ProvDP: Differential Privacy for System Provenance Dataset

Kunal Mukherjee[1]✉[0009−0004−9693−6520], Jonathan Yu[2][0009−0000−5457−0592], Partha De[1][0009−0005−0975−8744], and Dinil Mon Divakaran[3][0000−0001−8706−432X]

[1] The University of Texas at Dallas, Richardson, TX, USA
kunal.mukherjee@utdallas.edu, dparthade@gmail.com
[2] yucjonathan@gmail.com
[3] Institute for Infocomm Research, A*STAR, Singapore
dinil_divakaran@i2r.a-star.edu.sg

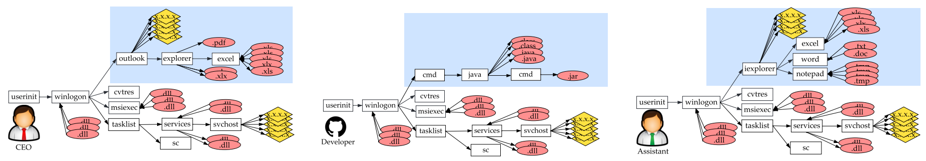
Abstract. *Provenance-based Intrusion Detection System* (PIDS) is getting widely deployed for safeguarding enterprises across various verticals against sophisticated cyberattacks such as Advanced Persistent Threat (APT) campaigns. Rich in detail, provenance data are essentially fine-grained activities of users logged at their endpoints. Therefore, a security breach of provenance data can result in the leak of private information pertaining to users and the enterprise they work for (e.g., the clients to which a user often communicates), raising significant privacy concerns. In this work, we propose a novel privacy-preserving solution, ProvDP, specifically tailored to protect the privacy of provenance data and focusing on provenance graphs. Our approach introduces multiple technical contributions: (1) a novel method for converting provenance graphs to and from provenance trees, enabling us to harness properties of trees to develop privacy-preserving techniques while preserving the semantic value of provenance graphs, (2) a novel *subtree differential privacy* framework for providing privacy guarantee on these provenance trees, and (3) empirical evidence that the application of differential privacy does not diminish the detection accuracy of PIDSes. Our evaluation demonstrates that PIDS trained on differentially private data maintain utility while preserving privacy.

Keywords: Differential Privacy · Sub-tree Differential Privacy · Provenance Analysis · Graph Privacy · Deep Learning · Privacy-Preserving Machine Learning · Sub-tree Differential Privacy.

1 Introduction

System provenance [24] has emerged as a critical component in safeguarding against sophisticated cyber threats like Advanced Persistent Threat (APT). The APT actors aim to infiltrate organizations and critical infrastructure, leading to catastrophic socio-economic consequences. The evolution of APT attacks [5, 7] has driven the development of advanced security monitoring systems, and since the influential work by King et al. [27], system provenance has solidified its role as an essential field for comprehensive system surveillance and protection. Among various defenses, Provenance-based Intrusion Detection System (PIDS) [12, 18, 33–35, 41, 46], stand out as particularly effective against APT campaigns [3, 6].

However, a significant challenge in deploying PIDS is the inherently sensitive nature of provenance data, which is composed of detailed audit logs and execution traces from an organization's endpoints. Therefore, a security breach can leak sensitive information pertaining to users and their online activities, *e.g.,* what applications do they use often, what directory do they store their sensitive information, etc. This sensitivity severely limits the sharing of provenance data across organizations, impeding collaborative development and evaluation of PIDS. For example, banks and financial institutions often encounter similar threats and attacks. By collaborating, these organizations can proactively prevent security threats that are seen by one institution but not the other.



(a) CEO (b) Developer (c) Assistant

Fig. 1: The *blue* provenance sub-graph shows the unique system interactions that cab be used to identify different users.

In system provenance analysis, provenance logs can be represented as graphs by connecting its resources and mapping its access to the resources. In provenance graphs, nodes represent system entities *e.g.,* processes, files, and network sockets, while edges denote the interactions between these entities, including actions like `CREATE` processes, `READ` or `WRITE` files, or `SEND` or `RECEIVE` data to a socket. The execution of a specific process or access to a particular file can inadvertently reveal user identification; and the attacker can infer sensitive information from the user identity. As illustrated in Figure 1, the structure of system programs in the provenance graph may appear similar across different users such as a CEO, a developer, and an assistant. However, the user-facing programs, *e.g.,* `EXCEL.EXE`, `OUTLOOK.EXE`, or `JAVA.EXE`, are distinctly different. This distinction can be exploited by an attacker to infer user identity or decrease user privacy by analyzing the graph structure and the constituent program usage.

In this work, we aim to develop a privacy-preserving approach for provenance data analysis, ensuring that the presence or absence of specific nodes and edges in a provenance graph does not compromise privacy. Our problem statement addresses this challenge by focusing on how to obscure the identification of specific user types—such as a CEO, developer, or assistant—from their activity patterns captured in the provenance graph, thereby preventing adversaries from inferring identification based on graph structure.

Specifically, we introduce a privacy mechanism based on *differential privacy* (DP) [16,17] that, in a principled and quantifiable manner, has the capability to protect the sensitive aspects of the graph. Our approach is the first to apply DP to provenance graphs, ensuring that the execution of any specific process or the interaction with any particular file remains confidential, even when the graph is subjected to detailed analysis.

Differential privacy offers a formal framework to ensure that the inclusion or exclusion of a single event in a dataset does not significantly affect the outcome of any analysis, thereby preserving individual privacy and dataset utility. While differential privacy has been successfully applied in various domains, its application in the context of system provenance presents unique challenges due to the structured nature of the data typically represented as graphs. Ensuring privacy while maintaining the utility of provenance graphs for security applications, particularly in PIDS, requires novel approaches.

One significant challenge in applying differential privacy to provenance domain is the sensitivity of edge manipulation. In a provenance graph, randomly deleting an edge is not trivial, *e.g.,* removing a process-to-process edge risks creating a disjoint graph, which could undermine the utility of the graph for security analysis. While deleting read or write edges is less detrimental, the deletion of a process creation edge, which holds substantial importance, must be deleted with respect to the context to avoid compromising the graph's structure.

Another challenge arises with edge addition as it is constrained by real-world systems. For example, adding a read edge between processes is not valid since a process can only be created or terminated, not read. The insertion of such irregular edges could result in an invalid graph that does not reflect feasible real-world scenarios. Provenance graphs are context-sensitive, where each edge type is specific to particular node types. Ensuring that any modifications maintain the natural connections is essential to preserve the validity and utility of the graph. Finally, the last and the most critical challenge is that the graph manipulation while providing privacy should not remove the implicit behavioral signals captured in the provenance graphs, which are used by PIDS to learn.

In this work, we design and develop ProvDP, a novel privacy-preserving framework specifically designed for provenance data analysis, with a focus on provenance graphs. ProvDP is built on three key technical innovations: *(1)* a novel method for converting provenance graphs into provenance trees and back, allowing us to utilize the structural properties of trees to develop privacy preserving techniques while maintaining the semantic integrity of the original graphs, *(2)* a novel *subtree differential privacy* algorithm to provide privacy guarantee (proved in §A.9) on provenance trees, and *(3)* a rigorous evaluation demonstrating that the application of differential privacy maintains the detection accuracy of PIDS [33]. In summary, our research makes the following contributions to the domain of system provenance and differential privacy:

- *Differential Privacy Framework. (§5)* We introduce a novel framework for applying differential privacy to provenance graphs, tailored for the unique challenges of system provenance data.
- *Problem Space Feasibility. (§5.6)* We introduce domain-specific validation rules to ensure that the differentially private data realistically represents problem space behavior.
- *Privacy-Utility Trade-off Study. (§6)* We conduct the first focused study on the impact of differential privacy on system provenance datasets, highlighting the trade-offs between privacy and utility.

To facilitate reproducibility and benefit the research community, we have released our codebase[4]. This release includes the FiveDirections DARPA Transparent Computing (TC) provenance dataset used in system provenance research [21, 22, 33, 43], as well as post-processing scripts that convert the dataset into the Deep Graph Library (DGL) [2] data loading framework for ease of use.

2 Background

System Provenance. System provenance is the field of tracking fine-grained system data (*e.g.,* `syscall`), from large enterprise and industrial systems. It traces information flow and control dependencies starting from a Point-Of-Interest (POI) event, associating dependent system events both forward and backward directions, which enables forensic analysis and advanced security defenses [12, 18, 22, 27, 29, 33, 34, 34, 35, 41, 43, 44, 46]. By examining system-call logs [1, 4, 11], system provenance graphs are created by capturing relationships (*i.e.,* `READ`, `WRITE`, `CREATE` and `EXECUTE`) among major resources (*i.e.,* process, files and network sockets). Provenance graphs offer significant advantages over traditional logs as the interactions define robust structural features defined that pose a significant hurdle for adversaries attempting to manipulate the system.

Formally, a provenance graph is a connected set of timestamped edges $e = (u, v, r)$, where $u, v \in \{processes \cup files \cup sockets\}$ and v is causally dependent on u (*e.g.,* a file u is written by a process v), and r is the relationship between the nodes. Therefore, the POI event is represented as $e_{poi} = (u_{poi}, v_{poi}, r_{poi})$. Nodes and edges can be annotated with attributes such as executable names, file paths, and IP addresses, making these graphs invaluable tools for forensic analysis such as discovering points of entry and infection propagations.

Provenance-based Machine Learning (ML) Research. System provenance, initially proposed to automate forensic investigations, has evolved into a crucial foundation for ML-based security detectors, driven by rapid advances in data analysis techniques. The numerous EDR (Endpoint Detection and Response) products [9, 10, 15, 38] are able to gather fine-grained details at user endpoints, such as processes, network connections, execution of commands, changes of registry, *etc.* Latest research works [13, 18, 20, 33, 41] have modeled the endpoint

[4] `https://github.com/provdp/prov-dp`

log data using models ranging from Bi-LSTM to transformers to Graph Neural Networks (GNNs).

Differential Privacy. Differential Privacy (DP) [16, 17] provides a framework for protecting individual privacy in datasets. It has been shown that simple anonymization is insufficient for preventing privacy breaches [14, 36, 37, 45], as such data can often be re-identified when combined with external information. DP addresses this issue by introducing statistical noise to obscure individual contributions, ensuring privacy guarantees.

DP is typically implemented within two main models: the centralized model, where a trusted server collects and processes raw data, and the local model, where users perturb their data locally before sharing it. The latter, known as Local Differential Privacy (LDP), ensures privacy even if the server or analysis infrastructure is compromised. Under LDP, an ϵ-local randomizer $\mathcal{R} : \mathcal{D} \to \mathcal{T}$ maps each user's data $v \in \mathcal{D}$ to a perturbed value in $\mathcal{T}$, satisfying: $\Pr[\mathcal{R}(v) = t] \leq e^{\epsilon}\Pr[\mathcal{R}(v') = t], \quad \forall v, v' \in \mathcal{D}, \forall t \in \mathcal{T}$, where ϵ is the privacy budget, quantifying the trade-off between privacy and utility. This ensures that observing $\mathcal{R}(v)$ provides little information about whether the original data was v or v', even when auxiliary information is available.

Differential Privacy for Graphs. Differential privacy analysis of graph data has largely focused on the centralized model [16, 25, 26, 39, 45], with two primary privacy definitions: node privacy and edge privacy.

Node privacy [25] ensures indistinguishability between two neighboring graphs $\mathcal{G}$ and $\mathcal{G}'$, where $\mathcal{G}'$ is obtained by removing a single node and its edges from $\mathcal{G}$. This provides strong privacy guarantees by making it nearly equally probable (differing by a factor of e^{ϵ}) for an adversary to infer that $\mathcal{G}$ or its neighbor was the input to the randomizer $\mathcal{R}$. Edge privacy [40] or k-edge privacy [23], on the other hand, focuses on protecting the presence of specific edges or k edges, offering weaker but often more practical privacy guarantees.

Node and edge DP is well-suited for many graph types, but it is not directly applicable to provenance graphs. Provenance graphs capture system constrained causal relationships between nodes, and removing a node can disrupt these relationships and removing an edge or k-edges can create disjoint graphs. As a result, novel privacy definitions and algorithms are required for analyzing provenance graphs, as we explore in subsequent sections.

3 Problem Statement and Threat Model

3.1 Problem Statement

This research addresses the challenge of *preserving user privacy* and *reducing the risk of user identification* in provenance graphs. Given a provenance graph $\mathcal{G}_i$, an adversary aims to determine whether a specific user $u \in \mathcal{U}$ (where $\mathcal{U}$ is the set of all users) is associated with $\mathcal{G}_i$. The adversary may leverage the detailed structure of $\mathcal{G}_i$ and auxiliary information $\mathcal{A}$ to infer the presence or absence of u, breaching user privacy.

ProvDP mitigates this risk by applying a differentially private mechanism $\mathcal{R} : \mathcal{G} \to \mathcal{G}'$, where $\mathcal{G}$ is the space of all possible provenance graphs and $\mathcal{G}'$ is the space of differentially private provenance graphs. The mechanism $\mathcal{R}$ satisfies ϵ-differential privacy, ensuring that for any two neighboring graphs $\mathcal{G}_i$ and $\mathcal{G}_j$ differing by the presence or absence of a single user's session data, the probability of inferring u's inclusion is bounded by e^{ϵ}, where ϵ is the privacy budget.

Given the above definition, the core problem is whether an adversary, equipped with $\mathcal{A}$ and knowledge of $\mathcal{R}(\mathcal{G}_i)$, can infer user identities from a differentially private provenance graph. Our goal is to develop a methodology that introduces noise and perturbations to $\mathcal{G}_i$, preserving its utility for legitimate security applications while significantly reducing the adversary's ability to identify u.

3.2 Threat Model

We assume a white-box threat model where adversary has domain knowledge, defense internals, DP mechanism internals, and access to auxiliary information, such as statistical data or prior distributions. The adversary can use this information to infer user identities from differentially private provenance graphs.

Adversary Knowledge. The adversary is highly skilled and has access to auxiliary information, such as publicly available datasets and statistical patterns within provenance graphs (*e.g.,* program lineage, causal relationships, and subgraphs). They have complete knowledge of the defense system and the differential privacy mechanism. Their goal is to identify whether specific entities or events are present in a differentially private graph.

Adversary Capability. The dedicated adversary has access to public provenance datasets and sufficient system resources. They can make refined queries to incrementally enhance their inferences, aiming to detect specific entities.

4 ProvDP Motivation

We will modify a State-of-The-Art (SOTA) graph-DP algorithm [39] for provenance domain and an analyze its shortcomings. Finally, use the learnings to generate insights for developing ProvDP.

4.1 Baseline approach

As our straw-man approach, we adapt the Top-M Filter (TmF) [39], which was designed for perturbing sparse social network graphs under an edge-differential privacy guarantee. The TmF algorithm is composed of two differentially private algorithms: the first perturbs the number of edges E in the graph G, $m = |E|$, using Laplace noise under budget ϵ_1, to produce perturbed $\tilde{m}$. The second algorithm is composed of two steps. The first step filters out existing edges by computing $1 + Lap(1/\epsilon_1)$ for each edge and if the value is less than θ the edge is removed. In the second step, two nodes are randomly selected and an edge is added if there is no existing edge between them.

Unlike social graphs, system provenance graphs are directed, acyclic and no self-loops exist. We adopt TmF to provenance domain constraints as described in Algorithm 1, and call it Extended Top-M Filter (ETmF). We show in §A.5 that ETmF is also edge-differential private. We first partition nodes into disjoint sets by their type: $V = V_p \cup V_f \cup V_i$ for process, file, and IP, respectively. Please note, $\forall e_i(u_i, v_i) \in E$, $V_s = \bigcup_{i=1..|V_s|}[u_i]$ and $V_d = \bigcup_{i=1..|V_d|}[v_i]$.

In ETmF, we first generate the set of all possible edges $E_{possible}$ by enumerating edges such that $E_{possible}$: $(u_{possible}, v_{possible}) \in V_s \times V_d$. This might introduce self-loop in the graph. We then remove the infeasible edges (*i.e.,* self-loop edges E_{sl}), thereby obtaining the set of valid edges E_{valid} and then run ETmF for each edge in E_{valid}. When choosing a random edge to add, we take a random sample from E_{valid}. This process is repeated until the graph contains the desired $\tilde{m}$ edges. This can lead to a graph with disconnected components; therefore only the component containing the source node of POI event *i.e.,* u_{poi} is kept.

Analysis. The privacy guarantee produced by ETmF are undesirable since it only provides edge-DP, an insufficient guarantee in provenance domain. For example, even if there is no edge from `JAVA.EXE` to `helloworld.java`, the presence of that file shows the type of user interacting with it. Additionally, ETmF produces in infeasible provenance edges. For example, this method can result in a process with multiple incoming `CREATE` edges from other processes to a single process, which is infeasible in the real world. While this issues can be addressed by rejecting infeasible relations, but it would leak sensitive information and incorporate bias as we are interfere with the randomization scheme.

4.2 Insights for developing ProvDP

To overcome the shortcomings in ETmF, we must *(1)* provide subtree-differential privacy, and *(2)* generate feasible graphs without rejection sampling. Our key observation is that provenance graphs are inherently tree-like structures. Trees are more desirable because they are innately simpler structures as they are acyclic, and each node has at most one incoming edge. This makes it much simpler to modify trees at a node level. For example, if we were to attach a single node to an existing node in a graph, the new node could also have edges to all other existing nodes in the graph. When appending a node to an existing node in a tree, we cannot add any additional edges without breaking the tree structure, so there is only one place it can be added, which is at the leaf level. Additionally, our graph-to-tree conversion process ensures that resource nodes (files and IP sockets) are *always* located at leaf nodes. We then leverage this fact to perturb our trees without rejection sampling.

Following this insight, we create the provenance graphs via two distinct breadth-first-search (BFS) procedures. First, we choose a POI event e_{poi} and perform BFS from v_{poi}, only considering events *forward* in time that originate from the current node and we perform another BFS from u_{poi}, only considering events *backward* in time that originate from the current node. Finally, we join the two graphs produced by the BFS procedures to create a complete provenance graph. This means the graphs thus generated are inherently tree-like in nature (*e.g.,* two graphs attached by e_{poi} event), except subgraphs where is a loop created by a process read and writing to a same file or network socket.

5 ProvDP Design

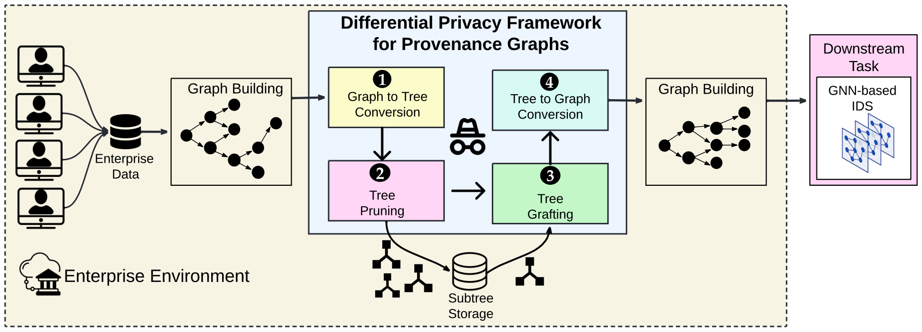


Fig. 2: ProvDP overview.

In this section, we describe the design of ProvDP, as illustrated in Figure 2. ProvDP is a post-processing framework that enterprises can use after they have collected their data and generated provenance graphs. These graphs are intended to be shared for downstream tasks, *e.g.,* integration with cloud-based *Intrusion Detection System* (IDS) [8]. ProvDP is ϵ-Subtree-DP as proved in §A.9 and is composed of four distinct stages:

(1) Graph to tree conversion
(2) Tree pruning
(3) Tree grafting
(4) Tree to graph conversion

In ProvDP, we first convert graphs to trees to simplify manipulation and then prune and graft subtrees to obscure a user's activity in a session, providing differential privacy (DP) guarantees.

5.1 Stage 1: Graph to Tree Conversion

We assume that the input provenance graphs are directed, acyclic, and that each process only has a single incoming edge from another process. The graph to tree conversion stage has three steps.

1. Break Cycles: To eliminate all cycles, all outgoing edges of a resource (*e.g.,* file, and IP) nodes are reversed. This is a lossless operation, because the edge direction can be identified from the edge type. For example, file read edges are always outgoing, and the converse is true for write edges.

2. Remove Lattice Structure: If a resource node has multiple incoming edges, duplicate the node and revise the edges to point to different nodes. For example, given two process nodes n_{p_1}, n_{p_2}, and resource node n_r, consider the two edges $e_1 = (n_{p_1}, n_r)$ and $e_2 = (n_{p_2}, n_r)$. The resource node would be duplicated, and an edge would be updated to point at the new node, resulting in $e_1 = (n_{p_1}, n_r)$ and $e_2 = (n_{p_2}, n'_r)$. Recall that process nodes never have more than one incoming edge from another process Therefore all nodes in the current graph have at most one incoming edge. Due to the node duplication, this graph may be disconnected, so it is currently a forest of trees.

3. Forest to Tree: To convert a forest into a single tree, we add a single "virtual" root node. We then create an edge between each disjoint tree's root and the virtual root node, after which we have a singular tree.

5.2 Stage 2: Tree Pruning

A system provenance tree captures the behavior of a user during a session, where a session is defined as the user's activity over a specific period on a particular machine. In the context of system provenance, user behaviors are reflected in the subtrees of the provenance tree. However, classical Differential Privacy (DP) [16, 25, 26] is defined in terms of neighboring databases, and ϵ-DP quantifies the notion of *indistinguishability* between neighboring databases. Inspired by ϵ-DP, we introduce the concept of ϵ_1-Subtree-Differential Privacy (ϵ_1-Subtree-DP) for provenance graphs. We define two trees, T_1 and T_2, as neighbors if they differ by one subtree T_v, rooted at a node v. Notably, if we prune (or remove) a subtree rooted at level 1, an entire session is removed. In this case, our framework aligns with the classical definition of neighboring datasets [16], where removing all user's sessions from a provenance graph is analogous to removing an entire user's data from a database.

Definition 1. *A randomizer R is ϵ_1-Subtree-DP if, for any pair of tree neighbors T_1, T_2, and for any output $x \in Range(R)$:*

$$\Pr[R(T_1) = x] \leq e^{\varepsilon_1} \Pr[R(T_2) = x],$$

where $\varepsilon_1 \geq 0$ is the privacy budget.

The Laplace mechanism [16] is a standard mechanism in DP. It is based on the concept of *global sensitivity* of a function f, defined as $\Delta f = \max_{T_1, T_2, t} ||f(T_1, t) - f(T_2, t)||_1$, where T_1, T_2 are all pairs of neighboring trees. Given a function f and a privacy budget ϵ, the noise is drawn from a Laplace distribution $p(x|\lambda) = \frac{1}{2\lambda} \exp^{-|x|/\lambda}$ where $\lambda = \Delta f / \epsilon$.

ϵ_1-Subtree-DP. The single point prune tree algorithm described in Algorithm 2 prunes a subtree $T_{v_{chosen}}$ rooted at node v_{chosen} from tree T to generate pruned tree $\tilde{T}$. Algorithm 2 introduces a probabilistic process to determine whether a subtree should be pruned from the tree during the traversal. For each node, v, a random number r is drawn from a normal distribution between 0 and 1, *i.e.,* $r \sim \mathcal{N}(\mu, \sigma^2)$, $r \in [0, 1]$. This random number is compared against the computed pruning probability, denoted as *pruning_prob*, which determines the node's pruning eligibility. *pruning_prob* is derived from the logistic function:

$$pruning\_prob = \frac{1}{1 + e^{\frac{\epsilon_1}{2S}}}$$

where ϵ_1 is the privacy parameter, and S is the sensitivity of the subtree rooted at node v. A smaller ϵ_1 enforces stricter privacy by increasing the likelihood of pruning and vice-versa.

The sensitivity, S, reflects the influence of the subtree within the tree, T_v, and is computed using the following equation: S = 1 / ($\alpha \cdot$ size + $\beta \cdot$ height + $\gamma \cdot$ depth + $\eta \cdot$ branching_factor), α, β, γ, and η being tunable hyper-parameters that weigh the importance of the node's structural properties—size of subtree rooted at v, height of subtree rooted at v, depth of v and branching factor of the subtree rooted at v, respectively, and $\alpha + \beta + \gamma + \eta = 1$.

Intuitively, a large distance indicates less noise needs to be added, and a small distance indicates more noise. The logistic function ensures that nodes with smaller sensitivity values S, representing less critical subtrees, are more likely to be pruned. As S decreases, the term $e^{\frac{\epsilon_1}{2S}}$ grows larger, reducing the value of *pruning_prob* and increasing the likelihood of pruning. Conversely, for nodes with larger S, *pruning_prob* increases, thus decreasing the probability of pruning. This balance allows us to modulate the pruning behavior based on the node's structural importance and the privacy requirements. Therefore, bigger subtrees are less probable of being pruned, and vice-versa. This lets us preserve the overall tree structure while pruning the parts containing privacy-sensitive information like files and external network socket connections.

Theorem 1. *Algorithm 2 is ϵ_1-Subtree-differentially private and proved in §A.6.*

ϵ_k-Subtree-DP. Empirically we have seen that a user performs similar activity in multiple sessions, so pruning one subtree (*i.e.,* from one session) is not enough protect the user's privacy—there are other subtrees (*i.e.,* different sessions) that can contain identifying information. Therefore, we prune the tree at k places, so that k sessions are affected. This will lead to a decrease in utility but it provides a stronger privacy guarantee.

We define two trees T_1 and T_2 as k neighbors if they differ by k subtrees. Therefore, we define ϵ_k-Subtree-Differential Privacy (ϵ_k-Subtree-DP) for provenance graphs as following.

Definition 2. *A randomizer R is ϵ_k-Subtree-DP if for any pair of tree neighbors T_1, T_2, and for any output $x \in Range(R)$:*

$$\textit{we have } \Pr[R(T_1) = x] \leq k \cdot e^{\varepsilon} \Pr[R(T_2) = x],$$

where $\varepsilon \geq 0$ is the privacy budget.

Algorithm 3 extends Algorithm 2 to allow k-subtree pruning (*e.g.,* prune the tree T at k places) to hide k user distinguishable patterns and we prove that it provides ϵ_k-Subtree differentially privacy (§A.7).

We begin by initializing a set $Pruned\_SubTrees$ and a key-value structure called bucket B to contain pruned subtrees, with the key being the size of the pruned subtree. We follow Algorithm 2 to prune tree T at k points to generate pruned tree $\tilde{T}$. When we prune a subtree T_s rooted at r from tree T, we keep track of the parent node of r, v_p, and the edge e_p connecting v_p and r. T_{vp} is the pruned subtree T_s with the addition of node v_p and edge e_p. We add (T_{vp}, v_p, e_p) to the set $Pruned\_SubTrees$. The presence of v_p and e_p is needed to preserve the legal relationships (*i.e.,* valid edges according to provenance domain) of the pruned subtrees in $Pruned\_SubTrees$.

After completing this stage, we have a pruned tree $\tilde{T}$ and a set $Pruned\_SubTrees$ containing the pruned subtrees. The pruned subtrees in $Pruned\_SubTrees$ hold valuable patterns, and pruning k of them may reduce utility. To mitigate this, we graft back some subtrees to preserve those patterns.

Theorem 2. *Algorithm 3 is ϵ_k-Subtree-differentially private and proved in §A.7.*

5.3 Stage 3: Tree Grafting

Keeping the downstream utility in perspective, if k is too large, we will decrease the utility to an abnormally low value by disrupting important patterns present in the tree that is utilized by downstream applications, *i.e.,* IDS. We need to graft back pruned subtrees to preserve signals in the tree. Therefore, the third stage of our algorithm involves grafting k' pruned subtrees from $Pruned\_SubTrees$ back into the pruned tree $\tilde{T}$, described in Algorithm 4.

$\epsilon_{k'}$-Subtree-DP. We prove that after grafting we get $\epsilon_{k'}$-Subtree differentially privacy, where k' is the number of times grafting has happened. After grafting, we expect some of the key signals that were removed due to pruning to be present again in the tree but at a different location, since the root node of the subtree will be different. Since, the IDS are trained to learn similar patterns which are considered benign, the presence of the signals in the tree are important.

Algorithm 4 takes as input a modified tree $\tilde{T}$, a set of pruned subtrees $Pruned\_SubTrees$, and a privacy budget ϵ_2. We initialize a variable k' to zero that will contain the number of times grafting has taken place. This algorithm iterates over each node v in $\tilde{T}$ that was marked for pruning during the prior pruning step (Algorithm 3). For each such node v, the subtree size s_v rooted at v before pruning is noted, and noise sampled from a Laplace distribution is added to this size to yield $\tilde{s}_v = s_v + Lap(1/\epsilon_2)$. A weight W is then computed as the absolute difference $|s_v - \tilde{s}_v| + 1$. The probability of grafting, $grafting\_prob$, is then defined as the inverse of W, ensuring that subtrees with smaller differences to $\tilde{s}_v$ have higher grafting probabilities. A random value r is sampled from a Gaussian distribution, bounded within $[0, 1]$, to decide whether grafting should occur. If $r < grafting\_prob$, a random subtree $\tilde{t}$ of size $\tilde{s}_v$ is chosen and grafted back into $\tilde{T}$ at node v. We count the number of times grafting has occurred and store the information in k'. Therefore, k' is always less than k.

All resource nodes in $\tilde{T}$ are leaf nodes, and edges between non-leaf nodes originate from process nodes. As a result, the parent nodes of any marked nodes in $\tilde{T}$ are guaranteed to be process nodes, which ensures that only valid process-to-process relationships are preserved during grafting. Each subtree $\tilde{t} \in Pruned\_Trees$ contains the root node v and its parent node v_p, connected by the edge e_p. When grafting $\tilde{t}$ to $\tilde{T}$ at node v_T, the algorithm modifies the edge e_p to connect v_T to v, effectively updating e_p to (v_T, v) and discarding v_p. This adjustment preserves the integrity of process creation relationships ensuring adherence to provenance constraints and eliminating the need for rejection sampling.

An argument can be made that grafting rare or uncommon program creation edges might introduce artificial signals, potentially reducing the dataset's utility. However, our evaluation shows that with optimal hyperparameter, the subtrees are grafted at shallower depths, where the programs are well-known user programs with diverse behavioral patterns. Therefore, the subtree grafted are not creating unnatural relationships that can introduce unintended signals.

Theorem 3. *Algorithm 4 is $\epsilon_{k'}$-Subtree-differentially private and proved in §A.8.*

5.4 Stage 4: Tree to Graph Conversion

Once we have a dataset of fully perturbed trees $\tilde{T}$, we convert the trees back to graphs. First remove the virtual node and its edges which results in a disconnected forest. Recall that we duplicated resource nodes during the graph-to-tree conversion. To reverse this, we join all resource nodes with matching labels. The final step is to revert the resource edges to their original direction. Edges are defined with an event type, thus we can use this type to determine the direction of the edge. We now have a dataset of *differentially private* provenance graphs.

Theorem 4. *ProvDP is ϵ-Subtree-differentially private, where $\epsilon = \epsilon_k + \epsilon_{k'}$ and proved in §A.9.*

5.5 Sensitivity and Utility tradeoff

In the ProvDP framework, the trade-off between sensitivity and utility is a critical aspect of ensuring differential privacy while maintaining the integrity of the provenance data. By adjusting the sensitivity bound in the pruning stage, smaller, more privacy-sensitive subtrees are subjected to higher perturbation, while larger, less sensitive subtrees remain relatively intact, preserving the overall structure of the provenance graph. The allocation of the privacy budget between the pruning and grafting stages (ϵ_1 and ϵ_2) allows for fine-tuning of this balance, where a lower sensitivity bound can enhance privacy but at the cost of increased noise and potential loss of utility. To measure this inherent trade-off between sensitivity and utility we designed experiments (*i.e.,* §6.4 and §6.1) containing ablation study to measure how varying different privacy parameters (*e.g.,* ϵ, δ, α, β, γ and η) will affect the pruning properties, tree modification, and downstream impact.

5.6 Problem Space Feasibility

Applying Differential Privacy (DP) in the provenance domain poses unique challenges due to the strict semantic and structural requirements of system provenance graphs. Unlike other domains, random edge addition or deletion can render provenance graphs invalid by disrupting their critical relationships. In ProvDP, we address these challenges by pruning only process nodes, ensuring structural integrity and preventing invalid edges. The graph-to-tree conversion stage further guarantees that no invalid edges arise during grafting, as the hierarchical tree structure inherently mitigates such risks.

Our subtree pruning approach preserves key behavioral signals while introducing differential privacy. By carefully allocating the privacy budget between pruning and grafting, we perturb the most sensitive parts of the graph while maintaining its utility for security analysis and intrusion detection. This balance ensures that the resulting graphs remain both private and useful.

6 Evaluation

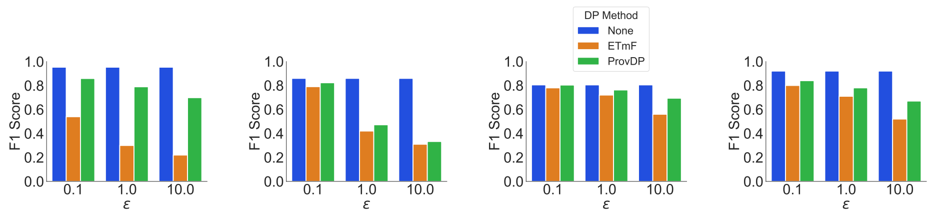
To comprehensively evaluate ProvDP, we investigate these research questions:

- **RQ1: Downstream Utility Impact.** How does ProvDP impact the performance of GNN-based IDS on provenance datasets (§6.1)?
- **RQ2: Sensitivity.** What is the trade-off between privacy sensitivity and utility using ProvDP (§6.2)?
- **RQ3: Parameter Exploration.** How does the different parameters of ProvDP impact the privacy guarantee (§6.3 and §6.4)?
- **RQ4: Overhead.** What is the overhead of ProvDP (§6.5)?

Methodology. We select the downstream task of APT detection using GNN-based IDS released by [33]; our experiments show that by applying ProvDP framework to the benign dataset the detection accuracy of the PIDS is not compromised. First, we analyze the critical trade-off between the privacy budget and detection accuracy loss, using metrics such as F1 score. Second, we conduct an ablation study to explore the impact of ProvDP's hyperparameters (*e.g.,* ϵ, δ, α, β, γ, and η) on the detection accuracy. We vary $\epsilon = \epsilon_1 + \epsilon_2$ and control the allocation of the budget between the two algorithms (Algorithm 3 and Algorithm 4) by varying the δ parameter such that $\epsilon_1 = \delta\epsilon$, and $\epsilon_2 = (1-\delta)\epsilon$. Third, we look at the overhead of ProvDP. Finally, we demonstrate that our novel graph-to-tree conversion algorithm achieves high coverage and look at the tree's characteristics and to delve deeper, we analyze the impact of hyperparameters on the characteristics of the pruned subtree also.

Downstream Task. We utilize four APT datasets similar to previous provenance research [33], three from the DARPA TC datasets [19] and [33] (Linux). These DARPA datasets were referenced by prior studies [12,18,33,41,46]: FiveDirections (Windows), TRACE (Linux), and THEIA (Linux). The DARPA dataset contained attacks that were carried out in systems with diverse workloads to simulate realistic benign background environment and were specifically designed to target systems with long-running processes, capturing the stealthy vectors often used by sophisticated adversaries. The DARPA datasets exceed 40 GB in size and comprise more than 36 million nodes and 100 million edges.

Baselines: To further investigate the impact of the privacy budget on detection performance, we evaluate against two distinct differential privacy algorithms. The baseline algorithm employs adapting TmF [39], we refer to it as ETmF (described in §4.1), which offers edge-differential privacy but results in the worst performance due to the disruption of the provenance graph structure and creation of unnatural edges, which significantly hampers the detection model's accuracy. Next, we evaluate using ProvDP (described in §5), using the same privacy budget to demonstrate that it strikes an optimal balance between robust detection performance and privacy protection.



(a) FiveD (b) Trace (c) Theia (d) [33]

Fig. 3: Detection performance of GNN-based IDS using different privacy budgets.

6.1 Downstream Impact

The downstream impact is shown in Figure 3, where we show the performance of GNN-based IDS from [33] on different differentially private APT datasets. The datasets without any DP ("None") consistently exhibits superior performance, serving as a baseline for comparison. Both ETmF and ProvDP show notable degradation in performance as the privacy budget ϵ increases, which somewhat deviates from the expected behavior in differentially private mechanisms. We believe during the grafting phase subgraphs are grafted to unrelated processes resulting in rare subgraphs and decreasing the accuracy. For instance, ETmF's F1 score on the FiveDirections dataset drops significantly from 0.54 at $\epsilon = 0.1$ to 0.22 at ϵ= 10, indicating a substantial impact of increased privacy levels on the model's accuracy. ProvDP, although demonstrating higher downstream utility compared to ETmF, also experiences a decline in performance, particularly on the TRACE dataset. ProvDP generally outperforms ETmF across the datasets, as it contains domain specific constraints and balanced approach between privacy and utility.

At a privacy budget of $\epsilon = 0.1$, ProvDP demonstrates superior performance compared to ETmF across all three datasets, indicating that ProvDP offers a more effective balance between privacy and utility at this level of privacy protection. Specifically, on the FiveDirections dataset, ProvDP achieves an impressive F1 score of 0.83, significantly outperforming ETmF's score of 0.54. Similarly, for the TRACE, THEIA, and [33] datasets, similar trends are seen. These results suggest that at lower epsilon values, where privacy protection is stronger, ProvDP is more effective than ETmF in preserving the data utility.

At a higher privacy budget of $\epsilon = 10$, both ProvDP and ETmF exhibit a decline in performance, an unexpected outcome given the increased budget, and the gap in their effectiveness becomes more pronounced. We believe this unexpected trend is shown due to rare subgraph structure being formed during grafting stage. ProvDP continues to outperform ETmF, albeit with reduced margins. On the FiveDirections dataset, ProvDP maintains an F1 score of 0.75, while ETmF's performance deteriorates to 0.22, highlighting ProvDP's superior ability to handle higher privacy budgets while maintaining model accuracy. The TRACE dataset shows a similar trend, with ProvDP scoring 0.33 compared to ETmF's 0.31. On the [33] dataset, ProvDP achieves an F1 score of 0.69, outperforming ETmF's 0.52. These comparisons indicate that even at lower epsilon values, where privacy constraints are stringent, ProvDP offers a more robust performance compared to ETmF.

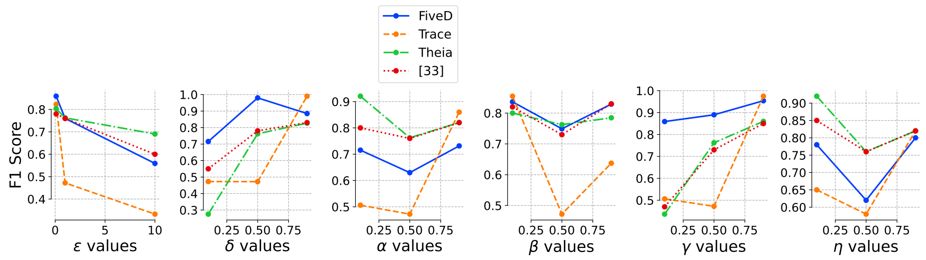


Fig. 4: Detection performance of GNN-based IDS using different hyperparameters. $\epsilon = 1, \delta = 0.5, \alpha = 0.5, \beta = 0.5, \gamma = 0.5$, and $\eta = 0.5$ are fixed, unless explicitly varied.

6.2 Ablation Study: Privacy Budget vs. Utility

Effect of ϵ. The impact of various hyperparameters on the performance of [33] is shown in Figure 4. As ϵ increases from 0.1 to 10, there is a noticeable fluctuation in the performance (*i.e.,* F1 score) metrics. Specifically, at $\epsilon = 0.1$, the FiveDirections dataset shows a peak F1 score of 0.95, indicating that lower privacy budget results in lower utility. However, for all the datasets, F1 score decreases sharply as ϵ increases to a large number suggesting that higher privacy budgets (lower privacy) may not always yield better results. As mentioned above, we believe that during the grafting phase, some subgraphs are attached to unrelated processes, leading to the formation of rare subgraphs that cause false positives and reduce accuracy. Thus, ϵ appears to have a strong impact, highlighting the main challenge of DP, how to maintaining high accuracy while ensuring strong privacy guarantees. In future work, we plan on using FrequencyDB [22] to ensure that only related subgraphs are grafted.

6.3 Ablation Study: Parameter Exploration

The impact of the δ reveals an interesting trend where lower values (indicating stronger privacy guarantees) do not consistently degrade performance across all datasets as seen in Figure 4. For instance, at $\delta = 10$, the TRACE dataset achieves an impressive performance of 0.94, which diminishes significantly with lower δ values. Conversely, the FiveDirections dataset shows peak performance at $\delta = 0.95$, suggesting that intermediate values of δ provide an optimal balance. The varying influence of δ across datasets underscores its role in determining the robustness of the privacy mechanism without uniformly affecting all datasets.

The parameters α, β, and γ exhibit distinct effects on the performance, with some clear patterns. For instance, β has high impact on the THEIA dataset, peak performance at 0.79 for $\beta = 0.1$ and $\beta = 0.9$. On the other hand, α and γ show more dataset-dependent behavior. Notably, $\alpha = 0.5$ and $\gamma = 10$ yields consistently good results across all datasets, indicating a relatively stable performance metric. It can be further stated that α, γ, and η parameters are related to each other since size, depth and branching factor of the subtree are interrelated. These observations suggest that while α, γ, and η contribute similarly to fine-tuning performance, but β effects are nuanced and dataset-dependent. Therefore, β emerging as a critical parameter for optimizing outcomes.

6.4 Effect of Hyperparameter on Pruning

Table 1: Tree statistics for the DARPA TC datasets.

Metric	**FiveD**	**Trace**	**Theia**	[33]
% of successful conversion	100%	100%	100%	100%
Tree Statistics				
Avg. Height	6.59	6.70	6.22	4.73
Avg. Diameter	8.09	9.44	8.97	7.19
Max. Degree	1001.00	972.00	653.74	1001.00
Avg. Degree	292.37	408.83	400.21	399.14
Avg. Node Depth	2.83	2.54	2.71	2.96

Graph to tree conversion rate. All graphs in the DARPA TC datasets were successfully converted to tree. Our algorithm §5.1 was specifically designed for the system provenance domain focusing on domain-specific constraints, we achieved a 100% converge.

Tree Statistics. The tree statistics presented in Table 1 indicate that the FiveDirections and TRACE datasets have similar average heights of 6.54, in contrast to THEIA's average height of 4.73. This suggests that the THEIA dataset produces more compact trees, indicating a less dense structure. Additionally, both the TRACE and THEIA datasets exhibit a higher average degree than FiveDirections. This difference can be attributed to the fact that TRACE and THEIA are derived from Linux environments, while FiveDirections is from a Windows environment, where graphs tend to be less dense. Also, the diameters of the datasets are similar, implying that the trees are more stretched out and less centralized. Since these trees capture user program behavior, a single tree can encompass multiple behaviors, leading to a more spread-out tree structure.

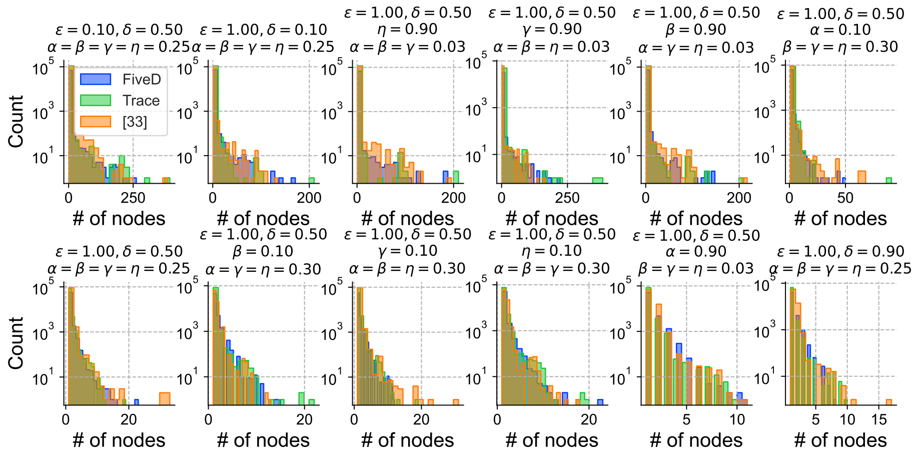
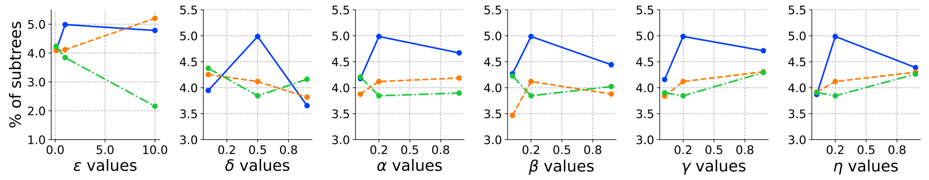
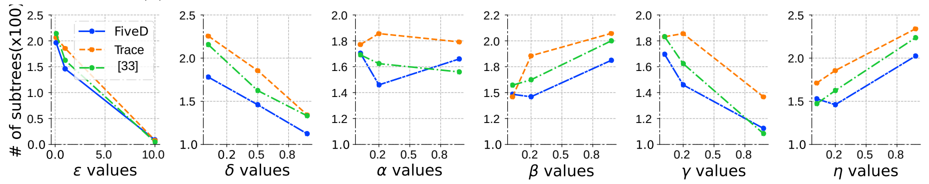


Fig. 5: Distribution of pruned subtree size based on different hyperparameter. Theia dataset is omitted due to showing similar trend as Trace and $\epsilon = 10$ is omitted due to insufficient samples due to heavy pruning.

Distribution of subtree pruned. The distribution is heavily biased toward smaller-node subtrees, as seen in all the subplots in Figure 5. As discussed in the previous section, most trees in the datasets have a large diameter compared to their height, resulting in a wide spread and many leaf nodes. Consequently, these leaf nodes are marked for pruning at a higher rate than subtrees at greater heights. The parameters ϵ, δ, and α have the most significant impact on the size of the pruned subtrees. As observed in the first two subplots, as ϵ increases, the size of the pruned subtrees decreases (e.g., when ϵ increased from 0.1 to 1.0, the range of the x-axis decreased from 200 to 100). A similar pattern is evident when comparing the second subplot with the last subplot: as δ increases, the size of the pruned subtrees decreases. This is expected, as increasing ϵ and δ increases the privacy budget, leading to the pruning of progressively smaller subtrees due to reduced perturbation of subgraphs. Finally, comparing the eighth and ninth subplots, we observe that increasing α helps discretize the buckets, causing only subtrees of specific sizes to be pruned, with some sizes not being pruned at all.



(a) Effect of hyperparameters on the percentage of unmoved subtrees.



(b) Effect of hyperparameters on number of subtrees perturbed.

Fig. 6: $\epsilon = 1.0, \delta = 0.5, \alpha = 0.5, \beta = 0.5, \gamma = 0.5$, and $\eta = 0.5$ are fixed unless explicitly varied. Theia dataset omitted due to showing similar trend as Trace.

Effect of hyperparameters on subtrees. When a subtree is placed in a bucket, it is marked with the ID of the graph it originally came from. After reattachment, a subtree is considered unmoved if it is placed into its original tree. It is clear to see from Figure 6a that all datasets, regardless of the choice of privacy budget, hover around 0.1% of subtrees unmoved. This is because the amount of single-node subtrees present in the bucket crowding out all other subtree sizes. We observe that as ϵ increases, the number of unmoved subtrees decreases significantly, especially in the FiveDirections and [33] datasets, indicating that higher ϵ values lead to greater perturbation but of small number of subtrees as seen in Figure 6b. Changes in α, β and γ show little impact on the percentage of unmoved subtrees, effectively showing that ϵ is the most critical parameter for controlling the trade-off between privacy and the structural integrity of provenance graphs.

The analysis of number of subtree perturbations in relation to different hyperparameters reveals ϵ exhibits a pronounced effect on the number of perturbed subtrees across all datasets, with a dramatic reduction observed as ϵ increases. For example, in the [33] dataset, the number of perturbed subtrees plummets from 443.24 at $\epsilon = 0.1$ to almost zero at $\epsilon = 10$. This finding underscores the role of ϵ in controlling the degree of perturbation applied to the provenance graph, with higher values leading to almost complete disruption of subtrees, thereby compromising the graph's utility.

In contrast, δ and γ show a general trend of decreasing perturbation with increasing parameter values, the impact is less severe compared to ϵ. Additionally, parameters α and β display varying levels of subtree perturbation across different datasets, indicating that their influence is more dataset-specific. For instance, in the TRACE model, α has a noticeable but moderate effect, with perturbed subtrees decreasing from 268.12 to 201.04 as α increases from 0.1 to 0.9. Interestingly, γ displays stronger impact than α and β, particularly in the TRACE and [33] datasets, where the number of perturbed subtrees decreases more noticeably with higher γ values. However, the reduction is less drastic compared to ϵ and δ, indicating that while γ is important among α, and β, its effects are less impactful than hyperparameter that control the privacy budgets.

6.5 Overhead of ProvDP

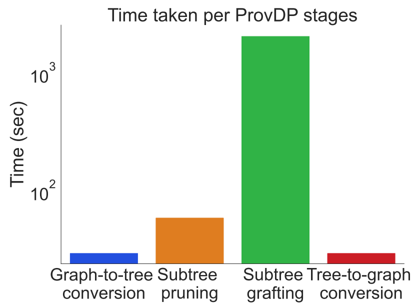


Fig. 7: Average overhead for ProvDP stages per dataset.

The overhead of ProvDP's four stages (§5) measured by the time taken per stage per graph is shown in Figure 7. The experiments were conducted on a system featuring 8 Intel virtual CPUs, 32 GB of RAM, an NVIDIA RTX3080 GPU, and running Ubuntu. Subtree grafting is the most time-consuming due to its complexity in finding the appropriate subtree to graft. Graph to tree and tree to graph conversion takes minimum duration of 30 seconds for an average graph size of 1200 nodes. Subtree pruning is much faster than grafting, since you only need to keep track of the node we are pruning from and the pruned tree, it does not require any subtree searches which has additional overhead.

7 Discussion and Limitations

Real-World Application. ProvDP demonstrates significant potential in preserving privacy without sacrificing utility in controlled environments. However, the deployment of ProvDP in real-world settings remains to be fully explored. Real-world applications often involve more complex and dynamic data flows, which might introduce unforeseen challenges in maintaining the balance between privacy and utility. Further research is needed to adapt and fine-tune ProvDP to handle the intricacies of real-world provenance data, including the development of case studies and field trials to validate its practical effectiveness.

Scalability of ProvDP. Ensuring ProvDP can handle the vast amounts of provenance data typically generated in large-scale, real-world environments is important. While our current implementation demonstrates promising results on datasets of moderate size, enterprise-scale deployments require optimization based on parameters such as size of the network, log volume, resources available for ProvDP, etc. As discussed in §6.5, subtree grafting is the most time-intensive stage in ProvDP due to the search strategy used to identify appropriate subtrees for grafting. To mitigate this overhead, we plan to sort and store pruned subtrees in an indexed in-memory database (*e.g.,* by bin number), thereby reducing the query time for matching suitable subtrees. We also intend to explore neural approaches for subtree identification, where subtrees are embedded and matched to candidate embeddings using cosine similarity. These optimizations will reduce grafting complexity and ensure efficient, large-scale operation of ProvDP in high-performance enterprise settings.

Applicability of ProvDP to Different GNN Architectures. Our study applies ProvDP specifically to the GAT-based IDS introduced by [33], which employs Graph Attention Network (GAT) [42]. While this provides a solid foundation for understanding ProvDP's privacy guaranteeing capabilities, extending it to other GNN architectures (*e.g.,* Graph Convolution Network (GCN) [28]) demands further investigation. A comprehensive extension to [33] would be required to incorporate an alternative GNN design, rigorously evaluate its correctness, and assess its detection performance—we consider this as a future work.

8 Related Work

Differential Privacy for Graph Datasets. Applying differential privacy to graph datasets presents unique challenges due to graph structures' complex and interdependent nature. Early works, such as [23] and [26], focused on ensuring privacy in social networks by perturbing graph structures or by adding noise to graph queries. These methods were designed to prevent adversaries from inferring sensitive relationships or attributes within the network. Recent advancements, such as [47] and [48], have explored the application of differential privacy in more general graph settings, including heterogeneous and multi-relational graphs.

9 Conclusion

The increasing deployment of PIDS across various industries underscores the importance of protecting sensitive provenance data from privacy breaches. While provenance data is invaluable for detecting sophisticated cyberattacks, its detailed nature poses significant privacy risks, particularly when exposed to malicious entities. In response to this challenge, we have introduced ProvDP, a novel framework that applies differential privacy techniques to provenance graphs, ensuring that the utility of such data for intrusion detection is preserved while safeguarding user privacy. Our contributions include a method for transforming provenance graphs into provenance trees, which allowed us to leverage subtree differential privacy to protect the data effectively. Our evaluations demonstrate that implementing these privacy-preserving techniques does not compromise the accuracy of PIDS models. Models trained on differentially private data generated by ProvDP exhibit similar performance on provenance non-private data.

This work represents a significant advancement in privacy preserving security analytics, offering a balanced approach that maintains both privacy and utility. We believe that the principles and techniques developed in ProvDP can be extended to other domains of provenance analysis, opening new avenues for research and application in privacy-preserving security technologies.

Acknowledgments. We thank the Shepherd and anonymous reviewers for their helpful feedback.

A Appendix

A.1 Extended Top-m Filter Algorithm

Algorithm 1 Extended Top-m Filter

```
1: **Input:** Graph $G = (V, E)$ and adjacency matrix $A$ for graph $G$
2: $\tilde{E} \leftarrow \emptyset$
3: $m = |E|$
4: $\tilde{m} \leftarrow \lceil m + Lap(1/\epsilon_2) \rceil$
5: $E_{possible} \leftarrow$ all possible edges in $V_s \times V_d$
6: $E_{valid} \leftarrow E_{possible} \setminus E_{sl}$
7: $\epsilon_t = \ln\left(\frac{|E_{valid}|}{\tilde{m}} - 1\right)$
8: // Set filter bound
9: if $\epsilon_1 < \epsilon_t$ then
10:     $\theta = \frac{\epsilon_t}{2\epsilon_1}$
11: else
12:     $\theta = \frac{1}{\epsilon_1} \ln\left(\frac{|E_{valid}|}{2\tilde{m}} + \frac{1}{2}\left(e^{\epsilon_1} - 1\right)\right)$
13: end if
14: // Filter existing edges
15: for $A_{ij} = 1$ do
16:     compute $\bar{A}_{ij} = A_{ij} + Lap(1/\epsilon_1)$
17:     if $\bar{A}_{ij} > \theta$ then
18:         Add directed edge $(u_i, v_j) \in E$ to $\tilde{E}$
19:     end if
20: end for
21: // Add new edges
22: while $|\tilde{E}| < \tilde{m}$ do
23:     Randomly pick edge $(u_i, v_j) \in E_{valid}$
24:     if $(u_i, v_j) \notin \tilde{E}$ then
25:         Add directed edge $(u_i, v_j)$ to $\tilde{E}$
26:     end if
27: end while
28: if $(V, \tilde{E})$ has component: $C_1, C_2, ..., C_k$ where $k > 1$ then
29:     $\tilde{G} \leftarrow \tilde{C} : u_{poi} \in \tilde{C}$
30:     $\tilde{V} \leftarrow v \in \tilde{G}$
31:     $\tilde{E} \leftarrow e \in \tilde{G}$
32: else
33:     $\tilde{V} \leftarrow V$
34:     $\tilde{G} \leftarrow (\tilde{V}, \tilde{E})$
35: end if
36: return $\tilde{G} = (\tilde{V}, \tilde{E})$
```

A.2 Prune Subtree at Single Point Algorithm

Algorithm 2 Prune Subtree at Single Point

```
1: Input: Tree $T$, privacy budget $\epsilon_1$, sensitivity parametrized by $\alpha$, $\beta$, $\gamma$, and $\eta$
2: pruned_eligible $\leftarrow \emptyset$ // Member of this set is eligible for pruning
3: // Initialize a node processing queue with the root node
4: $Q = [v_{root}]$
5: // conduct level order traversal
6: while size($Q$) $> 0$ do
7:     $v = Q$.pop()
8:     size, height, depth, branching_factor $\leftarrow$ getSubtreeStats($T$, $v$)
9:     $S \leftarrow 1 \ / \ (\alpha \cdot \text{size} + \beta \cdot \text{height} + \gamma \cdot \text{depth} + \eta \cdot \text{branching\_factor})$
10:    $\textit{pruning\_prob} \leftarrow 1 \ / \ (1 + e^{\frac{\epsilon_1}{2S}})$
11:    $r \leftarrow \mathcal{N}(\mu, \sigma^2)$, $r \in [0, 1]$
12:    if $r < \textit{pruning\_prob}$ and depth $> 0$ then
13:        pruned_eligible.add(v)
14:    end if
15: end while
16: // Randomly choose a node $v_{chosen}$ from the set pruned_eligible
17: // Subtree rooted at $v_{chosen}$ is $T_{v_{chosen}}$
18: $\tilde{T} \leftarrow T \setminus T_{v_{chosen}}$
19: return $\tilde{T}$
```

A.3 Prune Subtree at k Point Algorithm

Algorithm 3 Prune Subtree at k Points

```
1: **Input:** Tree $T$, privacy budget $\epsilon_1$, Sensitivity parametrized by $\alpha$, $\beta$, $\gamma$, and $\eta$, and
   Number of subtrees to be pruned $k$
2: // Copy the tree so that it can be manipulated
3: $\tilde{T} \leftarrow T$
4: // Set of pruned trees
5: $Pruned\_Trees \leftarrow \{\}$
6: // Initialize a dictionary, B, to contain pruned subtrees
7: // whose keys are the size of the pruned subtree
8: $B \leftarrow \{\}$
9: // Initialize a node processing queue with the root node
10: $Q = [v_{root}]$
11: // conduct level order traversal
12: **while** $Q$.size() $> 0$ **do**
13:     $v = Q$.pop()
14:     size, height, depth, branching_factor $\leftarrow$ getSubtreeStats($T$, $v$)
15:     $S \leftarrow 1 \ / \ (\alpha \cdot \text{size} + \beta \cdot \text{height} + \gamma \cdot \text{depth} + \eta \cdot \text{branching\_factor})$
16:     $pruning\_prob \leftarrow 1 \ / \ (1 + e^{\frac{\epsilon_1}{2S}})$
17:     $r \leftarrow \mathcal{N}(\mu, \sigma^2)$, $r \in [0, 1]$
18:     **if** $r < pruning\_prob$ and depth $> 0$ **then**
19:         $v.mark = True$
20:         $B$[size].append(v)
21:     **end if**
22:     **for** child $c$ of $v$ **do**
23:         $Q$.append($c$)
24:     **end for**
25: **end while**
26: **for** $i = \{1...k\}$ and $k - i <$ number of nodes in $B$ **do**
27:     Randomly choose bucket $B_s$ that contains trees of size $s$
28:     by generating a random number $s$ between 0 and $B.size - 1$
29:     Randomly choose a tree $T_s$ of size $s$ rooted at $r$ from $B_s$
30:     Remove $T_s$ from $B_s$
31:     $\tilde{T} \leftarrow \tilde{T} \setminus T_s$
32:     **for** $v \in T_s.V$ **do**
33:         **if** $v.mark$ = True **then**
34:             $v.mark \leftarrow False$
35:             $v.prune \leftarrow True$
36:             $v.subtree\_size \leftarrow s$
37:         **end if**
38:     **end for**
39:     // $T_s$ is the subtree rooted at $r$
40:     // $v_p$ is the parent node of $r$
41:     // $e_p$ is the edge connecting $v_p$ to $r$
42:     // $T_{vp}$ is the subtree $T_s$ with the addition of node $v_p$ and edge $e_p$
43:     $Pruned\_SubTrees$.append([$T_{vp}$, $v_p$, $e_p$])
44: **end for**
45: **return** $\tilde{T}$, $Pruned\_SubTrees$
```

A.4 Graft Subtree Algorithm

Algorithm 4 Graft Subtrees

```
1: Input: Modified tree $\tilde{T}$, set of pruned subtrees $Pruned\_SubTrees$ and privacy
   budget $\epsilon_2$
2: $k' \leftarrow 0$
3: for $v$ in $\tilde{T}.V$ do
4:     if $v.prune$ then
5:         $s_v \leftarrow v.subtree\_size$
6:         $\tilde{s}_v \leftarrow s_v + Lap(1/\epsilon_2)$
7:         $W \leftarrow |s_v - \tilde{s}_v| + 1$
8:         $grafting\_prob \leftarrow 1/W$
9:         $r \leftarrow \mathcal{N}(\mu, \sigma^2)$, $r \in [0, 1]$
10:        if $r < grafting\_prob$ then
11:            $\tilde{t} \leftarrow$ random.choice($Pruned\_SubTrees[\tilde{s}_v]$)
12:            Graft $\tilde{t}$ to $\tilde{T}$ at node $v$
13:            $k' += 1$
14:        end if
15:    end if
16: end for
17: return $\tilde{T}$, $k'$
```

A.5 ϵ-Edge-Differential Privacy (ϵ-Edge-DP) Proof for Algorithm 1

The ETmF algorithm which is adapt the TmF [39] is composed of two mechanisms which are ϵ_1-Edge-DP and ϵ_2-Edge-DP, and ETmF is ϵ-Edge-DP by sequential composition (based on Theorem 2.1 in [39]) where $\epsilon = \epsilon_1 + \epsilon_2$.

The first mechanism calculated the number of edges to be added $\tilde{m}$ by perturbing the number of edges, $m = |E|$ using Laplace noise under budget ϵ_2 to produce perturbed $\tilde{m}$.

$$\tilde{m} = \lceil m + Lap(\Delta f/\epsilon_2) \rceil$$

Based on the definition of neighboring graphs in [39], two graphs are neighboring if they differ in a single edge, thus the global sensitivity of function f which is defined as Δf is 1.

$$\tilde{m} = \lceil m + Lap(1/\epsilon_2) \rceil$$

Therefore, this step is ϵ_2-Edge-DP (based on Theorem 2.1 in [39]) since Laplacian noise is added as a scaled parameter $\frac{\Delta f}{\epsilon_2}$ with respect to global sensitivity function Δf.

The second mechanism adds edges to the graph of size $\tilde{m}$, by first identifying candidate edges in the graph by adding Laplace noise under budget ϵ_1 and then passing the candidate edges through a high pass filters with threshold θ.

The global sensitivity is also 1 in this case similar to first mechanism. By the assumption of edge independence, parallel composition (based on Theorem 2.2 in [39]) is applicable at edge level. Therefore, the second method is ϵ_1-Edge-DP. Therefore, ETmF is ϵ-Edge-DP by sequential composition (based on Theorem 2.1 in [39]) where $\epsilon = \epsilon_1 + \epsilon_2$.

In system provenance the edges are public knowledge, and with that assumption, TmF can be generalized to apply to any set of valid edges E_{valid}. We can brute-force enumerate over all possible edges and reject invalid edges to obtain our E_{valid}. To derive the new upper-bound for θ, we calculate the theoretical maximum of $|E_{\text{valid}}|$ since θ is upper bounded by ϵ_t. There ca be a directed edge between any two nodes. Therefore, the maximum number of edges is bounded by C_2^n *i.e.,* $n(n-1)/2$. Therefore, the upper bound of θ is $n(n-1)/2$.

A.6 ϵ_1-Subtree-Differential Privacy (ϵ_1-Subtree-DP) Proof for Algorithm 2

In Definition 1 we defined ϵ_1-Subtree-Differential Privacy.

Definition 3. *In Definition 1 we defined two trees (T_1 and T_2) as neighboring, if the differ by a single subtree, t. Now we define the global sensitivity of a function f as $\Delta f = \max_{T_1,T_2,t} ||f(T_1,t) - f(T_2,t)||_1$.*

Using Definition 1, we consider a pair of tree neighbors T_1, T_2 who differ by one subtree t, $\Pr[R(T_1) = x]$ which is proportional to $\exp\left(\frac{\epsilon_1 f(T_2,t)}{S}\right)$ using exponential distribution [30, 31]. Therefore, the ratio of $\Pr[R(T_1) = x]$ and $\Pr[R(T_2) = x]$ is equivalent to:

$$\frac{\exp\left(\frac{\epsilon_1 f(T_1,t)}{S}\right)}{\exp\left(\frac{\epsilon_1 f(T_2,t)}{S}\right)}.$$

This can be simplified to e^{ϵ_1} as the maximum value of $f(T_1,t) - f(T_2,t)$ is S,

$$\exp\left(\frac{\epsilon_1(f(T_1,t) - f(T_2,t))}{S}\right) \leq \exp\left(\frac{\epsilon_1 S}{S}\right) = e^{\epsilon_1}$$

For Algorithm 2, we will describe the effect of subtree size at that node, degree of the node, subtree height, and subtree depth on the global sensitivity calculation. We define global sensitivity of function f with respect to size as $\Delta f_{size} = \max_{v \in V \setminus V_{root}} \{|V| - (\text{number of nodes after pruning } T_v)\}$. It can be rewritten in terms of subtree T_v pruned, $\Delta f = \max_{v \in V} |T_v|$.

The height parameter does not contribute to the global sensitivity if there is more than one longest alternative root-to-leaf paths. If node v which is on one of the longest root-to-leaf paths is pruned, the other longest root-to-leaf paths survive. Therefore, the height of the tree $\tilde{T}$ remains same as the height of

T. If and only if there is one unique longest root-to-leaf path and the pruned node v is a member of the path, the sensitivity with respect to height becomes $\Delta f_{height} \leftarrow \max(height(T) - height(\tilde{T}), height(T_v))$. Therefore, the upper bound of $\Delta f_{height} \leftarrow height(T) - 1$.

The depth does not contribute to the global sensitivity as the depth is not a property of a tree rather it is a property of the node which is pruned. Therefore, it will contribute towards the local sensitivity of the pruned subtree. Let us consider L_T is the set of leaf nodes of the tree T. Therefore, the upper limit of the local sensitivity with respect to the depth becomes $\Delta f_{depth} \leftarrow \max_{v \in L_T} depth(v)$.

The branching factor b_T of a tree T is defined as the maximum over the degrees of all the nodes in a tree, $b_T \leftarrow \max_{v \in T} degree(v)$. Let us define the least common ancestor $LCA_b \leftarrow lca(v_1, v_2, ..., v_k) \mid degree_{i=1...k}(v_i) = b_T$. The branching factor of the tree reduces if and only if we prune a node which is either LCA_b or its ancestors. Therefore, the sensitivity with respect to the branching factor becomes $\Delta f_{branching\_factor} \leftarrow b_T$. Please note if the LCA_b is living at level 1 of the tree and only node at level 1 then the above mentioned scenario occurs.

Finally, we calculate the global sensitivity Δf by taking a weighted sum of Δf_{size}, Δf_{height}, Δf_{depth}, and $\Delta f_{branching\_factor}$.

$$\Delta f \leftarrow \alpha \cdot \Delta f_{size} + \beta \cdot \Delta f_{height} + \gamma \cdot \Delta f_{depth} + \eta \cdot \Delta f_{branching\_factor}$$

Special Scenario: Lets us assume that the session behaviors have similar structures. Therefore, a tree T contains user activities that are uniform in nature across different sessions and T contains multiple similar subtrees.

The subtree size strictly decreases as depth increases, thus this v that gives the global sensitivity must be located at depth 1. At depth ≥ 2, the subtrees cannot be a candidate as there always exists a subtree rooted at depth 1 which will be larger in size by at least 1 node. Using our assumption of uniform user activity, all subtrees at depth 1 are of equal size, then the size of any subtree rooted at depth 1 will be $\frac{|V|-1}{degree(v_{root})}$, where $degree(v_{root})$ is the number of user behavior contained in the tree. For height and branching factor since the subtree are not unique the effect on global sensitivity is nullified. The sensitivity due to the depth is maximum when we prune at leaf. In such scenario, the depth will be the $height - 1$ which is equal to $\frac{|V|-1}{degree(v_{root})}$.

These two cases cannot occur together. We can either prune at level 1 or we can prune at leaf. Therefore, Δf for this special scenario:

$$\Delta f \leftarrow \max(\alpha, \gamma) * \frac{|V| - 1}{degree(v_{root})}$$

A.7 ϵ_k-Subtree-Differential Privacy (ϵ_k-Subtree-DP) Proof for Algorithm 3

In Definition 2 we defined ϵ_k-Subtree-Differential Privacy. Using the sequential composition theorem (Theorem 3) from [30] which states any sequence of computations that each provide differential privacy in isolation also provide differential privacy in sequence. Mathematically, we denote it as, if a randomizer R provide ϵ_i- differential privacy, then the sequence of R provides $\sum_i \epsilon_i$-differential privacy.

In k-subtree pruning algorithm (Algorithm 3), we perform subtree pruning (Algorithm 2) in sequence k times on the tree T. In §A.6 we showed that Algorithm 2 is ϵ_1-Subtree-DP. Therefore applying Algorithm 2, k times in sequence will provide $\sum_{i=1}^{k} e_1$ Subtree differential privacy. We define ϵ_k as $\epsilon_k = \sum_{i=1}^{k} e_1$. Therefore, Algorithm 3 is ϵ_k-Subtree-DP (using the sequential composition theorem from [30]).

A.8 $\epsilon_{k'}$-Subtree-Differential Privacy ($\epsilon_{k'}$-Subtree-DP) Proof for Algorithm 4

In the grafting algorithm (Algorithm 4), we perform grafting once which is ϵ_2-Subtree-DP because in line 6, we add Laplacian noise (*e.g.,* $Lap(1/\epsilon_2)$) to s_v to get $\tilde{s}_v$ which affects the grafting probability. This step is ϵ_2-Edge-DP (based on Theorem 2.1 in [39]) since Laplacian noise is added as a scaled parameter $\frac{\Delta f}{\epsilon_2}$ with respect to global sensitivity function Δf.

In the grafting algorithm (Algorithm 4), we perform subtree grafting in sequence k' times on the tree $\tilde{T}$. We showed that grafting one subtree is ϵ_2-Subtree-DP. Therefore, applying it k' times in sequence will provide $\sum_{i=1}^{k'} e_2$ Subtree differential privacy. We define $\epsilon_{k'}$ as $\epsilon_{k'} = \sum_{i=1}^{k'} e_2$. Therefore, Algorithm 4 is $\epsilon_{k'}$-Subtree-DP (using the sequential composition theorem from [30]).

A.9 ϵ-Subtree-Differential Privacy (ϵ-Subtree-DP) Proof

We proved Algorithm 3 is ϵ_k-Subtree-DP in §A.7 and Algorithm 4 is $\epsilon_{k'}$-Subtree-DP in §A.8. Since, the algorithm are invoked in sequence, ProvDP is ϵ-Subtree-DP where $\epsilon = \epsilon_k + \epsilon_{k'}$, using the sequential composition theorem from [30].

A.10 Implementation.

Our ProvDP tool is implemented in Python, comprising approximately 1K lines of code. We developed custom Python functions to construct provenance trees from graphs, filter extraneous information, and abstract node entities. This preprocessing ensures that the raw graph data is optimally prepared for our downstream task, GNN-based anomaly detection [33]. For the GNN model, we utilize the DGL [2] library, specifically employing GAT layers. The GAT layer was chosen due to recent works [32, 35] showing explanation capability of the architecture which increases the user's confidence. These layers aggregate features from neighboring nodes to generate refined node representations. The model architecture incorporates dropout and ReLU activation functions between layers to enhance generalization.

References

1. The linux audit framework. https://github.com/linux-audit/ (2015)
2. Deep graph library: Easy deep learning on graphs. https://www.dgl.ai/ (2019)
3. Evasive attacker leverages solarwinds supply chain compromises with sunburst backdoor. https://tinyurl.com/bdz8s5yn (2019)
4. Event tracing for windows (etw) - windows drivers — microsoft docs. https://docs.microsoft.com/en-us/windows-hardware/drivers/devtest/event-tracing-for-windows--etw- (2019)
5. North korea's lazarus apt leverages windows update client, github in latest campaign. https://tinyurl.com/mr4h7d35 (2019)
6. U.s. said to find north korea ordered cyberattack on sony. https://tinyurl.com/5da2h9bx (2019)
7. Wildpressure targets industrial in the middle east. https://tinyurl.com/mr2n8hdu (2019)
8. Extended detection and response (xdr). https://www.cybereason.com/platform/xdr (2023)
9. Anderson, B., McGrew, D.: Machine learning for encrypted malware traffic classification: accounting for noisy labels and non-stationarity. In: Proceedings of the 23rd ACM SIGKDD International Conference on knowledge discovery and data mining. pp. 1723–1732 (2017)
10. Bilge, L., Balzarotti, D., Robertson, W., Kirda, E., Kruegel, C.: Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In: Proceedings of the 28th Annual Computer Security Applications Conference. pp. 129–138 (2012)
11. Cantrill, B.: Dtrace. In: Large Installation System Administration Conference (LISA) (2005)
12. Cheng, Z., Lv, Q., Liang, J., Wang, Y., Sun, D., Pasquier, T., Han, X.: Kairos: Practical intrusion detection and investigation using whole-system provenance. In: IEEE Symposium on Security and Privacy (SP) (2024)
13. Cheng, Z., Lv, Q., Liang, J., Wang, Y., Sun, D., Pasquier, T., Han, X.: Kairos: Practical Intrusion Detection and Investigation using Whole-system Provenance. In: IEEE Symposium on Security and Privacy (SP) (2024)
14. Dinur, I., Nissim, K.: Revealing information while preserving privacy. In: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. pp. 202–210 (2003)
15. Divakaran, D.M., Fok, K.W., Nevat, I., Thing, V.L.: Evidence gathering for network security and forensics. Digital Investigation **20**, S56–S65 (2017)
16. Dwork, C.: Differential privacy. In: International colloquium on automata, languages, and programming. pp. 1–12. Springer (2006)
17. Dwork, C., Roth, A., et al.: The algorithmic foundations of differential privacy. Foundations and Trends® in Theoretical Computer Science **9**(3–4), 211–407 (2014)
18. Goyal, A., Wang, G., Bates, A.: R-caid: Embedding root cause analysis within provenance-based intrusion detection. In: IEEE Symposium on Security and Privacy (SP) (2024)
19. Griffith, J., Kong, D., Caro, A., Benyo, B., Khoury, J., Upthegrove, T., Christovich, T., Ponomorov, S., Sydney, A., Saini, A., et al.: Scalable Transparency Architecture for Research Collaboration (STARC)-DARPA Transparent Computing (TC) Program. Tech. rep. (2020)

20. Gysel, P., Wüest, C., Nwafor, K., Jašek, O., Ustyuzhanin, A., Divakaran, D.M.: Eagleeye: Attention to unveil malicious event sequences from provenance graphs. arXiv preprint arXiv:2408.09217 (2024)
21. Han, X., Yu, X., Pasquier, T., Li, D., Rhee, J., Mickens, J., Seltzer, M., Chen, H.: Sigl: Securing software installations through deep graph learning. In: USENIX Security Symposium (SEC) (2021)
22. Hassan, W.U., Guo, S., Li, D., Chen, Z., Jee, K., Li, Z., Bates, A.: NoDoze: Combatting Threat Alert Fatigue with Automated Provenance Triage. In: Network and Distributed System Security Symposium (NDSS) (2019)
23. Hay, M., Miklau, G., Jensen, D., Towsley, D., Weis, P.: Accurate estimation of the degree distribution of private networks. In: 2009 Ninth IEEE International Conference on Data Mining. pp. 169–178. IEEE (2009)
24. Inam, M.A., Chen, Y., Goyal, A., Liu, J., Mink, J., Michael, N., Gaur, S., Bates, A., Hassan, W.U.: SoK: History is a Vast Early Warning System: Auditing the Provenance of System Intrusions. In: IEEE Symposium on Security and Privacy (SP) (2023)
25. Karwa, V., Raskhodnikova, S., Smith, A., Yaroslavtsev, G.: Private analysis of graph structure. Proceedings of the VLDB Endowment **4**(11), 1146–1157 (2011)
26. Kasiviswanathan, S.P., Nissim, K., Raskhodnikova, S., Smith, A.: Analyzing graphs with node differential privacy. In: Theory of Cryptography: 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings. pp. 457–476. Springer (2013)
27. King, S.T., Chen, P.M.: Backtracking intrusions. In: Proceedings of the nineteenth ACM symposium on Operating systems principles (2003)
28. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
29. Liu, Y., Zhang, M., Li, D., Jee, K., Li, Z., Wu, Z., Rhee, J., Mittal, P.: Towards a Timely Causality Analysis for Enterprise Security. In: Network and Distributed System Security Symposium (NDSS) (2018)
30. McSherry, F.D.: Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of data. pp. 19–30 (2009)
31. Miller, S., Childers, D.: Probability and random processes: With applications to signal processing and communications. Academic Press (2012)
32. Mukherjee, K., Harrison, Z., Balaneshin, S.: Z-rex: Human-interpretable gnn explanations for real estate recommendations. arXiv preprint arXiv:2503.18001 (2025)
33. Mukherjee, K., Wiedemeier, J., Wang, T., Wei, J., Chen, F., Kim, M., Kantarcioglu, M., Jee, K.: Evading provenance-based ml detectors with adversarial system actions. In: USENIX Security Symposium (SEC) (2023)
34. Mukherjee, K., Wiedemeier, J., Wang, Q., Kamimura, J., Rhee, J.J., Wei, J., Li, Z., Yu, X., Tang, L.A., Gui, J., et al.: Proviot: Detecting stealthy attacks in iot through federated edge-cloud security. In: International Conference on Applied Cryptography and Network Security. pp. 241–268. Springer (2024)
35. Mukherjee, K., Wiedemeier, J., Wang, T., Kim, M., Chen, F., Kantarcioglu, M., Jee, K.: Interpreting gnn-based ids detections using provenance graph structural features. arXiv preprint arXiv:2306.00934 (2023)
36. Narayanan, A., Shmatikov, V.: Robust de-anonymization of large sparse datasets. In: 2008 IEEE Symposium on Security and Privacy (sp 2008). pp. 111–125. IEEE (2008)
37. Narayanan, A., Shmatikov, V.: De-anonymizing social networks. In: 2009 30th IEEE symposium on security and privacy. pp. 173–187. IEEE (2009)

38. Nevat, I., Divakaran, D.M., Nagarajan, S.G., Zhang, P., Su, L., Ko, L.L., Thing, V.L.: Anomaly detection and attribution in networks with temporally correlated traffic. IEEE/ACM Transactions on Networking **26**(1), 131–144 (2017)
39. Nguyen, H.H., Imine, A., Rusinowitch, M.: Differentially private publication of social graphs at linear cost. In: Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015. p. 596–599. ASONAM '15, Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2808797.2809385, https://doi.org/10.1145/2808797.2809385
40. Nissim, K., Raskhodnikova, S., Smith, A.: Smooth sensitivity and sampling in private data analysis. In: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing. pp. 75–84 (2007)
41. Rehman, M.U., Ahmadi, H., Hassan, W.U.: FLASH: A Comprehensive Approach to Intrusion Detection via Provenance Graph Representation Learning. In: IEEE Symposium on Security and Privacy (SP) (2024)
42. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. arXiv preprint arXiv:1710.10903 (2017)
43. Wang, Q., Hassan, W.U., Li, D., Jee, K., Yu, X., Zou, K., Rhee, J., Chen, Z., Cheng, W., Gunter, C.A., Chen, H.: You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis. In: Network and Distributed System Security Symposium (NDSS) (2020)
44. Wang, T., Klancher, S., Mukherjee, K., Wiedemeier, J., Chen, F., Kantarcioglu, M., Jee, K.: Provcreator: Synthesizing graph data with text attributes
45. Yuan, Q., Zhang, Z., Du, L., Chen, M., Cheng, P., Sun, M.: {PrivGraph}: Differentially private graph data publication by exploiting community information. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 3241–3258 (2023)
46. Zengy, J., Wang, X., Liu, J., Chen, Y., Liang, Z., Chua, T.S., Chua, Z.L.: Shadewatcher: Recommendation-guided cyber threat analysis using system audit records. In: IEEE Symposium on Security and Privacy (SP) (2022)
47. Zhang, S., Ni, W.: Graph embedding matrix sharing with differential privacy. IEEE Access **7**, 89390–89399 (2019)
48. Zheng, X., Zhang, L., Li, K., Zeng, X.: Efficient publication of distributed and overlapping graph data under differential privacy. Tsinghua Science and Technology **27**(2), 235–243 (2021)