源消耗,且实时处理能力强、适用范围广。

1.3 攻击保留压缩相关研究

与冗余节点压缩和冗余边压缩不同,攻击保留压缩不局限于边或节点层面的压缩,而是基于攻击行为假设,将所有与攻击无关的实体节点或行为边视为冗余数据。该方法在特定的攻击检测场景中展现出较好的压缩效果。

Winnower^[17]对大规模分布式系统审计日志进行图压缩,采用确定性有限自动机学习技术识别和归纳跨节点的共同行为模式,仅当出现新的或异常行为(即不匹配现有模型的行为)时才传输和记录数据。NoDoze^[21]基于事件边的历史频率来识别并保留与攻击最相关的异常路径,以此减少依赖图中的良性数据,实现压缩效果。Zhu等^[5]提出了基于可疑语义的数据压缩方法,根据

预定策略保留重要、威胁程度高的事件边,同时删除与攻击无关的事件。李霄翔等^[22]提出通过挖掘并匹配正常行为子图压缩数据的方法,ProvGRP^[23]通过合并相似的行为路径实现压缩。此外,一些研究方法^[24-25]在实时检测过程中只保留与威胁图匹配的相关实体与事件,删除其余的节点与事件。

综上,冗余节点压缩的压缩率通常低于冗余边压缩,攻击保留压缩虽然压缩幅度更大,却建立在先验攻击假设之上,容易丢失与假设不符但实为攻击链的关键数据。相比之下,维护攻击调查结果的边压缩方法,既不依赖任何启发式攻击模型,又能保留图中所有实体间的交互关系,兼顾了压缩率与信息完整性。本文扩展了此类压缩方法的应用场景,实现了一种高压压缩率与信息完整性的冗余边压缩算法。

2 背景知识

2.1 溯源图

基于溯源图的入侵检测起源于BackTracker^[26]。BackTracker从操作系统级别的审计日志中提取进程、文件及其依赖关系,生成依赖图以追踪入侵起源。此后,相关研究陆续涌现。PASS^[27]是一种生成祖先图的存储系统,用于支持来源信息的查找。SPADE^[28]与CamFlow^[29]分别基于OPM^[30]和PROV-DM^[31]生成指向数据流源头的溯源图,其中CamFlow基于版本化将溯源图表示为有向无环图。溯源图在一些研究方案中^[32]被称为因果图。

现有研究对溯源图的命名与定义虽有差异,但均基于操作系统审计日志,提取系统中的实体(如进程、文件、网络)作为节点,将节点之间的交互事件作为边(如read、execve、unlink等系统调用)。对于溯源图,本文给出如下定义。

定义1 溯源图

溯源图是一种有向属性图,可表示为 $G = \langle V, E \rangle$,其中 V 表示系统中实体节点的集合, E 表示实体间交互的带有时间戳的有向边集合。实体节点分为主体和客体,主体为系统中的进程或线程,客体为文件、网络套接字、管道等。边表示系统审计日志中的操作事件,其方向代表信息流的方向。每个节点与边均有唯一的标识符(ID)。

溯源日志及对应溯源表示如图1所示，其中图1(a)为某日志收集系统产生的日志事件流，图1(b)为根据日志事件流中的实体及交互行为生成的简化溯源图，其中圆圈表示实体节点，带时间戳的有向边表示节点间的交互行为（本文以时间戳标识图中的边），箭头方向为信息流方向。

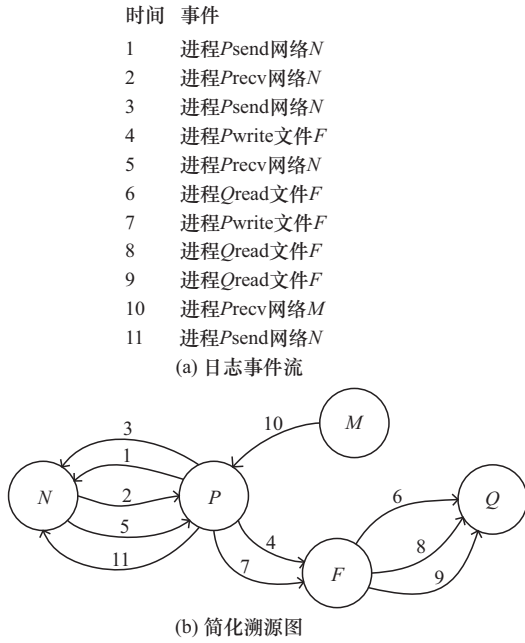


图1 溯源日志及对应溯源表示

2.2 因果关系与依赖

在攻击调查中，通常沿着有向边，前向追踪信息流传递所影响的节点，后向追踪事件起因（即信息流的源节点）。存在信息流传递的节点之间被认为存在因果关系。本文对因果关系与依赖的定义如下。

定义2 因果关系与依赖

若两个节点存在信息流传递，则二者存在因果关系，其中信息流起始节点是目的节点的依赖。具体而言，如果起始节点 S 与目的节点 T 之间存在一条路径 $Path$ 满足式(1)，那么节点 S 与节点 T 之间存在因果关系，且节点 T 依赖于节点 S ，记为 $T \text{ dp } S$ 。

$$\begin{aligned} \exists Path = \{ e_1, e_2, e_3, \dots, e_n \} \\ \wedge T(e_i) \leq T(e_{i+1}) \\ \wedge \text{end}(e_i) = \text{start}(e_{i+1}), \quad 1 \leq i \leq n-1 \end{aligned} \quad (1)$$

其中， $Path$ 表示溯源图中的一条以 S 为起点、 T 为终点的路径， e 表示有向边， $\text{start}(e)$ 表示边的

源节点， $\text{end}(e)$ 表示边的目的节点， $T(e)$ 表示事件边发生的时间。

以图1中的节点为例，节点 N 与节点 F 之间存在路径 $Path=\{2,4\}$ ，该路径以 N 为起点、 F 为终点，且时间上递增，因此节点 N 与节点 F 存在因果关系，且 $F \text{ dp } N$ 。节点 M 与节点 F 之间尽管存在路径 $Path=\{10,4\}$ 与 $Path=\{10,7\}$ ，但不满足时间递增条件，故二者不存在因果关系。

2.3 攻击调查

攻击调查^[33-34]是指威胁发现后的分析工作，主要目的是查清攻击的起源、过程及影响，为后续威胁处理与响应提供支撑。在溯源图的检测过程中，威胁节点通常源于某实体的异常行为事件，通过对威胁节点进行依赖源分析，找到攻击的入口点，再沿信息流方向分析攻击可能造成的影响。威胁来源查找与影响分析在图中表现为前后向分析，该过程也被称为取证分析^[35]。

定义3 后向分析

后向分析是追溯攻击起源的一种分析方法，通过反向追踪信息流获取影响当前节点的所有依赖节点。给定检测节点 x ，后向分析根据因果关系追踪溯源图中节点 x 依赖的所有节点（即节点 x 依赖的节点集合），如式(2)所示。

$$R_{\text{backward}} = \{ v | x \text{ dp } v, v \in V \} \quad (2)$$

定义4 前向分析

前向分析是追踪攻击影响的一种分析方法，通过正向追踪信息流获取当前节点的所有影响实体节点。给定检测节点 x ，前向分析根据因果关系找到溯源图中依赖于节点 x 的所有节点（即节点 x 影响的节点集合），如式(3)所示。

$$R_{\text{forward}} = \{ v | v \text{ dp } x, v \in V \} \quad (3)$$

以图1为例，若 $t=5$ 时发现进程 P 异常，则攻击调查首先追溯导致进程 P 异常的源头。根据式(2)可知， $P \text{ dp } N$ ，故可以获得后向分析的结果 $R_{\text{backward}}=\{N\}$ ，即可能是网络节点 N 导致 P 异常。随后分析此异常产生的影响，需要通过前向分析找到节点 N 影响的节点，由此得到影响结果集 $R_{\text{forward}}=\{P,F,Q\}$ 。根据分析取证，可以推测网络节点 N 传递某种信息使得进程 P 异常，且可能会进一步对文件 F 与进程 Q 造成影响。

攻击调查与信息流的传递密不可分,保留因果关系的压缩算法不改变节点间的依赖关系,可在不影响取证分析结果的情况下减少存储与后续分析的开销。

3 本文方法

溯源图压缩分为在线压缩与离线压缩两种场景。在线压缩在日志实时处理时决定是否压缩溯源数据;而离线压缩则根据告警点选取特定时间段的日志进行离线分析与压缩。本文针对两种场景设计了不同的压缩处理方法,算法的应用框架如图2所示。

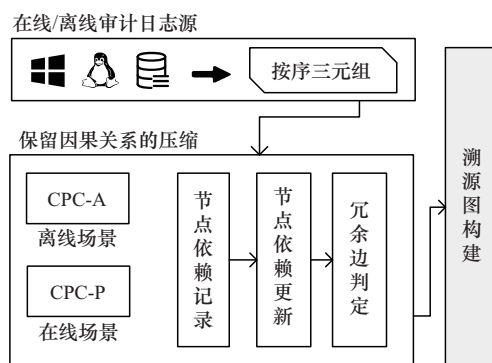


图2 本文压缩算法的应用框架

数据源为操作系统的日志流,可以从主机日志收集程序(如auditd或ETW)中获取,或从数据库提取告警相关事件日志。通过日志流解析,提取并生成归一化、按时间排序的三元组(源节点,操作边,目的节点),每个三元组对应一条边,可以表示为 $\langle \text{src}, \text{op}, \text{dst} \rangle$ 。

得到三元组数据后,可以根据场景选择合适的压缩算法进行处理。全依赖保留压缩算法(CPC-A)适用于离线场景,部分依赖保留压缩算法(CPC-P)适用于在线场景。算法通过记录、更新节点依赖来判定溯源边是否可压缩。压缩后的图数据可以存储起来供后续分析使用。

3.1 压缩策略

因果关系保留压缩的核心是识别并删除冗余边,其本质是找出不影响节点间依赖关系的冗余信息流,删除此类信息流不会改变攻击调查结果。由第2.3节的定义可知,后向分析结果是一个节点所依赖的节点的集合,前向分析结果是一个节点所影响的节点的集合,因此保证压缩前

后每个节点的依赖集合不变,即可保留因果关系。若节点 A 在依赖不变的情况下,重复向受其影响的节点 B 传递信息,则该信息流对应的边为冗余边,可以将其删除。

对于按时间排序的三元组集合Emap,其中每个三元组对应一条边,包含源节点src和目的节点dst。对于每个节点 x ,其依赖信息存储在 DP_x 中,依赖信息不变时直接影响的节点存储在集合 IN_x 中,初始时 DP_x 和 IN_x 均为空。

压缩策略实现流程如图3所示,具体步骤如下。

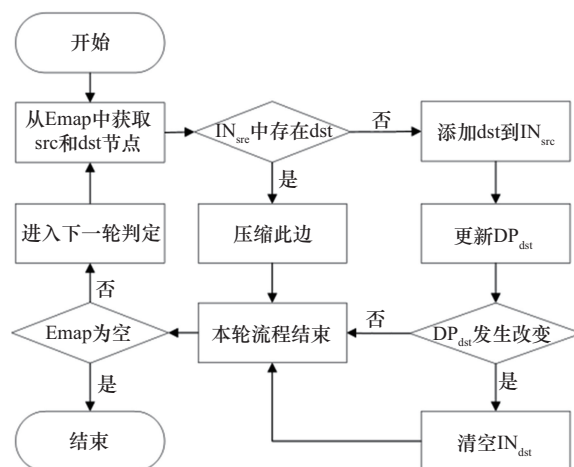


图3 压缩策略实现流程

1) 循环获取Emap中按顺序排列的三元组,提取三元组中的src与dst。

2) 若 IN_{src} 中存在dst,由于src在依赖不变时已向dst传递信息,再次传递信息, DP_{dst} 也不会发生变化,故此边可压缩,进入下一轮循环。

3) 若 IN_{src} 中不存在dst,此边不可压缩,需要更新节点状态信息——将dst添加到 IN_{src} 中(表示src给dst传递过信息流),根据src和dst, DP_{src} 更新目的节点的依赖信息 DP_{dst} ,并查看 DP_{dst} 是否发生改变。

4) 若 DP_{dst} 发生改变,则dst依赖改变后的出边必须全部保留,清空节点集合 IN_{dst} ,进入下一轮循环。

5) 若 DP_{dst} 未发生变化,则直接进入下一轮循环。

该策略删除节点依赖不变时的重复信息流边,可以保留节点间的因果关系,维护攻击调查的结果。其思路与GS压缩相似,但GS压缩基于模糊语义概念,节点有新的传入边即认为节点的

依赖信息发生改变，忽略了“信息传入但节点依赖不变”的情况，限制了压缩能力。本文方法解决了此问题，提升了压缩效果，扩展了适用范围。

上述压缩策略的实现关键在于节点依赖的记录方式、节点依赖更新与节点依赖改变的判定方法。下文将围绕这两点详细阐述CPC-A与CPC-P的实现。

3.2 全依赖保留

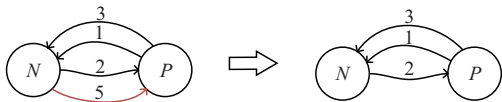
CPC-A将节点 x 的依赖信息 DP_x ，记录为 x 所依赖的所有节点的集合。每个节点的初始依赖为空。对于事件日志流解析出的每个三元组(src, op, dst)，采用式(4)更新节点依赖，源节点 src 与目的节点 dst 的依赖分别为 DP_{src} 与 DP_{dst} ，边信息传递后，目的节点依赖为 DP'_{dst} 。节点依赖改变的判断规则为：在事件发生后， DP'_{dst} 等于 DP_{dst} ，节点依赖未改变，否则节点依赖发生改变。

$$DP'_{dst} = DP_{src} \cup \{src\} \cup DP_{dst} - \{dst\} \quad (4)$$

以节点循环交互场景为例，验证CPC-A的压缩能力。图4展示了两个节点之间的循环交互压缩场景，图4(a)记录了不同时刻下的节点状态，图4(b)展示了压缩前后的溯源图。初始时，节点的 DP 、 IN 均为空。当 $T=3$ 时，节点 N 和 P 互相依赖，且在 N 和 P 依赖不变的情况下已完成信息传递，因此后续节点间的交互边均可看作冗余边，可以进行压缩且不影响因果关系。

时间	节点	DP	IN
$T=0(\text{initial})$	N	—	—
	P	—	—
$T=1(P \rightarrow N)N$	N	P (发生变化)	—
	P	—	—
$T=2(N \rightarrow P)$	N	P	P
	P	N (发生变化)	—
$T=3(P \rightarrow N)$	N	P	P
	P	N	N
$T=5(N \rightarrow P)$	N	P	P
	P	N	N

(a) 节点状态表示



(b) 溯源图压缩前后

图4 节点循环交互压缩场景

图4仅展示了两个节点之间的循环交互压缩。但是，由于CPC-A记录了节点所有的依赖，因此任意多个节点循环交互中的冗余信息流都可以被检测到并删除。

将CPC-A应用于图1所示的溯源图，压缩前后对比如图5所示。图5(a)展示了压缩前的溯源图 G ，图5(b)展示了压缩后的溯源图 G' 。压缩边的具体情况如下： $T=5$ 时，节点 N 的依赖未发生改变，且有指向节点 P 的信息流（即节点 N 的 IN 包含节点 P ），故 $T=5$ 时指向节点 P 的重复边可以删除；同理， $T=7$ 时，节点 P 的 IN 包含 F ， $T=8$ 和 $T=9$ 时，节点 F 的 IN 包含节点 Q ，因此对应边均可删除。

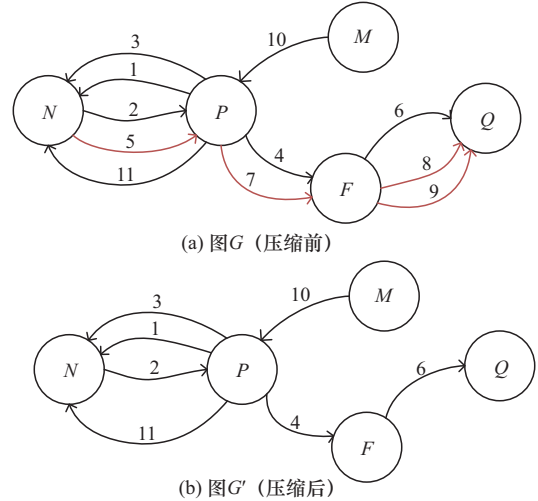


图5 溯源图压缩前后对比

以 P 异常为例，验证压缩前后取证结果的一致性。在攻击调查时，首先需要追溯威胁来源，进而分析威胁所产生的影响。由于 P 不会自发产生异常，其异常必然由新的信息流入引起，因此根据图 G 中节点 P 的入边将发现 P 异常的时间划分为4个区间，分别为 $\{t < 2, 2 \leq t < 5, 5 \leq t < 10, 10 \leq t\}$ 。下面分别讨论4种情况下的攻击调查结果。

1) 若在 $t < 2$ 时发现 P 异常：从图中观察到 P 从一开始就是异常的，因此后向分析结果为空，前向分析显示所有依赖 P 的节点均受到影响，即 $\{N, F, Q\}$ ，观察图 G' 也是如此，压缩前后取证分析的结果一致。

2) 若在 $2 \leq t < 5$ 时发现 P 异常：后向分析发

现 $P \text{ dp } N$, 得到威胁来源为 $\{N\}$, 前向分析得出受影响的节点为 $\{P, F, Q\}$, 观察图 G' 也是如此, 压缩前后取证分析的结果一致。

3) 若在 $5 \leq t < 10$ 时发现 P 异常, 后向分析得到威胁来源为 $\{N\}$, 前向分析显示影响的节点为 $\{P, F, Q\}$ 。虽然压缩后的图 G' 中无 $T=5$ 时 P 的入边, 但取证分析结果与未压缩时一致。这是因为 $T=5$ 时写入 P 的边与 $T=2$ 时的边在依赖传递效果上相同, 删除后不会影响节点 N 与 P 之间的因果关系。 $T=7$ 、 $T=8$ 和 $T=9$ 时刻的边压缩同理, 因此取证分析结果保持不变。

4) $10 \leq t$ 时发现 P 异常, 后向分析得到威胁来源为 $\{N, M\}$, 前向分析显示影响的节点为 $\{P, F, Q\}$, 从图 G' 中可以得到相同的取证结果。

上述分析表明, 删除冗余的依赖传递边不会影响取证分析的结果。

然而, CPC-A 在节点依赖记录方面的内存开销会随着节点数量的增加呈线性增长, 且后续的集合运算时间也会相应增加。若节点数量为 n , 每个节点记录存储占用内存为 m , 则在最坏的情况下每个节点都依赖于其余所有节点, 则总内存开销为 $n^2 \times m$ 。由于 m 为常数, 因此其空间复杂度为 $O(n^2)$ 。在运算方面, 时间复杂度与依赖集合大小相关, 为 $O(n)$ 。在实际应用中, 可以采用分层存储策略, 仅在内存中保留最常用的节点状态, 其余数据存入嵌入式数据库, 通过部分时间开销换取空间效率的提升。

鉴于节点数量增加会导致存储与 CPU 开销不断上升, 该方法仅适用于适度规模的离线溯源图构建与分析。

与现有方法相比, CPC-A 的最大优势在于能够检测所有因果关系冗余边, 从而实现溯源图的最大程度压缩。

3.3 部分依赖保留

为使依赖压缩策略适用于大规模实时溯源图压缩场景, 可以采用记录部分依赖的方法以平衡压缩效果与开销。在部分依赖保留 (CPC-P) 算法中, 节点依赖 DP 仅记录最新的 L 个依赖, 无法记录节点的全部依赖, 故 CPC-P 可能引发依赖缺失问题, 即节点依赖实际改变但未被检测到。

如图 6 所示, 当 $T=1$ 与 $T=3$ 时, 节点 P 的依

赖均发生改变, 但是由于依赖记录长度限制, 当 $L=1$ 时, DP 均为 F , 表现相同, 因此无法准确识别节点依赖的变化。

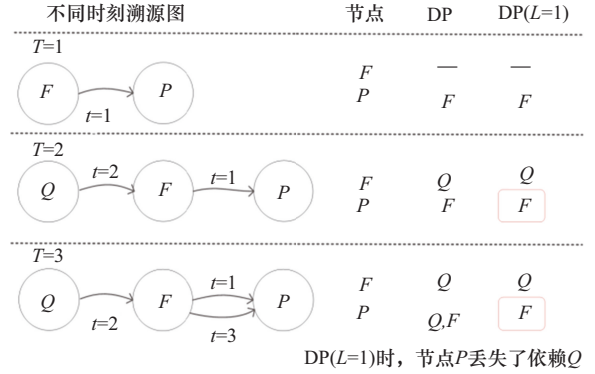


图6 依赖缺失问题示意

为解决依赖缺失问题, CPC-P 引入了节点版本信息: 将“节点+版本”作为依赖标识 (初始版本为 0), 节点依赖变化时版本递增。如此, 依赖变化可通过依赖节点的版本体现, 并被准确识别。

如图 7 所示, V_{num} 表示 num 版本的节点 V , 即使采用 DP ($L=1$) 作为依赖记录, 也可准确判定节点依赖变化。 $T=1$ 时, 节点 F 向 P 传递信息流, P 的 DP ($L=1$) 记录为 F_0 。 $T=2$ 时, F 有新的信息流入, 依赖发生变化, 版本更新为 F_1 。 $T=3$ 时, F 再次向 P 传递信息流, P 的 DP ($L=1$) 由 F_0 更新为 F_1 。与图 6 所示的无版本信息相比, 带有版本信息的依赖节点标识能够精确判定节点的依赖变化。需要注意的是, 该版本信息仅由算法内部维护, 不会体现在生成的溯源图中。

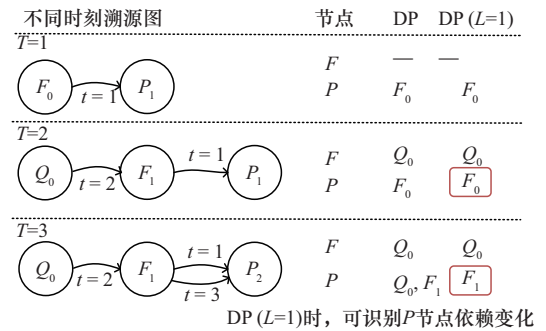


图7 带版本信息的依赖状态记录

因此, 在 CPC-P 算法中, 将带有版本信息的节点作为节点依赖, 节点依赖 DP 记录最新的 L 个

带版本信息的依赖节点，每个节点的初始依赖为空。对于事件日志流解析出的每个三元组 (src, op, dst) ，可以使用式(5)更新节点依赖。

$$DP'_{dst} = \text{updateLdp}(DP_{dst}, DP_{src}, src_v) \quad (5)$$

其中， src 、 dst 分别为边的源节点与目的节点，其初始依赖分别为 DP_{src} 与 DP_{dst} 。函数 $\text{updateLdp}()$ 根据传入参数保留最多 L 个不同的依赖节点信息，且优先保留最新传入的节点依赖（最右侧参数优先保留）。边信息传递后，目的节点依赖为 DP'_{dst} 。

节点依赖是否变化的判断需借助式(6)：若事件发生前后节点的新增依赖 DP_{new} 为空，表明节点依赖未改变，否则节点依赖发生改变，版本更新为 $v+1$ 。

$$DP_{new} = DP'_{dst} - DP_{dst} \quad (6)$$

当节点依赖变化且版本更新为 dst_{v+1} 时，存在引起该变化的依赖集合 DP_{new} 。若引起 dst_v 依赖变化的节点 $X \in DP_{new}$ ，同时依赖于节点 dst_v 和 DP_{new} ，则节点 X 也应依赖于 dst_{v+1} 。为防止依赖关系丢失，可以利用式(7)判断并更新 DP_X 中 dst 节点的版本。

$$\begin{aligned} & \text{update}(dst_v, dst_{v+1}) \text{ in } DP_X, \\ & \text{if } (dst_v + DP_{new} - X) \in DP_X, \\ & X \in DP_{new} \end{aligned} \quad (7)$$

算法1描述了依赖版本更新的具体流程，该机制可有效删除循环场景下的冗余边。

算法1 依赖版本更新算法

输入 1. 目的节点 dst_v 变为 dst_{v+1} ，其新增依赖集合 DP_{new} （非空）；

2. 任意节点 N 的依赖集合 DP_N

输出 无

- 1) **for** each node $X \in DP_{new}$ **do**
- 2) **if** $\{dst_v + DP_{new} - X\} \in DP_X$ **then**
- 3) **update** $DP_X(dst_v, dst_{v+1})$
- 4) **end if**
- 5) **end for**

为更直观地呈现 CPC-P 的运行机制，图8展示了节点循环冗余场景下 CPC-P 运行状态。左侧为不同时刻发生的事件及其对应的溯源边（send、recv 表示网络信息传输）；右侧为事件发生后各节点的状态信息，主要包括节点依赖 DP

及节点依赖不变时直接影响的节点集合 IN，节点 P 与 S 的初始状态均为空，版本号为0。设定 DP 记录长度 $L=1$ 。具体流程如下。

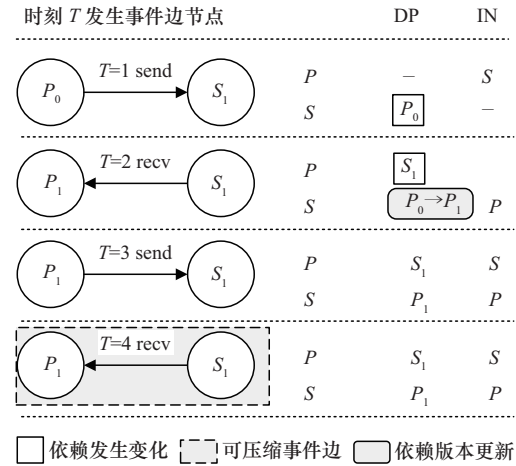


图8 节点循环冗余场景下 CPC-P 运行状态示意

1) 当 $T=1$ 时，发生 P 到 S 的事件边， S 的依赖发生变化，版本由0变为1。

2) 当 $T=2$ 时，发生 S 到 P 的事件边， P 的依赖发生变化，清空节点 P 的 IN 集合，且由于节点 S 依赖于 P_0 及引起节点 P 依赖改变的节点 S_1 （节点默认依赖自身），满足式(7)的依赖版本更新条件，因此将节点 S 的 DP 由 P_0 更新为 P_1 。

3) 当 $T=3$ 时，发生 P 到 S 的事件边，节点 P 的 IN 集合新增节点 S 。

4) 当 $T=4$ 时，发生 S 到 P 的事件边，由于节点 S 依赖未变且已记录于 P 的 IN 集合，因此该边可压缩。

若节点数量为 n ，每个节点的记录存储占用内存为 m ，每个节点记录 L 个依赖节点，则在最坏的内存使用情况下总内存占用量为 $n \times L \times m$ ，由于 m 、 L 为常量，因此 CPC-P 的空间复杂度为 $O(n)$ 。算法的运算时间与依赖集合大小有关，而依赖集合最多记录 L 个依赖， L 为常数，因此 CPC-P 的时间复杂度为 $O(l)$ 。在实际实现中，还可以采用分层存储技术进一步降低内存开销。

CPC-P 通过记录最新的 L 个依赖节点，可以解决涉及 $L+1$ 个节点的循环冗余交互问题。后续实验表明，即使仅记录1个或2个依赖节点，CPC-P 仍能取得良好的压缩效果。CPC-P 的开销远小于 CPC-A，因此适用于实时图构建场景。

CPC-P 通过牺牲少量冗余检测能力,大幅减少了计算与存储开销。

4 实验与结果分析

本节首先介绍实验环境与数据集,然后通过实验结果展示 CPC-A 与 CPC-P 的压缩效果及开销,最后验证因果关系保留压缩算法在前后向取证分析中的结果正确性。

实验环境配置如下:处理器为 Intel Core i5-12500H @ 2.50 GHz,内存为 DDR5 4800 MHz, 16 GB×2,软件环境为 Python 3.8。算法代码已开源于 Gitee 仓库。

4.1 数据集

本文在 DARPA TC E3 数据集上测试与评估本文所提的压缩算法。该数据集是溯源图检测研究工作中^[36-38]常用的公开数据集。DARPA TC 是美国国防高级研究计划局针对 APT 等问题提出的透明计算 (TC) 计划,其中 E3 数据集是 TC 计划在第三次战役期间生成的文件集合,其中包含多个参与者在不同平台主机中记录的系统实体与事件信息。参与者分别以 cadets、clearscope、fivedirections、theia 和 trace 等项目名称标识。

本文选取 E3 数据集中的 3 个子数据集 (cadets、fivedirections、theia) 开展评估实验。各数据集的基本信息见表 1。这 3 个子集分别基于 3 种不同的操作系统与日志收集工具构建,记录了约 10 天的系统活动 (其中包括少量攻击操作),经

数据过滤后,提取出千万量级的事件边。通过在不同的操作系统与审计工具上进行测试,可以验证算法的通用性与压缩效果。

图 9 展示了 3 个数据集中各类型事件的数量分布 (基于事件提取后的统计结果)。可以看出,不同数据集的事件分布差异显著:cadets 数据集包含其他数据集没有的特殊操作边 (如 mmap),具有较少的网络事件边;fivedirections 数据集中 read 事件占比最高;theia 数据集中网络和读写事件分布最广。通过多个不同数据集的压缩实验,可以在一定程度上证明算法的通用性。

数据集以通用数据模型 (CDM) 规范提供,每条数据都记录了系统中的实体信息或事件信息。在实验过程中,提取数据集中的所有可用实体节点,并获取按照时间排序的源节点和目的节点都可用的事件边。这些按时间排序的事件边流可用于模拟实时溯源图构建,评估压缩算法。每个事件可解析为四元组 ($s, o1, o2, type$), 其中 s 表示执行操作的进程主体、 $o1$ 和 $o2$ 表示被 s 操作的实体 (如其他进程主体或文件等客体)。生成溯源图的边至少要保证存在 s 与 $o1$, 少数事件会包含 2 个被操作的实体 $o1$ 和 $o2$ 。

表 2 列出了信息流方向与事件类型的对应关系,具体说明如下。

- 1) $s \rightarrow o1$: 进程 s 向实体 $o1$ 传递信息,如 write 表示写入文件, fork 表示创建子进程。
- 2) $o1 \rightarrow s$: 进程 s 从实体 $o1$ 获取信息,如

表 1 数据集的基本信息

数据集	系统	收集工具	持续时间	数据集大小/GB	边数 (提取后) /条
cadets	FreeBSD	Dtrace	10 d 23 h	35.7	41.35×10^6
fivedirections	Windows10	ETW	8 d 13 h	214	46.44×10^6
theia	Ubuntu	Auditd	10 d 7 h	79.3	44.52×10^6

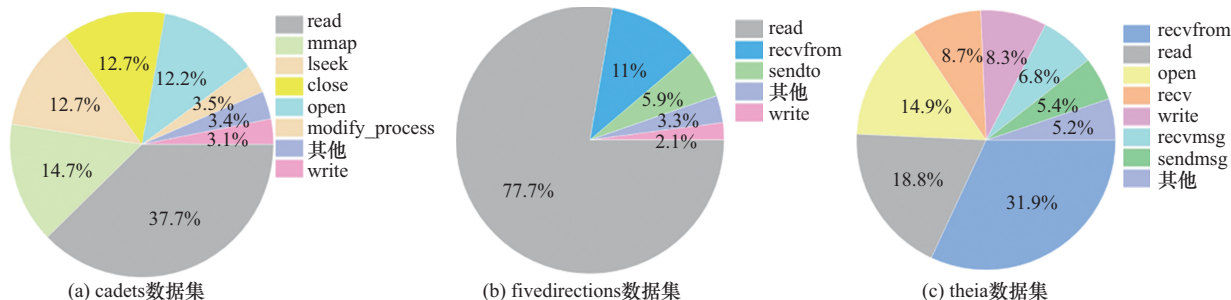


图 9 3 个数据集中各类型事件的数量分布

read表示从文件中读取信息、loadlib表示加载库文件到进程。

表2 信息流方向与事件类型的对应关系

信息流方向	对应事件类型
$s \rightarrow o1$	write、fork、clone、open等
$o1 \rightarrow s$	read、recvmsg、loadlib等
$o1, o2 \rightarrow s$	execute
$s, o1 \rightarrow o2$	rename、add_obj_attribute

3) $o1, o2 \rightarrow s$: 如execute事件表示进程 s 由父进程 $o1$ 生成, 并加载可执行映像 $o2$ 。

4) $s, o1 \rightarrow o2$: 如rename事件表示进程 s 将文件 $o1$ 重命名为 $o2$ 。

4.2 评估

本节首先评估因果关系保留压缩算法的压缩效果, 随后分析算法的运行开销, 最后通过多个攻击场景验证算法压缩后取证分析的正确性。实验选取GS算法作为对比基准, 因其是现有维护攻击调查结果的边压缩方法中效果最佳且支持实时压缩的算法(CPR与FD方法不支持实时压缩)。

(1) 算法压缩效果评估

图10展示了GS和CPC-P算法在各数据集上的压缩率(压缩边数/压缩前边数)。

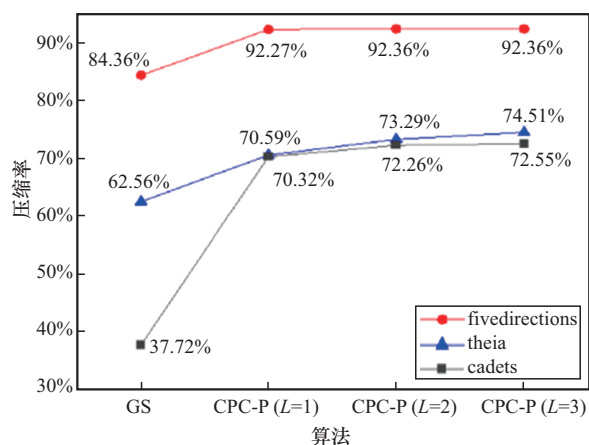


图10 不同算法在各数据集上的压缩率对比

从图10中可以看出, CPC-P ($L=1$) 在fivedirections、theia、cadets 3个数据集上的压缩率均高于GS算法压缩效果(计算公式为 $\frac{A-B}{B}$, 其中 A 、 B 分别为各算法的压缩率), 分别提高9.37%、12.84%、86.4%。在cadets数据集上, 压缩效果提升近一倍, 表明该数据集中两个节点之间循环交互较多。而CPC-P ($L=2$) 相较于CPC-P ($L=1$) 的压缩效果提升分别为0.09%、3.82%、2.75%, 说明循环冗余多发生于两个节点交互, 但随着节点依赖记录长度的增加, 压缩率仍有一定提升。尽管CPC-P节点依赖记录长度的增加带来的增益逐渐减小, 但CPC-P仍能以较少的依赖信息和较低的开销实现较好的压缩效果, 满足实时溯源图构建压缩需求。

需要注意的是, 算法仅保证节点间因果关系不变, 但压缩可能导致特殊信息流丢失。例如, 进程 P 写文件 F (P, write, F) 与修改文件权限 (P, chmod, F) 具有相同的依赖传递, 省略chmod边不会改变因果关系, 但可能导致后续分析丢失部分信息。因此, 在实际应用中通常压缩非关键操作边, 而保留chmod类的特殊操作边。

为进一步评估算法的压缩能力, 表3展示了GS和CPC算法的常见类型边(如文件读/写类型边及网络交互边)在不同数据集上的压缩因子(压缩因子=压缩前边数/压缩后边数)。其中, sum表示事件边数量。实验结果表明, 在读/写、网络传输等基本类型边的压缩上, CPC均优于GS, 且在部分场景下(如fivedirections的网络传输)有数十倍的提升。

上述实验未展示CPC-A的压缩效果。CPC-A记录了节点的所有依赖信息, 理论上是CPC-P算法能达到的最佳压缩效果。然而, 由于DARPA TC E3数据集规模庞大(各子数据集过滤后的节点数达百万级), 记录节点的全部依赖

表3 GS和CPC算法的常见类型边在不同数据集上的压缩因子

数据集	读			写			网络传输		
	sum	GS	CPC(L=1)	sum	GS	CPC(L=2)	sum	GS	CPC(L=3)
cadets	11.89×10^6	2.16	7.6	0.97×10^6	2.12	4.26	0.14×10^6	1.78	1.9
fivedirections	36.27×10^6	15.13	15.19	0.95×10^6	1.74	3.52	7.92×10^6	2.28	75.06
theia	8.36×10^6	2.13	2.7	3.67×10^6	7.54	24.52	24.38×10^6	4.74	6.94

信息将导致开销线性增长, 因此不适合将 CPC-A 用于整个数据集。为此, 研究人员从各子集中提取百万量级边数据构建新的数据集, 用于验证 CPC-A 的压缩效果。

此外, 还选取了 CPR、FD 两种主流方法进行对比。CPR 算法的实现思路如下: 若源节点 S 到目的节点 D 之间存在两条边, 且在这两条边对应的时间间隔内不存在流入 S 的边或流出 D 的边, 则可压缩其中一条冗余边, 从而保证压缩前后任意时刻的可达性不变。FD 算法放宽了条件: 保证图压缩前后, 节点任意时间的后向可达性、节点依赖变化时刻的前向可达性不变 (节点有入边时生成新版本节点, 视为依赖变化)。FD 通过构建有向无环的版本图反映依赖的变化情况, 结合版本节点优化策略减少存储与计算开销。

在新数据集上的测试结果如图 11 所示。实验结果表明, CPC-A 的压缩率高于其他算法。GS 与 FD 的压缩率相近, 均在源节点有新入边前对重复指向同一目的节点的冗余数据流进行了压缩, 但是 FD 因引入了版本节点与边, 故压缩率略低于 GS。两者均未考虑有新入边但节点依赖未变的情况, 因此无法有效解决循环交互场景的冗余边问题。CPR 对冗余边的判定条件更加严格, 故压缩率最低。

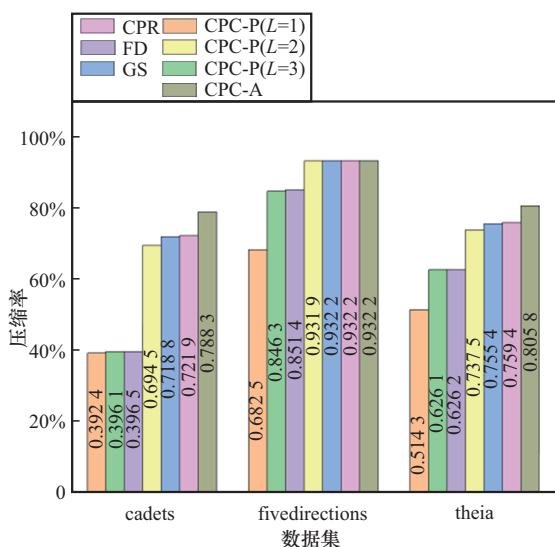


图 11 在新数据集上各算法的压缩率对比

(2) 算法运行开销评估

压缩算法的开销与节点数量有关, 因此本文选用一个包含 10 万个节点、300 万条边的数据集进行

测试, 评估算法的运行开销。评估时, 将按时间排序的溯源边的源节点、目的节点作为输入, 各算法判定百万条边是否可压缩。测试过程仅判定边是否可压缩, 不涉及数据预处理、解析与存储操作。

各算法的运行开销见表 4, 其中“内存”列是算法存储 10 万节点状态后的内存开销, “处理速度”列是根据算法处理完 300 万条边的总耗时计算得出的。

表 4 各算法的运行开销

算法	内存/MB	处理速度
GS	85	19.4×10^3
CPC-P ($L=1$)	145	24.3×10^3
CPC-P ($L=2$)	149	23.7×10^3
CPC-P ($L=3$)	152	24.5×10^3
CPC-A	20 000	9.5×10^3

实验结果表明, CPC-P 由于记录了节点的部分依赖信息, 其内存开销高于 GS 算法, 且内存开销随着依赖记录长度 L 的增加而上升, 但整体仍在可接受范围内。相反, CPC-A 记录了所有节点的依赖信息, 处理 300 万条边的内存开销达 20 000 MB 左右, 因此不适用于实时运行场景。在处理速度方面, CPC-P 略高于 GS 算法。DARPA TC E3 数据集来自真实系统, 系统每秒可生成约 50 条边, 而 CPC-P 压缩算法每秒可以处理约 24×10^3 条边, 远高于数据生成速度, 因此 CPC-P 完全能够支持实时溯源图压缩。

(3) 因果关系保留的正确性评估

本文通过对比压缩前后攻击场景的取证分析结果评估本文方法的因果关系保留能力。由于 theia 数据集缺失基本的文件类型数据, 难以直观展示节点间的数据交互, 因此本文仅在 cadets 和 fivedirections 数据集上还原了部分攻击场景, 结果见表 5, 其中攻击相关的节点数量通过前后向分析获得 (从恶意进程出发, 通过后向分析追溯攻击来源, 再进行前向分析获得攻击的影响)。表中“节点数量”列下的 CPC-P 和 CPC-A 子列, 分别表示经 CPC-P 算法和 CPC-A 算法压缩后的取证分析结果。从表 5 中数据可以看出, 压缩前与压缩后的前向/后向分析的节点数量完全一致, 表明算法保留了因果关系。

表5 部分攻击场景取证分析结果

数据集	分析类型	节点数量/个		
		压缩前	CPC-P	CPC-A
cadets	后向分析	3	3	3
	前向分析	22	22	22
fivedirections	后向分析	2	2	2
	前向分析	12	12	12

图12展示了在cadets数据集上还原的攻击场景（简化整理后），其中菱形表示网络连接，圆形表示进程节点，矩形表示文件节点。攻击过程简述如下： $T=1$ 时，攻击者控制webserver向存在漏洞的进程nginx发送恶意载荷； $T=2$ 时，nginx与shellcode主机通信，加载恶意植入； $T=3$ 时，nginx与命令和控制服务器C&C-1建立通信； $T=5\sim 8$ 时，攻击者通过受控的ngiix先后创建并提权tmux-1002、font、XIM文件，但是只有XIM文件成功执行； $T=9$ 时，XIM与C&C-2建立连接； $T=10\sim 16$ ，XIM进程在C&C-2的控制下先后执行文件删除、创建并提权等操作，生成test进程； $T=17\sim 19$ ，test进程与C&C-3通信，并通过执行uname命令和进行内部网络扫描与多台内部主机通信，获取侦察信息并将其传给C&C-3。

如图12所示，当nginx创建并提权tmux-1002文件时，发现nginx进程处于异常状态。通过对nginx节点进行后向分析，追溯到威胁来自3个未知的网络连接；再通过前向分析，可以发现后续22个（简化后）受影响的节点。实验结果显示，压缩前后的取证分析结果完全一致，证明算法能够很好地保留节点之间的因果关系。

综上，本文算法在压缩效果上优于GS压缩算法；同时，本文算法保留了节点之间的因果关系，且压缩前后的前向与后向取证分析结果一

致。CPC-A具有最好的压缩效果，但仅适用于百万量级规模的离线溯源图分析；而CPC-P适用于溯源图的实时压缩，能够大幅提升压缩效果。

5 结束语

攻击调查是溯源图的核心功能之一，保留攻击调查结果的图压缩方法已得到广泛应用，然而，现有压缩方法未能很好地解决节点间的冗余交互问题。针对这一问题，本文通过分析节点间的因果关系、记录节点依赖信息来识别冗余的依赖传递，从而提升溯源图的压缩能力，扩展算法的适用场景。此外，为解决算法效率问题，本文在CPC-A的基础上，实现了适用于实时场景的CPC-P算法。

在DARPA TC E3数据集上的实验结果表明，本文方法可以保留取证分析结果的正确性。相比于GS算法，CPC-P在增加少量内存开销的前提下，提高了运行效率与压缩效果，能够满足实时溯源图构建的压缩需求。

参考文献：

- [1] HOSSAIN M N, MILAJERDI S M, WANG J, et al. SLEUTH: real-time attack scenario reconstruction from COTS audit data[C]//Proceedings of the 26th USENIX Security Symposium (USENIX Security 17). Berkeley: USENIX Association, 2017: 487-504.
- [2] MILAJERDI S M, GJOMEMO R, ESHETE B, et al. HOLMES: real-time APT detection through correlation of suspicious information flows[C]//Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP). Piscataway: IEEE Press, 2019: 1137-1152.
- [3] XU Z, WU Z Y, LI Z C, et al. High fidelity data reduction for big data security dependency analyses[C]//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2016: 504-516.
- [4] HOSSAIN M N, WANG J, SEKAR R, et al. Dependence-preserving data compaction for scalable forensic analysis[C]//Proceedings of

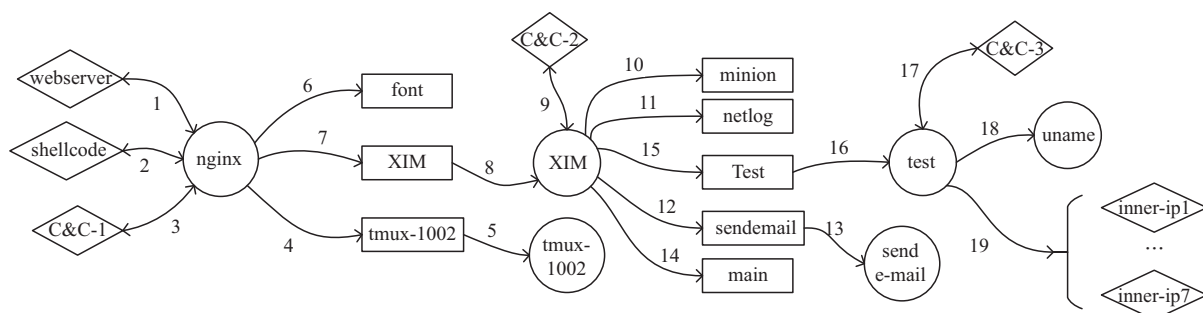


图12 CADETS攻击场景简图

- the 27th USENIX Conference on Security Symposium. Berkeley: USENIX Association, 2018: 1723-1740.
- [5] ZHU T T, WANG J Y, RUAN L Q, et al. General, efficient, and real-time data compaction strategy for APT forensic analysis[J]. IEEE Transactions on Information Forensics and Security, 2021, 16: 3312-3325.
 - [6] LI Z Y, CHEN Q A, YANG R Q, et al. Threat detection and investigation with system-level provenance graphs: a survey[J]. Computers & Security, 2021, 106: 102282.
 - [7] 冷涛, 蔡利君, 于爱民, 等. 基于系统溯源图的威胁发现与取证分析综述[J]. 通信学报, 2022, 43(7): 172-188.
 - LENG T, CAI L J, YU A M, et al. Review of threat discovery and forensic analysis based on system provenance graph[J]. Journal on Communications, 2022, 43(7): 172-188.
 - [8] ZIPPERLE M, GOTTWALT F, CHANG E, et al. Provenance-based intrusion detection systems: a survey[J]. ACM Computing Surveys, 2023, 55(7): 1-36.
 - [9] INAM M A, CHEN Y F, GOYAL A, et al. SoK: history is a vast early warning system: auditing the provenance of system intrusions[C]//Proceedings of the 2023 IEEE Symposium on Security and Privacy (SP). Piscataway: IEEE Press, 2023: 2620-2638.
 - [10] XIE Y L, MUNISWAMY-REDDY K K, LONG D D E, et al. Compressing provenance graphs[C]//Proceedings of the Workshop on the Theory and Practice of Provenance. 2015.
 - [11] LEE K H, ZHANG X Y, XU D Y. LogGC: garbage collecting audit log[C]//Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security - CCS '13. New York: ACM, 2013: 1005-1016.
 - [12] PEI K X, GU Z S, SALTAFORMAGGIO B, et al. HERCULE: attack story reconstruction via community discovery on correlated log graph[C]//Proceedings of the 32nd Annual Conference on Computer Security Applications. New York: ACM, 2016: 583-595.
 - [13] NI X Y, CHEN R R, HU M H, et al. Sardine: a threat-aware compression and querying system[C]//Proceedings of the 2023 7th Asian Conference on Artificial Intelligence Technology (ACAIT). Piscataway: IEEE Press, 2023: 1289-1296.
 - [14] DING H L, ZHAI J, DENG D, et al. The case for learned provenance graph storage systems[C]//Proceedings of the 32nd USENIX Security Symposium. Berkeley: USENIX Association, 2023: 3277-3294.
 - [15] 仇晶, 陈荣融, 朱浩瑾, 等. 基于溯源图的网络攻击调查研究综述[J]. 电子学报, 2024, 52(7): 2529-2556.
 - QIU J, CHEN R R, ZHU H J, et al. A survey of network attack investigation based on provenance graph[J]. Acta Electronica Sinica, 2024, 52(7): 2529-2556.
 - [16] TANG Y T, LI D, LI Z C, et al. NodeMerge: template based efficient data reduction for big-data causality analysis[C]//Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2018: 1324-1337.
 - [17] HASSAN W U, LEMAY M, AGUSE N, et al. Towards scalable cluster auditing through grammatical inference over provenance graphs[C]//2018 Network and Distributed System Security Symposium. 2018.
 - [18] SETAYESHFAR O, ADKINS C, JONES M, et al. GrAALF: supporting graphical analysis of audit logs for forensics[J]. Software Impacts, 2021, 8: 100068.
 - [19] MA S, ZHAI J, KWON Y, et al. Kernel-supported cost-effective audit logging for causality tracking[C]//Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference. Berkeley: USENIX Association, 2018: 241-253.
 - [20] LEE K H, ZHANG X, XU D. High accuracy attack provenance via binary-based execution partition[C]//20th Annual Network and Distributed System Security Symposium. 2013.
 - [21] HASSAN W U, GUO S J, LI D, et al. NoDoze: combatting threat alert fatigue with automated provenance triage[C]//2019 Network and Distributed System Security Symposium. Internet Society. 2019.
 - [22] 李霄翔. 基于正常行为建模的溯源图压缩算法研究与实现[D]. 北京: 清华大学, 2024.
 - LI X X. Research and Implementation of provenance graph compression algorithm based on normal behavior modeling[D]. Beijing: Tsinghua University, 2024.
 - [23] LI J W, ZHANG R, LIU J Y. ProvGRP: a context-aware provenance graph reduction and partition approach for facilitating attack investigation[J]. Electronics, 2024, 13(1): 100.
 - [24] LI Z, WEI Y, SHEN X, et al. TAGS: real-time intrusion detection with tag-propagation-based provenance graph alignment on streaming events[J]. arXiv: 2403.12541, 2024.
 - [25] XIONG C L, ZHU T T, DONG W H, et al. Conan: a practical real-time APT detection system with high accuracy and efficiency[J]. IEEE Transactions on Dependable and Secure Computing, 2020, 19(1): 551-565.
 - [26] KING S T, CHEN P M. BackTracking intrusions[J]. ACM SIGOPS Operating Systems Review, 2003, 37(5): 223-236.
 - [27] MUNISWAMY-REDDY K K, HOLLAND D A, BRAUN U, et al. Provenance-aware storage systems[C]//Proceedings of the USENIX Annual Technical Conference, General Track. Berkeley: USENIX Association, 2006: 43-56.
 - [28] GEHANI A, TARIQ D. SPADE: support for provenance auditing in distributed environments[M]//Middleware 2012. Berlin, Heidelberg: Springer, 2012: 101-120.
 - [29] PASQUIER T, HAN X Y, GOLDSTEIN M, et al. Practical whole-system provenance capture[C]//Proceedings of the 2017 Symposium on Cloud Computing. New York: ACM, 2017: 405-418.
 - [30] MOREAU L, CLIFFORD B, FREIRE J, et al. The open provenance model core specification (v1.1)[J]. Future Generation Computer Systems, 2011, 27(6): 743-756.
 - [31] MOREAU L, HUYNH T D. The PROV-JSONLD serialization: a JSON-LD representation for the PROV data model[C]//Proceedings of Provenance and Annotation of Data and Processes. Cham: Springer, 2021: 51-67.
 - [32] ALSAHEEL A, NAN Y, MA S, et al. ATLAS: a sequence-based learning approach for attack investigation[C]//Proceedings of the 30th USENIX Security Symposium. Berkeley: USENIX Association.

tion, 2021: 3005-3022.

- [33] HASSAN W U, NOUREDDINE M A, DATTA P, et al. OmegaLog: high-fidelity attack investigation via transparent multi-layer log analysis[C]//Proceedings of 2020 Network and Distributed System Security Symposium. SanDiego: NDSS, 2020.
- [34] FANG P, GAO P, LIU C, et al. Back-propagating system dependency impact for attack investigation[C]//Proceedings of the 31st USENIX Security Symposium (USENIX Security 22). Berkeley: USENIX Association, 2022: 2461-2478.
- [35] HOSSAIN M N, SHEIKHI S, SEKAR R. Combating dependence explosion in forensic analysis using alternative tag propagation semantics[C]//Proceedings of the 2020 IEEE Symposium on Security and Privacy (SP). Piscataway: IEEE Press, 2020: 1139-1155.
- [36] CHENG Z J, LV Q J, LIANG J Y, et al. Kairos: practical intrusion detection and investigation using whole-system provenance[C]//Proceedings of the 2024 IEEE Symposium on Security and Privacy (SP). Piscataway: IEEE Press, 2024: 3533-3551.
- [37] LI S F, DONG F, XIAO X S, et al. NODLINK: an online system for fine-grained APT attack detection and investigation[J]. arXiv: 2311.02331, 2023.
- [38] 梁若舟, 高跃, 赵曦滨. 基于序列特征提取的溯源图上 APT 攻击检测方法[J]. 中国科学: 信息科学, 2022, 52(8): 1463-1480.
- LIANG R Z, GAO Y, ZHAO X B. Sequence feature extraction-based APT attack detection method with provenance graphs[J]. Scientia Sinica (Informationis), 2022, 52(8): 1463-1480.

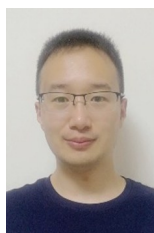
[作者简介]



张子龙 (2000-), 男, 河南驻马店人, 信息工程大学硕士生, 主要研究方向为主机入侵检测。



董卫宇 (1976-), 男, 辽宁沈阳人, 信息工程大学教授, 主要研究方向为网络空间安全。



王奕森 (1990-), 男, 河南沈丘人, 信息工程大学副教授, 主要研究方向为智能化漏洞挖掘、嵌入式系统安全。