ProvNavigator：基于影子路径引导的网络攻击调查方法

席昊[1]，范皓[2]，袁沈阳[1]，朱金宇[1]，陈昌骅[1]，万海[1]，赵曦滨[1]
（1.清华大学软件学院，北京 100084；2.中债金科信息技术有限公司，北京 100044）

摘　要：在网络攻击发生后，开展攻击调查以分析其根本原因及影响至关重要。目前，基于溯源图的技术已成为主流方法，但该方法面临依赖爆炸问题。最新研究通过整合审计日志和应用日志，在一定程度上缓解了这一问题，并展现出无须程序插桩、模型训练或污点分析的优点。然而，现有日志融合技术要么依赖复杂的融合规则，要么需要进行应用程序逆向工程，且在应对新应用时需重新调整算法，限制了其通用性。为解决这些问题，提出了一种新的基于影子路径引导的网络攻击调查方法ProvNavigator。该方法在构图阶段通过分析日志间的相关性，将不同日志源的独立溯源图合并为全局融合溯源图；在攻击调查阶段，当面对依赖爆炸的节点时，ProvNavigator利用“影子路径对”引导调查，选择适当的边进行追踪，以重构整个攻击链。ProvNavigator无须插桩或逆向分析，具备通用性。ProvNavigator的原型系统在包括4个DARPA TC数据集在内的6个真实攻击场景中进行了实验评估。实验结果显示，ProvNavigator能有效还原攻击故事，在仅有6.01%运行时开销的情况下，实现了94.3%的精确率。

关键词：攻击调查；依赖爆炸；日志融合；溯源图；影子路径
中图分类号：TP393
文献标志码：A
DOI：10.11959/j.issn.1000–436x.2025062

ProvNavigator: shadow path guided attack investigation method

XI Hao[1], FAN Hao[2], YUAN Shenyang[1], ZHU Jinyu[1], CHEN Changhua[1], WAN Hai[1], ZHAO Xibin[1]
1. School of Software, Tsinghua University, Beijing 100084, China
2. Chinabond Finance and Information Technology Co., Ltd., Beijing 100044, China

Abstract: After cyber attacks, conducting an investigation to analyze its root cause and impact is crucial. Currently, provenance graph-based techniques have become mainstream methods, but these methods face the challenge of dependency explosion problems. Recent research has alleviated this issue to some extent by integrating audit logs and application logs, showcasing advantages such as no need for program instrumentation, model training, or taint analysis. However, existing log fusion techniques either rely on complex fusion rules or require reverse engineering of applications, necessitating algorithm adjustments when facing new applications, which limits their general applicability. To address these issues, a new log fusion-based attack investigation method, ProvNavigator, was proposed. In the graph construction phase, individual provenance graphs from different log sources were merged into a global fused provenance graph by analyzing correlations between logs. In the attack investigation phase, nodes with dependency explosion were handled through “shadow path pairs,” and appropriate edges were selected to reconstruct the entire attack chain. ProvNavigator required no instrumentation or reverse analysis and demonstrated general applicability. A prototype system was developed and was experimentally evaluated in 6 real attack scenarios, including 4 DARPA TC datasets. The experimental results show that ProvNavigator can effectively reconstruct attack stories, achieving 94.3% precision with only 6.01% runtime overhead.

Keywords: attack investigation, dependency explosion, log fusion, provenance graph, shadow path

收稿日期：2024–12–26；修回日期：2025–03–25
通信作者：赵曦滨，zxb@tsinghua.edu.cn
基金项目：国家自然科学基金资助项目（No.6212780016）
Foundation Item：The National Natural Science Foundation of China (No.6212780016)

0　引言

随着信息技术的迅速发展，企业信息系统面临的网络攻击日益增多，给企业和社会造成了巨大的经济损失。因此，加强网络安全防护尤为重要。在遭受网络攻击后，应立即展开攻击调查，以识别攻击者并确定受影响的系统和数据范围。通过深入的攻击调查可以不断强化网络防御策略，防止类似事件再次发生。近年来，越来越多的研究提出利用溯源图进行安全事件的调查与分析[1-3]。

溯源图是通过审计日志生成的有向图，用以描述系统行为。图中节点分为2类：一类代表主体（如进程、线程等），另一类代表客体（如文件、注册表、网络套接字等）。溯源图中每条边表示一个事件，边方向指示数据流路径，而边的属性则包含时间戳、操作类型和操作参数等信息。图1为溯源图及其对应日志的示例[4-5]。该溯源图示例详细记录了一次结构化查询语言（SQL）注入攻击，其中攻击者IP为x.x.x.x:41402（事件①相关），此次攻击修改了位于/usr/local/db/datafile.db的敏感文件（事件②相关）。

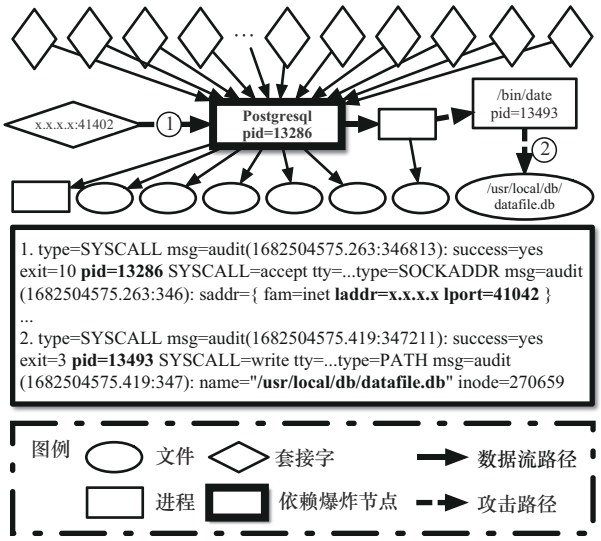


图1　溯源图及其对应日志的示例

传统基于溯源图的攻击调查方法利用前后向分析[2,6-8]：从症状事件（SE, symptom event）出发，通过后向分析追溯攻击的根源，再通过前向分析确定受影响的实体集合。这种攻击调查方法对简短且直接的攻击较为有效。但在调查例如高级持续威胁（APT, advanced persistent threat）攻击等持续时间较长的攻击时，会遇到依赖爆炸问题，即在溯源分析中依赖关系的指数增长问题。此问题常见于运行时间长的程序，如服务器应用程序。这些程序通常接收大量输入并产生大量输出，导致依赖关系急剧增加，使得在海量依赖中识别与攻击真正相关的少数对象变得异常困难。在溯源图中，这种有大量依赖关系的节点被称为依赖爆炸节点[5,9-13]（如图1中Postgresql进程节点）。如，在不加以约束的情况下，从图1中文件写入事件②沿着攻击路径分析时，会遇到依赖爆炸节点Postgresql。继续调查时，大量正常网络连接事件会淹没真正的入边攻击IP访问事件①，从而迫使安全人员遍历节点所有入边进行研判，导致攻击调查整体效率大幅下降。

为了解决依赖爆炸问题，研究人员提出了一系列方法，主要分为基于执行单元分区、模型训练和日志融合的方法。基于执行单元分区的方法利用进程结构特性将进程划分为多个执行单元[11-14]，通常依赖于程序插桩技术，侵入性较大。基于模型训练的方法通过评分或异常检测模型进行攻击调查[4,15-18]。然而，此类方法对训练数据敏感，且在解决依赖爆炸问题上效果有限。近期，提出了基于多层级日志融合（MLF, multi-layer log fusion）的攻击调查方法[19-22]。MLF的攻击调查方法不仅利用审计日志，还整合了应用程序日志。其主要优势在于能够在无须插桩和不依赖训练数据的情况下进行执行单元分区,从而解决依赖爆炸问题。其中，Hassan等[19]提出了通用溯源图的概念（OmegaLog）。通过拦截应用程序日志记录系统活动，并将其嫁接到系统层面的溯源图上，助力调查人员进行更精确的推理。Satapathy等[20]采用在服务器无感知应用上通过控制流图的精细分析实现执行分区的识别（DisProTrack)。然而，该方法依赖于精确的二进制静态分析，一旦二进制文件变化，需重新分析整个过程，从而降低了调查效率。Yang等[21]采用时间戳匹配的方法（UIScope），将由图形用户界面（GUI, graphical user interface）日志和审计日志生成的独立溯源图（IPG）进行合并，并通过GUI行为对审计日志溯源图中的依赖爆炸过程进行分区。然而，UIScope仅适用于有前端交互的程序，其应用场景非常有限。Yu等[22]设计了135条规则融合应用程序日志和审计日志（ALchemist），通过数据逻辑引擎

（Datalog）[23]推断出单元之间的相等关系和依赖关系。相等关系表示 2 个单元本质上是一个单元，而依赖关系反映 2 个单元之间有数据依赖。这个方法的主要限制在于需要大量人工设计规则。这些规则在不同应用程序间不通用。一旦出现新应用程序，便需重新分析应用程序结构来制定相应新规则。

总体而言，基于日志融合的攻击调查方法仍面临 2 个挑战。第一，通用的日志融合策略。在融合日志时应当具备高度的通用性，即在面对新应用程序时，无须对新程序进行逆向工程或专门设计融合规则。然而，不同应用程序的日志格式各异，且不同种类日志所表达的语义也存在差异，这就需要为不同应用制定针对性的融合方案。因此，设计一种能够克服应用日志间格式和语义差异的日志融合策略具有挑战性。第二，准确的依赖爆炸问题解决方法。同一进程会分时处理不同请求，而现有日志记录的粒度仅能达到进程级别，难以精确到单个请求。这就造成无法直接基于所服务请求划分同一进程中的事件，从而加剧依赖爆炸问题。因此，在不借助逆向工程分析和预定义融合规则的情况下，制定合适策略以确保在依赖爆炸节点上进行前后向分析的准确性，仍然是极其困难且具有挑战性的。

为应对上述 2 个挑战，本文提出了一种基于日志融合的攻击调查方法 ProvNavigator。该方法旨在基于症状事件展开回溯分析。当遇到依赖爆炸节点时，通过寻找影子路径对（SPP, shadow path pair）来选择合适的边，以准确重构整个攻击链。该方法涉及 2 个主要阶段。在第一阶段，基于不同类型的日志（如，审计日志、应用程序日志、流量日志等）各自构建独立溯源图。随后，通过分析日志条目之间的相关性生成日志关联图（CLG, correlated log graph），并利用该图将不同独立溯源图中的边进行关联，进而构成全局融合溯源图（HHPG, hybrid holistic provenance graph）。此方法有效应对了“通用的日志融合策略”挑战。在第二阶段，本文方法从初始症状事件出发，在全局融合溯源图中进行前向和后向分析展开攻击调查。当遇到依赖爆炸节点时，本文设计一种基于影子路径对的策略予以应对。具体而言，该策略充分利用多种日志数据，通过综合分析筛选出准确的溯源出边和入边，从而提高攻击调查的准确性。此方法有效应对了“准确的依赖爆炸问题解决方法”的挑战。由此可见，日志融合与影子路径对在 ProvNavigator 中对于高效解决依赖爆炸问题及进行攻击调查具有至关重要的作用。此外，本文方法的输出提供了丰富的关联字段说明，使得攻击调查结果具有较高的可解释性，便于安全专家快速理解重构的攻击事件流程。本文在 6 个攻击场景中评估了 ProvNavigator 的有效性。实验结果表明，该方法在遇到依赖爆炸节点时，能够准确选择相关的出边和入边。本文的主要贡献总结如下。

1) 提出了一种日志融合技术，该技术能够将不同日志源独立溯源图整合为一个全局融合溯源图。

2) 引入了影子路径对的概念，通过在全局融合溯源图中捕捉系统操作动作留下的影子路径对，并利用这些影子路径对来引导攻击调查，有效地在依赖爆炸节点周围选择正确的出边和入边。

3) 开发了 ProvNavigator 原型，并在 6 个攻击场景中进行了验证。实验结果显示，本文方法在仅有 6.01% 的运行开销下，实现了 94.3% 的精确率。

1　背景及相关工作

溯源图提供了丰富的操作系统级调用关系信息，因此许多攻击调查工作都集中在基于溯源图的研究上。然而，基于溯源图的攻击调查方法[2,6-7,24]在应对长期运行的进程节点时，常遇到依赖爆炸问题。这是因为这些进程在生命周期内与多个实体进行交互，导致其形成大量依赖关系，即众多的出边和入边。以后向分析为例，在溯源图中进行攻击调查时，若遇到依赖爆炸节点，即便仅有少数入边与攻击相关，仍须保留该节点的全部入边。若遇到多个依赖爆炸节点，所需保留入边数量将呈指数级增长，从而严重降低分析效率与准确性。近年来，国内外学者为解决依赖爆炸问题进行大量研究。然而，各项研究均存在不同程度的局限性，本文已在表 1 中对这些局限性进行了系统总结。本节将相关文献划分为 3 类，并对各类别展开详细阐述。

1) 基于执行单元分区（EUP, execution unit partitioning）的攻击调查。执行单元分区利用长时间运行进程的“无限循环”结构特性，通过将进程的

执行划分为多个小的执行单元来解决依赖爆炸问题。该方法的挑战在于如何识别单元之间的边界以及单元之间的数据流。为了解决这一问题，现有EUP[11-14, 25]通常采用二进制级别的插桩技术和源代码级别的程序注解技术。然而，二进制插桩和注解位置在软件变更时需频繁更新，这在生产环境中实施十分困难。文献[26]利用硬件支持的处理器跟踪（PT, intel process tracing）技术，有效避免了对应用程序的插桩。然而，其方法依赖于污点分析，容易受到被追踪对象特性的影响，尤其在I/O密集型程序中可能带来显著的性能开销。此外，近年来，文献[26-27]提出了重分区攻击，利用EUP防御和取证机制来混淆系统历史事件。针对上述问题，本文提出了基于应用日志划分依赖爆炸节点上边的方法，该方法无须对程序进行插桩与注解，亦不依赖EUP，从而避免引入重分区攻击。

表1 依赖爆炸问题解决方案比较

文献	插桩代码	依赖训练	应用受限
文献[11]	需要	需要	受限
文献[12]	需要	需要	受限
文献[13]	需要	不需要	受限
文献[26]	不需要	不需要	受限
文献[4]	不需要	需要	不受限
文献[15]	不需要	需要	不受限
文献[16]	不需要	需要	不受限
文献[17]	不需要	需要	不受限
文献[18]	不需要	需要	不受限
文献[28]	不需要	需要	不受限
文献[19]	不需要	不需要	受限
文献[20]	不需要	不需要	受限
文献[21]	不需要	不需要	受限
文献[22]	不需要	不需要	受限
ProvNavigator	不需要	不需要	不受限

2) 基于模型训练的方法。基于模型训练的方法分为两类：基于标记传播的方法与基于深度学习的方法。标记传播通过传播标记值来区分异常和正常活动，从而缓解依赖爆炸问题。例如，文献[4]和文献[15]通过将标记应用于症状事件，并通过数据项关联到溯源图中的相应实体，然后沿数据流传播这些标记。文献[16]则通过历史相关事件的频率为每条边分配权重，并使用扩散算法沿相邻边传播这些权重。然而，基于标记传播的方法的缺点在于其依赖于具有代表性的异常训练数据，训练数据集中的不准确性容易导致模型产生误报。基于深度学习的方法减少了对异常训练数据的依赖[17-18]。例如，文献[17]通过无监督学习，利用嵌入学习和单类支持向量机识别恶意事件，避免了对异常事件数据的标注。文献[18]利用自然语言处理和序列模型学习技术提高了模型的数据学习能力。这2种模型[17-18]能够在依赖爆炸节点上判断与可疑事件相关的边，从而提升攻击调查的效率。文献[28]通过构建精简的攻击摘要图，显著提升了攻击调查的效率。然而，如果攻击链中存在未被训练数据覆盖的零日漏洞，上述基于模型训练的方法可能失效。针对因训练数据不足导致的模型失效问题，本文提出影子路径对发现与相似度计算算法，减小对训练数据的依赖，同时在攻击调查任务中保持高精度和高召回率。

3) 基于日志融合的攻击调查方法。基于日志融合的攻击调查方法[19-22]通过引入程序应用日志作为溯源数据，并依据日志融合规则构建多层级精准的语义信息，从而缓解攻击调查中的依赖爆炸问题。基于日志融合的攻击调查方法能够有效解决基于EUP和模型训练方法的局限性，既无须对程序本身进行修改，又降低对训练数据的依赖。现有的日志融合方案、Omegalog[19]、ALchemist[20]、DisProTrack[21]和UIscope[22]已在本文引言中详细介绍，它们分别存在攻击调查效率低和适用场景有限等问题。针对这些问题，本文通过日志关联图构建全局融合溯源图的方法，提高了日志融合方案的通用性，从而提升了攻击调查效率。

综上所述，现有的3类方法均致力于解决基于溯源图进行攻击调查时遇到的依赖爆炸问题。尽管这些方法在特定场景下有效地缓解了该问题，但它们仍存在一些局限性，例如需对程序应用进行修改、对训练数据依赖较强以及算法通用性较低等。

2 ProvNavigator设计与实现

本文所提出的ProvNavigator基于审计日志和应用日志，通过构建日志关联图来制定通用的日志

融合策略，并利用影子路径对来引导攻击调查，以解决依赖爆炸问题。图 2 为 ProvNavigator 工作流，主要由全局融合溯源图的构建和影子路径对引导的攻击调查 2 个阶段组成。具体介绍如下。

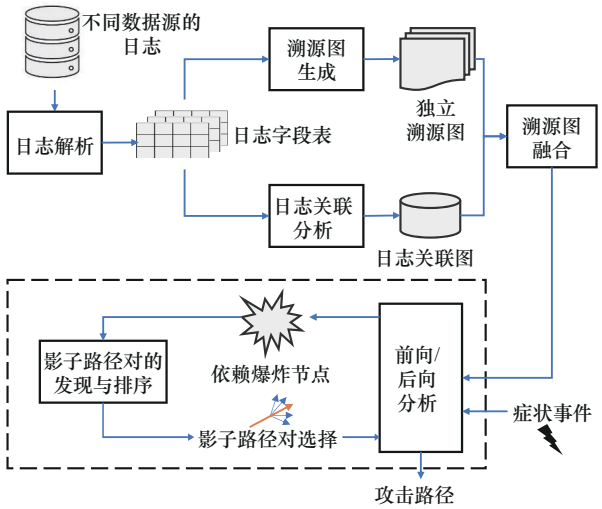
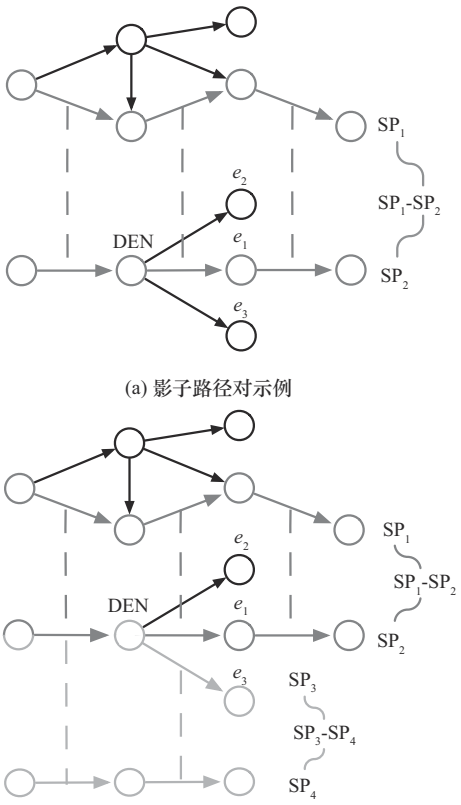


图 2　ProvNavigator 工作流

1) 全局融合溯源图的构建。ProvNavigator 为每个日志源（如审计、网络和应用程序日志）编写专用的解析器，用于将日志转换为规范日志格式（CLF, canonical log form）。ProvNavigator 基于规范化后的日志构建 IPG，通过分析日志条目之间的关联关系构建 CLG。最终，基于 CLG 将各个 IPG 整合，形成 HHPG。HHPG 的构建细节将在第 2.1 节详细讨论。

2) 影子路径对引导的攻击调查。通过实验发现，相同的系统操作动作会被不同日志源记录。以 SQL 注入操作为例，它会同时在 Web 应用程序日志、数据库日志、审计日志和网络流量日志中留下记录。因此，相同的系统操作动作会在不同日志源构建的独立溯源图中形成不同的因果路径。在全局融合溯源图中，源于同一系统操作动作的 2 个独立溯源图中的因果路径被称为影子路径对，其中每条因果路径称为影子路径（SP, shadow path）。例如，在图 3(a)中，一个系统操作动作产生了 2 条因果路径，即上半部分的影子路径 SP_1 和下半部分的影子路径 SP_2。本文称因果路径 SP_1 和 SP_2 为 1 个 SPP，并标记为 SP_1-SP_2。在图 3(a)下半部分进行后向分析遇到依赖爆炸节点（DEN, dependent explosion node）时，可以通过影子路径对 SP_1-SP_2 选择 e_1 作为出边来绕过。因此，解决依赖爆炸问题可以转化为找到最适合影子路径对的问题。为此，本文设计了 2 个算法：影子路径对发现算法和影子路径对相似度计算算法。如图 3(b)所示，通过影子路径对发现算法可以找到 2 个 SPP，即 SP_1-SP_2 和 SP_3-SP_4。由于 SPP 中的 2 条因果路径来自同一系统操作动作，它们之间存在内在联系。这种联系可通过衡量 SPP 中 2 条因果路径的相似度来表达，并用以 SPP 排序。如图 3(b)所示，影子路径对 SP_3-SP_4 的相似度较高。因此选择 SP_3-SP_4 中 e_3 所指向的节点 B 继续攻击调查过程。影子路径对于因果路径的详细定义将在第 2.2 节讨论。总体而言，基于影子路径对引导的方法显著提升了传统攻击调查的效果。从症状事件开始，在遍历全局融合溯源图过程中，不可避免地会遇到 DEN。影子路径对极大辅助了出边的选择，从而有效解决了依赖爆炸问题。



(a) 影子路径对示例

(b) 影子路径对发现算法示例

图 3　影子路径对相关示例

2.1　构建全局融合溯源图

全局融合溯源图对于后续的攻击调查至关重要。ProvNavigator 采用以下步骤构建全局融合溯源图。首先，引入规范日志格式的标准化数据结构，以此可以将来自不同来源的日志转换为统一

的标准格式，从而构建各自的独立溯源图。其次，定义日志关联图，该图用于关联不同独立溯源图中的边。最后，基于日志关联图，将不同日志生成的独立溯源图进行融合，以形成全局融合溯源图。

2.1.1 日志收集与独立溯源图构建

溯源图基于日志构建，用于捕捉系统实体及实体间的数据流动[1,3]。为了与后续定义的全局融合溯源图区分，本节首先定义由单一日志源构建的独立溯源图$G=(N,E)$，其中N是节点集，E是边集。每条边是有向的，用五元组$<n_1,n_2,r,t,\text{attrs}>$表示，其中$n_1$、$n_2\in N$，分别表示主体和客体，$r$为事件类型（如fileopen、write和read等），t为该事件发生的时间戳，attrs为事件属性（如ppid、url和path等）。边的方向与数据流的方向一致。独立溯源图可以从审计、应用程序和网络等日志中构建。为简化构建过程，本文设计一种称为规范日志格式的中间表示形式CLF。不同日志首先解析为统一的CLF格式，再基于此格式构建独立溯源图。

规范日志格式被定义为$f=(L)$，其中L是日志条目的集合。每个日志条目$e\in L$,e被表示为<start,end,time,relationtype,Fields>。其中，start、end分别是日志条目的主体和客体，time是时间戳，relationtype是事件的动作类型，Fields是包含额外信息的键值对集合$\{(\text{key,value})\}$。对于每个键值对p，p.key是键名，p.value是键值。图4展示了3种原始日志片段示例，下面给出其对应的规范日志格式示例。

```
type=EXECVE msg=audit(1679738589.407:27446795):
argc=2 a0="/bin/bash" a1="/home/alice/prov_nav/
experiment2/vcs-backend-weak/data/s1/s2/s3/malicious.tsv"
type=SYSCALL msg=audit(1679738589.407:27446795):
arch=c000003e syscall=execve ppid=2469485
pid=2469486 ··· comm="bash" exe="/usr/bin/bash"
```

(a) Auditd审计日志示例

```
1679738589405 - INFO - 1.2.3.4:59101[bob] - "POST /time/
offset/calculator HTTP/1.1" 200 Execute: os.system("/bin/
bash ~/prov_nav/experiment2/vcs-backend-weak/data/s1/s2/
s3/malicious.tsv")
```

(b) 应用程序日志示例

```
1679738589312 1.2.3.4 → 192.168.119.132 HTTP 864 POST
/time/offset/calculator HTTP/1.1  (application/x-www-form-
urlencoded)
```

(c) 网络流量日志示例

图4 3种原始日志片段示例

1)审计规范日志格式示例。图4(a)中的日志片段展示了Auditd日志中的execve系统调用事件。从原始日志中可以直接提取诸如pid、ppid和命令行等字段。可执行文件路径cmd_executable和参数路径cmd_path字段则从命令行字段中提取。表2展示了从原始审计日志解析出的规范日志格式。

表2 审计规范日志格式示例

等价键	值
timestamp	1679738589.407
start	Process_2469485
end	Process_2469486
relation_type	execve
ppid	2469485
pid	2469486
cmd	/bin/bash /USERDIR/···/malicious.tsv
cmd_executable	/bin/bash
cmd_path	/USERDIR/···/malicious.tsv

2)应用规范日志格式示例。图4(b)中的日志片段展示了一个服务器端应用程序关于命令执行系统操作动作的原始日志输出。从原始日志中可以直接提取timestamp、ip、port和url。IP地址、端口和用户名被解析并用作会话实体，归入start字段。命令执行的end字段表示已启动的进程，并由process_timestamp_session_serial表示。通过识别Web应用程序日志中相同的路径模式，可以区分出具有不同值的path字段。表3展示了从Web应用程序日志解析出的规范日志格式。

表3 应用规范日志格式示例

等价键	值
timestamp	1679738589405
start	Session_1.2.3.4_alice
end	Process_1679738589405_alice_1
relation_type	exec_cmd
ip	1.2.3.4
port	59101
url	/time/offset/calculator
cmd	/bin/bash /USERDIR/···/malicious.tsv
cmd_executable	/bin/bash
cmd_path	/USERDIR/···/malicious.tsv

3)流量规范日志格式示例。图4(c)中的日志片段展示了由tshark记录的网络流量日志。源IP和目标IP分别被解释为字段start和end。此外，规范日志格式中的字段，如时间戳、关系类型、URL和内容类型可以直接从原始日志条目中解析出来。表4展示了从网络应用日志解析出的规范日志格式。

表4　　流量规范日志格式示例

等价键	值
timestamp	1679738589312
start	1.2.3.4
end	192.168.119.132
relation_type	HTTP_POST
src_ip	1.2.3.4
dst_ip	192.168.119.132
url	/time/offset/calculator
cmd_type	/bin/application/x-www-form-urlencoded

对于每种特定的日志格式，ProvNavigator实现了不同的解析器，用以将原始多源异构日志转换为统一的规范日志格式。这使得本文方法能够处理不同应用程序生成的多种格式日志数据，提升了方法的通用性。由于不同应用或系统的日志格式通常较为稳定且不频繁变化，开发针对特定类型日志的解析器通常是一次性工作。每个日志条目字段为构建独立溯源图提供了必要信息。一旦从原始日志中提取出规范日志格式，就可以方便地构建独立溯源图。

2.1.2　日志关联分析

如前文所述，每个系统操作动作产生的事件会被不同日志源同时记录。相同事件产生的不同日志源日志条目是相关联的。为了描述这种相同系统操作动作产生事件的关联关系，ProvNavigator改进了现有日志关联分析工作（HERCULE）[29]中的CLG。HERCULE利用CLG来发现与攻击相关的社区，而本文中的CLG则被用来连接不同的独立溯源图，以构建全局融合溯源图。对此，本文的日志关联图构建算法主要在2个方面进行了改进。

首先，ProvNavigator中CLG描述相关性能力更强。HERCULE仅通过字段是否共享相同名称和值来评判日志间的关联。这种方法描述能力有限。例如，从网络流量日志条目中解析的CLF包含字段<src_ip,100.0.10.25>，而从Web应用程序日志中解析的另一个CLF包含字段<ip,100.0.10.25>。显然，这2个日志条目共享相同的IP地址，应该是相关的，但它们的字段名称不同（src_ip和ip）。因此，ProvNavigator除了等价连接之外，还引入了一种新的相关类型等价键组（KG, key group）。如果2个条目具有相同值的字段，且字段的名称属于相同的等价键组，那么这2个条目也被认为是相关的。在上述示例中，字段名称src_ip和ip属于相同的组IP，因此这2个日志条目是相关的。通过2种类型的相关性，可以建立条目之间广泛且全面的关联。

其次，ProvNavigator在构建CLG时具备更高的日志关联性能。在HERCULE中，日志关联图中的日志条目需要直接相连，因此必须遍历所有日志条目并根据预设规则进行匹配，导致时间复杂度为$O(N^2)$，其中N是日志条目的总数。相比之下，ProvNavigator通过标签哈希集合进行一次扫描即可完成日志关联。使算法复杂度降低至$O(N)$，具体复杂度计算依据将在后续算法介绍中详细解释。

本段定义了ProvNavigator中生成的CLG。给定一组CLF,其所构建的CLG定义为clg = < LN,TN,R >，其中日志节点（LN, log node）是CLF中日志条目的并集，日志节点的公式表达为$\mathrm{LN} = \{e \mid \exists \mathrm{clf} \in \mathrm{CLF}\ \text{s.t.}\ e \in \mathrm{clf}.L\}$。标签节点（TN, tag node）分为两部分：第一部分是日志条目中所有字段的并集，称为普通标签节点。第二部分是通过等价键组替换字段key得到的组标签节点，称为标签节点group。表5展示了常用的等价键组，例如，$\mathrm{Keys(PID)} = \{\mathrm{pid,ppid}\}$。标签节点被定义为

$$\mathrm{TN} = \{(\mathrm{key,value}) \mid \exists \mathrm{clf} \in \mathrm{CLF}, e \in \mathrm{clf}.L$$
$$\text{s.t.}\ (\mathrm{key,value}) \in e.\mathrm{Fields}\ \} \cup \{\ (\mathrm{group,value}) \mid \exists$$
$$\mathrm{clf} \in \mathrm{CLF}, e \in \mathrm{clf}.L, (\mathrm{key,value}) \in e.\mathrm{Fields}$$
$$\text{s.t.}\ \mathrm{key} \in \mathrm{Keys}(\mathrm{group})\ \}$$

R表示CLG中日志节点和标签节点之间的关联关系，包括日志节点与普通标签或组标签节点之间的连接，可表达为

$$R = \{\ (l,t) \mid \exists (t.\mathrm{key}, t.\mathrm{value}) \in l.\mathrm{Fields}$$
$$\text{s.t.}\ l \in \mathrm{LN} \wedge t \in \mathrm{TN}\ \} \cup \{\ (l,t) \mid \exists (\mathrm{key}, t.\mathrm{value}) \in$$

$$l.\text{Fields} \wedge \text{key} \in \text{Keys}(t.\text{group})$$

$$\text{s.t. } l \in \text{LN} \wedge t \in \text{TN} \}$$

表 5　常用的等价键组

组	等价键
IP	ip, snat_ip, dst_ip, src_ip, agent_ip
PATH	path, cmd_path, executable_path
NAME	ppname, ppuname, uname, pname, gname
UID	uid, auid, euid, ppuid
PID	pid, ppid

图 5 展示了由表 2~表 4 中 CLF 构建的 CLG 样例。例如，Web 应用日志节点有 6 个标签，网络流量日志节点有 4 个标签，审计日志节点有 5 个标签。不同日志节点通过相同键值标签节点进行关联。例如 Web 应用日志和网络流量日志节点都具有标签<IP,1.2.3.4>，这 2 个标签属于等价键组 IP。因此，这 2 个日志节点连接到同一个标签节点<IP, 1.2.3.4>上。

算法 1 是构建日志关联图的伪代码。其中，L_{all} 是包含不同来源的所有日志条目的集合，Tags 和 GroupTags 分别代表从每个日志条目 l_i 中解析出的普通标签集合 (key, value) 和组标签集合 (group, value)。函数 ParseTags 和函数 ParseGroupTags 解析出每条日志的普通标签 Tags 和组标签 ParseTags。算法首先判断标签是否已存在于哈希表中。如果标签不存在，算法将创建一个新的标签并将其添加到哈希表和日志关联图中。如果标签存在，算法将直接从哈希表中提取相应的标签。接着，函数 AddConnection 将日志条目 l_i 与相应的普通标签或组标签连接起来，以构建日志关联图。在整个算法流程中，每条日志的长度不会无限增长，因此在最坏情况下，函数 ParseTags 与函数 ParseGroup 的执行时间是恒定的，这 2 个函数的算法复杂度为 $O(1)$。此外，哈希表的插入和查询复杂度也为 $O(1)$。因此，如果有 N 条日志，构建日志关联图的算法复杂度为 $O(kN)$，其中 k 为 ProvNavigator 总共支持标签类型数目。由于 k 为常数，日志关联图构建的总体复杂度为 $O(N)$。由算法 1 构建的 CLG 用于关联在不同日志源中相同系统操作动作产生的事件，其中标签集合会被包含在 ProNavigator 的输出结果中。这些标签中的字段与数值使得输出更易理解，增强了可解释性。

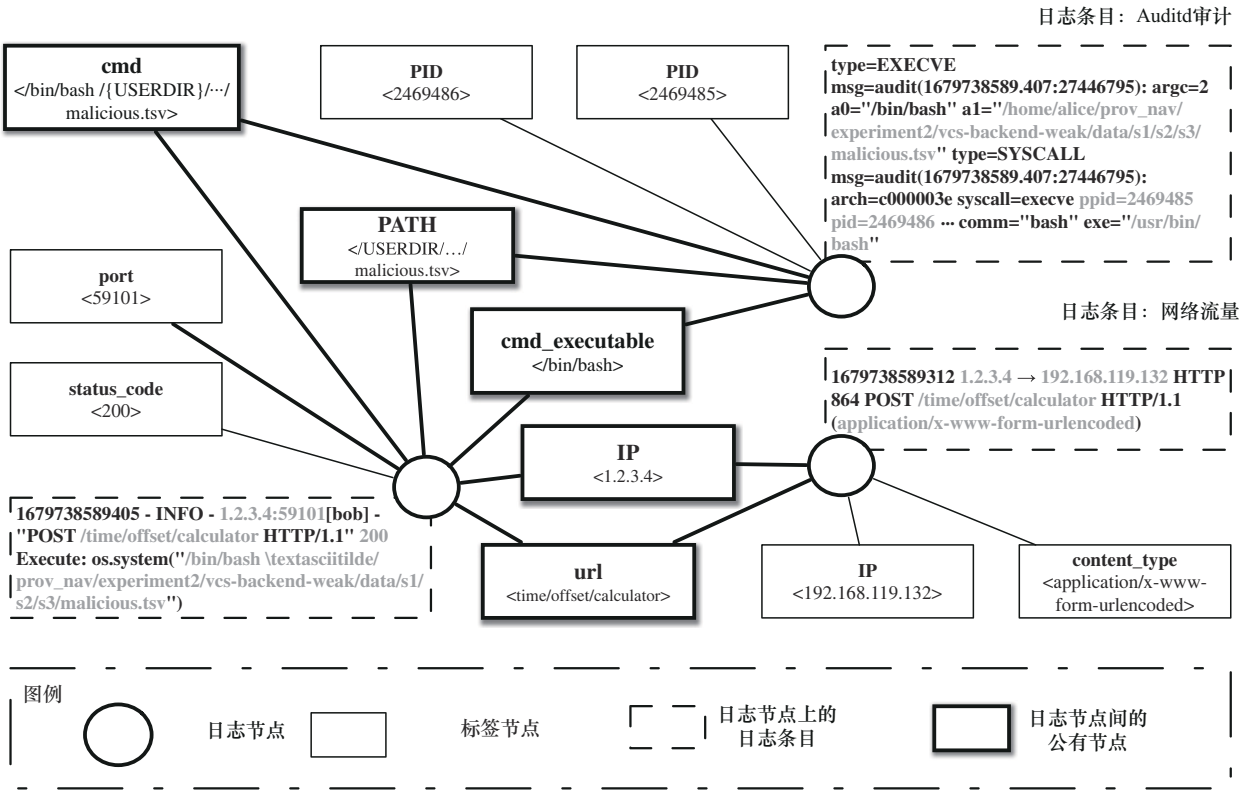


图 5　日志关联图示例

算法 1　构建日志关联图

输入　日志集合 L_{all}

输出　日志关联图 clg

1) for log l_i in L_{all} do
2)　Tags ← ParseTags(l_i)
3)　GroupTags ← ParseGroupTags(l_i)
4)　clg.insert(l_i)
5)　for Tag_i in Tags ∪ GroupTags do
6)　　if Tag_i not in hashset then
7)　　hashset.insert(Tag_i)
8)　　clg.insert(Tag_i)
9)　end if
10)　AddConnection(Tag_i,l_i)
11)　end for
12) end for

2.1.3　生成全局融合溯源图

给定一组独立溯源图 G 及其对应的 CLF 和 CLG，生成的全局融合溯源图被定义为 hhpg = < N,E,CR >，其中 N 是图中的节点，定义为 $N = \bigcup_{g \in G}\{ g.N \}$，$E$ 是每个独立溯源图上的边，定义为 $E = \bigcup_{g \in G}\{ g.E \}$。不同独立溯源图间边的关系集合（CR , correlation relation）定义为

$$\text{CR} = \{(e_i,e_j) \mid \exists g_i, g_j \in G$$
$$\text{s.t. } e_i \in g_j.E \wedge e_i \in g_j.E \wedge \exists t \in \text{clg.TN}, \ln_i \in$$
$$\text{clg.LN}, \ln_j \in \text{clg.LN}$$
$$\text{s.t.}(\ln_i,t) \in \text{clg.LN}$$
$$\text{s.t.}(\ln_i,t) \in \text{clg}.R \wedge (\ln_j,t) \in \text{clg}.R \wedge$$
$$\ln_i \sim e_i \wedge \ln_j \sim e_j\}$$

其中，ln ~ e 表示日志关联图中 ln 和独立溯源图中 e 是从规范日志格式的相同行解析出的。简而言之，对于来自不同独立溯源图的 2 条边 e_i 和 e_j，如果它们对应的日志关联图日志条目 $\ln_i$ 和 $\ln_j$ 存在等价连接，则称 e_i 和 e_j 是 CR 关联的。图 6 展示了从网络流量、应用程序和审计日志中解析出的全局融合溯源图。图 6 所示的全局融合溯源图描述了攻击者利用应用程序漏洞上传并执行恶意脚本的攻击行为。其中，菱形、矩形和椭圆形节点表示来自节点集合 N 的不同类型的实体，分别为套接字、进程和文件。有向箭头表示来自溯源边集合 E 且不跨越多个日志源实体之间的事件。无向虚线表示在不同数据源独立溯源图之间，来自日志关联图的 CR 关联。该攻击行为在不同源日志中均留下痕迹，可以构建出各自的独立溯源图。具体来说，在网络流量日志构建的独立溯源图上，攻击行为表现为攻击者 IP 1.2.3.4 访问在 IP 192.168.110.132 上运行 Web 服务的/time/offset/calculator 路径并 POST 恶意数据。在应用程序日志构建的独立溯源图上，则体现为攻击者 IP 1.2.3.4 通过一个会话上传 malicious.csv 恶意文件。在审计日志构建的独立溯源图上，该攻击表现为 bash 进程被触发并执行 malicious.tsv 恶意文件。这 3 个由不同日志源构建的独立溯源图可以通过不同标签进行关联。例如，网络流量日志溯源图和应用程序日志溯源图可以通过 IP=1.2.3.4 和 url =/time/offset/calculator 2 个标签关联，而应用程序日志溯源图和审计日志溯源图可以通过 executable_path = /bin/bash 和 cmd_path = /{userdir} /⋯/

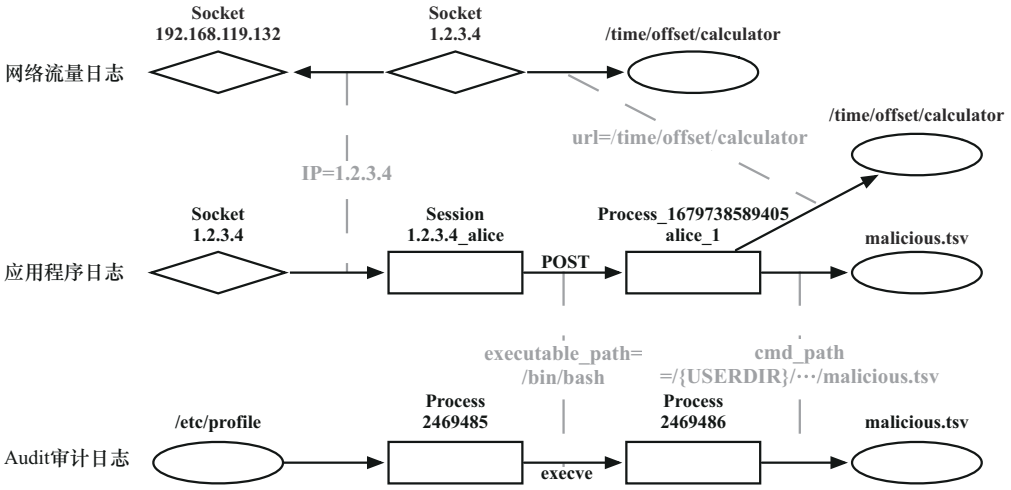


图 6　全局融合溯源图示例

malicious.tsv 2个标签关联。最终，3个独立溯源图利用日志关联图的标签相互关联，拼接成一个全局融合溯源图。

算法2描述了构建全局融合溯源图的过程。其中，clg代表日志关联图，gs代表独立溯源图集合$\{g_1,g_2,\cdots,g_n\}$。该算法首先将每个独立溯源图g中的所有节点和边添加到hhpg中。然后，对于clg中的每个日志节点ln，函数FindNeighbors识别出与ln相关联的邻近日志节点集neighbor_lns，这些相关节点通过数据库中的标签节点进行连接。最后，函数BuildCorr负责在hhpg中建立日志节点ln与其邻近节点neighbor_lns在溯源图中边的连接。

算法2　构建全局融合溯源图

输入　日志关联图clg，独立溯源图集合gs

输出　全局融合溯源图hhpg

1) for g in gs do
2)　hhpg.N.add($g.N$)
3)　hhpg.E.add($g.E$)
4) end for
5) for ln in clg. ln do
6)　neighbor_lns ← FindNeighbors(clg.ln)
7)　BuildCorr(hhpg,ln,neighbor_lns)
8) end for

2.2　路径引导的攻击调查

ProvNavigator在全局融合溯源图中进行遍历时，不可避免地会遇到依赖爆炸节点。正如之前讨论所述，解决依赖爆炸问题的关键在于如何基于依赖爆炸节点的入边（或出边）正确选择其关联的出边（或入边）。为此，本文首先提出了影子路径对的形式化定义。影子路径对能够有效地指导关联边的选择。接下来，本文设计了影子路径对发现算法用于枚举依赖爆炸节点周围的所有影子路径对。最后，本节提出了影子路径对相似度计算算法。基于该算法，可以对所有影子路径对进行排序，以便找出相似度最高的影子路径对，从而指导关联边的选择。

2.2.1　影子路径对

本节定义了与影子路径对引导攻击调查相关的基本概念。

定义1　因果路径。一个因果路径是独立溯源图中一条路径，其路径上事件时间戳呈单调递增。给定图$G=(N,E)$，因果路径p定义为$\{e_1,e_2,\cdots,e_n\}$，其中$\forall i\in[1,n-1],e_i.n_2=e_{i+1}.n_1\wedge e_i.t<e_{i+1}.t$。图7展示了一个进程先读取文件后写入文件的因果路径，其中事件时间戳沿箭头方向递增。

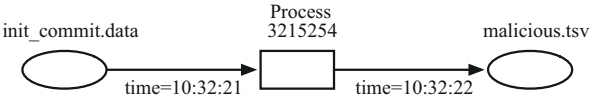


图7　因果路径示例

定义2　影子路径对。本文通过解析审计日志得到的溯源图称为审计溯源图（APG, audit provenance graph），通过解析应用程序日志或网络流量日志得到的溯源图称为高层溯源图（HPG, high- level provenance graph）。系统操作动作会在审计溯源图和高层溯源图中形成不同因果路径。影子路径对用于表示同一系统操作动作形成的因果路径。给定审计溯源图中的因果路径ap: = $\{e_{a_1},e_{a_2},\cdots,e_{a_m}\}$和高层溯源图的因果路径hp: = $\{e_{h_1},e_{h_2},\cdots,e_{h_n}\}$，如果$e_{a_1}$和$e_{h_1}$是CR相关的，并且$e_{a_m}$和$e_{h_n}$也是CR相关的，则hp和ap构成一个影子路径对。影子路径对SPP标识为一个二元组< ap,hp >。图8展示了一个影子路径对示例，其中审计溯源图和高层溯源图中因果路径的第一个和最后一个事件均相关（用虚线关联）。

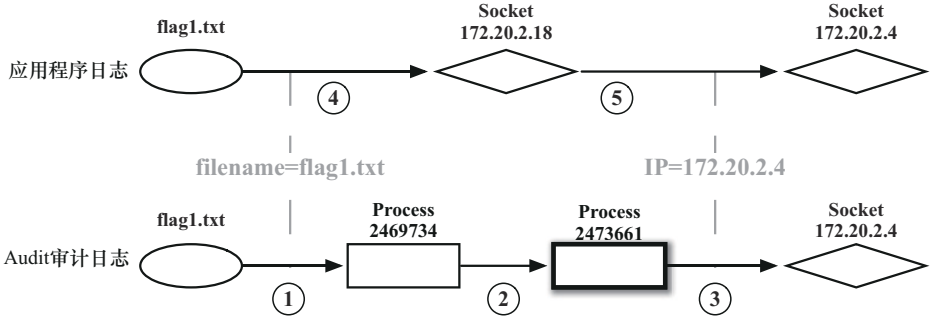


图8　影子路径对示例

2.2.2　影子路径对发现算法

在APG中，给定一条边e，可以通过以下4个步骤搜索影子路径对。1) 在HPG中找到与e相关的所有事件边 E_{start}。2) 对于每个E_{start}，执行前向分析以找到高层子图sg_h。3) 对于sg_h中的每个事件边e_h，在APG中找到与其相关的所有事件边e_{end}。4) 对于每个事件边e_{end}，执行后向分析以找到审计溯源图子图sg_a。如果在sg_a中找到包含e的子图，则可以形成一个影子路径对。算法3描述了影子路径对发现算法伪代码。其中，函数ForwardAnalysis和BackwardAnalysis是在引言中介绍的前后向分析，用于搜索sg_a与sg_h子图。给定独立溯源图集合gs，函数CorreHiEvent和CorrAuEvent通过CLG获取相关的高层和审计层事件边。函数CausalPath在给定子图及起始和终止事件边后，返回2个事件边之间的合法因果路径ap与hp。最后得到影子路径对的二元组< ap,hp >。

在构建与攻击相关的影子路径对时，能够匹配到攻击相关字段的边数量相对有限。这是因为攻击者通常不会执行大量恶意行为[9]，以避免被终端检测与响应（EDR, endpoint detection and response）发现。因此，与攻击事件相关的因果路径较少，对算法3复杂度不会产生显著影响。然而，当攻击事件解析出较多与正常事件相同的字段时，能够与攻击事件匹配的因果路径数量将增加，从而导致影子路径对发现算法的复杂度上升。为此，本文实现了影子路径对相似度计算算法，以降低影子路径对发现算法复杂度，具体内容将在下文中详细说明。

算法3　影子路径对发现算法

输入　日志关联图clg，独立溯源图集合gs，APG中的边e

输出　影子路径对集合spps

1)spps = ∅, $E_{\text{end}} = \varnothing$

2)E_{start} = CorreHiEvent(clg,gs,e)

3)sg_h = ForwardAnalysis(E_{start})

4)for e_h in sg_h do

5)　$E_{\text{end}} = E_{\text{end}} \cup$ CorrAuEvent(clg,gs,e_h)

6)end for

7)for e_{end} in E_{end} do

8)　sg_a = BackwardAnalysis(e_{end})

9)　if e is in sg_a then

10)　ap = CausalPath(sg_a,e,e_{end})

11)　hp = CausalPath(sg_h,E_{start},E_{end})

12)　spps = spps ∪ (ap,hp)

13)　end if

14) end for

2.2.3　影子路径对相似度计算算法

在算法3中，给定一条边e所找到的SPP可能不是唯一的。在这种情况下为了选出最合适的SPP来提高攻击调查结果的准确性，ProvNavigator引入了相似度（SV, similarity value）来衡量SPP中2个影子路径之间的相似性。利用相似度可以实现SPP的排序，从多个SPP中筛选出与攻击事件最相关的。给定一个影子路径对spp =< ap,hp >，其相似度通过式(1)、式(2)计算。

$$\text{SVE}(e_a,e_h) = \sum_{\text{tag} \in e_a.\text{tags} \cap e_h.\text{tags}} \frac{1}{\text{CN}(\text{tag})} \tag{1}$$

$$\text{SV}(\text{spp}) = \sum_{\substack{e_a \in \text{ap} \wedge e_h \in \text{hp} \\ \wedge < e_a,e_h > \in \text{CR}}} \text{SV} \tag{2}$$

$\text{SVE}(e_a,e_h)$衡量了e_a和e_h之间的相似性，其中CN(tag)是在CLG中标签节点tag能关联到的日志节点数量。例如，在图5中，CN(IP,<1.2.3.4>)为2，因为有2个日志节点（即应用程序日志节点和网络流量日志节点）与标签节点(IP,<1.2.3.4>)相关联。SV(spp)是SPP中所有具有CR相关的ap和hp之间SVE分数之和。在日志数据中，攻击事件相较于良性事件通常占比较低。例如，在一次针对APT攻击的事件记录中，攻击相关事件的平均占比低于0.01%[9]，其余大多数事件均为良性事件。因此，在重构攻击链时，如果遇到依赖爆炸节点，通常每个依赖爆炸节点上与攻击事件相关的标签数量会少于良性事件的标签数量，从而导致CN值较小。因此，在式(1)中，攻击事件的SVE分数相对较高，从而可以通过SVE分数从重叠标签中区分出与攻击相关的标签。此外，相同系统操作动作在不同独立溯源图中通常具有许多共同属性，存在较多的CR关联。给定溯源图上的攻击事件，通过式(2)可以叠加不同独立溯源图中与攻击相关的SVE分数，从而相同系统操作动作的SV会更高。

总体而言，较高的相似度可以在攻击调查时，用来从不同独立溯源图中筛选出与攻击最相关的影子路径对。影子路径对相似度越高，表明高层与审计日志之间相关性越强，且有助于找到与攻击相关

的影子路径对，从而提高重构攻击链的准确性。

2.2.4 攻击调查

ProvNavigator与传统攻击调查方法在处理依赖爆炸节点时存在显著区别。传统攻击调查方法在分析审计溯源图的因果路径时若遇到DEN，需要遍历该节点上的所有边。例如，在图3(b)中，DEN上所有的3条边e_1、e_2和e_3。ProvNavigator则利用影子路径对发现算法和影子路径对相似度计算算法来筛选可能涉及攻击的影子路径对。如，在图3(b)中，ProvNavigator筛选出相似度更高的SP_3-SP_4，仅需关注e_3作为下一步调查对象。此外，本文方法会将当前因果分析节点移至影子路径对中的最后一个节点。例如在图8中，攻击调查的起点为底层溯源图中的flag.txt节点，通过<filename,flag1.txt>和<ip,172.20.2.18>2个标签，找到了影子路径对①"②"③和④"⑤。通过这个影子路径对，可以将攻击调查的分析从影子路径对的第一个节点flag1.txt移至最后一个节点172.20.2.18。ProvNavigator具有高度可解释性，从全局融合溯源图构建到影子路径对的发现与排序，直至攻击故事的完整重构，ProvNavigator各步骤的运作机制对用户都是透明的。此外，ProvNavigator的结果不仅展示了在依赖爆炸节点上影子路径对的因果路径，还提供了关键词，以帮助安全专家理解选择因果路径的原因。例如，在图8中，标签<filename, flag1. txt>和<ip, 172.20.2.18>标识了依赖爆炸节点前后的关键动作，使得安全专家可以立即理解ProvNavigator在依赖爆炸节点前后的决策原因。

3 实验与评估

为了评估ProvNavigator的有效性，本文提出了以下研究问题。

研究问题1：独立溯源图之间是否普遍存在CR关联？此问题旨在探讨通过CR关联构建全局融合溯源图的通用性和可行性。

研究问题2：在全局融合溯源图中，围绕依赖爆炸节点是否存在影子路径对？这一问题关注在攻击调查过程中，遇到依赖爆炸节点时，是否可以利用影子路径对有效选择合适的入边或出边。

研究问题3：ProvNavigator是否能准确重构攻击故事？具体而言，ProvNavigator在处理依赖爆炸节点时，选择的出边或入边是否恰当？

研究问题4：进行攻击调查的系统开销如何？此问题旨在评估使用ProvNavigator进行攻击调查的计算成本及其对系统性能的影响。

3.1 威胁模型与假设

构建ProvNavigator的数据基础源自审计系统提供的多种日志类型。审计系统的重要考量之一是记录机制的安全性，因为系统入侵者可能会篡改审计日志，以掩盖其活动痕迹。与相关文献[19,22]一致，本文要求日志工具能够有效保障日志数据的完整性。具体而言，ProvNavigator和相关工作[11,13,19]一样，要求被监测应用程序及其相关操作系统日志可以保证所有攻击相关操作均能被记录。当攻击者入侵系统时，现有成熟的软件和内核强化技术[30-31]可以用来保护日志存储，从而防止攻击者篡改由多个数据源收集的日志记录。该实验前提在相关文献中亦有所体现[11,13,16]。

3.2 实验设置

本文实验在2种不同的操作系统环境下进行：Ubuntu 18.04.4和Windows 7。这2个操作系统均启用了审计和应用级别的日志模块。审计日志来源于Auditd和Sysmon工具。本文实验在一台服务器上进行，以实验测试ProvNavigator的开销和有效性。该服务器配备了Intel Xeon Platinum 8255C 2.50 GHz CPU和32 GB内存。

为了回答4个研究问题，本文在不同场景下进行了6个实验。实验使用的数据集在表6中详细描述。实验数据集包含4个由基于序列攻击调查学习方法（ATLAS）[18]开源的DARPA TC数据集（表6中数据集1~4）和2个APT案例（表6中数据集5、6）组成。其中，DARPA TC数据集通过模拟现实世界的APT事件得到，该数据集的实验在安装有32位Windows 7的虚拟机环境中进行，每轮攻击的持续时间约为1 h。APT-1与APT-2数据集模拟了现实世界中的用户注入攻击和信息窃取攻击。实验在安装有Linux Ubuntu 20.04操作系统的虚拟机中进行，每轮攻击的持续时间同样约1 h。这2个APT数据集的细节将在3.5节中进行介绍。综上所述，本文实验数据集涵盖了Linux和Windows操作系统和5种不同的应用。每个数据集都包括了3种类型的日志：审计日志、应用程序日志和网络日志。

表 6　评估 ProvNavigator 的攻击数据集概览

序号	数据集名称	漏洞场景	操作系统	应用程序	节点|边数	审计|应用日志大小/MB
1	TC-S-1	网站入侵	Windows	Firefox	8 729|35 066	62.14|319.66
2	TC-S-2	恶意广告	Windows	Firefox	37 722|90 352	261.52|753.44
3	TC-S-3	垃圾邮件	Windows	Firefox	9 896|49 711	87.10|434.83
4	TC-S-4	钓鱼攻击	Windows	Firefox	13 780|48 595	84.81|363.21
5	APT-1	用户植入	Linux	Minihttpd & Pro-ftpd	1 806|16 085	45.98|0.15
6	APT-2	信息窃取	Linux	PostgreSQL & Apache	1 010|8 327	37.58|0.64

3.3　日志关联的存在性

ProvNavigator 通过 CR 关联连接不同独立溯源图以构建全局融合溯源图，因此 CR 关系的存在性至关重要。本节通过 6 个数据集评估 HPG 和 APG 之间存在 CR 关联的普遍性。在表 7 中，第 2 列和第 3 列分别展示了 APG 和 HPG 中边的数量，第 4 列展示了 APG 中边具有 CR 关联边的比例，第 5 列展示了 HPG 中边具有 CR 关联边的比例。结果显示，在 HPG 中超过 90% 的边在 APG 中具有 CR 关联。在 APT-2 数据集中（也即表 7 中第 6 行），CR 关联边的比例相对较低。这种差异源于在这些场景中，大多数网络行为在 Auditd 审计日志中记录为本地 IP 地址（例如 127.0.0.1），而网络日志中则记录外部 IP 地址（例如 192.168.119.1）。尽管这些记录源自同一网络连接操作，但由于缺少共同标签，无法建立关联。然而，这种限制对攻击调查的影响不大，因为其他应用日志（如 access.log）记录的本地 IP 地址同样可以建立 CR 关系。综上所述，在实验数据集中不同应用程序和操作系统产生的 HPG 与 APG 中都普遍存在 CR 关联。因此，ProvNavigator 通过 CR 关联拼接独立溯源图来构建全局融合溯源图的算法是可行的。

表 7　数据集中不同层级日志间存在 CR 关联占比

序号	APG 边/条	HPG 边/条	APG → HPG	HPG→ APG
1	26 945	8 121	2.56%	90.15%
2	83 898	6 454	2.35%	98.91%
3	44 796	4 915	2.28%	99.61%
4	44 385	4 210	3.29%	99.82%
5	15 005	1 080	3.82%	99.43%
6	6326	2 001	12.14%	52.92%

3.4　影子路径对的存在性

ProvNavigator 通过分析 SPP 来绕过依赖爆炸节点。因此，依赖爆炸节点周围是否存在 SPP 对本文方法的实现至关重要。表 8 中的观察结果包括：所有数据集中均存在依赖爆炸节点（如第 2 列所示）；每个依赖爆炸节点至少关联一个影子路径对（如第 3 列所示）；平均每个依赖爆炸节点具有 2 475 个入边和 1 982 个出边（如第 4 列所示）；每个依赖爆炸节点周围的影子路径对平均数量为 224 个，远少于入边与出边的平均数量（如第 5 列所示）。综上所述，实验数据集构建的全局融合溯源图中，都存在依赖爆炸节点。每个依赖爆炸节点至少存在一个相应 SPP，使得攻击调查能够利用这些 SPP 在依赖爆炸节点上选择合适的边。因此，影子路径对在不同应用程序与操作系统间均表现出通用性。

表 8　数据集中依赖爆炸节点上影子路径对数目概览

序号	依赖爆炸是否存在	SPP 是否存在	入边|出边/条	每个节点 SPP 数量
1	是	是	3 119|1 679	536.0
2	是	是	2 963|3 952	68.0
3	是	是	4 391|2 424	67.5
4	是	是	2 591|1 700	34.3
5	是	是	1 032|825	29.1
6	是	是	752|1 313	610.0

3.5　攻击故事重构的有效性

本节评估了 ProvNavigator 是否能够利用影子路径对在依赖爆炸节点上有效选择合适的边，并重构正确的攻击故事。本文通过恶意用户注入攻击（APT-1）和信息窃取攻击（APT-2）2 个案例，展示了 ProvNavigator 重构攻击链的详细过程。实验

结果如表9所示。实验结果显示，当攻击调查遇到依赖爆炸节点时，ProvNavigator能够选择正确的入边或出边，在2个主流操作系统和不同应用程序生成的数据集上，攻击故事重构的精确率平均达到94.3%（第6列所示），召回率平均达到100%（第8列所示）。这表明本文方法在不同应用程序与系统间具备高度通用性，同时具有良好的准确性和完备性。表9展示了ProvNavigator在攻击调查过程中产生的假阳性（FP, false positive）事件数量。这些假阳性代表被错误识别为恶意的良性活动，通常是由于影子路径对选择不准确造成的。在所有数据集中，FP<10，说明本文方法的精确度较高。消融实验结果表明，影子路径对有助于减少重构攻击链中的FP，这得益于影子路径相似度计算算法在攻击事件上的筛选能力。

与ATLAS[18]中只要攻击链中包含恶意事件即视为准确不同，本文采用更细粒度的边级别评估。精确率评估了本文方法重构攻击链所有边中真实攻击事件边占重构攻击链所有边的比例。精确率越高，代表对攻击事件的判断越准确，安全专家处理误判所需时间越少。召回率评估了本文方法重构攻击链中是否包含了所有攻击事件，召回率越高代表遗漏的攻击事件越少，保证了本文方法结果的完备性。由于ATLAS开源算法中无法处理Linux日志，本文实验通过表9中1~4号Windows系统生成的数据集与ATLAS输出结果进行对比。在边级精确率与召回率计算方式下，ProvNavigator重构出的攻击链精确率较高，在1~4号数据集上平均达到97.8%，这得益于完善的影子路径对引导攻击调查机制。相比之下，ATLAS的平均精确率仅为80.4%。在召回率方面，ProvNavigator和ATLAS分别达到100%和93.5%。因此，ProvNavigator在重构攻击链时没有遗漏攻击事件，相比于现有工作[18]展现出更好的完备性。

3.5.1 APT-1（恶意用户注入攻击）

APT-1攻击针对企业客户端−服务器架构的数据版本控制系统。黑客利用恶意文件上传后门渗透系统。通过上传恶意载荷，攻击者发起反向Shell以实现未授权访问。这种连接使得攻击者在系统内获得持久化落脚点，从而辅助后续攻击操作。实验场景包括2个主要攻击步骤：攻击者利用泄露的文件传输协议（FTP）凭据上传名为attackk.sh的恶意脚本，以触发从服务器发起的反向Shell；攻击者利用超文本预处理器（PHP）服务的远程代码执行漏洞（CVE-2018- 20062），执行恶意脚本并建立反向Shell。APT-1场景通过Auditd审计日志构建APG；通过tshark记录的网络流量、Proftpd和Apache日志构建HPG。图9展示了在APT-1恶意用户注入攻击场景中，2个攻击步骤（恶意脚本上传和执行）构建出的全局融合溯源图。图中加粗的proftpd和apache2矩形分别代表2个依赖爆炸节点。当攻击者发起的反向Shell连接触发入侵检测系统警报时，ProvNavigator从这一症状事件①开始重建攻击故事。攻击调查沿着进程创建的因果路径：①→②→③进行。然而，在遇到第一个依赖爆炸节点apache2时，传统的后向分析需要遍历依赖爆炸节点上的所有边。为了在依赖爆炸节点上选择合适的回溯边，ProvNavigator利用边①的标签<ip, 192.168.1.125>和边④的标签<file, attackk.sh>匹配高层溯源图因果路径⑦解析出的标签。因此，在图9中，审计溯源图的后向因果路径①→②→③→④可以使用这一影子路径对（图9中深灰色部分所示）在依赖爆炸节点apache2上选择合适的回溯边④。

表9 ProvNavigator攻击故事重构表现

序号	攻击/良性		FP		精确率		召回率	
	节点数	边数	有SPP	没有SPP	本文	ATLAS	本文	ATLAS
1	266/8 463	2 802/32 264	9	5 067	98.9%	77.6%	100%	95.7%
2	154/37 568	6 189/84 163	0	21 629	100%	83.3%	100%	96.3%
3	2 422/7 474	9 584/40 127	3	7 514	96.9%	82.2%	100%	90.0%
4	3 050/10 730	9 787/38 808	6	4 569	95.2%	77.9%	100%	92.1%
5	67/1 739	2 687/1 339	9	12 863	82.7%	—	100%	—
6	4/1 006	6 072/2 255	1	775	92.3%	—	100%	—
平均	431/6 423	6 425/23 405	—	—	94.3%	80.4%	100%	93.5%

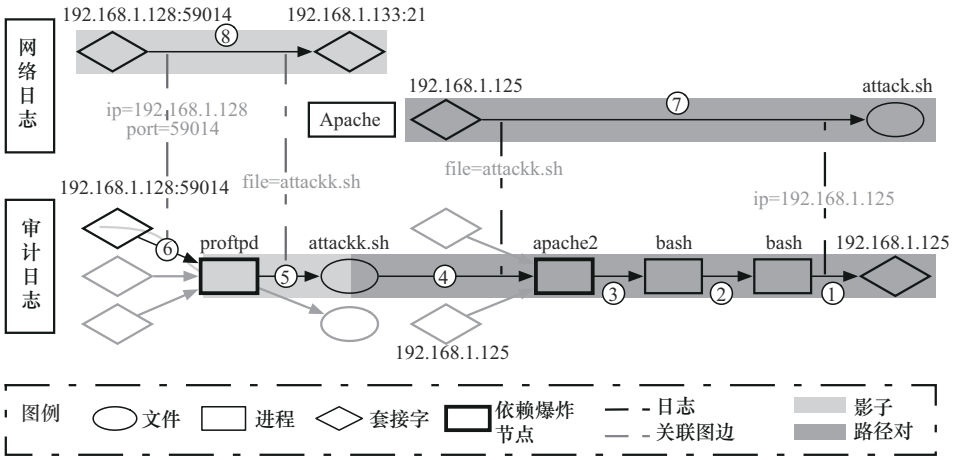


图 9　APT-1 局部全局融合溯源图示例

当从边④继续后向分析时，将遇到另一个依赖爆炸节点 proftpd。在依赖爆炸节点 proftpd 的另一侧记录了许多 proftpd 进程在文件上传过程中使用到的套接字，其中只有一个套接字与攻击行为相关。为了在这个依赖爆炸节点选择合适的回溯边，ProvNavigator 将因果路径⑤→⑥中的影子路径对与高层溯源图中因果路径⑧的标签<ip, 192.168.1.128>、<port, 59014>和<file, attackk.sh> 匹配。因此，在重构攻击故事时，ProvNavigator 能够将与攻击无关的良性套接字从调查结果中剔除。最终，完整的攻击故事序列路径⑥→⑤→④→③→②→①被准确重构。在整个调查过程中，ProvNavigator 使用影子路径对在溯源图上依赖爆炸节点有效选择合适回溯边，显著提高攻击调查的效率和准确性。

3.5.2　APT-2(信息窃取攻击)

APT-2 场景设定在一个使用浏览器-服务器架构的公司内部网站上。类似于 APT-1 攻击，APT-2 场景中的 ProvNavigator 部署在网站服务器上。通过收集包括审计日志、MySQL 数据库日志、网络请求日志及网页应用日志，构建出了如图 10 中所示的 APT-2 局部全局融合溯源图，涵盖从路径①到路径⑩。

APT-2 场景包括 3 个攻击步骤。

1) 通过 SQL 注入获得管理员权限（路径⑥、路径⑧）。

2) 获得管理员权限，通过路径遍历漏洞访问网站源代码（app.py）（路径⑤、路径⑨）。

3) 在审计网站源代码后，攻击者发现了一个远程命令执行漏洞。利用该漏洞向/etc/crontab 写入命令，以发起反向 Shell 连接，最终窃取网站服务器中的机密数据（路径①、路径⑦）。

图 10 展示了在 APT-2 信息窃取攻击场景中，包含 SQL 注入、遍历路径和反向 Shell 连接 3 个攻击步骤的全局融合溯源图。该图中加粗的矩形 Process_2469734 代表依赖爆炸节点。在网站服务器端，对敏感文件/etc/crontab 的修改触发了入侵检测系统警报。ProvNavigator 将此警报视为症状事件③。在溯源图中，网站服务端的主进程 Process_2469734 是一个依赖爆炸节点。攻击者向 Process_2469734 发送恶意请求以读取源代码 app.py。这一行为导致网站服务端主进程 Process_2469734 读取页面文件 hint.txt。随后，Process_2469734 派生出子进程 Process_2473661 用于向/etc/crontab 写入数据。本节将详细说明 ProvNavigator 如何结合后向分析和前向分析来有效完成对攻击行为的攻击调查。

后向分析：后向分析从症状事件③开始，首先遇到依赖爆炸节点 Process_2469734。此时，ProvNavigator 通过标签<filename,/etc/crontab>将事件③和事件⑦关联起来。随后，后向分析继续在节点 172.20.2.18 的 HPG 上进行。在调查 HPG 的下一跳事件时，事件⑦和⑩可以与 APG 关联。此时，可以使用影子路径对相似度计算算法选择更有可能是恶意的事件。考虑到 hint.txt 是网站后端常返回的文件，因此与之相关的良性日志条目较多。标签<filename,hint.txt>的匹配频率较高，但影子路径对相似度会降低（影子路径对相似度与标签出现的频率成反比）。相反，日志中 app.py 的出现频率较低，其相似性得分会较高。因此，ProvNavigator 选择关联⑨和⑤，并依此从 HPG 回到 APG 继续后向分析，然后找到了泄露的代码源文件 app.py。最终后向分析

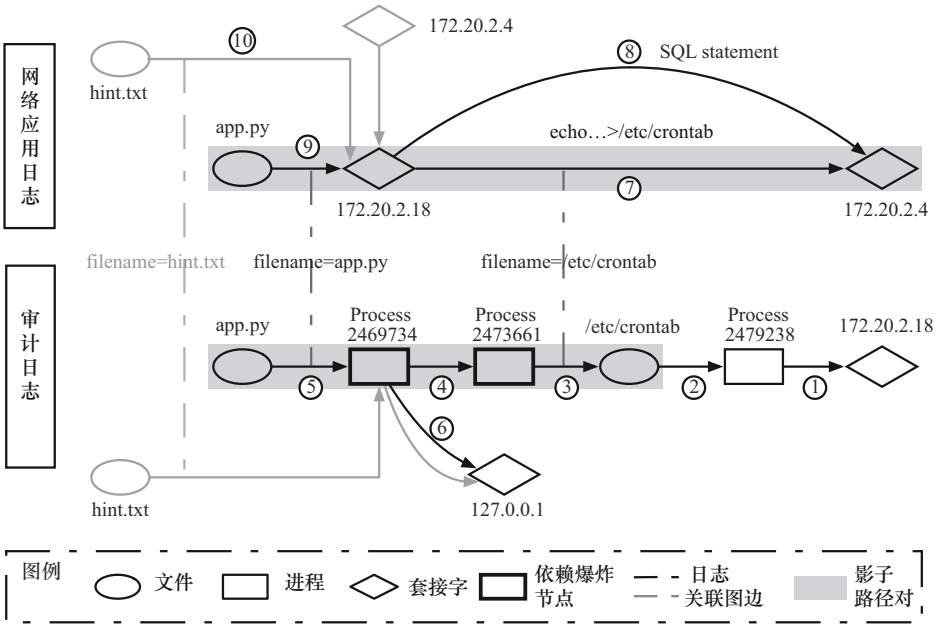


图 10　APT-2 局部全局融合溯源图示例

重构了 2 个攻击链，分别为⑤→④→③和⑨→⑦。

前向分析：ProvNavigator 在 2 个方面提高了前向分析的效果：减少误报。传统的前向分析将所有满足因果关系的可达边标记为可疑事件，容易错误地纳入大量良性活动，增加误报率。例如，网站服务端与 MySQL 通信时，从 Process_2469734 到 127.0.0.1 生成的边代表了正常且频繁的行为。由于这些活动在时间上存在因果关系，传统前向分析会将其纳入攻击链中。然而，当 ProvNavigator 前向分析遇到 Process_2469734 依赖爆炸节点时，由于缺乏影子路径对，ProvNavigator 并不会将良性事件⑥包含在攻击子图中。因此，影子路径对引导的溯源调查算法有效减少了误报。与仅利用审计日志的算法相比，ProvNavigator 能够利用来自 HPG 的高层次语义信息，提高攻击调查的效果。例如，在图 10 中，HPG 的高层次语义信息帮助识别了 SQL 注入攻击。具体而言，从反向分析中捕获的攻击者 IP（172.20.2.18）通过高层 HPG 中的前向分析揭示了恶意边⑧。此外，通过对/etc/crontab 执行的前向攻击调查，ProvNavigartor 发现了由 crontab 启动的反向 Shell。最终，ProvNavigartor 重构了全部攻击步骤：⑤→④→③→②→①以及⑨→(⑧,⑦)。

3.6　性能评估

ProvNavigator 在 2 个方面引入了额外性能开销。首先，该方法需要额外的存储空间来保存全局融合溯源图，以便后续的攻击调查。其次，前后向分析会增加响应时间开销。本节对 TC-S-1 至 TC-S-2 的 4 个公开数据集评估了相关工作 ATLAS 的性能表现，并在全部 6 个实验场景中评估了 ProvNavigator 的性能表现。

1) 存储空间开销是指存储全局融合溯源图所需的额外空间与原始日志文件的大小之比。图 11 展示了溯源图相关的存储空间开销统计。ATLAS 的平均存储开销为 9.15%。相比之下，ProvNavigator 的平均存储开销仅为 6.14%。这得益于 ProvNavigator 通过溯源图对应用与审计日志进行精简的表达。

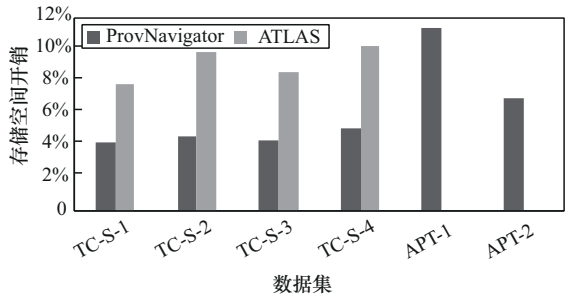


图 11　ProvNavigator 与 ATLAS 存储空间开销对比

2) 运行时开销是指在攻击调查中消耗的时间。攻击调查主要包括全局融合溯源图构建以及前后向分析。然而，构建全局融合溯源图是攻击调查之前的一次性过程，因此不会影响运行时的开销。图 12 中的结果显示，ATLAS 的平均运行时开销为 12.75%。相比之下，ProvNavigator 无须使用复杂

的学习模型推理，因此运行时开销仅为6.01%，处于可接受的范围内。

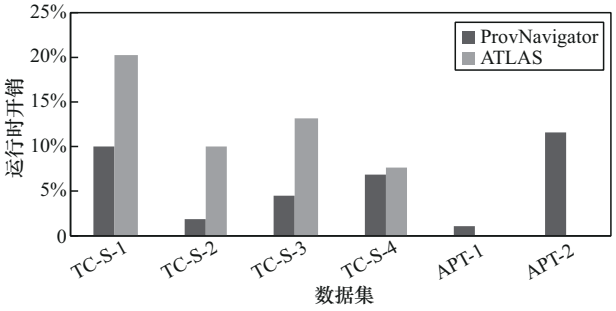


图 12　ProvNavigator与ATLAS运行时开销对比

4　结束语

为了解决基于溯源图攻击调查中遇到的赖爆炸挑战，本文提出了一个名为ProvNavigator的攻击调查方法。该方法通过日志关联分析构建全局融合溯源图，并引入了一种新颖的攻击调查方法，即通过影子路径对在依赖爆炸节点上选取合适的边。相较于先前的相关文献[16,18,22]，本文方法具有无须程序插桩、模型训练或污点分析的优点。实验评估涵盖了6个不同的攻击场景，评估结果显示ProvNavigator成功关联了各种独立溯源图，并通过影子路径对有效缓解了依赖爆炸的问题。此外，案例研究证明了ProvNavigator在较低开销下可以准确重构攻击路径。然而，目前越来越多的应用服务部署在分布式环境下，例如微服务和服务器无感知计算。ProvNavigator并未针对跨主机请求行为进行设计，因此无法准确地关联主机间请求行为以重构攻击链，我们将此问题作为未来的研究方向。

参考文献：

[1] KING S T, CHEN P M. Backtracking intrusions[C]//Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles. New York: ACM Press, 2003: 223-236.

[2] KING S T, MAO Z M, LUCCHETTI D G, et al. Enriching intrusion alerts through multi-host causality[C]//Proceedings of the Symposium on Network and Distributed System Security. Piscataway: IEEE Press, 2005: 1-12.

[3] PASQUIER T F J, SINGH J, EYERS D, et al. Camflow: managed data-sharing for cloud services[J]. IEEE Transactions on Cloud Computing, 2017, 5(3): 472-484.

[4] 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.

[5] 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.

[6] ZHANG H, YAO D F, RAMAKRISHNAN N, et al. Causality reasoning about network events for detecting stealthy malware activities[J]. Computers & Security, 2016, 58: 180-198.

[7] ZHU N N, CHIUEH T C. Design, implementation, and evaluation of repairable file service[C]//Proceedings of the 2003 International Conference on Dependable Systems and Networks. Piscataway: IEEE Press, 2003: 217-226.

[8] HOSSAIN M N, WANG J, WEISSE O, et al. Dependence-preserving data compaction for scalable forensic analysis[C]//Proceedings of the 27th USENIX Security Symposium (USENIX Security 18). Berkeley: USENIX Association, 2018: 1723-1740.

[9] 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.

[10] 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.

[11] LEE K H, ZHANG X, XU D. High accuracy attack provenance via binary-based execution partition[C]//Proceedings of the 2013 Network and Distributed System Security Symposium. Piscataway: IEEE Press, 2013: 1-16.

[12] MA S Q, ZHANG X Y, XU D Y. ProTracer: towards practical provenance tracing by alternating between logging and tainting[C]//Proceedings of the 2016 Network and Distributed System Security Symposium. Piscataway: IEEE Press, 2016: 1-15.

[13] MA S Q, ZHAI J, WANG F, et al. MPI: multiple perspective attack investigation with semantic aware execution partitioning[C]//Proceedings of the 26th USENIX Security Symposium.Berkeley: USENIX Association, 2006: 1111-1128.

[14] BATES A, HASSAN W U, BUTLER K, et al. Transparent web service auditing via network provenance functions[C]//Proceedings of the 26th International Conference on World Wide Web. New York: ACM Press, 2017: 887-895.

[15] 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.

[16] HASSAN W U, GUO S J, LI D, et al. NoDoze: combatting threat alert fatigue with automated provenance triage[C]//Proceedings of the 2019 Network and Distributed System Security Symposium. Piscataway: IEEE Press, 2019: 1-15.

[17] DING H, ZHAI J, NAN Y, MA S Q. AIRTAG: Towards automated attack investigation by unsupervised learning with log texts[C]///Proceedings of the 32nd USENIX Security Symposium (USENIX Security 23). Berkeley: USENIX Association, 2023: 373-390.

[18] ALSAHEEL A, NAN Y, MA S Q, et al. ATLAS: A sequence-based learning approach for attack investigation[C]//Proceedings of the 30th USENIX Security Symposium (USENIX Security 21). Berkeley: USENIX Association, 2021: 3005-3022.

[19] HASSAN W U, NOUREDDINE M A, DATTA P, et al. OmegaLog: high-fidelity attack investigation via transparent multi-layer log analy-

sis[C]//Proceedings of the 2020 Network and Distributed System Security Symposium. Piscataway: IEEE Press, 2020: 1-16.
[20] SATAPATHY U, THAKUR R, CHATTOPADHYAY S, et al. DisPro-Track: distributed provenance tracking over serverless applications[C]// Proceedings of the IEEE INFOCOM 2023 - IEEE Conference on Computer Communications. Piscataway: IEEE Press, 2023: 1-10.
[21] YANG R Q, MA S Q, XU H T, et al. UISCOPE: accurate, instrumentation-free, and visible attack investigation for GUI applications[C]//Proceedings of the 2020 Network and Distributed System Security Symposium. Piscataway: IEEE Press, 2020: 1-18.
[22] YU L, MA S Q, ZHANG Z, et al. ALchemist: fusing application and audit logs for precise attack provenance without instrumentation[C]// Proceedings of the 2021 Network and Distributed System Security Symposium. Piscataway: IEEE Press, 2021: 1-18.
[23] JORDAN H, SCHOLZ B, SUBOTIĆ P. Soufflé: on synthesis of program analyzers[C]//Proceedings of the 28th International Conference on Computer Aided Verification. Berlin: Springer, 2016: 422-430.
[24] BABIKER M, KARAARSLAN E, HOSCAN Y. Web application attack detection and forensics: a survey[C]//Proceedings of the 2018 6th International Symposium on Digital Forensic and Security (ISDFS). Piscataway: IEEE Press, 2018: 1-6.
[25] IRSHAD H, CIOCARLIE G, GEHANI A, et al. TRACE: enterprise-wide provenance tracking for real-time APT detection[J]. IEEE Transactions on Information Forensics and Security, 2021, 16: 4363-4376.
[26] ZENG J, ZHANG C Q, LIANG Z K. PalanTír: optimizing attack provenance with hardware-enhanced system observability[C]//Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM Press, 2022: 3135-3149.
[27] YAGEMANN C, NOUREDDINE M A, HASSAN W U, et al. Validating the integrity of audit logs against execution repartitioning attacks[C]// Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM Press, 2021: 3337-3351.
[28] 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.
[29] 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 Press, 2016: 583-595.
[30] BATES A, TIAN D, BUTLER K R B, et al. Trustworthy whole-system provenance for the linux kernel[C]// Proceedings of the 24th USENIX Security Symposium.Berkeley: USENIX Association, 2015: 319-334.
[31] HASAN R, SION R, WINSLETT M. Preventing history forgery with secure provenance[J]. ACM Transactions on Storage, 2009, 5(4): 1-43.

[作者简介]

席昊（1997– ），男，内蒙古鄂尔多斯人，清华大学博士生，主要研究方向为网络安全、异常检测、攻击溯源等。

范皓（1974– ），男，上海人，中债金科信息技术有限公司高级工程师，主要研究方向为网络安全、异常检测、攻击溯源等。

袁沈阳（2000– ），男，江苏南通人，清华大学硕士生，主要研究方向为网络安全、攻击溯源。

朱金宇（2000– ），男，江苏徐州人，清华大学硕士生，主要研究方向为网络安全、攻击溯源。

陈昌骅（1999– ），男，河南洛阳人，清华大学博士生，主要研究方向为网络安全、漏洞分析、攻击溯源等。

万海（1981– ），男，湖南郴州人，博士，清华大学副研究员、研究生导师，主要研究方向为网络安全、信息系统安全、工业网络、实时系统、形式化方法、可信软件等。

赵曦滨（1973– ），男，江苏扬州人，博士，清华大学副教授、博士生导师，主要研究方向为信息系统安全、工业网络及控制、企业信息化、人工智能、智能制造等。