R-CAID: Embedding Root Cause Analysis within Provenance-based Intrusion Detection

Akul Goyal
Computer Science
University of Illinois at Urbana-Champaign
akulg2@illinois.edu

Gang Wang
Computer Science
University of Illinois at Urbana-Champaign
gangw@illinois.edu

Adam Bates
Computer Science
University of Illinois at Urbana-Champaign
batesa@illinois.edu

***Abstract*—In modern enterprise security, endpoint detection products fire an alert when process activity matches known attack behavior patterns. Human analysts then perform *Root Cause Analysis (RCA)* over event logs to determine if the alert is indicative of an actual attack. *Data Provenance* can help to automate RCA by representing event logs as a causal dependency graphs; in fact, researchers are now examining whether provenance-based anomaly detection should replace pattern-based detection altogether. Unfortunately, we observe that current approaches leverage off-the-shelf graph embedding techniques that are unable to associate events with their root causes. This shortcoming not only fails to capitalize on the RCA capabilities of provenance, but also leaves provenance-based IDS vulnerable to mimicry and evasion attacks.**

This work presents the design and implementation of R-CAID, a novel approach to incorporate RCA into provenance-based IDS. R-CAID precomputes each node's root causes during graph construction, then directly links those nodes to their root causes during embedding. Further, R-CAID's classification model is node/process-level, rather than graph/system-level, bringing it more in line with the precision of commercial systems. Under a passive adversary model, we find that R-CAID consistently outperforms baseline graph neural networks, sequence-based log IDS, and even a commercial endpoint detection system. Under a white-box active adversary model, R-CAID maintains a high level of performance (e.g., for DARPA Theia, 0.94 AUC adversarial down from 0.99 passive). R-CAID achieves this by associating each system entity with its immutable and unforgeable root causes, preventing adversaries from being able to masquerade as legitimate processes. This work is thus the first to demonstrate the promise of provenance-based IDS in a manner that avoids the pitfalls of mimicry and evasion.

***Index Terms*—Intrusion detection**

1. Introduction

Data provenance is changing how we reason about system intrusions. Most of today's endpoint detection products are pattern-based, raising an alert when a series of system

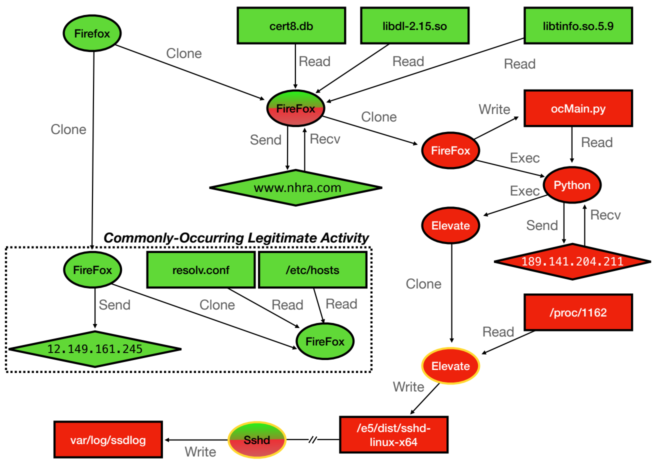


Figure 1: Provenance of the Drakon attack from a DARPA Transparent Computing engagement. Processes, files, and sockets are depicted as ovals, rectangles, and diamonds, respectively. Benign activity is green, malicious activity red, and green-to-red represents transfer to attacker control. The yellow line around process nodes refers to an process with elevated privileges.

events matches a pre-defined rule about a known attack behavior (e.g., MITRE ATT&CK techniques [1]). As such, systems are known to generate a high volume of false alarms (e.g., [2]–[4]), a human analyst must then examine related events to find out if the alert is cause for concern. While traditionally, this has entailed preparing a tedious and error-prone series of queries over event logs , data provenance has partially automated this process by iteratively parsing each event into a causal dependency graph (Fig. 1). As a result, analysts can perform *Root Cause Analysis* (RCA) of alerts with a simple backward graph traversal. Prior work has demonstrated numerous methods of precisely reconstructing an attack chain (e.g., [5]–[8]) using provenance.

Provenance's success in automating investigation tasks has led many to consider its use in detection tasks as well [9]–[15] However, *we observe that the transition of provenance analysis from human-in-the-loop investigation to procedural detection, has come at the cost of an enormous* *loss of graph context.* To make provenance graphs amenable to machine learning, current approaches encode the graph into a concise fixed-length representation using off-the-shelf commodity graph embedding techniques. These techniques describe a graph as the "sum of its parts," aggregating over many *highly localized* neighborhood traversals to generate an embedding causing less-localized graph relationships – the kind that forms the the basis for RCA – to be omitted. Perhaps unsurprisingly, this approach leads to rampant opportunity for attackers to evade detection – recent results by Goyal et al. [16] demonstrate that virtually all state-of-the-art provenance-based IDS are vulnerable to evasion, reporting evasion rates of 100% on community-standard evaluation datasets.

Given that commodity off-the-shelf embedding techniques are sacrificing security-relevant graph context, tailored solutions for provenance graph embedding are needed. In particular, current node embedding schemes lose the connection between suspicious activities and their root causes, simplifying evasion. An attack's root causes are particularly insightful for several reasons. First, an attacker's behaviors on a system often contradict our expectations about how descendants of a given root cause should behave. For example, the root causes of the attack in Figure 1) include Firefox and `www.nhra.com` – is it typical for visits to this website to execute Python scripts and write to SSH files? Second, an attacker's initial access point is immutable – while they may masquerade as legitimate processes after establishing a presence on the system, they cannot go back in time to modify their method of entry. Thus, we predict that preserving root-cause relationships will provide resilience against mimicry attacks.

To test this hypothesis, we present the design and implementation of R-CAID, the first Root-Cause Analysis Intrusion Detection system. The foundation of R-CAID is a traditional graph neural network (GNN) that performs process-level classification, bringing provenance-based IDS in line with the precision of a commercial endpoint detection models (e.g., [17]–[19]), R-CAID then introduces a novel and efficient method to incorporate RCA – rather than attempting to associate every node with its entire provenance history, which would be both ineffective and prohibitively costly, R-CAID precomputes every node's root causes during graph construction and then directly connects each node to its root causes during embedding. We also introduce an optimization to this procedure that skips over highly connected root causes (e.g., the GNU C Library) that increase storage and analysis costs without adding any discriminative power.

In our comprehensive evaluation of R-CAID, we first compare its performance to numerous baselines including a Vanilla GNN, several log sequence analyzers [20]–[23], and VMWare CarbonBlack's commercial enterprise Endpoint Detection and Response (EDR) [17]. We demonstrate that R-CAID can dramatically outperform all of these systems; its closest competitor, the Vanilla GNN, achieves 0.65 AUC for the DARPA Trace dataset as compared to R-CAID' 0.99 AUC. We then test R-CAID against an *extremely overpowered* adaptive adversary that performs a gradient descent-based evasion attack with full knowledge of R-CAID' configuration, training data, and test data. We find that R-CAID experiences less degradation during adversarial attacks than the Vanilla GNN and is still able to detect malicious processes in every attack. In the worst case against DARPA Trace, R-CAID maintains a 0% FPR while detecting 52% of the attacker's processes, while in the best case against DARPA Theia it maintains 100% TPR with just 10% FPR.

Our paper makes the following contributions:

- *Embedding Root Cause Analysis.* We demonstrate an effective and efficient method of incorporating the insights of RCA in the embedding space. Our root node embedding technique improves both detection and resilience to adversarial evasion.
- *Comprehensive Evaluation.* The rigor of our evaluation significantly raises the bar for Host Intrusion Detection research. Our baselines include both state-of-the-art graph- and sequence-based anomaly detectors, and we are also (to our knowledge) the first work in this space to directly compare against a commercial pattern-based EDR. Finally, our adversarial evaluation considers a significantly more powerful threat model than prior work.
- *Reproducible Research.* All code and data generated throughout this study can be found on BitBucket, including R-CAID itself, a replication of the ATLAS [5] dataset with EDR alerts and telemetry, and a precise ground truth labeling of the DARPA Transparent Computing datasets.

The rest of this paper is organized as follows. In Section 2, we motivate our approach and consider the limitations of prior work. Section 3 presents the threat model that guides our design. We present the design of R-CAID in Section 4 and discuss implementation considerations in Section 5.1. Section 5 presents our passive attacker experiments, adaptive attacker experiments, and performance results. We revisit related work in Section 6, consider limitations of our approach in Section 7, and conclude in Section 8.

2. Motivation

To motivate this work, we consider an example of a realistic APT attack behavior from DARPA Transparent Computing Attack Engagement 5, shown in Figure 1. Initially hijacking a previously trusted website (`nhra.com`), the attacker waits for a victim with a vulnerable Firefox browser to create a connection with the malicious server. Upon the victim visiting the webpage, the attacker downloads a Drakon exploit onto the target system and executes *loaderDrakon* within the Firefox process's memory. As a result, Firefox connects to an attacker-controlled server at IP address `189.141.204.211` and awaits attacker commands. Leveraging a previously installed driver, "BinFmt Elevate", the adversary elevates privileges on the Firefox process and gains root access. Finally, the attacker injects shellcode into the sshd using the Inject2 Process injection techniques, which results in a file - `sshdlog` - written to disk. From here, the attacker can connect to other malicious web servers to exfiltrate data. A system analyst conducting root cause analysis of this attack would identify `firefox` (top left), `www.nhra.com`, and `189.141.204.211` as attack-related root causes.

2.1. Prior Approaches

Using anomaly detection could differentiate many adversary activities from normal system behavior. However, Provenance-based IDS designers face numerous challenges when learning a proper representation of system activity. First, provenance graphs are *massive* data structures (e.g., gigabytes per day on a single machine [24]) that are immutable and append-only. The classifier must be able to learn a concise, *fixed-length* representation of this structure. Second, provenance is a feature-rich property graph that encodes various forms of system information. These properties, while rich with useful information, also make it challenging to produce a *generalizable* representation; even repeated executions of the same process may differ in process identifiers, the relative ordering of system calls due to non-determinism, differences in files accessed due to semi-random filenames, etc. Finally, because attacks represent a vanishingly small proportion of system activity, the representation must be extremely *sensitive* to small changes in the graph.

To address these challenges, Provenance-based IDS designers have borrowed embedding techniques introduced by the machine learning community. The goal of embedding algorithms is to transform an input into a real-valued vector such that similar inputs are closer to one another in the vector space. When embedding more complex data structures (e.g., a text document [25]), prior work first embeds all substructures (e.g., words) and then aggregates these embeddings to summarize the complex structure. In the case of graph learning, such as in Graph Neural Networks (GNNs), all subgraphs are embedded and then aggregated to represent the entire graph [26]. Returning to our example in Figure 1, a GNN could model this system by embedding a 1-hop traversal of each node's local neighborhood, and then aggregating. Training on the green subgraphs and testing against the entire graph would lead to significantly different representations in the embedding space. Such results prove that describing complex data structures by the "sum of their parts is safe and effective."

2.2. The Curse of Locality

Unfortunately, the assumptions made in order to produce *fixed-length*, *generalizable*, and *sensitive* representations for anomaly detection are often confounded by an adaptive adversary. Dating back to the earliest host anomaly detection systems, it was quickly discovered that a "sum of parts" approach to system modeling created opportunities for attackers to evade detection. Against the Forrest IDS [27], which modeled processes according to a fixed N-length slide window of their system calls, Wagner and Soto, demonstrated that an adversary could produce useful attack patterns that did not appear during training but in fact appear benign when broken down into N length chunks [28].

Unfortunately, even though modern provenance-based IDS benefit from revolutionary advancements in deep learning, they still fall prey to the same fundamental vulnerabilities. Returning to our example, the adversary could adapt Wager and Soto's mimicry attack to provenance-based IDS by having its processes (e.g., red `Firefox`) access files commonly accessed by the legitimate application (e.g., `cert8.db`, `libdl-2.14.so`). By changing the embedding of the attacker-controlled process' local neighborhood, the attacker could force misclassification of the aggregated graph embedding. Alternatively, the attacker could directly attack the aggregation stage repeating the *Commonly-Occurring Legitimate Activity* subgraph, causing the abnormal behaviors to account for a smaller proportion of the aggregate embedding. Goyal et al. [16] recently demonstrated such attacks were successfully able to evade state-of-the-art systems, including Unicorn [12], ProvDetector [29], and SIGL [10].

We observe that the fundamental problem enabling such attacks could be seen as a *curse of locality*. The curse of locality refers to the loss of context when decomposing a provenance graph into a set of overlapping local neighborhoods. For example, consider a provenance-based IDS that describes a given node v by traversing its K-hop neighborhood. The relationship between v and all nodes $K + 1$ or more hops away is lost; even though those nodes are eventually visited, their relationship to v is not part of the final embedding. Notably, this also means that *root cause analysis* capabilities are also absent in the final graph embedding. Thus, the curse of locality dramatically increases attacker degrees of freedom when attempting to evade detection.

In this work, our key insight is that the most severe consequence of the curse of locality is the disassociation between a local event and its root cause(s) within the system. Returning again to Figure 1, one of the most obvious signs of compromise in the attack graph is that `sshd` counts `firefox` among root causes. Unfortunately, this dependency would not not appear in `sshd`'s embedding using traditional approaches. If the IDS retained this association, it would dramatically restrict the the attacker's options for remaining undetected – they could only engage in behaviors that the IDS associates with `firefox`, preventing them from being able to achieve their objectives covertly.

3. Threat Model

This work considers an "Advanced Persistent Threat" (APT) adversary attempting to gain covert access to a victim endpoint system. We broadly define the adversary's capabilities based on the MITRE ATT&CK knowledge base of attacker tactics, techniques, and procedures [1]. To avoid detection, the attacker may employ Defense Evasion tactics. These include techniques such as "Exploitation for Defense Evasion," exploiting vulnerabilities within existing tools to bypass security controls, and "Masquerading," manipulating features of attacker processes and artifacts to make them appear legitimate.

We make the following assumptions about the target system. Similar to most work in the space (e.g., [30]–[35]), the OS, auditing framework, and EDR (i.e., the provenance-based IDS) compromise the Trusted Computing Base (TCB) of the system. We assume these components function correctly and cannot be manipulated by the attacker. We also assume that the attacker cannot directly change the contents of audit logs, which is possible via tamper-evident logging techniques [36]–[39] that are outside of the scope of this work. We do not consider attacks that employ implicit flows (side-channel attacks) that the audit system cannot capture.

That said, we place no further restrictions on the runtime behaviors of the adversary on the victim machine; in other words, we conservatively assume that attacker-controlled and -affected processes may take any action on the system to confuse the EDR and evade detection.

4. R-CAID Design

We now present the design for R-CAID. First, we describe the foundation of R-CAID applying GNNs to provenance graphs in §4.1. Next, we extend GNNs to incorporate RCA capabilities through a novel "Root Node Embedding" technique, explained in §4.2. Finally, to adapt GNNs to anomaly detection, we explain our training procedure in §4.3 and our testing procedure in §4.4. Finally, in §4.5, we present an optimization for R-CAID that exposes a performance/security trade-off continuum for root node embedding.

4.1. Graph Neural Network

First, we describe a traditional Graph Neural Network (GNN) and how to apply it to provenance graphs.

Let us define a provenance graph G by a set of vertices $V = \{v_j\}_{j=1}^{|V|}$ and edges $E = \{e_j\}_{j=1}^{|E|}$. Each vertex in V maps to a system entity, such as a file, process, or network socket, and each edge $e = (v_i, v_j, rel)$ connects two vertices by the system call event *rel* such that for any two given edges $(v_i, v_j, rel_k), (v_j, v_i, rel_l) : rel_k \neq rel_l$. There exists a path P between any two nodes $P(v_s, v_d)$ if there occurs a set of edges such that $v_s(rel_{s-i})v_i, \cdots, v_j(rel_{j-t})v_d$ where each edge $v_{src}(rel)v_{dst}$ can be found within E. A node v_i's K-hop neighborhood N_i^K is the set of *ancestors* reachable from v_i within K hops; in other words $\forall v_k \in N_i^K : \exists P(v_k, v_i) \in E$. The goal for any GNN is to create a D-dimensional vector that accurately describes each node v_i in G in terms of v_i and its K-hop neighborhood N_i^K. One possible way GNNs achieve this is by using a technique called message passing, where information flows according to the direction of the edge.

To start, let $\{h\} = \{h_1, \cdots, h_{|V|}\}, h_i \in \mathcal{R}^D$ be a set of feature vectors corresponding to each node in G where D is the number of features. A GNN takes in as input $G, \{h\}$ where h_i initializes each node v_i feature vector x_i^0 for the first layer of the GNN. For each subsequent layer l, a node v_i's new representation x_i^l is an update between its intermediate representation x_i^{l-1} and an aggregation of all the intermediate representations of nodes in N_i^K such that $x_i^l = U(x_i^{l-1}, A(N_i))$. During model training, aggregation A and update U functions are optimized to represent the node accurately. An example of message passing on a provenance graph is visualized in Figure 2. Here, we see the GNN updating process P_2's embedding to h_7' by first aggregating P_2's parent's intermediate representations from the previous layer (h_4, h_6) and then concatenating that with P_2's intermediate representation of the prior layer (h_7).

A GNN comprised K layers where each layer l in the network aggregates features from nodes l hops away. [1] The final layer of the GNN outputs an embedding matrix $\mathcal{E}^*$ representing every node in G such that $\mathcal{E}^* = \mathcal{R}^{(|V| \times L)}$ where L is the size of an individual embedding. [2] $\mathcal{E}^*$ can be used in downstream tasks like node classification or outlier detection.

4.2. Root Node Embedding

Given the design of a baseline GNN, we now consider mitigation to the curse of locality. A naïve approach to solving this problem would be to increase the hop value K to the point where $K = \infty$. While this would superficially prevent the loss of dependency context, it would be costly and result in overfitting and unusable representation; previous work shows that GNNs with more than 4 layers (i.e., hops) face a vanishing gradient problem [40], [41]. Instead, we propose to *directly link* each node to its root causes, regardless of the root cause's distance from the node in the provenance graph.

Classically, a node $v_j \in V$ is a root node if there exists no $v_i \in V$ for which $P(v_i, v_j)$ - that is, there are no inbound edges associated with v_j. As such, a GNN should never update a root node's embedding under message passing, as no information flows into the node. In a true provenance graph, which is acyclic [42], it is straightforward to maintain this definition of root nodes. However, when applying provenance to operating systems in practice, cyclic behaviors between entities (e.g., socket I/O) are allowed, creating the risk of no root nodes existing within the final graph. One method to break cycles is to *version* the graph [43], [44], wherein every system entity is associated with one-to-many nodes, and a new version is created each time information flows into the entity. While this prevents cycles, it also causes the graph to explode exponentially in size and is thus more costly to analyze. Alternatively, most provenance work in the security community tolerates cycles and resolves the issue by attributing timestamp values to edges (see Figure 3). Given such a graph, we can redefine

1. Note that we purposely reuse the term K; by design, the depth of the neighborhood traversal, K hops, is equal to the number of layers in the neural network.

2. D and L do not need to be equal. The architecture of the GNN defines these sizes.

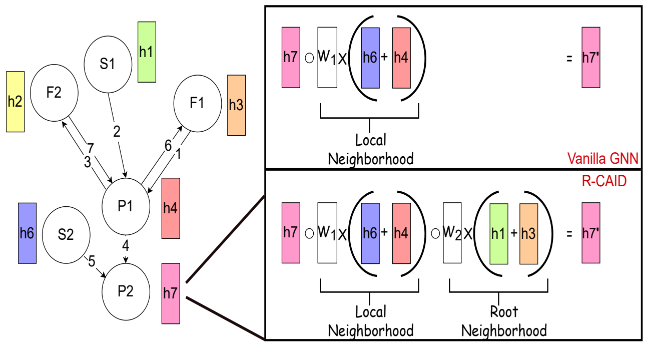


Figure 2: A colored coded visualization of the node embedding procedure. Circles represent provenance nodes, adjacent rectangles denote the feature vector for each node, with edges marking provenance relationships. P-nodes are processes, S-nodes are sockets and F-nodes are files. Shown here is the 1-hop neighborhood for process P_{ii}. In a vanilla GNN, Process 2 is updated by an weighted summation of feature vectors (h4, h6) and an aggregation with (h7): Process 2 feature vector. In R-CAID, Process 2 is additionally updated with a weighted summation of its root cause nodes' feature vectors (h1, h3).

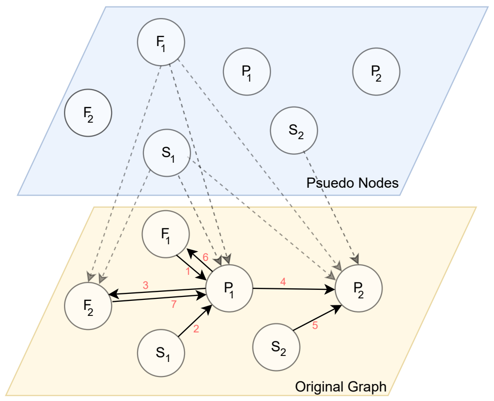


Figure 3: A visualization of the pseudo root node concept. In the figure, node F2 is not a pseudo-root as it had an incoming edge before an outgoing edge. Conversely, F1 is a pseudo-root as its outgoing edge occurred before its incoming edge.

a root causes to be a node whose first event timestamp is on an outbound edge. [3]

Prior work has introduced GNNs that support logical timestamps [45], but these models carry significant computation and memory costs and are specific to a single architecture. Instead, we provide a solution that avoids these pitfalls without requiring changes to the GNN's structure. pseudo-graph $\mathcal{G}$ is an overlay graph on top of G that retains the initial embeddings of all root nodes in G.

We concretely detail the construction of $\mathcal{G}$ as follows: let V_f be the set of nodes representing files or sockets in G. For any node $v_k \in V_f$, if its first action within G is an outgoing flow $e_k = (v_k, v_i, rel)$, then we identify v_k as a root cause. For example, in Figure 3, node F_1 first timestamped edge is outgoing in G and is therefore considered a root node. In contrast, node F_2's first timestamp edge is incoming and, as such, not a root node.

For each root v_k, we first create a new node p_k^r known as v_k's pseudo-root and then add p_k^r to the pseudo-graph $\mathcal{G}$ assigning it the same initial feature vector h_k associated with v_k. After identifying a root cause, we keep track of its descendants defined as any node v_j such that a path $P(v_k, v_j)$ within G exists. For each descendant node v_j, we add an outgoing edge from p_k^r in $\mathcal{G}$ directly connecting p_k^r and v_j. Again, in Figure 3, within $\mathcal{G}$ F_1's pseudo-root connects to all of F_1's descendants nodes $\{P_1, P_2, F_2\}$. Note, under this construction, it is impossible for an outbound edge from G to $\mathcal{G}$ to occur. In the end, information can only flow out of $\mathcal{G}$ during message passing, but not into $\mathcal{G}$, thus assuring that pseudo-root node embeddings are not updated.

Given $\mathcal{G}$ as additional input, we can update the embedding function defined in Section § 4.1, $x_i^K = h_K(U(x_i^{K-1}, A(N_i)))$, as follows: $x_i^K = h_K(U(x_i, C(A(N_{v_i}), A(\mathcal{R}_{v_i}))))$. Here C represents a concatenation function that joins the aggregated representation of all nodes within N_{v_i} and the aggregated representation over all pseudo-nodes in $\mathcal{G}$ that directly connect into v_i represented by $\mathcal{R}_{v_i}$. Figure 2 demonstrates how this new definition affects the embedding of P_2 on the lower right side of the image.

Root-Paths: It is possible for us to generalize and extend root cause embedding to provide ancestral paths of arbitrary depth (L-hop traversals). Given a root node v_i, let N_i^L be v_i's' L-hop neighborhood such that for every node v_j in N_i^L, there exists a path $P(v_i, v_j)$ where the length of $P \leq L$. To extend to root cause neighborhoods, pseudo-graph $\mathcal{G}$ would be modified to host pseudo-root paths $\mathcal{P}$ where each node p_i in $\mathcal{P}$ is a pseudo-node. Like pseudo-root nodes mirroring root nodes within the provenance graph G, $\mathcal{P}$ would emulate each path of length L in the given root's L-hop neighborhood N_i^L. For a given node v_i in a provenance graph G, its embedding utilizing a GNN would be, $\mathrm{e}_i = h_K(U(x_i, C(A(N_{v_i}), A(\mathcal{P}_i))))$ where $\mathcal{P}_i$ are all the pseudo-root paths that indirectly connect with v_i, i.e., there exists a path from the last node in P to v_i in the provenance graph. By enlarging the size of the overlay pseudo-graph $\mathcal{G}$ to include pseudo-root paths $\mathcal{P}$, more information from the roots can be pulled down to each child node's local neighborhood allowing for a more informed embedding. There is a balance in choosing how much of the root neighborhood to embed - larger L values propagate more information and increase the noise at each node's neighborhood. Our evaluation considers paths (L=0, L=1) or pseudo-roots and pseudo-root edges.

4.3. Training Procedure

Traditionally, GNNs are trained in a supervised setting utilizing each node's well-defined label to optimize its update U and A aggregation functions. Because a node's label

3. We define the root node whose first event timestamp is an outbound edge because the attacker must take some action to initiate an attack chain.

signals similarity, a GNN learns to embed nodes such that nodes with similar labels are close together. However, in the most conservative and appropriate design for anomaly-based IDS, a single class is assumed for all nodes in the training dataset. This way, a model is only trained on benign activity, preventing it from placing any assumptions or restrictions on the attacker's behaviors. In a one-class learning problem, class labels are only used at test time to determine if a sample is a member of the class (benign) or not (malicious). Unfortunately, in the presence of only a single label during training, a GNN cannot mispredict. As a result, the GNN cannot optimize its U, A functions or learn a suitable similarity measure.

One common approach to adapting GNNs to a one-class setting utilizes self-supervised learning, which uses the supervisory signals from the data to train the model. To distinguish between different training samples, we utilize process tags, where we annotate each process node by its function - automatically generated by using the associated file path to its binary executable and extracting the process name. This strategy quickly allows us to group like-for-like process nodes regardless of differences in the command line, environment, or symbolic links. As a result, **during training** R-CAID is optimized to embed processes with similar binary names close together. It is important to note that while processes are not the only node type within provenance graphs, we focus on identifying malicious programs because this is typically the focus of EDR systems. However, since files and sockets also play an essential role in attacks, it could be beneficial to tag and test non-process nodes by following a similar procedure. In Section 5, we evaluate how including non-process tags impacts the model's overall performance.

Using self-supervised learning allows us to approximate the one-class process anomaly detection model and maintain the integrity of R-CAID' training routine as "benign data only." In other words, R-CAID **does not train on attack data.** Moreover, the process tag is only used for improving the training of R-CAID- R-CAID does not need the process tag for testing/detection. Instead, we use the output for R-CAID final layer to embed processes and identify outliers.

Node Features: In § 4.1, for each node v_i, we defined a corresponding feature vector h_i that GNN took in as input to initialize the intermediate node embedding x_i^0 for the first layer. Given the diversity of audit logs and telemetry data, many options for encoding system entities' features are available. To preserve the generality of our approach, we focus on encoding the system identifiers – filenames and paths for files and IP addresses for sockets – common to almost all audit log formats. We consider two methods for encoding this information, Doc2Vec [25] and one hot embedding. Given a group of sentences, Doc2Vec learns a metric that groups similar sentences. Each file path in the training dataset is considered a single sentence and gets passed into the Doc2Vec model. The Doc2Vec model then provides an embedding associated with each file path as an initial node's features. One hot embedding provides a single numerical mapping for each component in the file path. For example, given the two file paths: /usr/bin/python and /home/bin/firefox, the resulting embedding would be [1,1,1] and [2,1,2] with a map {1:{usr: 1, home:2}, 2:{bin:1}, 3:{python:1, firefox:2}}. For both approaches, when embedding the node features of a process, we truncate the process name from the path so that the process tags described above do not appear in the node features. We compare the performance between these two approaches in our evaluation.

4.4. Testing Procedure

R-CAID generates an anomaly score for each process node in the test set through three steps. First, using a fully trained GNN, each process node in the training and test set is embedded into a D-dimensional vector. Next, the IDS clusters all the node embeddings in the training dataset using K-Means. To identify the optimal K, we utilize the elbow method [46], which iteratively samples through a range of possible K and finds the one that provides the least inter- and intra-cluster variation. Finally, for the last part of our testing procedure, each node in the test set is assigned an anomaly score by calculating the Median Absolute Deviation (MAD). For every node in the test set, MAD generates the Euclidean distance between the nodes and the median of its nearest cluster. Then any node whose Euclidean distance has more than a specific variation, given all the Euclidean distances of nodes in the test set, is considered abnormal. In all our experiments, we provide results that show the IDS's performance under different variation thresholds.

4.5. Optimization: Prune Highly-Connected Roots

To implement root cause analysis, R-CAID creates an overlay graph where an edge connects each root node to all its descendants to allow for more information to be present during node embedding. A side effect of this is that the size of the overlay graph exponentially increases with the size of the provenance graph. For smaller graphs, R-CAID can retain all edges between pseudo-roots and their children in memory. However, effectively scale to larger graphs, R-CAID must reduce the number of root causes recruited into the pseudo-graph. In our implementation, R-CAID utilizes a graph attentional operator (GAT) to weigh each node's roots based on their importance to the node's classification. Root nodes connected to a large portion of the graph (diverse in many different processes) provide limited information as they do not offer a strong signal to a given node's class (process tag). For this reason, we prune roots from the pseudo-graph connected to more than a prune threshold percentage of the provenance graph. Pruned roots, by definition, connect to a massive number of nodes within the provenance graph, making it unlikely that an attacker could abuse this optimization to force suspicious root causes out of the model.

5. Evaluation

5.1. Implementation

We developed R-CAID using 1956 lines of code in Python. While are many different out-of-the-box implementations available for building GNN (e.g., Node2Vec [47], GraphSage [48], SGC [49]); in this work, we created a simple 3-layer GAT [50] using Pytorch Geometric [51] as the building framework. For the update U and aggregation A function, we used Attention and Multi-Layer Perceptron (MLP). In the appendix, we provide more details about these layers and our reasons for using them (). We trained our model with AMD Threadripper Pro 3995WX CPUs @ 2.70 GHz, 1024GB of physical memory, and NVIDIA RTX A6000 GPU. The model was optimized using the Adam optimizer [52] with a batch size of 64. We randomly initialized the model's parameters and set the maximum number of training epochs to 1000. However, because we employed early stopping, training never reached 1000 epochs - our early stopping condition stopped training if the validation loss did not decrease after five consecutive epochs. We used the model weights associated with the lowest validation loss for testing. We also used a dropout [53] with a ratio of 0.5 to prevent overfitting. We employed a hyper-parameter search over the learning rate ($\{0.001, 0.0001, 0.00001\}$) and the size of the embedding vectors ($\{32, 64, 128\}$). The parameters that provided the best performance were a learning rate of 0.0001 and an embedding vector of size 128.

We also implement provenance graph parsers for each log format described in §5.2. Our parser supports all processes, files, and network activity in these logs. When translating the resulting provenance graph into an adjacency matrix for the GNN, note that our GNN implementation does not support edge attributes such as the system call relationship. Fortunately, this information is implicitly present in the adjacency matrix as nodes' types indicate the relationship, e.g., a provenance edge from a process to a file denotes a read event.

5.2. Datasets and Ground Truth Labeling

We evaluate R-CAID using four different datasets.
Streamspot is an open-source dataset [54] that uses SystemTap [55] to record five benign browsing behaviors (five benign workloads (video game, YouTube, file downloading, CNN, email) and one malicious behavior (drive-by download). The attack details a Command and Control (C&C) scenario where the adversary exploits a Firefox vulnerability after the victim clicks on an attacker-controlled URL. Once on the system, the attacker exploits a Flash vulnerability to gain root access. Streamspot repeats each behavior 100 times, resulting in 600 graphs. On average, each graph in StreamSpot has a total of $27,792,491$ edges and $822,998$ nodes.
DARPA Transparent Computer (TC) Engagement 3 describes system activity from multiple hosts over 2 weeks, during which a professional penetration team utilizes sophisticated methods to launch attacks. We use the two Linux platforms in DARPA TC 3, the Theia and the Trace datasets, and evaluate against the 2 web-based attacks in Theia and the 3 web-based attacks in Trace. A complete description of these attacks is available online [56]. Theia contains $46,303,154$ edges and $3,721,210$ nodes, while Trace contains $771,554,076$ edges and $169,763,844$ nodes.
ATLAS (v2): To evaluate the ATLAS threat investigation system [5], Alsaheel et al. prepared ten attack chains (4 single hosts, 6 multi-host) and subsequently released both the data and attack scripts. We replicated this attack engagement on two Windows VMs instrumented with the Carbon Black EDR. Two students used the VMs as their primary workstations for approximately 8 hours a day over a five-day week. On the final day, the students took turns periodically launching each of the ten attack chains while continuing to use the VMs for other work. We then trained R-CAID using the four days of benign Microsoft Windows Security Auditing logs.

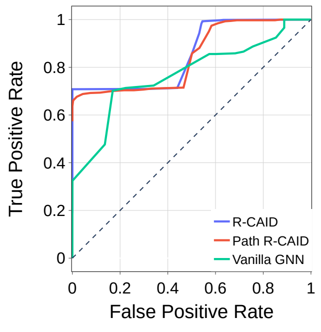
5.3. Labeling Methodology

While these datasets represent the community standard for IDS evaluations, unfortunately, there is no standardized ground truth labeling methodology. Streamspot labels at the granularity of whole graphs/logs, while ATLAS labels only a small percentage of entities present in the attack chain, and the DARPA TC datasets provide no labels at all. This lack of standardization of ground truth labels creates a risk for a researcher to bias evaluation results.

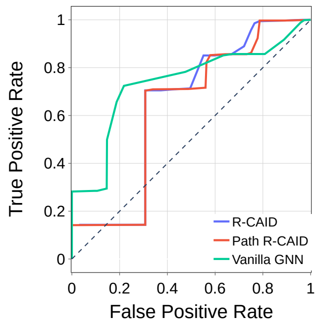
We propose the following labeling methodology to improve the transparency and reproducibility of results. For each dataset, we identified the initial access points of each attack chain using a combination of available documentation, provenance graph visualization, and speaking with the dataset authors. We ensured the correctness of the access points by matching unique identifiers (file names, time, etc.) between ground truth documentation and the parsed audit logs. We then performed a temporal depth-first search from the initial access points to identify all nodes in the provenance graph that contained attack dependencies. We labeled all nodes in this graph as malicious (for node-level classification), then all edges linked to malicious nodes were labeled as malicious (for event-level classification). This strategy *overapproximates* the true attack graph but guarantees that all attack entities are labeled malicious. It also reduces researcher degrees of freedom by minimizing qualitative judgments in the labeling process. Nonetheless, three senior graduate students in our group repeated this procedure independently and then met to confirm the results and resolve any inconsistencies. Our labels can be found on BitBucket.

5.4. Hyper-parameter Search

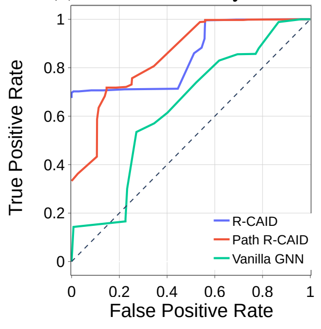
We utilized the StreamSpot dataset to perform a hyperparameter tuning on critical parameters in R-CAID. We



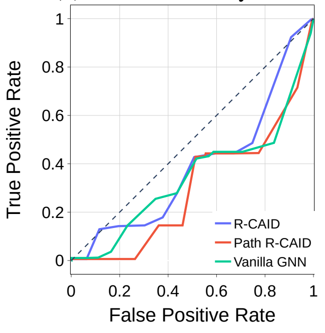
(a) Doc2Vec, 2 Layer



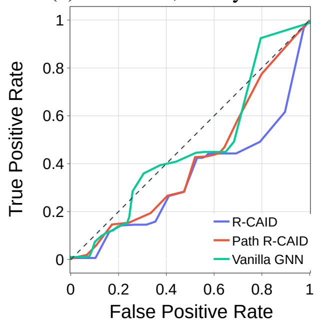
(b) Doc2Vec, 3 Layer



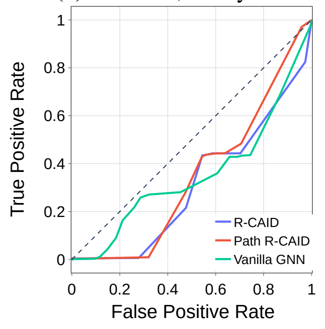
(c) Doc2Vec, 4 Layer



(d) One Hot, 2 Layer



(e) One Hot, 3 Layer



(f) One Hot, 4 Layer

Figure 4: Hyper-Parameter tuning of R-CAID on the StreamSpot dataset.

chose Streamspot for this experiment because its smaller size made it amenable to repeated testing. Figure 4 visualizes the results from our experiments. Each subfigure depicts a ROC curve that compares the model's the model's True Positive Rate (TPR) to False Positive Rate (FPR) under different MAD score thresholds. Here, we manipulate three hyper-parameters: 1) the node feature embedding method (Doc2Vec, One Hot); 2) the number of layers in the GNN, which is also the hop size of neighborhoods in message passing; and 3) whether the GNN embeds the root node (R-CAID), the root path (Path R-CAID), or no root (Vanilla GNN).

Comparing Doc2Vec (Fig. 4a-4c) to One Hot (4d-4f), Doc2Vec is the better method for encoding node features. Among the Doc2Vec ROC curves, an interesting trend occurs in which R-CAID beats the Vanilla GNN with two or four layers but slightly underperforms it with three layers.

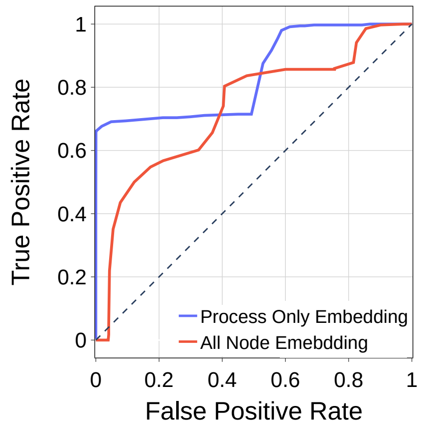


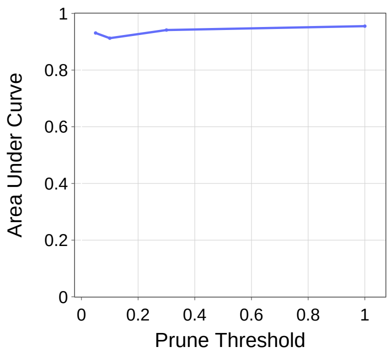
Figure 5: Classification performance of R-CAID on the Streamspot dataset when classifying only process nodes as compared to all nodes.

We attribute this dip to the structure of a provenance graph – because I/O dominates process activity, process nodes are usually an even number of hops away from other processes (e.g., `process A writes file1, file1 read by process B`). Because we trained GNN for process identification, the odd-numbered hop typically encoded additional I/O information that added more noise than signal. While offering similar performance, R-CAID performance is slightly better with two layers and is much more efficient to train. Finally, we select the baseline root node embedding function as the root node, and root path embedding performs similarly in Doc2Vec, 2 Layer (Figure 4a). Thus, for the remainder of these experiments, we use use Doc2Vec for node features, a two-layer GNN, and root node embedding.

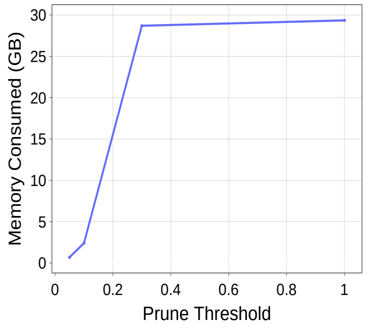
Wait, are these results any good?: A skeptical reader may, at this point, express some doubt about the performance of R-CAID – our chosen parameterization observed a 73% TPR for the Streamspot dataset. To interpret this result, recall that our classification task attempts to detect *individual malicious processes*, not attacks. The 73% TPR reflects that on average R-CAID identified 8 out of the 12 malicious processes in the attack graphs without flagging a single benign entity. It is difficult to compare this result to other provenance-based IDS because past systems have only performed whole-graph anomaly detection, which is far more coarse-grained. However, we will soon demonstrate that R-CAID dramatically outperforms other fine-grained detection methodologies, such as log sequence analyzers and commercial pattern-based detectors.

5.4.1. Process vs. All Node Classification.

We now briefly evaluate our design decision to only classify process nodes instead of all nodes. We first train R-CAID on just processes tagged/embedded, then again with all nodes tagged/embedded. The results are shown in Figure 5. Process tagging identifies 73% of attack processes without any false positives. When classifying all nodes, reaching the same TPR concedes a 37% False Positive Rate. This result



(a) AUC Performance



(b) Memory Consumption

Figure 6: Performance and Memory comparison of pruning highly-connected roots using the ATLASv2 dataset.

indicates that classifying non-process nodes is not helpful and is likely detrimental. GNN still represents non-process nodes in the final model without being directly embedded because they appear in the local neighborhood traversals and the pseudo-node layer.

5.4.2. Pruning Highly-Connected Nodes. Another parameter in our design is the pruning optimization, which removes highly-connected root nodes from the pseudo-graph. Recall that we predicted that highly-connected nodes would likely contain very little practical information for anomaly detection. To test this, we evaluated the classification performance and storage costs of R-CAID under various root pruning thresholds on the ATLASv2 dataset. Figure 6 visualizes the results. *Smaller* prune thresholds indicate that *more* roots are pruned for the pseudo-graph; e.g., at threshold 0.2, any root connected to 20% or more of the provenance graph is pruned. The difference in AUC from threshold 1.0 to threshold 0.05 was just 2.53%, but led to a 97.6% decrease in memory consumption. As predicted, pruning highly-connected roots has a vanishingly small impact on AUC while potentially significantly reducing the memory consumption of R-CAID. Nonetheless, for the remainder of our evaluation, we use the most conservative prune threshold permitted by the size of the dataset. On our machine, this meant that the pruning threshold was 1.0 for Streamspot and ATLASv2 (no pruning, and 0.5 for the DARPA TC datasets.

	LSTM		TF		AE		R-CAID	
	FPR	TPR	FPR	TPR	FPR	TPR	FPR	TPR
Streamspot	0.24	1.0	0.25	1.0	0.22	1.0	0.03	0.68
Theia	0.22	1.0	0.34	1.0	0.31	1.0	0.002	1.0
Trace	0.34	1.0	0.37	1.0	0.37	1.0	0.02	1.0

TABLE 1: Comparison of R-CAID performance to state-of-the-art log sequence analyzers: LSTM [20], [21], TF (Transformers) [22], and AE (AutoEncoders) [23])

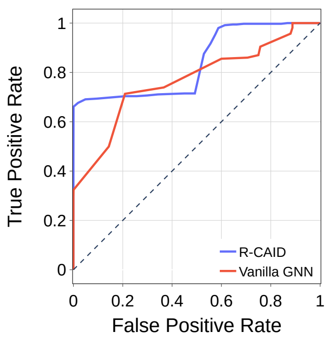
5.5. Comparison to Related Work

Past provenance-based IDS that have appeared in the literature, such as StreamSpot [57], ProvDetector [14], SIGL [10], are *whole-graph* classifiers that are not directly comparable to R-CAID. Instead, we continue our experiments by comparing R-CAID to baselines that can provide similarly fine-grained classification decisions: the traditional GNN, a battery of state-of-the-art log sequence analyzers, and a commercial pattern-based EDR.

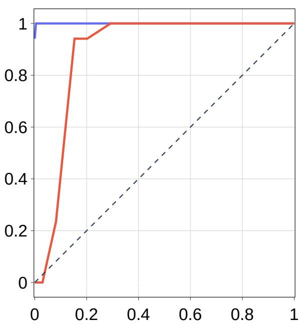
5.5.1. Vanilla GNN. Recall that R-CAID is an extension of a traditional GNN presented in §4.1, hereafter referred to as "Vanilla GNN." We compare the performance of the two GNNs on Streamspot, Theia, and Trace. Figure 7 summarizes our results. R-CAID consistently outperforms the Vanilla GNN, detecting many more attack processes before admitting false positives. For Theia and StreamSpot, R-CAID spotted 94% and 73% of the attack before flagging any benign behavior. Similarly, for Trace, R-CAID flagged 83.3% of the attack while marking just 0.2% of normal behavior.

Interestingly, both R-CAID and the GNN's performance on the DARPA TC 3 datasets is markedly improved compared to the Streamspot dataset – the AUC for R-CAID and Vanilla GNN were 0.999 and 0.897, respectively, for Theia, 0.990 and 0.645 for Trace, but drop to 0.619 and 0.469 for Streamspot. We attribute this to the structural differences between these datasets. The Streamspot dataset exclusively describes browsing behaviors, and with our conservative labeling methodology, it was challenging to detect the more innocuous Firefox processes in the attack chain. Further, Streamspot is a smaller dataset with fewer positive samples (12 malicious processes per graph), causes a more significant drop in TPR when a single misclassification occurs.

5.5.2. Log Sequence Analyzers. Although not graph-based, log sequence analyzers are similar to R-CAID making extremely fine-grained determinations of abnormal behavior. Each sequence-based IDS assumes an inherent structure within the input logs. Given a sliding log event window of size n, the IDS classifies the nth event based on the previous $n-1$ events. The IDS outputs a binary classification for each window, indicating whether the nth event is anomalous. Deeplog [20], LogAnomaly [21], Logsy [22], and AE [23] are all deep learning-based IDS that utilize sequential system logs to identify attack behavior. Deeplog and LogAnomaly use an LSTM backbone to model system activity, while Logsy uses a Transformer-based alternative. AE uses an autoencoder to determine anomalous behavior.



(a) StreamSpot


(b) Theia

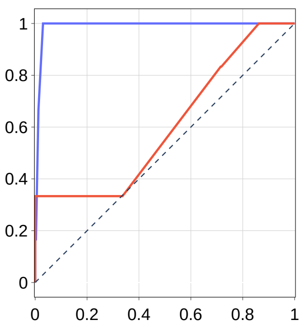

(c) Trace

Figure 7: Performance comparison of R-CAID as compared to a Vanilla GNN.

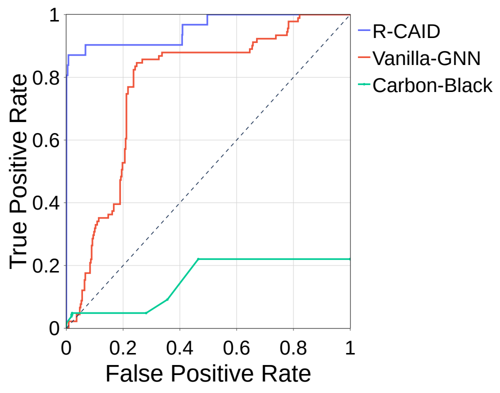


Figure 8: Performance comparison of R-CAID as compared to a Vanilla GNN and the VMWare CarbonBlack EDR on the ATLASv2 dataset.

We use the implementations provided by [58] to evaluate each of these systems using a fixed window size of 10. Chen et al.'s framework does not output a continuous anomaly score but makes a binary prediction over each log sequence (benign, malicious). Thus, we report a single TPR and FPR rate for each system rather than ROC curves. To simplify the comparison to R-CAID, we report comparable TPR and FPR for each dataset from Figure 7.

Our results are reported in Table 1. All of the sequence-based IDS perform admirably at detecting the event windows containing attack steps; however, they do so by admitting far more false positives. Because benign data dominate these datasets, false alarms would dominate alert streams, leading to threat alert fatigue [2]. In contrast, R-CAID can achieve a high TPR with near-zero FPR. Because an analyst can reconstruct the entire attack from log data given reliably true alerts, in many ways, low FPR is more critical to the functional performance of these systems.

5.5.3. Commercial EDR. Commercial EDR products perform pattern-based intrusion detection [19], as opposed to anomaly-based. Analysts define rules that encode known attacker behaviors matched against a telemetry stream of system events. While rule-based detection offers many legitimate advantages, such as providing built-in explanations as to why every alert occurred (e.g., MITRE ATT&CK annotations [1]), conventional wisdom also claims that rule-based products offer better precision (i.e., fewer false alarms) than anomaly-based solutions. To test if this is universally the case, we compare the performance of R-CAID to the VMWare CarbonBlack's EDR As EDRs do not issue per-alert scores, we use the severity score of each rule to generate a ROC curve; the first point on the curve captures the TPR/FPR for alerts of severity 10, then for alerts of severity 9 or greater, etc.

Our results for the ATLASv2 dataset are shown in Figure 8. R-CAID identifies 74.19% of the attack processes before encountering false positives. While detecting all attack processes with FPR 49.67%. In contrast, at best, the commercial EDR identifies 22.04% TPR, which it achieves at an FPR of approximately 45%.[4] The EDR can detect at least one process in each of the ten attack chains at $Severity \geq 3$, which incurs a 35.6% FPR.

While this is a valuable point of comparison, *we do not claim to have designed a better threat detection system than a commercial vendor*. Pattern-based detection is significantly performance efficient, and at present, our GNN-based approach could not scale to the millions of endpoints that need to be supported by commercial products. Commercial solutions are also multi-modal, leveraging complementary techniques such as antivirus to detect a broader range of attacks, which our experiments do not reflect. The quality of public test data may also bias our results; the ATLASv2 dataset captures just a tiny percentage of the threats against which a commercial product must defend against, Bearing in mind these concessions, we are encouraged by the accumulating evidence of the efficacy of provenance-based approaches such as R-CAID.

5.6. Performance of R-CAID

We briefly describe the computational and memory overhead of R-CAID. In Table 2, we see that as any individual

4. Note that the EDR does not reach an FPR of 100%, this is an artifact of the visualization.

	% PT	N(K)	E(M)	RE(M)	s/e	TT
StreamSpot	100	853	0.825	5	2.83	26:01
ATLAS	100	454	1.08	834	3.37	110:02
THEIA	50	613	21.5	803	3.30	105:75
TRACE	50	1039	40.7	873	3.44	128:28

TABLE 2: Performance of R-CAID on different datasets; PT refers to the pruning threshold, N(K) refers to the number of nodes within the graph in the thousands. E(M) refers to the number of edges in the millions, and RE(M) stands for the number of root edges within the graph in the millions. s/e is the seconds R-CAID took to run through one epoch. TT is the total time it took for R-CAID to run.

graph gets bigger, the overall memory overhead of R-CAID also grows. Note, although StreamSpot has a larger number of nodes than ATLAS, it has far fewer root edges as it does not contain a single graph but is instead of a collection of 600 different provenance graphs. The size of the graph impacts the amount of time each epoch of R-CAID takes to run, as there is a more significant number of edges for R-CAID to process. However, the difference in time between the different datasets is not extensive because we use early stopping to dictate when to stop training.

5.7. Adversarial Attack Results

Given the recent discovery that provenance-based IDS are equally vulnerable to mimicry attacks as prior approaches [16], it is necessary to consider the resilience of R-CAID to active evasion. We adopt an extreme adversarial model in which the attacker is effectively unbounded in knowledge and ability. In addition to understanding the underlying model and the training dataset, we assume the attacker can access the testing dataset. In other words, the attacker knows the exact attacks used to evaluate the model. Furthermore, we assume the underlying system places no constraints on the adversary – that is, any modification to the graph embedding in the feature space is assumed to be a valid provenance graph in the problem space. Presuming such an adversary enables us to approximate a lower bound on the performance of our model. It also means we can more easily adopt existing adversarial attacks that otherwise would not apply to provenance graphs.

5.7.1. Graph-Based Adversarial Attack. For a given adjacency matrix A and GNN f, we can define an adversarial attack as the following:

$$\begin{aligned} f(A) &\neq f(A') \\ \text{st. } A &\neq A' \\ A - A' &\leq \Delta \end{aligned} \tag{1}$$

where A' is defined as $A + D \circ S$. D is an adjacency matrix representing all the potential edges the adversary can add to A. S is a binary matrix where 1 indicates the attack adds the respective edge to the graph. Δ is a perturbation limit, or the number of edges the adversary can add to A without raising an alert, such that $sum(S) < \Delta$.

The attacker aims to learn the optimal S matrix so that A' successfully evades classification. We focus on attacks that utilize gradient descent against the model's loss function to discover S. Given that loss functions allow models to classify input data correctly, the attacker can generate modifications that will most likely cause the model to mispredict by maximizing the model's loss.

However, creating an efficient gradient-based adversarial attack becomes problematic if the model does not use a loss function. Our intrusion detection system identifies anomalies by clustering the node embeddings outputted by the neural model. Clustering does not utilize a loss function the adversary can invert to induce evasion. Previous work [59] has constructed attacks against node embedding models, but these papers assume a secondary neural model downstream.

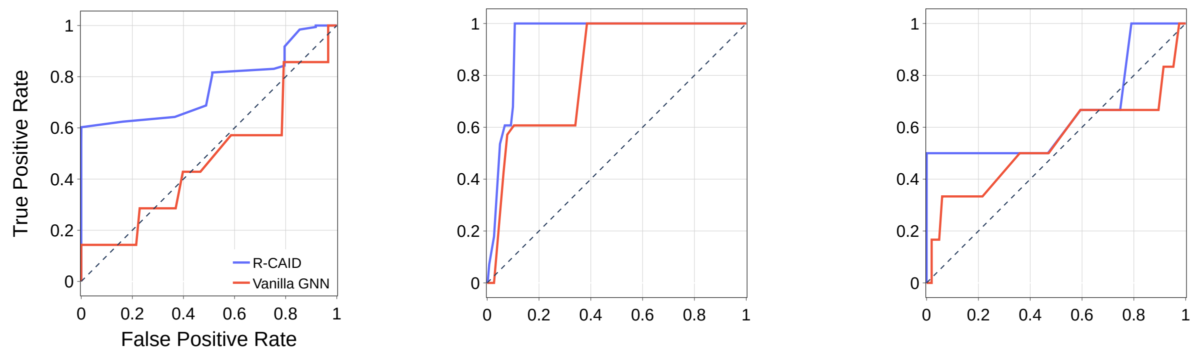
5.7.2. Adapting Adversarial Attacks To R-CAID. We reformulate the current dataset to adapt the adversarial attack to R-CAID. Given that the attacker has complete knowledge of the training and testing dataset, we assume the attacker labels each node benign or malicious. The adversary can then use this dataset to train a node classification in a supervised fashion and define a loss function. This loss function can then be later employed to construct gradient-based adversarial attacks. [60] defines a gradient-based adversarial attack which globally reduces the node classification performance of the model as:

$$\min_S \big(\max \sum_{i \in V} f_i(S, A, x_i, y_i) \big) \tag{2}$$

$$f_i(S, A, x_i, y_i) = \max\{Z_{y_i} - \max_{c_i \neq y - I}(Z_{c_i}) - \kappa\} \tag{3}$$

Z_{y_i}, Z_{c_i} represent the probability of node i being classified as class y and c respectively. Equation 2 optimizes for the smallest number of edges added while maximizing the sum of classification probability of the incorrect over all nodes. In short, Equation 2 maximizes the likelihood of misclassifying all attack nodes with the smallest amount of change possible. To ensure a convex hull, S is a continuous matrix. However, in Equation 1, S is a discrete binary matrix. [60] resolves this by normalizing S to represent a probability matrix and then sampling Δ edges.

On the StreamSpot dataset, given the smaller size of the attack graph, generating and storing a matrix that holds the probabilities for all possible edge addition is possible. However, maintaining a similar probability matrix for Theia and Trace is computationally infeasible because their attack is part of a larger graph. Therefore, we modify the adversarial strategy to be more memory efficient by greedily perturbing the input graph. For each edge addition, the adversarial attack only considers nodes directly part of the attack or nodes directly connected to the attack nodes. The attack graph gets updated with the optimal edge addition at the end of each iteration. Under this strategy, the number of potential edges grows slowly, allowing the adversarial attack to occur within the memory constraints. While these graphs are not globally optimal, at each iteration, they produce the locally optimal edge for that iteration.



(a) StreamSpot (b) Theia (c) Trace

Figure 9: Performance of R-CAID against adversarial samples. In the StreamSpot dataset, R-CAID has a 42% true positive rate while maintaining 0% false positives, while on Theia, R-CAID achieves a 100% true positive rate with 10% false positive rate. On Trace R-CAID identifies 52% of attack nodes with a 0% false positive.

5.7.3. Adversarial Attack Results. We perform the above procedure for two different systems, R-CAID with root node embedding and the Vanilla GNN. We generate adversarial attacks against all three datasets (for StreamSpot, we generate 100 evasion graphs). Figure 9 shows the ROC curves for the adversarial attacks, which we contrast to the results in Figure 7. For the Streamspot dataset, R-CAID's AUC lowers from 0.672 to 0.619 (7.8% decrease) as compared to the Vanilla GCN dropping from 0.614 to 0.469 (24% decrease). For the Theia dataset, R-CAID's AUC drops from 0.999 to 0.941 (5.81% decrease), compared to the Vanilla GCN dropping from 0.897 to 0.82 (8.58% decrease). For the Trace dataset, R-CAID's AUC drops from 0.990 to 0.660 (33.33% decrease), compared to the Vanilla GCN dropping from 0.645 to 0.541 (16.124% decrease).

Although an attacker can impact R-CAID's performance, we still see a highly favorable TPR/FPR tradeoffs on the ROC curve. On Streamspot's dataset, R-CAID identifies 42% of malicious processes with 0% FPR or 60% of malicious processes with near-0% FPR. In the Theia dataset, R-CAID is still able to identify 100% of the malicious processes with a 10% FPR rate. In Trace, R-CAID's TPR at near-0% FPR was previously 100%, whereas its TPR at exactly 0% is now 56.2%; interestingly, this is an increase from its previous value of 33.8% in the passive model. While anecdotal, we suspect this deviation in behavior between Theia and Trace to artifacts from the attack engagement. Trace's workload generator operated uniformly throughout Engagement 3, while through inspecting the logs, we identified that the Theia performers faced several outages during the engagement and appeared to modify their workload generator between outages. It may be that the highly well-formed nature of Trace's benign data caused R-CAID to place greater weight on graph attributes than the adversarial the attack was able to leverage a minority of the process embeddings.

Overall, R-CAID experiences less degradation during the adversarial attack due to the power of root note embedding. We examined several successful evasion attempts to understand the general strategy identified through gradient descent. Broadly, the adversarial sample introduced edges that connected the attack entities to benign nodes in the graph. In Theia, `shell` is spawned off by the attacker after running privilege escalation. During the adversarial attack, the attacker connects `shell` to benign nodes that usually spawn it, making the entire behavior appear more normal. However, because R-CAID links `shell` back to its immutable root causes, which includes an external IP address, the model can still correlate the anomalousness of the activity back to `shell`.

6. Related Work

Host Intrusion Detection utilizes information from the host system to predict whether an anomaly is occurring. Past works explore different avenues of information to use when detecting an intrusion. Such works includes argument dataflows [61], call stack information [62], and process configuration and environment [63]. System calls (syscalls) are predominately used by HIDS [27], [61], [64]–[70]. Previous techniques syscall anomaly detection proposed include hidden Markov models [71], finite state automaton [68], rule induction [64], sequence learning [27], [65], and policy specification [72], [73]. Additional work considers related information such as the appropriate length for syscall sequences [66], [69], ensemble and randomized classifiers [67], [74], [75], false positive reduction [76], multi-log analysis [77], and alert correlation [70], [78], [79]. **Log Sequence Based IDS** are a form of Host IDS that have received near-continuous attention since Stephanie Forrest [27], with modern systems incorporating notions of deep learning. Deeplog [20] and LogAnomaly [21] monitor each system call based on a sliding window LSTM model. Logsy [22] performs a similar function via transformers, while AE [23] instead opts for autoencoders. Log2Vec [80] is a word2vec-based model that uses graph embedding; however, its graphs are not causal provenance graphs but instead encode temporal log dependencies much like other sequence-based IDS. We are not aware of an open-source implementation of [80], so we refrain from commenting on its performance relative to R-CAID.

Provenance Graphs are a graphical representation of system call activity. Previous work in Provenance provenance-based IDS look for anomalous behavior within the graphical structure. StreamSpot [9] and Unicorn [11], [12] generate a vector summarizing the graphs K-hop neighborhood while ProvDetector [7], [14] and Pagoda [15] vectorize the provenance graph through all its paths. SIGL [10] uses an autoencoder to learn normal behavior and spots anomalous program installation. All of the above systems perform whole-graph classification tasks, and [16] shows all systems were systemically vulnerable to mimicry attacks. It is interesting to consider how R-CAID's root cause embedding technique might be adapted to these systems and improve their resilience to evasion.

Attack Reconstruction systems attempt to automate the threat investigation process using data provenance. Such systems assume that a point of interest (i.e., a true alert) in the graph has already been identified, making this task fundamentally different than intrusion detection. Hercule [81] performs clustering over heterogeneous log streams to identify neighborhoods of related activity. Holmes [13] and RapSheet [82] identify related alerts in the provenance graph using heuristics based on the MITRE ATT&CK framework [1]. ATLAS [83] performs supervised learning over attack and benign sequences to learn generalized attack patterns. DEPIMPACT [6] leverages heuristics, including data flow, temporal locality, and concentration of connections, to identify related attack events. Given its ability to identify multiple malicious processes in an attack chain with consistency, R-CAID provides rich "points of interest" for attack reconstruction, which in turn can leverage heuristics to identify any attack process missed by the detection stage.

Graph Neural Networks (GNN) are machine learning models that learn based on graphical data. Our IDS uses a GAT [50] where the model provides attention weights to each of the roots and all the nodes in the graph. There are a variety of other different GNN models: convolutional [84], heterogeneous [85], and temporal [45]. These GNN models can be adapted to provenance graphs and interchanged with the GAT in R-CAID and supply varying performance levels.

7. Discussion

We now consider limitations and threats to the validity of our approach.

Static Graphs: Any HIDS aims to detect intrusions as close to real-time as possible. However, to simplify implementation and experimentation, we make use of static datasets in this work, rather than an online streaming system. While a static IDS can approximate an online one through retraining as frequently as possible, R-CAID would benefit from incorporating notions of streaming detection as have been demonstrated by StreamSpot [57] and Unicorn [12], among others. As the mechanisms for transitioning from static to streaming are well-known and compatible with our root node embedding function, we leave such optimizations to future work.

Memory Overhead: R-CAID's most notable performance cost Adding new "pseudo" nodes and edges to the existing provenance graph. Provenance graphs are already known to be unwieldy in size , and to train a Provenance-based IDS, the graph must be stored in fast memory, so this cost is non-trivial. Indeed, our experiments hit a memory wall when using the larger Theia and Trace datasets. Fortunately, we demonstrated that by pruning the most highly-connected root nodes from the pseudo-graph, it was possible to manage memory costs while maintaining classification performance. Alternate heuristics could exist to select root nodes for the pseudo graph to further reduce costs. An alternative approach is to use existing forensic reduction techniques (see [86], [87]) to reduce the overall size of the graph.

Adversarial Attack: In 5.7, we limited the number of perturbations the attacker could create when evaluating R-CAID's robustness against adversarial attackers. As noted by [16], there is a lack of adversarial attacks that can efficiently scale to the size of provenance graphs and insert additional attack behavior to evade detection. We modified an open-source library to carry out the adversarial attack, and each attack perturbation took about 1.5–2hrs, exponentially increasing as the attacker inserted additional perturbations. It may be that an attacker with sufficient resources could calculate a feasible attack strategy against R-CAID based on considerable perturbation. While we can not disprove this possibility, our adversarial evaluation demonstrates that the cost difference between attacking R-CAID and a regular GNN makes R-CAID considerably more robust. We leave it up to future work to further investigate the robustness of these systems.

Pruning Root Nodes: To reduce the memory overhead of R-CAID, we introduced a root pruning mechanism to reduce the number of edges added to the graph. This optimization creates a possible attack surface wherein an attacker is able to prune *malicious* root nodes by causing them to become connected to a large proportion of system entities. Our adversarial evaluation did not rule out this possibility because those experiments assumed that root causes were immutable, so we discuss it further here. Consider the prune threshold of 5%, which in Figure 6 minimally impacted R-CAID's performance while dramatically reducing memory consumption. In the Theia dataset, a root node connected to 5% of the graph is connected to $186,061$ or more nodes, while the average attack root node connects to just $4,156$ nodes. The adversary would therefore need to establish dependencies between existing malicious entities and $181,905 - 191,478$ additional system entities to force the malicious root causes above the pruning threshold.[5] The attacker could achieve this by simply creating new system entities, or establishing information flows between attack nodes and existing entities, with the latter minimizing the transformation size. This behavior is possible and would

5. These numbers assume the pruning threshold is fixed at $186,061$ nodes and does not increase as more nodes are added to the graph.

force an attacker's root causes out of the root node set, but *the attacker would still need to evade the IDS*. As shown in section 5.7, this means calculating an adversarial sample that, in addition to the base attack, must mask a highly conspicuous addition of $190k$ edges. From our experiments, generating a single-edge modification using an adversarial attack could take $1.5 - 2$ hours, and we would expect this cost to increase dramatically given the size of the new attack graph. Further, without the root nodes R-CAID becomes a Vanilla GNN, which as shown in Figure 9b is still able to detect a large proportion of the attack with low FPR. Thus, it is not clear that this is a practical threat; we leave further exploration to future work.

Omitted Provenance Context: Provenance graphs are so rich in contextual information that tends to be discarded in most provenance systems. A notable omission in R-CAID is edge attributes, including event types and timestamps, as off-the-shelf GNNs do not support them. While the source and destination node types imply the event type, timestamps are an important component of provenance graphs and their omission may give rise to some future form of adversarial attack. A promising direction for future Provenance-based IDS research would be to leverage heterogeneous GNNs [85] so that the model could harness this important context during embedding.

Ground Truth Labeling: Work on provenance graph analysis suffers from a lack of consistency when labeling attack data; strategies range from tagging a small subset of attack-related data sources [5] to labeling an entire system graph as malicious [9]. Having precise ground truth was especially important for R-CAID due to it being the first node-level Provenance-based IDS. Our labeling procedure, described in § 5.2, attempts to label every attacker-controlled or influenced process as malicious. To facilitate community discussion and review, our labels alongside the R-CAID source code can be found on BitBucket.

8. Conclusion

This work presents R-CAID, the first Provenance-based IDS to incorporate root cause analysis. We demonstrate that R-CAID performs effectively against a passive attacker in the more challenging node-level (as opposed to whole-system) classification model, providing improved signal-to-noise as compared to state-of-the-art systems. We go on to show that our incorporation of RCA provides intrinsic resilience to mimicry attacks, even against unrealistically powerful adversaries. We hope that further incorporation of principles from the provenance literature will continue to improve HIDS.

Acknowledgment

We would like to thank the anonymous reviewers for their helpful feedback to improve the paper. This work was supported by NSF under contracts CNS-20-55127 and CNS-20-55233. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of their employers or the sponsors.

References

[1] "MITRE ATT&CK," https://attack.mitre.org, 2019.

[2] FireEye, Inc., "How Many Alerts is Too Many to Handle?" https://www2.fireeye.com/StopTheNoise-IDC-Numbers-Game-Special-Report.html, 2019.

[3] "Automated Incident Response: Respond to Every Alert," https://swimlane.com/blog/automated-incident-response-respond-every-alert/, 2019.

[4] J. Basra and T. Kaushik, "MITRE ATT&CK as a Framework for Cloud Threat Investigation," https://cltc.berkeley.edu/publication/mitre-attck/, Oct 2020.

[5] A. Alsaheel, Y. Nan, S. Ma, L. Yu, G. Walkup, Z. B. Celik, X. Zhang, and D. Xu, "ATLAS: A sequence-based learning approach for attack investigation," in 30th USENIX Security Symposium (USENIX Security 21). USENIX Association, Aug. 2021, pp. 3005–3022. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/alsaheel

[6] P. Fang, P. Gao, C. Liu, E. Ayday, K. Jee, T. Wang, Y. F. Ye, Z. Liu, and X. Xiao, "Back-Propagating system dependency impact for attack investigation," in 31st USENIX Security Symposium (USENIX Security 22). Boston, MA: USENIX Association, Aug. 2022, pp. 2461–2478. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/fang

[7] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates, "Nodoze: Combatting threat alert fatigue with automated provenance triage."

[8] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. N. Venkatakrishnan, "Propatrol: Attack investigation via extracted high-level tasks," in Information Systems Security, V. Ganapathy, T. Jaeger, and R. Shyamasundar, Eds. Cham: Springer International Publishing, 2018, pp. 107–126.

[9] E. Manzoor, S. M. Milajerdi, and L. Akoglu, "Fast memory-efficient anomaly detection in streaming heterogeneous graphs," in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 1035–1044.

[10] X. Han, X. Yu, T. Pasquier, D. Li, J. Rhee, J. Mickens, M. Seltzer, and H. Chen, "{SIGL}: Securing software installations through deep graph learning," in 30th USENIX Security Symposium (USENIX Security 21), 2021, pp. 2345–2362.

[11] X. Han, T. Pasquier, T. Ranjan, M. Goldstein, and M. Seltzer, "Frappuccino: fault-detection through runtime analysis of provenance," in 9th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 17), 2017.

[12] X. Han, T. Pasqueir, A. Bates, J. Mickens, and M. Seltzer, "Unicorn: Runtime Provenance-Based Detector for Advanced Persistent Threats," in 27th ISOC Network and Distributed System Security Symposium, ser. NDSS'20, February 2020.

[13] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "Holmes: real-time apt detection through correlation of suspicious information flows," in 2019 IEEE Symposium on Security and Privacy (SP). IEEE, 2019, pp. 1137–1152.

[14] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Chen, W. Cheng, C. A. Gunter et al., "You are what you do: Hunting stealthy malware via data provenance analysis." in NDSS, 2020.

[15] Y. Xie, D. Feng, Y. Hu, Y. Li, S. Sample, and D. Long, "Pagoda: A hybrid approach to enable efficient real-time provenance based intrusion detection in big data environments," IEEE Transactions on Dependable and Secure Computing, vol. 17, no. 6, pp. 1283–1296, 2020.

[16] A. Goyal, X. Han, A. Bates, and G. Wang, "Sometimes, You Aren't What You Do: Mimicry Attacks against Provenance Graph Host Intrusion Detection Systems," in 30th ISOC Network and Distributed System Security Symposium, ser. NDSS'23, February 2023.

[17] VMware, "Carbon Black Cloud," https://www.vmware.com/products/carbon-black-cloud.html, Last accessed April 2022.

[18] Splunk Inc., "splunk," https://www.splunk.com, Last accessed August 2018.

[19] "Endpoint Detection and Response Solutions Market," https://www.gartner.com/reviews/market/endpoint-detection-and-response-solutions, 2019.

[20] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in Proceedings of the 2017 ACM SIGSAC conference on computer and communications security, 2017, pp. 1285–1298.

[21] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun et al., "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs." in IJCAI, vol. 19, no. 7, 2019, pp. 4739–4745.

[22] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "Self-attentive classification-based anomaly detection in unstructured logs," in 2020 IEEE International Conference on Data Mining (ICDM). IEEE, 2020, pp. 1196–1201.

[23] A. Farzad and T. A. Gulliver, "Unsupervised log message anomaly detection," ICT Express, vol. 6, no. 3, pp. 229–237, 2020.

[24] S. Ma, J. Zhai, Y. Kwon, K. H. Lee, X. Zhang, G. Ciocarlie, A. Gehani, V. Yegneswaran, D. Xu, and S. Jha, "Kernel-supported cost-effective audit logging for causality tracking," in 2018 USENIX Annual Technical Conference (USENIX ATC 18). Boston, MA: USENIX Association, 2018, pp. 241–254. [Online]. Available: https://www.usenix.org/conference/atc18/presentation/ma-shiqing

[25] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in International conference on machine learning. PMLR, 2014, pp. 1188–1196.

[26] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," IEEE Transactions on Knowledge and Data Engineering, vol. 30, no. 9, pp. 1616–1637, 2018.

[27] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for unix processes," in Proceedings 1996 IEEE Symposium on Security and Privacy, May 1996, pp. 120–128.

[28] D. Wagner and P. Soto, "Mimicry attacks on host-based intrusion detection systems," in Proceedings of the 9th ACM Conference on Computer and Communications Security, ser. CCS '02. New York, NY, USA: Association for Computing Machinery, 2002, pp. 255–264. [Online]. Available: https://doi.org/10.1145/586110.586145

[29] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Zhen, W. Cheng, C. A. Gunter, and H. chen, "You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis," in 27th ISOC Network and Distributed System Security Symposium, ser. NDSS'20, February 2020.

[30] K. H. Lee, X. Zhang, and D. Xu, "LogGC: Garbage Collecting Audit Log," in Proceedings of the 2013 ACM SIGSAC conference on Computer and Communications Security, ser. CCS '13. New York, NY, USA: ACM, 2013, pp. 1005–1016. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516731

[31] S. Ma, X. Zhang, and D. Xu, "ProTracer: Towards Practical Provenance Tracing by Alternating Between Logging and Tainting," in Proceedings of NDSS '16, Feb. 2016.

[32] S. Ma, J. Zhai, F. Wang, K. H. Lee, X. Zhang, and D. Xu, "MPI: Multiple Perspective Attack Investigation with Semantic Aware Execution Partitioning," in 26th USENIX Security Symposium, August 2017.

[33] W. U. Hassan, N. Aguse, M. Lemay, T. Moyer, and A. Bates, "Towards Scalable Cluster Auditing through Grammatical Inference over Provenance Graphs," in Proceedings of the 25th ISOC Network and Distributed System Security Symposium, ser. NDSS'18, San Diego, CA, USA, February 2018.

[34] Y. Kwon, F. Wang, W. Wang, K. H. Lee, W.-C. Lee, S. Ma, X. Zhang, D. Xu, S. Jha, G. Ciocarlie, A. Gehani, and V. Yegneswaran, "Mci: Modeling-based causality inference in audit logging for attack investigation," in Proc. of the 25th Network and Distributed System Security Symposium (NDSS'18), 2018.

[35] A. Bates, W. U. Hassan, K. R. Butler, A. Dobra, B. Reaves, P. Cable, T. Moyer, and N. Schear, "Transparent Web Service Auditing via Network Provenance Functions," in 26th World Wide Web Conference, ser. WWW'17, Perth, Australia, April 2017.

[36] V. Karande, E. Bauman, Z. Lin, and L. Khan, "SGX-Log: Securing System Logs With SGX," in Proceedings of the 2017 ASIA CCS, ser. ASIA CCS '17, 2017.

[37] R. Paccagnella, P. Datta, W. U. Hassan, A. Bates, C. W. Fletcher, A. Miller, and D. Tian, "Custos: Practical Tamper-Evident Auditing of Operating Systems Using Trusted Execution," in 27th ISOC Network and Distributed System Security Symposium, ser. NDSS'20, February 2020.

[38] R. Paccagnella, K. Liao, D. Tian, and A. Bates, "Logging to the danger zone: Race condition attacks and defenses on system audit frameworks," in Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1551–1574. [Online]. Available: https://doi.org/10.1145/3372297.3417862

[39] C. Yagemann, M. Noureddine, W. U. Hassan, S. Chung, A. Bates, and W. Lee, "Validating the integrity of audit logs against execution repartitioning attacks," in Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '21, 2021.

[40] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in Thirty-Second AAAI conference on artificial intelligence, 2018.

[41] G. Li, M. Muller, A. Thabet, and B. Ghanem, "Deepgcns: Can gcns go as deep as cnns?" in Proceedings of the IEEE/CVF international conference on computer vision, 2019, pp. 9267–9276.

[42] Groth, Paul and Moreau, Luke, "Prov-overview: an overview of the prov family of documents," 2013.

[43] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer, "Provenance-aware Storage Systems," in Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference, ser. Proceedings of the 2006 Conference on USENIX Annual Technical Conference, Jun. 2006.

[44] T. F. J. . Pasquier, J. Singh, D. Eyers, and J. Bacon, "Camflow: Managed data-sharing for cloud services," IEEE Transactions on Cloud Computing, vol. 5, no. 3, pp. 472–484, July 2017.

[45] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," arXiv preprint arXiv:2006.10637, 2020.

[46] R. Thorndike, "Who belongs in the family?" Psychometrika, vol. 18, no. 4, pp. 267–276, 1953.

[47] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, 2016, pp. 855–864.

[48] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" arXiv preprint arXiv:1810.00826, 2018.

[49] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in International conference on machine learning. PMLR, 2019, pp. 6861–6871.

[50] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," arXiv preprint arXiv:1710.10903, 2017.

[51] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in ICLR Workshop on Representation Learning on Graphs and Manifolds, 2019.

[52] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

[53] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," The journal of machine learning research, vol. 15, no. 1, pp. 1929–1958, 2014.

[54] E. Manzoor, S. M. Milajerdi, and L. Akoglu, "StreamSpot: Detecting network anomalies in edge streams (Source Code and Data)," https://sbustreamspot.github.io/, 2016.

[55] B. Jacob, P. Larson, B. Leitao, and S. Da Silva, "Systemtap: instrumenting the linux kernel for analyzing performance and functional problems," IBM Redbook, vol. 116, 2008.

[56] D. I2O, "Transparent computing engagement 5 data release," https://github.com/darpa-i2o/Transparent-Computing, 2020.

[57] L. Akoglu, "Online detection of anomalous heterogeneous graphs with streaming edges," in 2017 IEEE International Conference on Data Mining Workshops (ICDMW). IEEE, 2017, pp. 968–968.

[58] Z. Chen, J. Liu, W. Gu, Y. Su, and M. R. Lyu, "Experience report: deep learning-based system log analysis for anomaly detection," arXiv preprint arXiv:2107.05908, 2021.

[59] A. Bojchevski and S. Günnemann, "Adversarial attacks on node embeddings via graph poisoning," in International Conference on Machine Learning. PMLR, 2019, pp. 695–704.

[60] K. Xu, H. Chen, S. Liu, P.-Y. Chen, T.-W. Weng, M. Hong, and X. Lin, "Topology attack and defense for graph neural networks: An optimization perspective," arXiv preprint arXiv:1906.04214, 2019.

[61] S. Bhatkar, A. Chaturvedi, and R. Sekar, "Dataflow anomaly detection," in 2006 IEEE Symposium on Security and Privacy (S P'06), May 2006, pp. 15 pp.–62.

[62] H. H. Feng, O. M. Kolesnikov, P. Fogla, W. Lee, and Weibo Gong, "Anomaly detection using call stack information," in 2003 Symposium on Security and Privacy, 2003., May 2003, pp. 62–75.

[63] J. T. Giffin, D. Dagon, S. Jha, W. Lee, and B. P. Miller, "Environment-sensitive intrusion detection," in Recent Advances in Intrusion Detection, A. Valdes and D. Zamboni, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 185–206.

[64] W. Lee and S. J. Stolfo, "Data mining approaches for intrusion detection," in Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7, ser. SSYM'98. USA: USENIX Association, 1998, p. 6.

[65] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting intrusions using system calls: alternative data models," in Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No.99CB36344), May 1999, pp. 133–145.

[66] A. Wespi, M. Dacier, and H. Debar, "Intrusion detection using variable-length audit trail patterns," in Recent Advances in Intrusion Detection, H. Debar, L. Mé, and S. F. Wu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 110–129.

[67] T. Bass, "Intrusion detection systems and multisensor data fusion," Commun. ACM, vol. 43, no. 4, pp. 99–105, Apr. 2000. [Online]. Available: http://doi.acm.org/10.1145/332051.332079

[68] R. Sekar, M. Bendre, D. Dhurjati, and P. Bollineni, "A fast automaton-based method for detecting anomalous program behaviors," in Proceedings 2001 IEEE Symposium on Security and Privacy. S P 2001, May 2001, pp. 144–155.

[69] D. Gao, M. K. Reiter, and D. Song, "On gray-box program tracking for anomaly detection," in Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13, ser. SSYM'04. USA: USENIX Association, 2004, p. 8.

[70] G. Gu, A. A. Cárdenas, and W. Lee, "Principled reasoning and practical applications of alert fusion in intrusion detection systems," in Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security, ser. ASIACCS '08. New York, NY, USA: ACM, 2008, pp. 136–147. [Online]. Available: http://doi.acm.org/10.1145/1368310.1368332

[71] K. Xu, K. Tian, D. Yao, and B. G. Ryder, "A sharper sense of self: Probabilistic reasoning of program behaviors for anomaly detection with context sensitivity," in 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), June 2016, pp. 467–478.

[72] C. Ko, G. Fink, and K. Levitt, "Automated detection of vulnerabilities in privileged programs by execution monitoring," in Tenth Annual Computer Security Applications Conference, Dec 1994, pp. 134–144.

[73] D. Wagner and R. Dean, "Intrusion detection via static analysis," in Proceedings 2001 IEEE Symposium on Security and Privacy. S P 2001, May 2001, pp. 156–168.

[74] J. E. Tapiador and J. A. Clark, "Masquerade mimicry attack detection: A randomised approach," Computers & Security, vol. 30, no. 5, pp. 297–310, 2011.

[75] W. Khreich, S. S. Murtaza, A. Hamou-Lhadj, and C. Talhi, "Combining heterogeneous anomaly detectors for improved software security," Journal of Systems and Software, vol. 137, pp. 415 – 429, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121217300420

[76] G. P. Spathoulas and S. K. Katsikas, "Using a fuzzy inference system to reduce false positives in intrusion detection," in International Conference on Systems, Signals and Image Processing, 2009.

[77] T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda, "Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks," in Proceedings of the 29th Annual Computer Security Applications Conference, ser. ACSAC '13. New York, NY, USA: ACM, 2013, pp. 199–208. [Online]. Available: http://doi.acm.org/10.1145/2523649.2523670

[78] F. Valeur, G. Vigna, C. Kruegel, and R. A. Kemmerer, "Comprehensive approach to intrusion detection alert correlation," IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 3, pp. 146–169, July 2004.

[79] A. Sadighian, J. M. Fernandez, A. Lemay, and S. T. Zargar, ONTIDS: A Highly Flexible Context-Aware and Ontology-Based Alert Correlation Framework. Cham: Springer International Publishing, 2014, pp. 161–177. [Online]. Available: https://doi.org/10.1007/978-3-319-05302-8_10

[80] F. Liu, Y. Wen, D. Zhang, X. Jiang, X. Xing, and D. Meng, "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, ser. CCS â€™19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1777â€"1794. [Online]. Available: https://doi.org/10.1145/3319535.3363224

[81] K. Pei, Z. Gu, B. Saltaformaggio, S. Ma, F. Wang, Z. Zhang, L. Si, X. Zhang, and D. Xu, "Hercule: Attack story reconstruction via community discovery on correlated log graph," in Proceedings of the 32Nd Annual Conference on Computer Security Applications, 2016, pp. 583–595.

[82] W. U. Hassan, A. Bates, and D. Marino, "Tactical Provenance Analysis for Endpoint Detection and Response Systems," in 41st IEEE Symposium on Security and Privacy (SP), ser. Oakland'20, May 2020.

[83] A. Alsaheel, Y. Nan, S. Ma, L. Yu, G. Walkup, Z. B. Celik, X. Zhang, and D. Xu, "{ATLAS}: A sequence-based learning approach for attack investigation," in 30th USENIX Security Symposium (USENIX Security 21), 2021, pp. 3005–3022.

[84] S. Zhang, H. Tong, J. Xu, and R. Maciejewski, "Graph convolutional networks: a comprehensive review," Computational Social Networks, vol. 6, no. 1, pp. 1–23, 2019.

[85] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, 2019, pp. 793–803.

[86] M. Inam, Y. Chen, A. Goyal, J. Liu, J. Mink, N. Michael, S. Gaur, A. Bates, and W. U. Hassan, "SoK: History is a Vast Early Warning System: Auditing the Provenance of System Intrusions," in 2023 IEEE Symposium on Security and Privacy. Los Alamitos, CA, USA: IEEE Computer Society, may 2023, pp. 307–325. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP46215.2023.00018

[87] M. A. Inam, A. Goyal, J. Liu, J. Mink, N. Michael, S. Gaur, A. Bates, and W. U. Hassan, "FAuST: Striking a Bargain between Forensic Auditing's Security and Throughput," in Proceedings of the 38th Annual Computer Security Applications Conference, ser. ACSAC '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 813–826. [Online]. Available: https://doi.org/10.1145/3564625.3567990

Appendix A. Graph Neural Network Layers

For our implementation of R-CAID, we used Attention and Multi-Layer Perceptron (MLP) for the update U and aggregation A function, respectively.

- Attention learns a weighting such that the essential features of a node have the most substantial influence on its classification. In our setting, we can represent Attention in our aggregation function as follows: $A(N_i) = \sum_{j=1}^{N_i} x_j^{l-1} * w^j$, where w_j is a learned weight associated with the feature vector x_j of each node in N_i.
- An MLP can learn the optimal update for x_i^{l-1} and $A(N_i)$. MLP are simple neural network layers that reduce the dimension such that given an input $x_i^{l-1}||A(N_i)$[6] it outputs x_i^l. Reducing the dimension forces MLP to learn the best combination between $x_i^{l-1}, A(N_i)$ such that x_i^l is a good representation for both inputs.

We used Attention as our aggregator because it learns a weight for every incoming edge connection. Provenance graphs are call graphs representing fine-grained system activity and are inherently noisy. For example, within section 5.2, the StreamSpot dataset contains 100 provenance graphs representing the same behavior occurring on the same system. The largest graph for a user browsing CNN is 900106 edges, while the size of the smallest graph has 204263 edges. Moreover, inserting additional edges between the pseudo-provenance graph and base provenance graph creates additional noise for R-CAID to model. Attention is helpful to cut through the noise and identify the essential parts of the underlying data allowing R-CAID to provide a more accurate embedding for each node concerning anomaly detection.

6. || denotes concatenation

Appendix B. Meta-Review

B.1. Summary

This paper proposes an anomaly detector that leverages a provenance graph of system processes, with the goal of detecting abnormal processes using a root-cause analysis. The central insight that this paper leverages is that summarizing the graph into local neighborhoods causes blind spots in performing anomaly detection with the graph. So, the paper creates a pseudo-graph that, roughly speaking, essentially forms the transitive closure of the original graph, so that local neighborhoods in the pseudo-graph represent more global neighborhoods in the original graph. Then, graph-based anomaly detection is performed using the pseudo-graph.

B.2. Scientific Contributions

- Creates a New Tool to Enable Future Science
- Addresses a Long-Known Issue
- Provides a Valuable Step Forward in an Established Field

B.3. Reasons for Acceptance

1) This paper does a very good job of introducing the problem and discussing the issue with the current approach which is basically that depth of the graph is proportional to size of the neighborhood that will be investigated for root cause analysis. This has the potential to miss paths that are longer than a certain length. It may be the case that the observed malicious effect (e.g., sshd writes to a file it's not supposed to) is many hops behind the root (e.g., firefox clones a process it should not have). The augmentation of pseudo root information in the graph embedding is a step forward to addressing the issue.
2) The experimental results are rich, comprehensively comparing the accuracy and performance of R-CAID and baseline, and verifying the experimental results of parameter selection and adversarial attacks.

B.4. Noteworthy Concerns

The paper is correct to point out that leveraging only local neighborhoods of events/processes for anomaly detection gives opportunities for mimicry, but this is not the only one. Nearly any shortcut that an IDS must make to manage its resource usage is an opportunity for mimicry — the adversary simply has to find a way to push its behavior into the part of the space to which the IDS designers have chosen to blind the IDS, in order to work within the available resources. The opportunity for mimicry that stood out to reviewers in this work is one whereby an attacker would cause its roots to be pruned from the graph by inducing connections to more than the required threshold number of nodes; the submitted paper simply described this as ”unlikely”, but it was not entirely clear to reviewers why that should be the case.

Appendix C. Response to the Meta-Review

We want to thank the reviewers for their insightful feedback. In response to the noteworthy concern of an adversary abusing the root pruning mechanism, we have extended our discussion section to consider this issue more fully. To summarize, forcing a malicious root cause to be omitted would require applying a significant transformation to the provenance graph, and attempting to do so would further increase the costs of a successful evasion. This is because, without root causes, R-CAID is effectively a standard GNN; we demonstrate the GNN to be modestly resilient to adversarial perturbation (Figure 9) even prior to the attacker’s conspicuous attempt to prune malicious root causes.