Prov2vec: Learning Provenance Graph Representation for Anomaly Detection in Computer Systems

Bibek Bhattarai
GraphLab
The George Washington University
USA
bhattarai_b@gwu.edu

H. Howie Huang
GraphLab
The George Washington University
USA
howie@gwu.edu

ABSTRACT

Modern cyber attackers use advanced zero-day exploits, highly targeted spear phishing, and other social engineering techniques to gain access, and also use evasion techniques to maintain a prolonged presence within the victim network while working gradually towards the objective. To minimize damage, detecting these Advanced Persistent Threats as early in the campaign as possible is crucial. This paper proposes, Prov2vec, a system for the continuous monitoring of enterprise host's behavior to detect attackers' activities. It leverages the data provenance graph built using system event logs to get complete visibility into the execution state of an enterprise host and the causal relationship between system entities. It proposes a novel provenance graph kernel to obtain the canonical representation of the system behavior, which is compared against its historical behaviors and that of other hosts to detect the deviation from the norm. These representations are used in several machine learning models to evaluate their ability to capture the underlying behavior of an endpoint host. We have empirically demonstrated that the provenance graph kernel produces a much more compact representation compared to existing methods while improving prediction ability.

CCS CONCEPTS

• **Security and privacy → Intrusion/anomaly detection and malware mitigation**.

KEYWORDS

Provenance Graph, Machine Learning, Anomaly Detection

ACM Reference Format:
Bibek Bhattarai and H. Howie Huang. 2024. Prov2vec: Learning Provenance Graph Representation for Anomaly Detection in Computer Systems. In *The 19th International Conference on Availability, Reliability and Security (ARES 2024), July 30–August 02, 2024, Vienna, Austria.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3664476.3664494

ARES 2024, July 30–August 02, 2024, Vienna, Austria

ACM ISBN 979-8-4007-1718-5/24/07
https://doi.org/10.1145/3664476.3664494

1 INTRODUCTION

Large enterprises and government networks have seen a significant rise in targeted attacks from experienced cyber criminals with substantial financial backing [45, 49, 56, 62]. These sophisticated attacks often referred to as advanced persistent threats (APTs), are carried out in multiple steps over a prolonged period, each designed to blend in with benign activity. These att acks bypass traditional signature-based defense mechanisms [57] via zero-day exploits. While anomaly detection processes such as [9] can detect the events that diverge from the norm, traditional sequence-based contexts can not reliably capture the causality between involved system entities and thus miss the 'low and slow' attacks.

To provide better visibility into attack campaigns, existing threat detection systems [5, 20, 42, 43] combine the **Provenance Graph** with alert generation capabilities of Endpoint Detection and Response (EDR) systems. This integration allows for automated alert correlation, reducing false positives and providing contextual information around alerts. By capturing the information flow between system objects using a Directed Acyclic Graph (DAG), these provenance graphs provide historical context and the impact of an alert through backward and forward graph traversal [31]. Formally, we define a provenance graph snapshot at time t as $G_t = (V_t, E_t, A^v, A^e)$, where V_t and E_t are set of nodes and edges at time t, and A^v and A^e are functions that maps all nodes $v \in V_t$ and edges $e \in E_t$ in the graph to set of node and edge attributes respectively.

Unfortunately, several challenges remain for existing provenance graph-based systems. First, fine-grained causal analysis on lengthy APT campaigns presents significant computational challenges. With a median dwell time of three weeks and the generation of several terabytes of log events daily, holding the entire provenance graph in memory is highly impractical. Storing graphs in databases and conducting graph traversal [44] for alert correlation significantly escalates the cost. Achieving real-time detection using compute-intensive alert correlation across the entire enterprise network is a daunting task. It is important to note that the majority of computational resources allocated to alert correlation should contribute minimally to the overall detection process. In a recent study [5] involving a network with 500 hosts, out of 28 hosts that were compromised over a three-day evaluation period, the alert correlation on a single host was able to detect the APT campaigns only on 5 hosts(1% of the total hosts). While alert correlation is crucial for retracing the attackers' steps and comprehending attacks on compromised hosts, applying it as a proactive detection measure universally across all hosts and at all times is cost-prohibitive.

Second, the detection capability of alert correlation-based systems is inherently limited by the coverage of rules employed in the

alert generation process. These rules are manually crafted by security practitioners, relying on the cyber threat intelligence reports from security forums, blogs, social media, and previous attacks. The matching semantics vary based on the platform and underlying sensors used for data collection. Replicating such rules in a new platform requires a substantial manual effort and expertise. Furthermore, although incorporating novelty detection on a node or edge level [4, 10, 53] can potentially detect previously unseen attacks, new benign activities also emerge constantly, necessitating a broader perspective on activities for threat detection.

To tackle these challenges, we introduce Prov2vec, an innovative system that leverages a novel **provenance graph kernel** to derive a canonical form for a given graph snapshot, capturing the aggregated host behavior at a specific time point in a fixed-size vector representation. Prov2vec operates by mining label-aware backward walks, with a maximum length specified by the user as **h**, for each node in the provenance graph. These walks encompassing the execution history of nodes over varying hop lengths from $0 \leq i \leq h$ are then compressed into a label that succinctly describes the nodes' causal history. A node label histogram is constructed by tallying the frequencies of distinct labels across all nodes in the graph for each hop length from 0 to h. These histograms are stored in memory using a fixed-size probabilistic data structure called histosketch [73], which is utilized by downstream machine learning tasks to model the behavior of the hosts and detect when they behave abnormally.

This work makes the following three contributions:

- We develop an end-to-end system for unsupervised anomaly detection on network systems. Leveraging the provenance graphs built from logs gathered during normal operations, our system creates comprehensive benign host behavior profiles. Any provenance graphs deviating from those generated by benign activities are identified as anomalies. We utilize these anomalous graph snapshots, along with their associated users and hosts, to traverse the authentication graph and uncover all compromised entities.
- We propose a novel graph kernel that enhances the generalization of similar provenance graph structures using compact node label histograms. Our approach achieves superior or comparable accuracy in downstream machine learning tasks while maintaining the histogram of size an order magnitude smaller than Wesfeller-Lehman subtree (WLSubtree) kernel [58] and temporally ordered WL subtree kernel from Unicorn [18].
- We showcase the effectiveness of Prov2vec in profiling host behavior and detecting compromised network entities through three machine learning tasks – graph classification, graph clustering, and graph anomaly detection – on provenance graphs generated from Windows and Linux hosts.

2 THREAT MODEL

We focus on a typical APT life cycle, where adversaries gain unauthorized access to the enterprise hosts and aim to maintain stealth for an extended period. To achieve their objectives, attackers carry out various post-exploitation activities, including internal reconnaissance, privilege escalation, lateral movement, and data exfiltration [63]. Our goal is to detect compromised hosts based on a given snapshot of the provenance graph at a specific time t, using Prov2vec. We assume that Prov2vec has sufficient historical data to establish a behavior profile of enterprise hosts during normal operations. We also assume that the provenance graph obtained during an attack exhibits distinct differences from the graphs observed during prior normal operations.

Prov2vec does not make assumptions about the specific actions performed by an attacker, apart from the fact that their intent and/or actions leave indicators in the audit logs and, consequently on the provenance graph. To accurately capture this information, Prov2vec assumes the correctness of log collection frameworks. The remainder of this paper assumes the validity of log collecting frameworks and log data used in our experiments, focusing on Prov2vec's ability to model system behavior based on them. For modeling the system behavior, this work assumes that provenance graphs with similar structures indicate comparable operational behavior. Therefore, the detection of abnormal behavior entails the computation of (dis)similarities across provenance graphs.

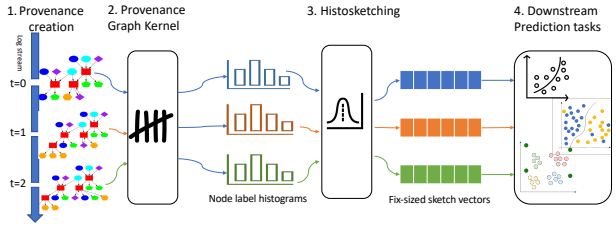

Figure 1: The system diagram of Prov2vec.

3 PROV2VEC DESIGN

3.1 Overview

Figure 1 shows the high-level overview of the Prov2vec system. In the first step (1), given the stream of log events generated by auditing tools [11, 15, 27], Prov2vec updates the provenance graph continuously with new events. That is, periodically it creates the snapshots of the provenance graph $G_t = (V_t, E_t)$.

In the second step (2), the provenance graph kernel is used to convert the graph snapshots into node-label histograms. While aggregating over the specified neighborhood, these histograms can have different sizes depending on the number of distinct provenance labels in a given graph snapshot.

In the third step (3), to compare the histograms with one another, we convert them into vectors of the same size. In the static setting, this can be done by building a vector of size equal to node label vocabulary, which is built using the histograms of all the graphs in question. In the streaming setting, the vocabulary size is constantly increasing. To enable easy comparison of streaming histograms, we utilize a probabilistic data structure called histosketch [73] that uses the consistent weighted hashing [34] to sample the histograms into fixed size vectors while preserving the similarity between them.

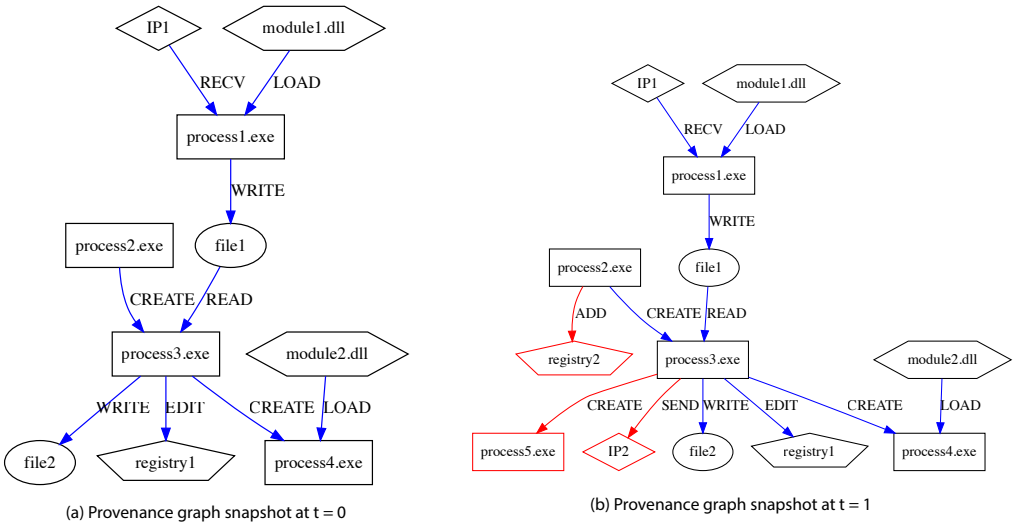

Figure 2: The sample provenance graphs captured. (a) the graph at time t = 0 and (b) the graph at time t = 1, with new nodes and edges denoted in the red color.

In the final step, (4), the series of feature vectors representing provenance graph snapshots is fed to machine learning models to learn the behavior of a network system. They can be designed for one of many tasks such as graph classification, outlier detection, and graph clustering. During deployment, the first three steps are performed and the resultant feature vector is tested against the model learned to detect whether the behavior at any instance is anomalous. The anomalies generated from these models serve as the leads for analysts, providing indications of potentially malicious activity. These anomalies can prompt further investigation to gain insights into the underlying causes and potential countermeasures. To extract subgraphs that capture the sequence of actions performed by the attacker, alert generation and correlation can be conducted using systems like Rapsheet [20] or SteinerLog [5]. We believe that PROV2VEC can play a crucial role in identifying suspicious endpoint hosts, enabling alert correlation systems to focus their fine-grained analysis on those specific hosts.

3.2 Provenance Graph Creation

The system logs are parsed into a triplet of $(subject, action, object)$ and inserted into a provenance graph. The direction of edges signifies the flow of data or information. For instance, an edge corresponding to a process writing on a file will point from the process to the file, whereas a process reading a file will have the opposite direction. Figure 2 shows two snapshots of a provenance graph at time $t = 0$ and later at $t = 1$. The red edges and nodes on the second snapshot represent the part added after the first snapshot.

To reduce the graph size and avoid dependency explosion during the forensic analysis, we utilize causality preserving duplicate elimination [72] and node versioning. When inserting an edge $(src, event, dst)$, if there already is another edge with the same triplet in the provenance graph, and there are not any outgoing edges from the latest version of dst, i.e., dst_i, we simply update the time information of edge and avoid inserting the edge again.

On the other hand, if the newest version of the node dst, i.e., dst_i already has outgoing edges at time i, the insertion of an incoming edge at time $i + 1$ changes the provenance of all neighboring nodes. In this case, for time $i + 1$, we create a new version of that node dst_{i+1}, and insert the edge $(src, event, dst_{i+1})$ instead. In addition, an edge needs to be inserted between dst_i and dst_{i+1} to indicate that the latter is the newer version of the former node.

In our experiment data, an average of 1.2 node versions are created for subjects and objects. With dependency preserving reduction, we were able to reduce the number of edges in the graph by a factor of 3.38× compared to the number of events. Node versioning and redundant edge elimination enable efficient incremental computation of node-label histograms. By creating different versions of nodes in the provenance graph as subjects or objects change, we can focus the computation only on newly inserted nodes. This approach minimizes redundant processing and improves computational efficiency, ensuring that the node-label histograms are efficiently updated as the provenance graph evolves.

3.3 Provenance Graph Kernel

To capture the heterogeneity of the provenance graph, we perform label-aware backward walks from each node in the given snapshot. These walks traverse the graph up to a user-defined length **h**. By accumulating the provenance labels of all nodes, we construct a provenance label histogram for the snapshot. This approach allows us to capture the diverse characteristics of the graph and generate a comprehensive representation of the node labels within the specified walk length.

DEFINITION 1. ***Label-aware backward walk:*** *Given a node $v \in V_t$, a backward walk of length i starting at v is defined as ($l(e_0)$, $l(e_1)$, $l(e_2)$, ... $l(e_{i-1})$, $l(u)$), where $(e_{i-1}, e_{i-2}, ..., e_0)$ is the sequence of edges representing the information flow from u to v, and $l(e_*)$ and $l(u)$ represent the type of events and objects on the walk respectively.*

Backward walk set *$W_i(v)$ of a given node v is the set of all possible backward walks of length i from v.*

Each backward walk of length i describes how node v is impacted by the set of nodes $\{u\}$ with a sequence of i consecutive activities. For $i = 0$, the walk corresponds to the node itself, i.e. $(l(v))$. For instance, in Figure 2(a), length 2 backward walks for `registry1` are $(EDIT, CREATE, PROCESS)$ and $(EDIT, READ, FILE)$. Similarly, the walks of length 1 and 0 for `registry1` are $(EDIT, PROCESS)$ and $(REGISTRY)$ respectively.

Given the set of backward walks $W_i(v)$ consisting of every length i backward walks from node v, we group labels at equal distances from v in these walks. Formally, $\tau_i^j(v) = \{w\ [i-j]\ \forall w \in W_i(v), 0 \leq j \leq i\}$ for $0 \leq i \leq h$. Here, each backward walk w of length i is the sequence of labels $(\ l(e_0), l(e_1), l(e_2), \ldots l(e_{i-1}), l(u)\ )$ as defined in Definition 1. The labels τ_i^j for $0 \leq j \leq i$ are then stacked together to form **i-provenance label** , i.e., $\psi_i(v) = (\tau_i^i, \tau_i^{i-1}, \ldots, \tau_i^1, \tau_i^0)$. If no backward walk of length i exists, then the empty set $\{\}$ is used to denote both $W_i(v)$ and i-provenance label $\psi_i(v)$. The process is repeated for each depth i for $0 \leq i \leq h$. Let's look at the 0-, 1-, and 2-provenance labels of node `registry1` in Figure 2(a),

- For i = 0, $\psi_0(registry1) = (\{REGISTRY\})$, where $\tau_0^0 = \{REGISTRY\}$.
- For i = 1, $\tau_1^0(registry1) = \{PROCESS\}$, $\tau_1^1(registry1) = \{EDIT\}$, and $\psi_1(registry1) = (\{EDIT\}, \{PROCESS\})$.
- For i = 2, $\tau_2^0(registry1) = \{PROCESS, FILE\}$, $\tau_2^1(registry1) = \{CREATE, READ\}$, $\tau_2^2(registry1) = \{EDIT\}$. Stacking all together, we get $\psi_2(registry1) = (\{EDIT\}, \{CREATE, READ\}, \{PROCESS, FILE\})$.

For each provenance graph snapshot G_t, a histogram is constructed containing the frequency of different provenance labels for all nodes in the graph. The histogram keys are generated based on the unique $\psi_i(v)$ values for all $v \in V_t$ and $0 \leq i \leq h$, where h is the maximum walk length. The histogram size of the graph snapshots obtained in Prov2vec is significantly smaller compared to the WL subtree kernel [58] and temporally sorted subtree kernel [18].

In contrast to the multi-set approach used in the WL subtree kernel [58] and the temporally sorted multi-set approach in Unicorn [18], the provenance kernel in Prov2vec utilizes a set to aggregate labels from the neighborhood. This distinction is important because the multi-set approach has been deemed to provide better discrimination power necessary in many domains. However for the provenance graph, since we use entity and event types as labels, these can generate spurious labels and weaken the generalization.

For instance, take three graphs in Figure 3 all of which represent a very similar set of actions, i.e., a process $p1$ reads from file(s), loads a module, and edits a registry item. In Prov2vec, after mining length 1 backward walks, the same provenance label ($\{LOAD, READ\}$, $\{FILE, MODULE\}$) is generated for $p1$ in each of the graphs G1, G2, and G3. However, the WL-subtree kernel maps $p1$ in G3 to a different label ($\{LOAD, MODULE\}$, $\{READ, FILE\}$, $\{READ, FILE\}$) compared to $p1$ in G1 and G2, i.e., ($\{LOAD, MODULE\}$, $\{READ, FILE\}$). Similarly, the Unicorn's kernel also considers the temporal order of $m1$ and $f1$, resulting in a different label for $p1$ in each graph. The ability of the provenance graph kernel to map similar behaviors to identical labels helps better generalize underlying behavior, reducing false positives in downstream tasks. This means that Prov2vec can capture similarities between different instances of $p1$ across the graphs, while the other kernels may treat them as distinct. By providing consistent labels for similar behavior, the provenance graph kernel enhances the accuracy and effectiveness of subsequent analysis tasks.

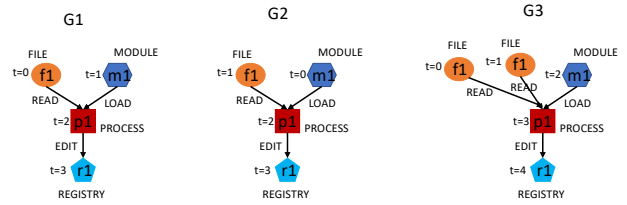

Figure 3: Three toy graphs representing a similar set of actions of a process reading from a file, loading module, and editing a registry. The provenance graph kernel maps all of them to an identical histogram while existing kernels make distinctions based on temporal order or repeated events.

Algorithm 1: Incremental algorithm for computing provenance label histogram

Data: Provenance graph snapshot G_t, current histogram *hist*, inserted edges ΔE, new nodes V_{new}, max walk length h
Result: An updated provenance label histogram *hist*

▷ `Initialize the labels`
for $v \in V_{new}$ **do**
  for $0 \leq i \leq h$ **do**
    $\psi_i(v) \leftarrow ()$;
    for $0 \leq j \leq i$ **do**
      $\tau_i^j(v) \leftarrow \{\}$;
  $\tau_0^0 \leftarrow \{l(v)\}$, $\psi_0 \leftarrow (\tau_0^0)$;
▷ `Iterate over inserted edges to infer other provenance labels`
for $1 \leq i \leq h$ **do**
  for $e = (u, v) \in \Delta E$ **do**
    if $\psi_{i-1}(u)$ *is empty* **then**
      skip the edge;
    $\tau_i^i(v)$.insert($l(e)$);
    for $0 \leq j \leq i-1$ **do**
      $\tau_i^j(v) \leftarrow \tau_i^j(v) \cup \tau_{i-1}^j(u)$;
    ▷ `Update the histogram, if there is an old label, we need to remove it`
    if $\psi_i(v)$ *is not empty* **then**
      $hist[\psi_i(v)] - -$;
    $\psi_i(v) = (\tau_i^i(v), \tau_i^{i-1}, \ldots, \tau_i^0)$;
    $hist[\psi_i(v)] + +$;

3.4 Incremental Provenance Graph Kernel

Algorithm 1 presents a real-time streaming approach for updating the provenance label histogram. It iterates through the newly inserted edges to obtain the provenance labels for them and updates the labels of the affected old nodes. First, it initializes the placeholders (lines 1 - 6) for provenance labels to hold $\psi_i(v)$ for all new nodes $v \in V_{new}$ and $0 \leq i \leq h$. In order to get $\psi_i(v)$, we need placeholder for $\tau_i^j(v)$ for $0 \leq j \leq i$. Once the initialization is done, we iterate through all inserted edges for h iterations to obtain the provenance labels corresponding to the backward walks of length 0 through h (lines 7-17). Once the provenance labels are obtained, we update the label histogram to reflect the newly formed provenance labels (lines 14 - 17).

In the graph snapshot of Figure 2(b), three new edges are inserted in the earlier snapshot, which creates three new nodes in the graph. Once the placeholders for ψ_i and corresponding τ_i^j for each of these nodes are initialized, it obtains the provenance labels of new nodes *registry2*, *process5.exe*, and *IP2*, using the labels of their in-neighbors, i.e., (*process2.exe*), (*process3.exe*), and (*process2.exe*) respectively. The new labels are then updated in the histogram.

The runtime complexity of Algorithm 1 is $O(h^2|\Delta E|)$ for a given batch of edge insertions ΔE. For the initial snapshot, the runtime complexity is $O(h^2|E_0|)$, where E_0 represents the number of edges in the initial snapshot. The initialization phase (lines 1-6) can be completed in $O(h^2|V_{new}|)$, where V_{new} is the set of newly inserted nodes in the given snapshot, and the entire vertex set for the initial snapshot. After the initialization, the computation of provenance labels occurs in $h \times |\Delta E| \times h$ operations, as the process needs to update i-provenance labels for each inserted edge for $0 \leq i \leq h$.

While the complexity is higher than $O(h|\Delta E|)$ of WL subtree kernel [58] with the h-hop neighborhood, it is important to note that the value of h is typically small(e.g., ≤ 4). In the WL subtree kernel, the authors use cross-validation to find the best value of tree depth and always land on values between 2, 3, or 4. In Unicorn, the neighborhood radius R parameter equivalent to h is set to 3 for all experiments. Due to this affinity to small h, the overhead from the quadratic scaling is generally negligible.

3.5 Featurization of Histograms

Most machine learning algorithms require a fixed-size input vector. The node label histograms from different snapshots have a different number of bins, i.e., distinct node labels. We need to convert these variable-size histograms to fixed-sized vectors. Let us assume histograms $H_0, H_1, ..., H_k$ are generated from graph snapshots $G_0, G_1, ..., G_k$. A label vocabulary Σ is the set of all the distinct labels computed for all the nodes in all the graph snapshots, i.e., $\Sigma = \cup_{i=0}^{k} L_i$, where L_i is the bins (labels) from H_i.

3.5.1 Sparse label frequency. In a static setting, where all the graph snapshots are already available, we accumulate the unique labels from all the graph snapshots to build a label vocabulary Σ. Then we convert the histogram H_i of each snapshot into a sparse vector V_i of size equal to $|\Sigma|$, where the element at the i^{th} index is the frequency of label i in the histogram of the given snapshot. The vector elements corresponding to the labels not present in the snapshot are set to 0. For each label l in vocabulary Σ,

$$V_i[l] = \begin{cases} H_i[l] & \text{if } l \in H_i \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

To assess the similarity between two vectors $V_i \in \mathbf{R}^{\Sigma}$ and $V_j \in \mathbf{R}^{\Sigma}$, we can compute the distance between them using normalized min-max, a popular distance measure for non-negative vectors.

$$D_{NMM}(V_i, V_j) = \frac{\sum_{l \in \Sigma} min(V_i[l], V_j[l])}{\sum_{l \in \Sigma} max(V_i[l], V_j[l])} \tag{2}$$

3.5.2 Histosketch. In the streaming setting, where the label vocabulary is continuously expanding, we utilize a histosketch data structure to convert the variable-sized histogram H_i into a fixed-size sketch vector S_i of size K. Histosketch employs consistent weighted hashing to transform the histogram into a compact sketch. By applying this technique, we can represent each snapshot with a fixed-size vector, regardless of the growing label vocabulary.

For a node label histogram H, where each label l has a frequency value, $H[l] \geq 0$, Consistent Weighted Sampling (CWS) produces $(l, a_l) : 0 \leq a_l \leq H[l]$, which is both uniform and consistent. This CWS sample (l, a_l) corresponds to the node label histogram bin (l) and its scaled weight (a_l) and is uniformly sampled from $\cup_l \{l\} \times [0, H[l]]$. This means the probability of selecting label l from H is proportional to its label frequency in the histogram, i.e., $H[l]$, and y is uniformly distributed on $[0, H[l]]$. The sample is also consistent, which means given two histograms H_1 and H_2, if $\forall l$, $H_1[l] \leq H_2[l]$, a sub-element (l, a_l) is sampled from H_1 and satisfies $y \leq H_2[l]$, then (l, a_l) will also be sampled from H_2 [34, 73].

To generate a consistent sample for a member of the node label histogram, CWS uses three distributions using all labels from the histogram. For each $l \in H$, and $k = 1, 2, ..., K$, CWS samples from $\gamma_{l,k} \approx$ gamma(2,1), $\beta_{l,k} \approx$ uniform(0,1), and $c_{l,k} \approx$ gamma(2,1). Once these distributions have been sampled, CWS generates a consistent sample for all labels in the histogram as follows.

$$y_{l,k} = exp(log(H[l]) - \gamma_{l,k}\beta_{l,k}) \tag{3}$$

$$a_{l,k} = \frac{c_{l,k}}{y_{l,k} exp(\gamma_{l,k})} \tag{4}$$

Equations 3 and 4 generate 'active indices' and hash a label l in proportion to its weight respectively. For each sketch element, it chooses a label i.e., $S_k = argmin_{l \in H} a_{l,k}$, and corresponding hash value $A_k = min_{l \in H} a_{l,k}$. Given two sketches S_i and S_j constructed from two histograms H_i and H_j respectively, the collision probability in sketches is exactly the normalized min-max similarity between the histograms(or sparse label frequency vectors):

$$Pr[S_i[k] = S_j[k]] = D_{NMM}(H_i, H_j) \tag{5}$$

where $k = 1, 2, ..., K$. The normalized min-max similarity between two histograms can be approximated by computing the hamming distance between two sketches. The computation over sketches S, which are compact and of fixed size, is much more efficient than the one over the full histogram H, which is a large, ever-growing vector.

Histosketch can support a real-time update where every single update in the histogram can be reflected in sketch vector in $O(K)$.

For each distinct label l in histogram H, it requires $O(K)$ space to store the pre-computed distributions $\gamma_{l,k}$, $\beta_{l,k}$, and $c_{l,k}$, where $k = 1, 2, 3, ...K$, thereby making the resultant space complexity $O(K \times |H|)$.

4 EVALUATION

We utilize the X-Stream [54], an edge-centric graph computing framework, to implement the graph kernels. This framework supports both in-memory and out-of-core graphs, enabling scalable computing on shared memory machines. In our implementation, node labels are stored on the vertices, and in each iteration of the graph kernel, the labels are scattered via edges and aggregated on the affected nodes to compute the set of newly formed labels from the streamed edges. This approach allows for efficient computation and maintenance of histograms and sketches in memory, while storing the provenance graph itself on disk. Other components of the Prov2vec system, such as downstream task modeling and data parsing, are implemented using Python.

We evaluate Prov2vec with three different datasets:

StreamSpot dataset generated by [39] contains information flow graphs derived from one attack and five benign scenarios. Each of the benign scenarios involves a normal task: watching YouTube, downloading files, browsing cnn.com, checking Gmail, and playing video games. The attack graphs are captured while a drive-by-download is triggered by visiting a malicious URL that exploits a flash vulnerability and gains root access to the visiting host. Each task is run 100 times on a **Linux** machine collecting a total of 600 graphs, where each graph encompasses all the system calls on the machine from boot up to shut down. In total, there are 5 different subject/object types and 29 different event types.

SupplyChain attack scenarios dataset [18] contains a whole system provenance including background activity captured by CamFlow (v0.5.0) [46] while simulating two supply chain attacks SC-1 and SC-2 on a continuous integration (CI) platform. They follow a typical cyber kill chain with 7 non-exclusive phases, i.e., reconnaissance, weaponization, delivery, exploitation, installation, command and control (C&C), and actions on objective [40]. In SC-1, GNU wget version 1.17 is exploited (CVE-2016-4971) using remote file upload when the victim requests a malicious URL to a compromised server. In SC-2, a vulnerability (CVE-2014-6271) from GNU Bash version 4.3 is exploited to allow remote attackers to execute arbitrary code via crafted trailing strings after function definitions in Bash scripts. Each scenario generates 125 graphs from the benign activities and 25 graphs from the attacker's activities.

Operational Transparent Cyber (OpTC) data [67] was collected over nine days at National Cyber Range in a simulated network with one thousand hosts, with half of the client machines turned off during data collection. Each host was running **Windows** 10 on VMware and was scripted to mimic daily user activities by performing common tasks such as creating, editing, and deleting word, powerpoint, excel, and text files; sending, receiving, and downloading files via emails; and browsing the internet. Three red-team APT exercises were performed, each on a separate day, where randomly chosen machines were targeted, compromised, and used to laterally move on to the other network clients. This dataset contains more than 17 billion events, from 500 hosts and 627 different users. Among these log events, there are 11 object types and 32 different event types. The most popular objects are FLOW (71.7%), FILE (12.4%), PROCESS (8.6%), MODULE (3.9%), THREAD (3.0%), and REGISTRY (0.3%). The rest of the objects constitute less than 0.1% of overall events. Only 0.3 million, approximately 0.0016% of the total events, are malicious [1].

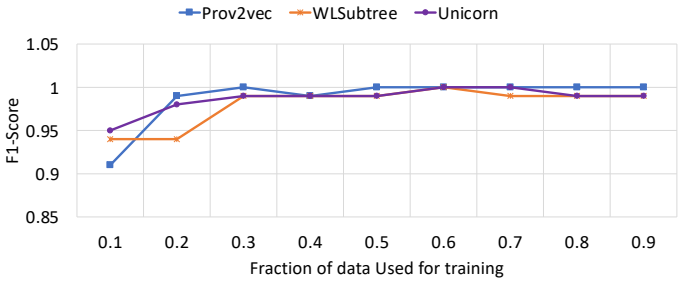

Figure 4: Performance of graph classification.

For comparison, we implemented two other existing graph kernels: (1) Weisfeiler-Lehman Subtree kernel (**WLSubtree**) [58] is implemented to include both edge labels and node labels in their aggregation. Using the edge and node label of each incoming neighbor of the given node v, a sorted multi-set of labels is built which is concatenated with the label of v. (2) The temporally ordered Weisfeiler-Lehman Subtree (**Unicorn**) kernel [18] is implemented. The underlying setups can be reused for each approach, where we only need to re-implement the kernel functions.

Downstream Tasks: We utilize the representation obtained from provenance graph kernel in three distinct downstream tasks to measure the effectiveness of different representation for specific applications.

- **Graph classification** classifies the provenance graphs based on the underlying action being performed on the system. We use XGBoost classifier [69] for graph classification.
- **Novelty detection** is used to detect anomalous behavior in homogeneous systems using one-class support vector machine [55].
- **Anomaly detection** using K-Medoids Clustering. It uses the partitioning around medoids (PAM) algorithm to minimize the distance between points labeled to be in a cluster and a point designated as the center of that cluster [68]. It is useful for detecting anomalous behavior in a heterogeneous system, i.e., a system with multiple benign behavior profiles.

The average performance from 5-fold cross-validation is reported in all of the prediction task reporting. The five-fold split is only performed in benign graphs for the task of anomaly/novelty detection, i.e., four-fifths of benign data are used to train the model.

4.1 Graph Classification

We obtain the static histograms on StreamSpot datasets, i.e., for each task and each run, one graph is built, and one histogram is constructed. We convert the histograms to sparse label frequency vectors, i.e., the feature vectors used here have sizes equal to the number of distinct node labels among all graphs, i.e., vocabulary

size. We evaluate the ability of Prov2vec to distinguish between six activities (youtube, download, cnn, gmail, vgame, and attack) based on the provenance label histogram they generate. We use h = 3, i.e., the 3-hop neighborhood labels are collected for all of the different kernels. We use supervised learning by training the XGB Classifier with a varying number of graphs and use the remaining graphs to test the classification performance. As depicted in Figure 4, all three kernel-based classifiers can reach the peak classification performance in as little as around 20 graphs per task, and Prov2vec performs slightly better than the other two methods. This depicts the ability of the provenance kernel to identify similar tasks, via a comparison of their provenance labels, with a reasonable amount of data.

4.2 Static Novelty Detection

Using unsupervised learning, we can predict the graphs that correspond to the attacks. We utilize 80% of all benign tasks (400 graphs in StreamSpot and 100 graphs in SC-1 and SC-2) as normal behavior profiles and use them to train One-class SVM. The remaining 20% of the benign activity graphs and all the graphs generated from the attack scenarios are used to test the anomaly detector, i.e., 200 graphs in StreamSpot and 50 graphs each in SC-1 and SC-2 respectively. Table 1 shows the performance for all three graph kernels and Figure 5 shows the area under ROC curve for three kernel functions on the three datasets. Prov2vec outperforms both WLSubtree and time-ordered WL Subtree kernel from Unicorn [18].

Despite having a significantly smaller histogram size (Figure 8), the Prov2vec outperforms both WLSubtree and time-ordered WL Subtree kernel from Unicorn [18]. The lower dimension of features helps the runtime of training and testing, while the better generalization of provenance using the concise histogram helps us to minimize the false positives, thereby improving the prediction ability of the anomaly detector.

Table 1: The performance of one-class svm based anomaly detection on three different graph kernels (used h = 3 on each kernel). P, R, A, and F1 represents precision, recall, accuracy, and f1-score respectively.

Dataset	Kernel	P	R	A	F1	Runtime (Sec)
StreamSpot	**Prov2vec**	**0.9708**	**1.0**	**0.985**	**0.9852**	**0.061**
	WLSubtree	0.76	0.99	0.84	0.8609	1.281
	Unicorn	0.7353	1.0	0.82	0.8475	3.034
SC-1	**Prov2vec**	**0.7742**	**1.0**	**0.8571**	**0.8727**	**1.445**
	WLSubtree	0.6857	1.0	0.7755	0.8136	5.281
	Unicorn	0.7059	1.0	0.7959	0.8276	8.016
SC-2	**Prov2vec**	**0.7353**	**1.0**	**0.82**	**0.8475**	**1.751**
	WLSubtree	0.7143	1.0	0.8	0.8333	10.687
	Unicorn	0.6579	1.0	0.74	0.7937	14.539

4.3 Real-time Anomaly Detection

The OpTC data provides a much better representation of real-world enterprise networks. The host logs for 500 different Windows-10 hosts are collected over 9 days. During the first six of 9 data collection days, only normal activities are performed on each host such as browsing the internet, playing video games, using Gmail, etc. Those 6 days are divided into 4 different boot-up to shut down sessions, i.e., (1) 17-18th, (2) 18-19th, (3) 19th, and (4) 20th - 23rd September 2019. We build different graphs for each host during each of these sessions, where the node label histogram is maintained incrementally and a snapshot is taken periodically. The series of histogram snapshots are then converted into fixed-sized sketch vectors of length 2,048. Next, all the sketches are clustered using the k-medoid algorithm where an optimal number of clusters is determined by maximizing the silhouette coefficient [48]. The trained k-medoid is used for anomaly detection during the evaluation period.

The APT attack exercises were performed during the last three days, where one attack campaign was carried out each day. During the evaluation period, we create a provenance graph on each host every day and incrementally run graph kernels to compute node label histograms. The snapshots of histograms are taken every hour and are converted to sketch vectors. The resultant sketch vector is then tested against the k-medoids model trained during benign activity duration. If the sketch does not fit on any of the underlying clusters in the trained model, the snapshot is considered an anomaly. If a host on a given evaluation day has at least one anomalous snapshot, we raise an alert indicating that the host may have been compromised.

Table 2: The anomaly detection results on 3 attack campaigns using k-medoids algorithm for h = 3 and sketch size = 2048.

Attack	Kernel	P	R	A	F1
Day1- Powershell Empire	**Prov2vec**	**1.0000**	**0.1765**	**0.9720**	**0.3000**
	WLSubtree	0.6000	0.1765	0.9680	0.2727
	Unicorn	0.4000	0.1176	0.9640	0.1818
Day2-Deathstar	**Prov2vec**	**1.0000**	**0.3333**	**0.9880**	**0.5000**
	WLSubtree	0.6667	0.2222	0.9840	0.3333
	Unicorn	0.3333	0.2222	0.9780	0.2667
Day3-Malicious Update	**Prov2vec**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
	WLSubtree	0.2000	1.0000	0.9840	0.3333
	Unicorn	0.2857	1.0000	0.9900	0.4444

Table 2 shows the performance for detecting compromised hosts on each day of the attack. We use a period of one hour between snapshots, neighborhood size of *h=3* for graph kernels, and sketch the size of *2048*. The precision represents the fraction of detected hosts that were actually compromised, while recall represents the fraction of compromised hosts that are detected. First, the precision of Prov2vec kernel is much better than both *WLSubtree* and *Unicorn* kernels. This is down to the better generalization of labels on histogram resulting from Prov2vec. This results in a more succinct histogram for Prov2vec kernel compared to the other two techniques, thereby enabling a smaller sketch to capture the system behaviors. Notice that the recall is noticeably low for all of the kernels during day1 and day2. This is because, during these campaigns, there is hardly any activity on some of the compromised hosts where an attacker simply logs in after obtaining the credential from the domain controller. Below we discuss each of these attack campaigns in detail.

Attack Day 1 Analysis. The attack campaign on day 1 uses PowerShell empire [14], where it manually connects to *Sysclient201* as the user *zleazer* and downloads malicious Powershell Empire stager. It then uses privilege escalation methods to obtain elevated agents, Mimikatz to collect credentials, registry edits to establish persistence, and discovery techniques to gather system and network information. It then pivots to *Sysclient402* using WMI invoke as an elevated agent where it performs a ping sweep of the local

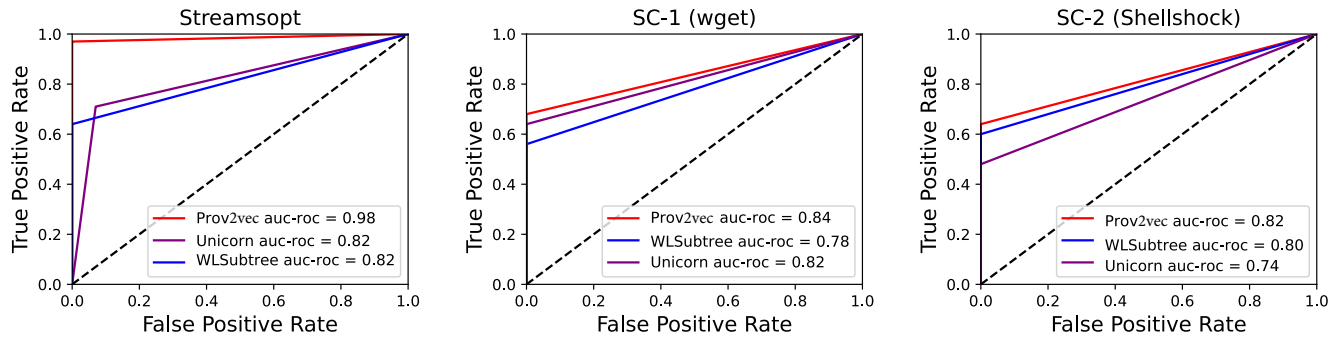

Figure 5: ROC curve of one-class SVM based novelty detection for three different graph kernels on different datasets. The area under ROC curve for Prov2vec kernel is consistently better than *WLSubtree* and *Unicorn* kernels.

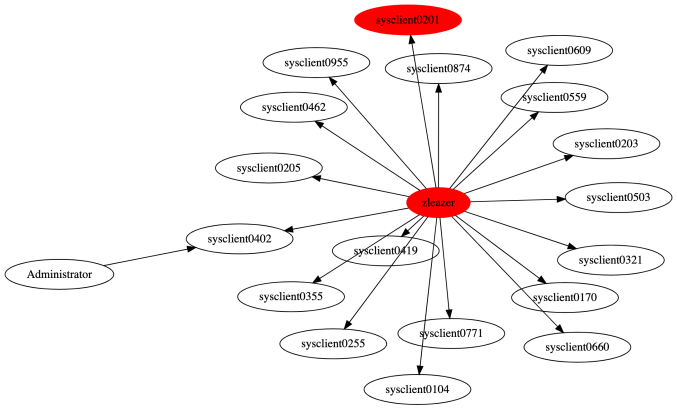

Figure 6: The movement of compromised user across network during attack campaign of attack day 1.

network and pivots to *Sysclient660*. Finally, it obtains domain controller information by using Powershell commands, pivots to *DC1* (domain-controller 1), where it obtains the user hashes using lsa, and pivots to 14 different hosts.

All kernels including Prov2vec flags *Sysclient201* and *Sysclient660* as compromised, however the Unicorn kernel missed *Sysclient402*. The remaining 14 hosts are missed as they do not have enough log data produced during the attacker's presence, and we could not flag the domain controller since there is no log collected for it. Figure 6 shows the attacker's movement across the network during the attack day 1.

Attack Day 2 Analysis. The attack campaign was carried out using Deathstar, which started with a phishing email containing malicious Powershell stagers to two users *bantonio* and *rsantilli*. On *Sysclient501*, *bantonio* opened the malicious attachment. Once checked in the attacker runs a series of commands to list domain controllers, SID, and admins. It used several UAC bypass techniques available in Powershell Empire such as *eventvwr*, *fodhelper*, *wmi invoke*, and *windir value modification* to escalate the privilege. It then started reverse shell to the attacker, which downloaded a netcat application with a different alias, compressed the content of *Documents* folder into a file named *export.zip*, and copied it to news.com hosted at *132.197.158.98*. The attacker pivoted to *Sysclient974* and explored files in the Documents folder. Similarly, it pivoted to *Sysclient005*, where it exfiltrated the data from the Downloads folder. The hosts *Sysclient501*, *Sysclient974*, and *Sysclient005* were 3 out of 9 compromised hosts that were detected by all three kernels. Figure 7 shows the attacker's movement across the network during the attack day 2.

Attack Day 3 Analysis. Here two hosts installed *notepad.exe* susceptible to malicious upgrade, which when updated reached out to the attacker's server hosted at *53.192.68.50* and downloaded a reverse tcp meterpreter payload that connected back to the attacker. Once connected, it runs discovery techniques to gather information on the local system, applications, domain controllers, and network shares. It then migrated to the *lsass* process, which used *Mimikatz* to collect clear-text passwords and hashes. Afterward, persistence was maintained by installing run keys, and user 'admin' was added to administrators and the RDP group. A similar approach was taken on both hosts *Sysclient351* and *Sysclient051*, where they left large enough footprints for an anomaly detector to trigger the alert.

We utilize a user-host interaction graph built using the user-session logs to flag potentially compromised hosts and users to quickly extract the impacted agents. The user-session logs in the OpTC data contain information such as user logins, logouts, and remote desktop protocol accesses. When we detect a compromised host using real-time anomaly detection on provenance graph snapshots, we extract the metadata from such anomalies, mainly the user, host, and the timestamp of the first anomaly. Following those agents and time information, we perform a temporal traversal on the user-host graph to obtain the potentially compromised hosts.

With the help of the temporal traversal, we can detect all the compromised hosts on day 1 as shown in Figure 6, except domain controller 1 (DC1) as we do not have user-session logs for DC1. In addition, it produces one false positive *sysclient0203* which was not mentioned in the ground truth. On day 2 as shown in Figure 7, this traversal led to a few false positives as *bantonio* logs into hundreds of hosts following the detection of an anomaly on *Sysclient501*. However, the user with the elevated privilege, i.e., *Administrator* connected to all 9 hosts mentioned in the ground truth, which can be traced from the user-session logs. Again with temporal traversal, we can detect the compromised hosts that were missed by anomaly

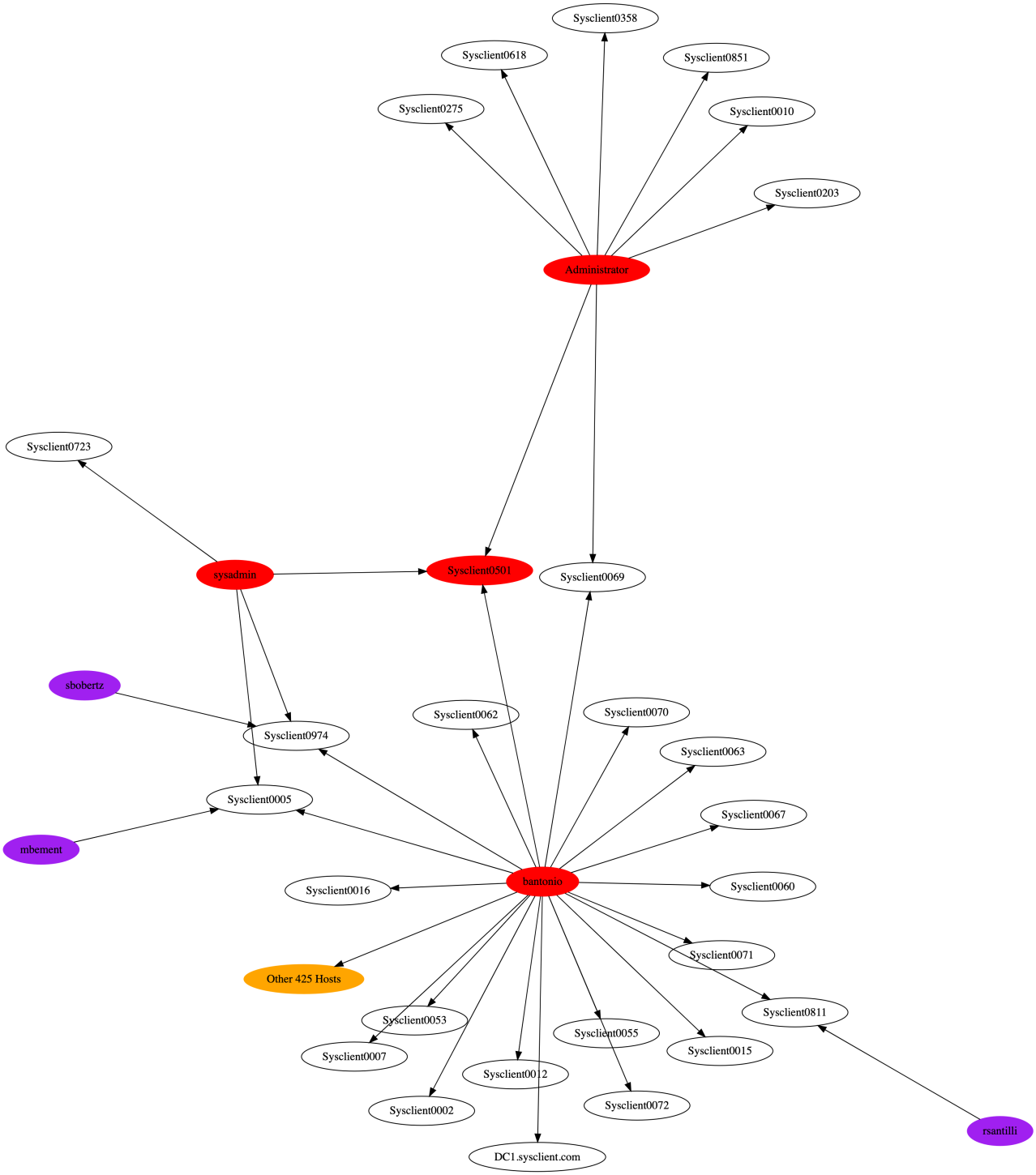

Figure 7: The movement of the compromised user across the network during the attack campaign of day 2.

detection as long as the anomaly detection finds at least one of the compromised hosts.

4.4 Effect of Sketch Size

We evaluate the impact of using a fixed-size sketch vector in the performance of downstream tasks compared to the use of a sparse label histogram of size equal to the number of distinct labels among

Table 3: The evaluation of effect of different sized histosketches on the anomaly detection performance on StreamSpot data. K is sketch vector size.

Sketch	Prov2vec Kernel				WLSubtree kernel				Unicorn Kernel			
	P	R	A	F1	P	R	A	F1	P	R	A	F1
32	0.81	1	0.88	0.89	0.82	1	0.89	0.9	0.84	1	0.91	0.91
64	0.83	1	0.9	0.9	0.8	1	0.88	0.89	0.83	1	0.9	0.9
128	**0.9**	**1**	**0.94**	**0.95**	0.83	1	0.9	0.91	0.76	1	0.85	0.87
256	0.9	1	0.94	0.95	0.88	1	0.93	0.94	0.85	1	0.91	0.92
512	0.89	1	0.94	0.94	**0.89**	**1**	**0.94**	**0.94**	0.86	1	0.92	0.92
1024					0.89	1	0.94	0.94	**0.89**	**1**	**0.94**	**0.94**
2048					0.89	1	0.94	0.94	0.89	1	0.94	0.94

all graphs. We vary the size of the sketch from 32 to 2,048, doubling each time to represent the node label histogram obtained by running all three kernels for $h = 3$. The histogram sketch obtained is thus used as the feature representation for the given graph. We train the k-medoids clustering algorithm using 80% of the graphs generated by benign activities. The remaining 100 benign graphs and 100 graphs generated during the attack are used for testing. During testing, each graph is tested against every cluster formed during training and flagged as an anomaly if it does not fit in any of the clusters. A graph is considered to fit in a cluster if its distance from the given clusters' medoid is within d standard deviation of the mean distance of all training samples in that cluster. In our experiments, we used $d = 2$, i.e., if a sample is farther than $mean + 2std$ away from all the medoids, it is considered an anomaly. The performance for varying sizes of sketches is shown in Table 3 for anomaly detection on StreamSpot data.

The results in Table 3 show that sketch size much smaller than the node label vocabulary size can match the performance for all kernels. The performance for Prov2vec kernel saturates after a sketch size of 128. Similarly the performance for *WLSubtree* and *Unicorn* kernels saturates at sketch sizes of 512 and 1,024 respectively. The peak performance of *WLSubtree* and *Unicorn* kernels match that of their sparse histogram vector counterpart from Table 1. However, the precision of Prov2vec kernel is slightly amiss from its static counterpart. Nevertheless, sketching constantly changing and different-sized histograms with fixed-size feature sketches preserves their similarity and provides a viable option for comparing continuously changing provenance graphs.

4.5 Effect of Neighborhood Size

We compare the resource consumption for using different kernels to compute the node label histograms in different datasets. We varied the value of h, i.e., the size of the neighborhood, and recorded the histogram size as well as the runtime for different graph kernels. As illustrated in Figure 8(a)-(f), the histogram for the 0-hop neighborhood is identical for all kernels, i.e., histograms built on node types. As the value of h increases, the differences between the sizes of the histogram for *Unicorn* and *WLSubtree* kernels compared to Prov2vec kernel get larger.

We evaluate the impact of neighborhood size (h) based on the performance of corresponding histograms in downstream machine-learning tasks. We use two SupplyChain datasets to evaluate the impact of neighborhood size on anomaly detection. We convert the histograms of corresponding snapshots to sketch vectors of size 2,048. The performance for anomaly detection is shown in Table 4 for two attack scenarios SC-1 (wget) and SC2 (shellshock). As expected, the performance for each kernel improves as we increase the neighborhood size, reaches the peak for the value of h = 3 or 4, and start to decline afterward. The provenance graph kernel does incur longer runtime as presented in Figure 8(g)-(i). However, the optimal value of h is usually small, thereby alleviating the impact of quadratic scaling.

The comparison of histogram size growth over time for three kernels is shown in Figure 9. The number of labels and rate of arrival of unseen labels are much smaller in the provenance graph kernel. Despite this succinct representation, the performance on downstream task for Prov2vec kernel is consistently better or comparable to the other two kernels as illustrated earlier.

5 DISCUSSIONS AND LIMITATIONS

Prov2vec makes certain assumptions and has limitations that should be considered.

First, it operates under the **closed-world assumption**, assuming that all benign behaviors have been observed during training [60]. However, in real enterprise networks, it is challenging to cover all possible benign cases. This may result in false alarms for previously unseen normal behaviors. To address this, system administrators can periodically update the model with new benign data. The incremental nature of Prov2vec makes it easy for the model to update.

Second, Prov2vec assumes an **integrity of training data** during a modeling period. It assumes that the newly observed normal behavior used for model updates is not corrupted by poisoning attacks [61] or graph backdoors [70]. The robustness of Prov2vec against such attacks is an area for future study.

The **datasets used in the experiments are synthetic**, which limits the representation of real-world APT attacks. While efforts have been made to make the datasets realistic, they lack some characteristics of APT attacks in the wild. Testing Prov2vec against actual enterprise systems or more realistic APT scenarios is a priority for future research.

Granularity of data provenance: Some attacks do not produce the attack pattern in the data provenance graphs. For example, malicious code in a file and thread-based attacks have the text information on the corresponding files and threads that are too fine granular to be recorded in the provenance graph. Like all provenance-based detection methods, Prov2vec will fail to detect those attacks. Incorporating more host-based data into the threat detection process or improving the information capture process

Table 4: The evaluation of the effect of different sized neighborhoods on the anomaly detection performance on SupplyChain data.

		Prov2vec				WLSubtree				Unicorn			
SC-1	h	P	R	A	F1	P	R	A	F1	P	R	A	F1
	1	0.5333	0.3333	0.5306	0.4103	0.5333	0.3333	0.5306	0.4103	0.5333	0.3333	0.5306	0.4103
	2	0.7778	0.875	0.8163	0.8235	0.7308	0.7917	0.7551	0.76	0.6667	0.8333	0.7143	0.7407
	3	**0.8148**	**0.9167**	**0.8571**	**0.8627**	0.7778	0.875	0.8163	0.8235	0.8148	0.9167	0.8571	0.8627
	4	0.7333	0.9166	0.7959	0.8148	**0.84**	**0.875**	**0.8571**	**0.8571**	**0.8148**	**0.9167**	**0.8571**	**0.8627**
	5	0.75	0.875	0.7959	0.8077	0.8333	0.8333	0.8367	0.8333	0.7586	0.9167	0.8163	0.8302
SC-2	h	P	R	A	F1	P	R	A	F1	P	R	A	F1
	1	0.5	0.04	0.5	0.0741	0.5	0.04	0.5	0.0741	0.5	0.04	0.5	0.0741
	2	0.7222	0.52	0.66	0.6047	0.6	0.6	0.6	0.6	0.5862	0.68	0.6	0.6296
	3	**0.7407**	**0.8**	**0.76**	**0.7692**	**0.7143**	**0.8**	**0.74**	**0.7547**	0.6552	0.76	0.68	0.7037
	4	0.7727	0.68	0.74	0.7234	0.6333	0.76	0.66	0.6909	**0.75**	**0.72**	**0.74**	**0.7347**
	5	0.75	0.48	0.66	0.5853	0.5909	0.52	0.58	0.5532	0.6552	0.76	0.68	0.7037

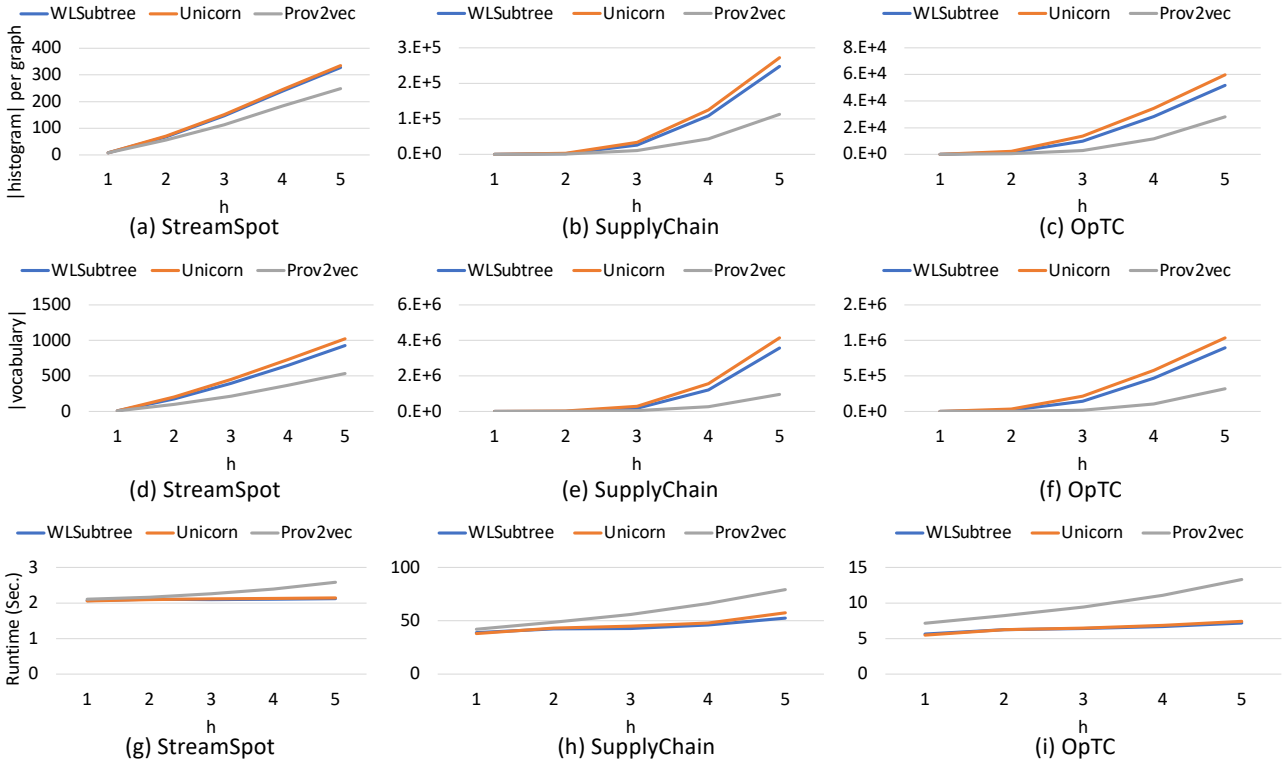

Figure 8: The comparison of resource consumption for different kernels. The plots (a)-(c) shows the average size of the histogram per graph, plots (d)-(f) shows the vocabulary size for different kernels, and plots (g)-(i) compares the runtime of different kernels for increasing neighborhood size.

for finer-grained provenance graph generation can be the research directions to further investigate this limitation.

The explainability of anomalies is a challenge in black-box machine learning systems. Prov2vec may struggle to provide detailed explanations for the detected anomalies. However, methods such as LIME and EDR systems can be used to explain individual predictions and understand the series of activities leading to an anomaly.

The provenance graph kernel **only supports discrete labels**, which limits its ability to capture continuous attributes. Including such attributes may require the use of deep learning techniques or graph kernels that support continuous attributes. Overall, while Prov2vec has shown promising results, addressing these limitations will be crucial for its broader applicability and effectiveness in detecting sophisticated attacks.

6 RELATED WORKS

Provenance graph has been popular tool for threat hunting research in last few years. Several works have been proposed to improve the provenance data collection [3, 46, 50], redundancy elimination [19, 33, 38, 72], intrusion detection using provenance

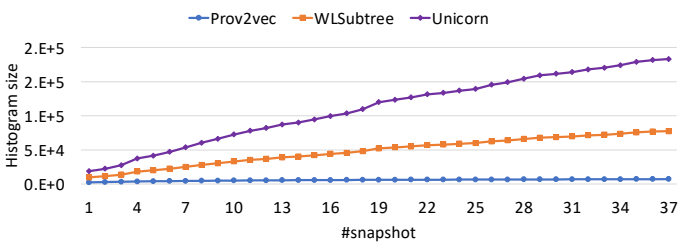

Figure 9: The histogram size trend with each hourly snapshot on host 201 during 16-17Sep on OpTC data.

graphs [5, 9, 12, 13, 18, 20, 22, 24, 36, 42, 43]. We refer interested readers to the comprehensive survey on threat detection techniques using provenance graph [77]. Traditional query systems are not optimized for provenance analysis. Several **provenance query systems** [12, 13, 47, 59] have been proposed to provide threat investigation abilities such as streaming queries, causality tracking, graph pattern matching, and anomaly analysis. These systems are implemented on top of mature stream processors or databases and take the provenance graphs specific data model and query engine.

Provenance data reduction is important for storage and computational efficiency. Causality preserving reduction [72] and subsequent dependence preserving reduction [25] merge the events if they do not alter the causality or forward and backward reachability respectively. LogGC [33] proposes a provenance garbage collection, that finds the isolated "temporary" nodes and removes them. Since garbage collection and causality/dependency preserving reduction can remove correlation between alerts or alert themselves, we modified these reduction systems to preserve alerts.

Threat detection with provenance graphs: Sleuth [24] uses policy based rules to trigger alerts and uses **tag propagation** technique to store and transmit the system execution history. The **abnormal behavior detection** systems [21, 37] learn host behavior from historical data or parallel systems and try to find abnormal interaction between system entities. The **graph pattern matching and alignment** based works such as Holmes [43], Poirot [42], Rapsheet [20], and SteinerLog [5] use indicator of attacks (IOAs) to generate suspicious events and chain them together using graph exploration techniques. They use those chain of alerts to detect the attacks as well as to reconstruct the individual steps taken by an attacker. However, a substantial amount of manual effort and domain expertise is required to come up with the relevant IOAs for matching. For example, Poirot requires one to write a different query for each of the attack campaigns and find their alignment on a provenance graph. Holmes [43], Rapsheet [20] and SteinerLog [5] use more fine-grained behavioral patterns representing different TTPs relevant to their system and follow the causal dependency in provenance graph to construct the attack campaigns.

Recent works have applied machine learning techniques on the provenance graph for providing behavioral modeling based threat detection approaches. Shadewatcher [75] formulates the threat detection as recommendation problem by likening the system interactions on audit logs to the entity relations on recommendation systems. PROGRAPHER [74] uses graph2vec in to obtain the representations of the graph snapshots, and provides detection on node level granularity. ANUBIS [2] uses rather probabilistic approach by using Poisson distribution to model the causal neighborhood of a given event. Log2Vec [36] combines the random walk with word2vec to obtain the node representations. Prov2vec follows the **graph kernel based** representation such as Unicorn [18] closely, where it uses discretely mined features to obtain graph representation and perform anomaly detection based on it. In Prov2vec, more compact histogram and consequently better generalization is achieved and we are able to outperform Unicorn in series of tasks.

Graph kernels are widely used for learning node and graph representations in machine learning tasks. These techniques iteratively accumulate and compress information from a node's neighborhood to derive a new node label. Various methods, such as random walks [28, 66, 76], subtrees [52, 58], cyclic patterns [23], shortest paths [6], and graphlets [51], are employed to capture node neighborhoods. Recently, Graph Neural Networks (GNNs) [16, 32, 35, 65, 71] have gained popularity for representation learning, with promising results in cybersecurity applications [26, 29, 30]. GNNs utilize recursive aggregation to compute a node's representation vector by incorporating information from its neighborhood, with each iteration encompassing a larger one-hop neighborhood. Node representations are then aggregated to obtain the feature vector for the entire graph.

Sequence-based learning techniques, which involve converting log sequences into key vectors representing system events, have gained popularity in operational anomaly detection [17, 41, 64]. Models based on recurrent neural networks (RNNs) or Transformers are then trained with these key sequences [7–9]. During deployment, these models predict anomalous behavior by forecasting the next event based on the observed sequence. However, their effectiveness is limited as they mainly examine short system call sequences and struggle to capture long-term behavior, leaving them vulnerable to evasion techniques. To detect stealthy and slow Advanced Persistent Threat attacks, which require a broader context, graph-based techniques leveraging the causal relationships among events in provenance graphs offer more promising solutions.

7 CONCLUSION

We design and implement a fully unsupervised technique in Prov2vec, which is able to successfully learn the system host behaviors from their provenance graphs and identify the potentially malicious behaviors that differ from the normality. The new provenance graph kernel, while incurs a slight overhead in histogram computation compared to state-of-the-art graph kernels, achieves an order magnitude smaller node label histogram sizes and significantly improves the performance of downstream machine learning tasks. The result from Prov2vec can be used as the first level of filtering for fine-grained alert correlation systems, where the anomalous hosts are further inspected to understand the context around underlying behavior.

ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation grant 212720.

REFERENCES

[1] Md. Monowar Anjum, Shahrear Iqbal, and Benoit Hamelin. 2021. Analyzing the Usefulness of the DARPA OpTC Dataset in Cyber Threat Detection Research. *CoRR* abs/2103.03080 (2021). arXiv:2103.03080 https://arxiv.org/abs/2103.03080

[2] Md Monowar Anjum, Shahrear Iqbal, and Benoit Hamelin. 2022. ANUBIS: a provenance graph-based framework for advanced persistent threat detection. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*. 1684–1693.

[3] Adam Bates, Dave Jing Tian, Kevin RB Butler, and Thomas Moyer. 2015. Trustworthy whole-system provenance for the linux kernel. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*. 319–334.

[4] Siddharth Bhatia, Bryan Hooi, Minji Yoon, Kijung Shin, and Christos Faloutsos. 2020. Midas: Microcluster-based detector of anomalies in edge streams. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

[5] Bibek Bhattarai and Howie Huang. 2022. SteinerLog: Prize Collecting the Audit Logs for Threat Hunting on Enterprise Network. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security (ASIA CCS '22)*. Association for Computing Machinery, 97–108. https://doi.org/10.1145/3488932.3523261

[6] Karsten M Borgwardt and Hans-Peter Kriegel. 2005. Shortest-path kernels on graphs. In *Fifth IEEE international conference on data mining (ICDM'05)*. IEEE, 8–pp.

[7] Biplob Debnath, Mohiuddin Solaimani, Muhammad Ali Gulzar Gulzar, Nipun Arora, Cristian Lumezanu, Jianwu Xu, Bo Zong, Hui Zhang, Guofei Jiang, and Latifur Khan. 2018. LogLens: A real-time log analysis system. In *2018 IEEE 38th international conference on distributed computing systems (ICDCS)*. IEEE, 1052–1062.

[8] Min Du, Zhi Chen, Chang Liu, Rajvardhan Oak, and Dawn Song. 2019. Lifelong anomaly detection through unlearning. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1283–1297.

[9] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 1285–1298.

[10] Dhivya Eswaran and Christos Faloutsos. 2018. SedanSpot: Detecting Anomalies in Edge Streams. *2018 IEEE International Conference on Data Mining (ICDM)* (2018).

[11] FreeBSD. 2018. DTrace on FreeBSD. https://wiki.freebsd.org/DTrace.

[12] Peng Gao, Xusheng Xiao, Ding Li, Zhichun Li, Kangkook Jee, Zhenyu Wu, Chung Hwan Kim, Sanjeev R Kulkarni, and Prateek Mittal. 2018. {SAQL}: A Stream-based Query System for Real-Time Abnormal System Behavior Detection. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*.

[13] Peng Gao, Xusheng Xiao, Zhichun Li, Fengyuan Xu, Sanjeev R Kulkarni, and Prateek Mittal. 2018. {AIQL}: Enabling Efficient Attack Investigation from System Monitoring Data. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*.

[14] Github. 2022. Powershell Empire. https://github.com/EmpireProject/Empire.

[15] Steve Grubb. 2022. auditd - The Linux Audit daemon. https://linux.die.net/man/8/auditd.

[16] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* (2017).

[17] Hossein Hamooni, Biplob Debnath, Jianwu Xu, Hui Zhang, Guofei Jiang, and Abdullah Mueen. 2016. Logmine: Fast pattern recognition for log analytics. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. 1573–1582.

[18] Xueyuan Han, Thomas Pasquier, Adam Bates, James Mickens, and Margo Seltzer. 2020. Unicorn: Runtime provenance-based detector for advanced persistent threats. *arXiv preprint arXiv:2001.01525* (2020).

[19] Wajih Ul Hassan, Lemay Aguse, Nuraini Aguse, Adam Bates, and Thomas Moyer. 2018. Towards scalable cluster auditing through grammatical inference over provenance graphs. In *Network and Distributed Systems Security Symposium*.

[20] Wajih Ul Hassan, Adam Bates, and Daniel Marino. 2020. Tactical provenance analysis for endpoint detection and response systems. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE.

[21] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. 2019. Nodoze: Combatting threat alert fatigue with automated provenance triage. In *Network and Distributed Systems Security Symposium*.

[22] Wajih Ul Hassan, Mohammad A Noureddine, Pubali Datta, and Adam Bates. 2020. OmegaLog: High-fidelity attack investigation via transparent multi-layer log analysis. In *Proc. NDSS*.

[23] Tamás Horváth, Thomas Gärtner, and Stefan Wrobel. 2004. Cyclic pattern kernels for predictive graph mining. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. 158–167.

[24] Md Nahid Hossain, Sadegh M Milajerdi, Junao Wang, Birhanu Eshete, Rigel Gjomemo, R Sekar, Scott Stoller, and VN Venkatakrishnan. 2017. SLEUTH: Real-time attack scenario reconstruction from COTS audit data. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*.

[25] Md Nahid Hossain, Junao Wang, R. Sekar, and Scott D. Stoller. 2018. Dependence-Preserving Data Compaction for Scalable Forensic Analysis. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association.

[26] Yuede Ji and H. Howie Huang. 2022. NestedGNN: Detecting Malicious Network Activity with Nested Graph Neural Networks. In *ICC 2022 - IEEE International Conference on Communications*. 2694–2699.

[27] Karl-Bridge-MIcrosoft, v kents, DCtheGeek, mcleanbyron, drewbatgit, and msatranjr. 2021. Event Tracing. https://docs.microsoft.com/en-us/windows/win32/etw/event-tracing-portal.

[28] Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. 2003. Marginalized kernels between labeled graphs. In *Proceedings of the 20th international conference on machine learning (ICML-03)*. 321–328.

[29] Isaiah J. King and H. Howie Huang. 2022. Euler: Detecting Network Lateral Movement via Scalable Temporal Link Prediction. In *the Network and Distributed System Security Symposium (NDSS '22)*.

[30] Isaiah J. King, Xiaokui Shu, Jiyong Jang, Kevin Eykholt, Taesung Lee, and H. Howie Huang. 2023. EdgeTorrent: Real-time Temporal Graph Representations for Intrusion Detection. In *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses* (Hong Kong, China) *(RAID '23)*. 77–91.

[31] Samuel T King and Peter M Chen. 2003. Backtracking intrusions. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*.

[32] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[33] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. 2013. LogGC: garbage collecting audit log. In *SIGSAC*.

[34] Ping Li. 2015. 0-bit consistent weighted sampling. In *Proceedings of the 21th ACM SIGKDD International conference on knowledge discovery and data mining*. 665–674.

[35] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2015. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493* (2015).

[36] Fucheng Liu, Yu Wen, Dongxue Zhang, Xihe Jiang, Xinyu Xing, and Dan Meng. 2019. Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1777–1794.

[37] Yushan Liu, Mu Zhang, Ding Li, Kangkook Jee, Zhichun Li, Zhenyu Wu, Junghwan Rhee, and Prateek Mittal. 2018. Towards a Timely Causality Analysis for Enterprise Security.. In *Network and Distributed Systems Security Symposium*.

[38] Shiqing Ma, Xiangyu Zhang, and Dongyan Xu. 2016. Protracer: Towards Practical Provenance Tracing by Alternating Between Logging and Tainting.. In *NDSS*.

[39] Emaad Manzoor, Sadegh M Milajerdi, and Leman Akoglu. 2016. Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1035–1044.

[40] Lochheed Martin. 2022. The Cyber Kill Chain. https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html.

[41] Salma Messaoudi, Annibale Panichella, Domenico Bianculli, Lionel Briand, and Raimondas Sasnauskas. 2018. A search-based approach for accurate identification of log message formats. In *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*. IEEE, 167–16710.

[42] Sadegh M Milajerdi, Birhanu Eshete, Rigel Gjomemo, and VN Venkatakrishnan. 2019. Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1795–1812.

[43] Sadegh M Milajerdi, Rigel Gjomemo, Birhanu Eshete, Ramachandran Sekar, and VN Venkatakrishnan. 2019. Holmes: real-time apt detection through correlation of suspicious information flows. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1137–1152.

[44] Neo4J. 2022. Neo4j Graph Database. https://neo4j.com/product/neo4j-graph-database/.

[45] Thor Olavsrud. 2014. 11 Steps Attackers Took to Crack Target. https://www.csoonline.com/article/2601021/11-steps-attackers-took-to-crack-target.html.

[46] Thomas Pasquier, Xueyuan Han, Mark Goldstein, Thomas Moyer, David Eyers, Margo Seltzer, and Jean Bacon. 2017. Practical whole-system provenance capture. In *Proceedings of the 2017 Symposium on Cloud Computing*. 405–418.

[47] Thomas Pasquier, Xueyuan Han, Thomas Moyer, Adam Bates, Olivier Hermant, David Eyers, Jean Bacon, and Margo Seltzer. 2018. Runtime analysis of whole-system provenance. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*.

[48] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[49] Nicole Perlroth. 2017. All 3 Billion Yahoo Accounts Were Affected by 2013 Attack. https://www.nytimes.com/2017/10/03/technology/yahoo-hack-3-billion-users.html.

[50] Devin J Pohly, Stephen McLaughlin, Patrick McDaniel, and Kevin Butler. 2012. Hi-Fi: collecting high-fidelity whole-system provenance. In *Proceedings of the 28th Annual Computer Security Applications Conference*. 259–268.

[51] Nataša Pržulj. 2007. Biological network comparison using graphlet degree distribution. *Bioinformatics* 23, 2 (2007), e177–e183.

[52] Jan Ramon and Thomas Gärtner. 2003. Expressivity versus efficiency of graph kernels. In *Proceedings of the first international workshop on mining graphs, trees and sequences*. 65–74.

[53] Stephen Ranshous, Steve Harenberg, Kshitij Sharma, and Nagiza F Samatova. 2016. A scalable approach for outlier detection in edge streams using sketch-based approximations. In *Proceedings of the 2016 SIAM International Conference on Data Mining*. SIAM.

[54] Amitabha Roy, Ivo Mihailovic, and Willy Zwaenepoel. 2013. X-stream: Edge-centric graph processing using streaming partitions. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 472–488.

[55] scikit-learn developers. 2022. One Class SVM. https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html.

[56] Security-X. 2018. [FireEye]Operation DeputyDog: Zero-Day (CVE-2013-3893) Attack Against Japanese. https://forum.security-x.fr/news/(fireeye)operation-deputydog-zero-day-(cve-2013-3893)-attack-against-j-27872/.

[57] SentinalOne. 2022. What Is A Malware File Signature (And How Does It Work)? https://www.sentinelone.com/blog/what-is-a-malware-file-signature-and-how-does-it-work/.

[58] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research* 12, 9 (2011).

[59] Xiaokui Shu, Frederico Araujo, Douglas L Schales, Marc Ph Stoecklin, Jiyong Jang, Heqing Huang, and Josyula R Rao. 2018. Threat intelligence computing. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*.

[60] Robin Sommer and Vern Paxson. 2010. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE symposium on security and privacy*. IEEE, 305–316.

[61] Xianfeng Tang, Yandong Li, Yiwei Sun, Huaxiu Yao, Prasenjit Mitra, and Suhang Wang. 2020. *Transferring Robustness for Graph Neural Network Against Poisoning Attacks*. Association for Computing Machinery, 600–608. https://doi.org/10.1145/3336191.3371851

[62] SecureWorld News Team. 2020. Data Breach a 'Huge Cyber Espionage Campaign Targeting the U.S. Government'. https://www.secureworldexpo.com/industry-news/data-breach-cyber-espionage-campaign-targeting-u.s.-government.

[63] The Mitre Corporation. 2022. ATT&CK Matrix for Enterprise. https://attack.mitre.org/.

[64] Risto Vaarandi and Mauno Pihelgas. 2015. Logcluster-a data clustering and pattern mining algorithm for event logs. In *2015 11th International conference on network and service management (CNSM)*. IEEE, 1–7.

[65] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[66] S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. 2010. Graph kernels. *Journal of Machine Learning Research* 11 (2010), 1201–1242.

[67] Dr. Carl Weir, Rody Arantes, Henry Hannon, and Marisha Kulseng. 2021. Operationally Transparent Cyber (OpTC). https://dx.doi.org/10.21227/edq8-nk52. https://doi.org/10.21227/edq8-nk52

[68] Wikipedia. 2022. k-medoids. https://en.wikipedia.org/wiki/K-medoids.

[69] xgboost developers. 2022. XGBoost Classifier. https://xgboost.readthedocs.io/en/stable/python/python_api.html.

[70] Zhaohan Xi, Ren Pang, Shouling Ji, and Ting Wang. 2021. Graph backdoor. In *30th USENIX Security Symposium (USENIX Security 21)*. 1523–1540.

[71] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[72] Zhang Xu, Zhenyu Wu, Zhichun Li, Kangkook Jee, Junghwan Rhee, Xusheng Xiao, Fengyuan Xu, Haining Wang, and Guofei Jiang. 2016. High fidelity data reduction for big data security dependency analyses. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*.

[73] Dingqi Yang, Bin Li, Laura Rettig, and Philippe Cudré-Mauroux. 2017. HistoSketch: Fast Similarity-Preserving Sketching of Streaming Histograms with Concept Drift. In *2017 IEEE International Conference on Data Mining (ICDM)*. 545–554. https://doi.org/10.1109/ICDM.2017.64

[74] Fan Yang, Jiacen Xu, Chunlin Xiong, Zhou Li, and Kehuan Zhang. 2023. {PROGRAPHER}: An Anomaly Detection System based on Provenance Graph Embedding. In *32nd USENIX Security Symposium (USENIX Security 23)*. 4355–4372.

[75] Jun Zengy, Xiang Wang, Jiahao Liu, Yinfang Chen, Zhenkai Liang, Tat-Seng Chua, and Zheng Leong Chua. 2022. Shadewatcher: Recommendation-guided cyber threat analysis using system audit records. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 489–506.

[76] Zhen Zhang, Mianzhi Wang, Yijian Xiang, Yan Huang, and Arye Nehorai. 2018. Retgk: Graph kernels based on return probabilities of random walks. *Advances in Neural Information Processing Systems* 31 (2018).

[77] Michael Zipperle, Florian Gottwalt, Elizabeth Chang, and Tharam Dillon. 2022. Provenance-Based Intrusion Detection Systems: A Survey. *ACM Comput. Surv.* (2022). https://doi.org/10.1145/3539605