# ORTHRUS: Achieving High Quality of Attribution in Provenance-based Intrusion Detection Systems

Baoxiang Jiang[1*†], Tristan Bilot[2345*†], Nour El Madhoun[3], Khaldoun Al Agha[2], Anis Zouaoui[4]
Shahrear Iqbal[5], Xueyuan Han[6], Thomas Pasquier[7]
[1]*Xi'an Jiaotong University,* [2]*Université Paris-Saclay,* [3]*LISITE, ISEP,*
[4]*Iriguard,* [5]*National Research Council Canada,* [6] *Wake Forest University*
[7]*University of British Columbia*

## Abstract

Past success in applying machine learning to data provenance graphs – a structured representation of the history of operating system activities – to detect host system intrusions has fueled continued interest in the security community. Recent solutions, particularly anomaly-based approaches using graph neural networks (GNNs) to detect previously unknown attacks, have reported near-perfect accuracy. Surprisingly, despite this high performance, the industry remains reluctant to adopt these intrusion detection systems (IDSs).

We identify *Quality of Attribution* (QoA) as the key factor contributing to this disconnect. QoA refers to the amount of effort required from a human analyst to investigate an IDS's detection output, uncover the root causes of an attack, understand its ramifications, and dismiss potential false alarms. Unfortunately, prior work often generates large volumes of low-QoA output, much of which is irrelevant to attack activities, leading to *alert fatigue* and *analyst burnout*. We introduce ORTHRUS, the first IDS to achieve high-QoA detection on data provenance graphs at the node level. ORTHRUS detects malicious hosts using a GNN encoder designed to capture the fine-grained spatio-temporal dynamics of system events. It then reconstructs the attack path through dependency analysis to ensure high-QoA detection.

We compare ORTHRUS against *five* state-of-the-art IDSs. ORTHRUS reduces the number of nodes requiring manual inspection for attack attribution by several orders of magnitude, significantly easing the burden on security analysts while achieving strong detection performance.

## 1 Introduction

In recent years, graph-based machine learning (ML) techniques have gained increasing traction for anomaly detection across various fields, such as finance, social sciences, and biology [10, 19, 21, 62, 67, 74]. Graphs are used to model interactions within complex systems, leveraging domain-specific semantics – typically based on aggregated topological information of nodes and their surrounding neighborhoods – to identify unusual nodes, edges, paths, subgraphs, or even entire graphs [11, 43].

In computer systems, a popular approach to intrusion detection and investigation – the practice of identifying the presence of an attacker in a host system and reasoning about the cause and damage of the attack [35, 41, 76] – is to analyze a special type of graph called a *provenance graph*. In these graphs, nodes represent low-level operating system objects, such as processes, files, and sockets, and edges represent various types of interactions between these objects as a result of system calls [52, 53]. A provenance graph is *directed*, *attributed*, and *dynamic*. It is directed and attributed because two nodes are *causally* connected by a *typed* edge to represent a system object (e.g., a process) acting upon another object (e.g., a file) through a specific system call (e.g., `open`). Different node types also have distinct attributes describing the entities they represent. For example, a `process` node has *command line options*, and a `file` node has a *file name* as attributes. A provenance graph is dynamic because it evolves over time as the underlying system runs, with its temporal structure capturing the *history* of the system's execution.

Provenance graphs can be used to detect malicious behaviors, such as Advanced Persistent Threats (APTs), a particularly stealthy type of attack [27, 40], during system execution. Systems designed for this purpose are often referred to as *Provenance-based Intrusion Detection Systems* (PIDSs). They can be divided into two broad categories: (1) *prior-knowledge-based* solutions; and (2) *anomaly-based* ones. Prior-knowledge-based solutions rely on known attack patterns and are unable to detect unknown attacks. In contrast, anomaly-based systems are unsupervised and learn benign behavior patterns from system execution, enabling them to detect previously unknown attacks. We focus on anomaly-based systems.

Prior work [18, 27, 29, 32, 40, 44, 65, 66, 69, 72, 73] has proposed various techniques for detecting anomalies in prove-

*These authors contributed equally.
†Work done while at the University of British Columbia.

nance graphs that indicate attack activity. These solutions primarily focus on improving detection performance; however, they rarely consider the *attribution quality* of detection signals [20]. Attribution is a crucial aspect of intrusion detection, where security experts must carefully analyze the output of the system to identify an intrusion's root cause and mitigate its impact. Low-quality signals provide either no contextual information or an overwhelming amount of irrelevant data, contributing to *alert fatigue* [31] and frequent *burnout* [61]. We identify two main factors contributing to this problem: (1) the evaluation strategies adopted by prior work (see §2), and (2) the metrics used to evaluate detection performance fail to account for the significant class imbalance in intrusion detection problems (see §5).

We discuss how past evaluation strategies, due to fundamental flaws in their design, have led to systems that overwhelm security analysts with irrelevant information. Building on this insight, we propose a conservative evaluation approach that favors systems producing a minimal amount of data, better aligning with security analysts' needs and guiding the design of ORTHRUS. As a result, ORTHRUS generates data several orders of magnitude smaller than state-of-the-art methods, effectively combating alert fatigue. This reduction would allow analysts to spend less time on report analysis and more time actively *handling* threats [12]. To achieve this, ORTHRUS combines a novel high-precision anomaly detection system to identify key attack-related nodes with a dependency analysis technique to extract relevant ancestors and descendants of those nodes. We evaluate ORTHRUS against five state-of-the-art baselines (Kairos [18], ThreaTrace [66], SIGL [29], MAGIC [36], and Flash [57]). We show ORTHRUS' superiority in terms of detection performance, attribution quality, training time, and memory consumption.

### Contributions

- ORTHRUS is the first PIDS capable of performing meaningful node-level detection on whole-system provenance graphs. We carefully designed a detection pipeline that reduces the amount of data security experts need to analyze by several orders of magnitude.
- We generate and open-source a comprehensive and solid ground truth labeling for PIDSs' standard benchmark datasets (DARPA E3 [8] and E5 [9]).
- We open-source our solution and evaluate it using publicly available datasets (see our compliance with the open science policy in §9).

## 2 Issues with Current Evaluation Strategies

Provenance-based Intrusion Detection Systems (PIDSs) represent a computer system's execution as a directed graph, known as a *provenance graph*, which depicts interactions between system objects (e.g., processes, files, sockets, etc.). This information can be used to detect anomalies in system behavior. PIDSs have evolved from detecting anomalies in repeated executions of the same application [28, 44] to analyzing whole-system provenance [27], achieving increasing accuracy and precision [18, 36, 57]. Unfortunately, the attribution quality of state-of-the-art systems remains a significant barrier to their practical adoption [20], as they often generate an overwhelming number of alerts.

Security analysts must examine the information reported by a detection system to identify attacks and distinguish between true and false positives. This task becomes particularly challenging when the number of anomalous graph elements reported is disproportionately large compared to the actual anomaly [20]. This shortcoming is a direct result of how researchers have evaluated past work. We identify three evaluation strategies used in assessing state-of-the-art systems, highlight their flaws, and propose a new approach to address these shortcomings.[1]

Table 1 compare the number nodes considered malicious under each evaluation strategy. We describe these approaches and explain their flaws below.

**Neighborhood Approach** (e.g., ThreaTrace [66], Flash [57], and MAGIC [36]). In this evaluation strategy, any node within 2 hops (both ancestors and descendants) of a node mentioned in the textual description of the ground truth is considered to be contributing to the attack. For instance, in a scenario where a malicious process and a benign one access a shared library, both nodes are considered part of the attack, even if one process is completely unrelated. In general, a node is labeled as malicious simply by sharing a dependency with an attack-related event. This strategy leads to a significant overestimation of the number of malicious nodes.

**Batch Approach** (e.g., Kairos [18] and EdgeTorrent [38]). In this strategy, the evaluated system processes a batch of events (e.g., a fixed number in the case of EdgeTorrent or based on elapsed time in the case of Kairos) and computes an anomaly score for the entire batch. As with the previous strategy, this tends to overestimate the number of malicious nodes, as events occurring simultaneously with an attack are considered malicious.

**Source Approach** (e.g., R-CAID [25]). In this strategy, the source nodes of an attack are identified, and all descendants are considered malicious. This method is problematic because it overestimates the scope of attack activities. For example, if an attacker hijacks a Firefox process to download a malicious payload, all subsequent activities performed by the browser are labeled as malicious. This can include hours of benign browsing activities that are completely unrelated to the attack. As previous strategies, this significantly overestimate attacks activities.

As Arp et al. [15] warn us about how the choice of poor

[1] We acknowledge that our own work UNICORN [27], SIGL [29], and Kairos [18] have contributed to this trend and share these flaws.

| Datasets | Training | Validation | Test | Total | Neigh. | Batch | Source | Ours | Prevalence |
|---|---|---|---|---|---|---|---|---|---|
| **E3-CADETS** | 449,325 | 40,581 | 268,153 | 758,059 | 12,852 | 4,929 | 2,062 | 68 | $2.5 \times 10^{-4}$ |
| **E3-THEIA** | 410,023 | 34,365 | 699,295 | 1,143,683 | 25,362 | 51,098 | 35,794 | 118 | $1.7 \times 10^{-4}$ |
| **E3-CLEARSCOPE** | 132,121 | 797 | 111,394 | 244,312 | 32,451 | 8,727 | 2,750 | 41 | $3.7 \times 10^{-4}$ |
| **E5-CADETS** | 3,275,875 | 1,245,539 | 3,111,378 | 7,632,792 | 20,524 | 717,783 | 401,065 | 123 | $4.0 \times 10^{-5}$ |
| **E5-THEIA** | 745,773 | 234,896 | 747,452 | 1,728,121 | 162,714 | 61,368 | 9,374 | 69 | $9.2 \times 10^{-5}$ |
| **E5-CLEARSCOPE** | 171,771 | 3,842 | 150,725 | 326,338 | 48,488 | 8,636 | 1,020 | 51 | $3.4 \times 10^{-4}$ |

Table 1: Comparison of evaluation strategies. Ground truth for the *neighborhood* (Neigh.) approach is calculated based on the methodology from Wang et al. [66], the *batch* approach from Cheng et al. [18], and the *source* approach from Goyal et al. [25]. We contacted the authors to confirm our understanding of their methodology. In all cases, our ground truth is smaller by several orders of magnitude. We also report the prevalence of malicious nodes in the test set using our methodology. It is important to note the significant class imbalance, with the proportion of malicious nodes ranging from approximately 1:10,000 to 1:1,000,000 across different datasets. A visual representation of the output of a "perfect" detection system for each strategy is provided in Appendix D.

evaluation methodology can lead to the design of impractical systems, while Pendlebury et al. [54] show how it leads to misleading performance reports in Malware classification task. While systems developed under the above evaluation methodologies may appear capable of identifying perturbations potentially correlated with attacks, they fail to achieve precise node-level attribution. Previous work [18, 57] acknowledges this issue and recognizes the challenge posed by the large volume of data produced by state-of-the-art PIDSs for security analysts. Kairos [18] employs community discovery and summarization techniques to group output data into meaningful subgraphs, while Flash [57] adopts a similar strategy by generating so-called "Attack Evolution Graphs". However, these approaches address only the symptom, not the root problem. They merely organize and summarize the vast amounts of unnecessary data into more manageable chunks [17] without resolving the fundamental inability of these systems to pinpoint nodes directly relevant to malicious behaviors. Practical systems cannot emerge until an appropriate methodology for evaluating them is established.

**Our approach.** We manually and painstakingly analyzed the datasets, comparing the textual documentation with the data they contain, to identify individual nodes that are part of the attacks. While previous approaches considered several thousand nodes as malicious, we identified between 41 and 123 nodes. When we attempted to evaluate previous systems using our approach, we quickly uncovered their inadequacies (see §5). We design ORTHRUS to achieve high attribution quality.

## 3 Threat Model

We follow the same threat model as past work [18, 27, 47, 48, 65]. We assume that attackers seek to take control of and maintain their presence in the target system. Activities not captured by standard kernel-level systems are considered out of scope (e.g., covert and hardware-level side channels). We also assume that the system is outside of attacker control when the training data is captured and when the training occurs (i.e., we exclude data and model poisoning from our threat model). ORTHRUS' software, the provenance capture mechanism, and the underlying OS form our trusted computing base. We assume the capture mechanism and OS are protected from attackers using hardening techniques described in past provenance capture work [16, 52]. Finally, we assume that the provenance record is protected against tampering; we leverage tamper-evident logging techniques [50, 51] to ensure log integrity and detect any attempts at tampering.

## 4 ORTHRUS Framework

ORTHRUS is an anomaly-based intrusion detection system that leverages advanced temporal graph learning techniques and causality analysis to (1) perform node-level anomaly detection *without* prior knowledge of attacks, and (2) reconstruct attack scenarios. ORTHRUS generates concise *attack summary graphs* with high attribution quality, enabling labor-efficient attack investigation. Fig. 1 illustrates the overall architecture of ORTHRUS, which comprises five components:

① **Graph Construction (§4.1).** Provenance graphs are built from raw logs, and redundant edges are pruned while preserving the sequence of events.

② **Edge Featurization (§4.2).** ORTHRUS converts provenance graphs into a sequence of vectorized edges used for graph learning.

③ **Temporal Graph Learning (§4.3).** ORTHRUS employs an encoder-decoder architecture to capture both the structural and temporal aspects of provenance graphs. The model processes the edge vector sequence as input, encoding it into edge embeddings. These embeddings are then decoded to predict the type of each edge. Reconstruction errors are determined by calculating the loss between the predicted and actual edge types.

④ **Anomaly Detection (§4.4).** First, we apply an automated

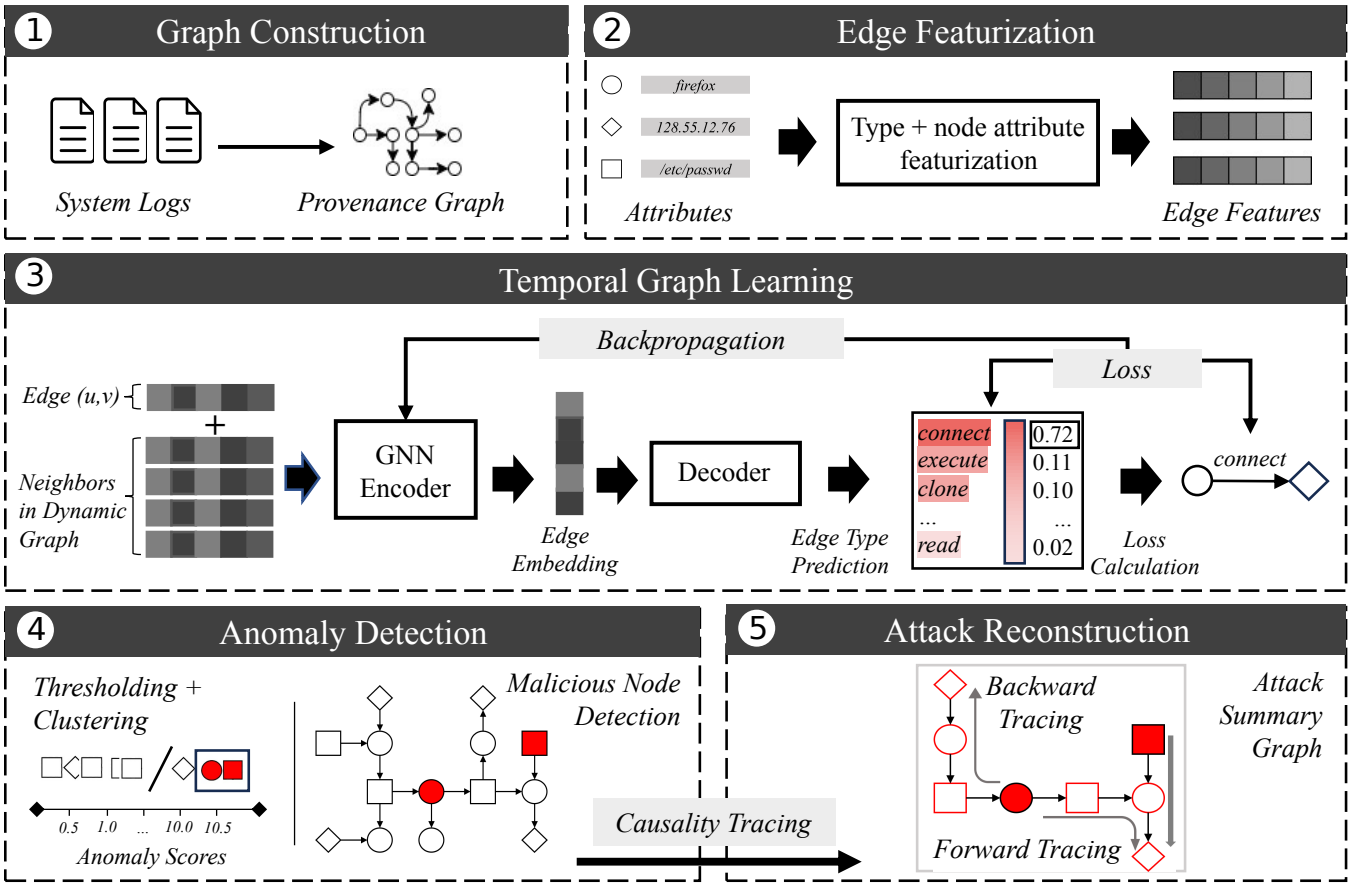



Figure 1: Overview of ORTHRUS framework.

thresholding technique to detect abnormal nodes. Then, we cluster the outliers to differentiate between benign anomalies (i.e., infrequent patterns) and malicious ones (i.e., truly malicious behaviors). This two-step approach eliminates the need for manual threshold setting, significantly reduces false positives, and enhances both precision and attribution quality.

⑤ **Attack Reconstruction (§4.5).** ORTHRUS takes the output of the anomaly detection stage to perform causality analysis. It takes the nodes identified as anomalous and automatically reconstructs attack scenarios. Rather than generating multiple discrete alarms, it provides analysts with a concise set of attack summary graphs, enabling more efficient investigation.

## 4.1 Graph Construction

ORTHRUS parses raw logs generated by various capture mechanisms (e.g., ETW [1], Linux Audit [4], CamFlow [52], eAudit [59]) to construct the input graph. We focus on three types of entities and their interactions: *processes*, *files*, and *netflows*. Table 2 outlines the information we extract from these logs. We reduce redundancy by performing edge pruning through Causality Preserved Reduction [68], which removes redundant edges while preserving the sequential information of events between nodes. Capture systems such as CamFlow [52] and SPADE [23] can automatically handle such redundancy reduction during capture. ORTHRUS can operate effectively on top of various capture mechanisms (see §5).

| Logs | Attributes |
|---|---|
| **Process** | name, cmd |
| **File** | path |
| **Netflow** | local address, local port, remote address, remote port |
| **Event** | subject, object, event type, timestamp |

Table 2: Attributes of logs.

## 4.2 Edge Featurization

At this stage, the provenance graph consists of various types of entities and relations, with entities annotated with textual information such as file paths, process command lines, and IP addresses. This information must be converted into feature vectors, and the provenance graph should be vectorized for the subsequent graph learning phase. To create these feature vectors, we extract five features and concatenate them to generate an edge feature vector $\mathbf{e}_{uv}$ (see Fig. 2).

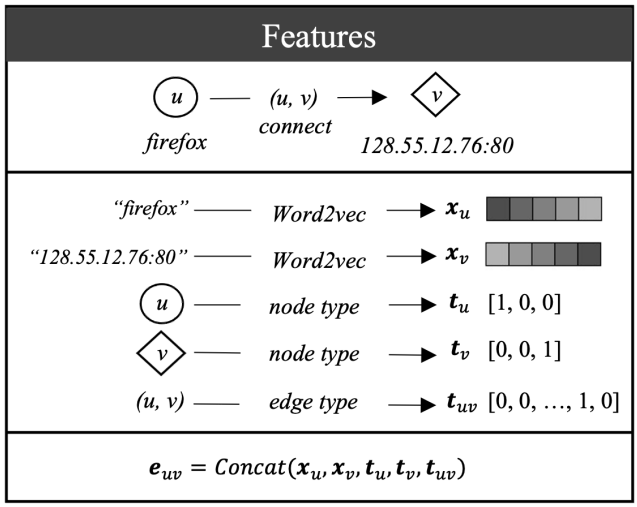



Figure 2: Features extracted from system entities to create edge features for graph learning.

**Encoding Types.** System entities and the edges connecting them contain valuable information that should be integrated into the graph to effectively learn interactions between specific types of entities. We use one-hot encoding to represent these types, converting each type into a fixed-size vector where all positions are zeros except for a single one at the index corresponding to the specific type. Consequently, for each edge $(u, v)$ in the graph, three types are extracted: the type of the event, denoted as $\mathbf{t}_{uv}$, and the types of the two endpoint entities, denoted as $\mathbf{t}_u$ and $\mathbf{t}_v$.

**Encoding Node Attributes.** While the type of a system entity is crucial for distinguishing between different types of entities, such as a process and a file, it is insufficient for differentiating between entities of the same type. For instance, we expect to see distinct behaviors associated with */etc/passwd* and */users/me/Downloads/file.txt*. To address this, we employ Word2vec [46] to transform textual attributes into feature vectors. We tokenize each textual attribute into a sequence of words $(w_1, w_2, ..., w_k)$. For example, the file path */users/me/Downloads/file.txt* would be tokenized into "users, me, Downloads, file, txt". The Word2vec model is trained on the corpus of all textual attributes, embedding each word $w_i$ into a fixed-size vector $\mathbf{w}_i$ that captures the similarity between words based on their context. Prior work [18, 75] suggests that system entities with similar semantics tend to share a similar hierarchy. By using the Skip-gram approach, which predicts context words based on a target word, this hierarchical relationship within attributes can be effectively captured.

Formally, we derive feature vectors for textual attributes by computing the average of their words' embedding:

$$\mathbf{x}_v = \sum_{i=1}^{n} \mathbf{w}_i, \tag{1}$$

where $\mathbf{x}_v$ is the feature vector of node $v$ and $n$ is the number of tokenized words in its attribute. As a result, file paths in the same parent directory (e.g., */var/log/wdev* and */var/log/xdev*), commands of the same process (e.g., *firefox -contentproc -childID 21* and *firefox -contentproc -childID 23*) or netflows on the same network segment (e.g., *128.55.12.110* and *128.55.12.118*) are closely mapped in the embedding space.

## 4.3 Temporal Graph Learning

Provenance graphs are powerful tools for modeling system activity, capturing critical structural features that enhance the profiling of user behaviors. These behaviors, whether benign or malicious, unfold as sequences of ordered events, represented as temporal edges that effectively encode temporal information within the graph. Leveraging both spatial and temporal dimensions is crucial for detecting attacks, as they inherently unfold across both. ORTHRUS models benign user behavior at the system level by analyzing interactions between entities such as processes, files, and sockets, utilizing both spatial and temporal dimensions.

To achieve this, ORTHRUS employs an encoder-decoder architecture specifically designed to capture the spatiotemporal context of system events within provenance graphs. The encoder utilizes a GNN architecture, which has proven effective in detecting complex attacks [18, 36, 57, 66]. To better capture the temporal dynamics of provenance graphs, ORTHRUS employs a *dynamic graph* structure, which incorporates temporal information by assigning timestamps to each edge, unlike static graphs that represent only the spatial dimension. This design enables the encoder to learn *embeddings* of nodes and edges that effectively preserve the structural and temporal patterns generated by system entities.

Our model does not rely on labeled attack data, as such labels are challenging to obtain in the cybersecurity domain. Additionally, supervised learning on specific attacks can limit the ability of a model to generalize to unseen attacks [18]. Therefore, ORTHRUS' encoder and decoder are trained in a self-supervised manner on benign data, functioning as an anomaly detection system. Specifically, the decoder reconstructs the type of each edge, as shown in Fig. 1, by learning to infer the event type from its spatiotemporal context. By training on a substantial amount of benign data, the model optimizes this reconstruction process, enabling it to learn benign user activity without the need for handcrafted labels.

During inference, ORTHRUS calculates reconstruction errors for each incoming edge. Edges with high reconstruction errors, which deviate significantly from typical benign patterns, are flagged as potentially malicious. The system then identifies the entities responsible for the attack and the timing of the attack by analyzing the source nodes of these high-error edges and their associated timestamps. In a subsequent step (see §4.5), an attack graph can be constructed from the predicted nodes and edges, assisting analysts in tracing the attack's path and reducing their workload.

**Encoder.** To enhance ORTHRUS' applicability to real-world scenarios, we propose a lightweight encoder (relatively low memory and computational complexity) that can learn useful temporal edge embeddings in a stateless manner while maintaining high detection performance. Unlike most GNN encoders used by other systems, ORTHRUS' encoder retains past events involving each system entity during the aggregation process using a temporal sampling approach. While KAIROS employs a similar approach, its encoder's reliance on state memory leads to significantly higher memory usage (nearly threefold) and weaker performance, as shown in the "Encoding" ablation in Table 7. This overhead arises because KAIROS stores an additional memory vector for each system entity and directly uses it during GNN propagation, substantially increasing the memory footprint. In contrast, ORTHRUS leverages the existing node features of entities without requiring additional storage. Although the state vector in KAIROS is intended to capture long-range dependencies, our ablation experiments show that our simple temporal sampling approach is sufficient.

For each incoming system event from node $u$ to $v$ occurring at time $t$, we aim to aggregate the information from source node $u$ within $v$. To capture the structural and temporal patterns surrounding $v$, the aggregation process should also consider the edges that occurred prior to time $t$, ensuring that node $v$ is aware of its context within the graph. We define the set of all edges that occurred prior to time $t$ as

$$S_t(v) = \{(u,v) \in E \mid t_{uv} < t,\ t_{uv} \in \mathcal{T}\}, \tag{2}$$

where $t_{uv}$ denotes the timestamp of the edge from node $u$ to node $v$, $E$ represents the set of all edges and $\mathcal{T}$ denotes the timestamps associated to edges in $E$. From $S_t(v)$, we sample the $\mathcal{N}$ most recent edges that interacted with node $v$, ensuring that the temporal information of previous events is preserved. By using a fixed-size sample for aggregation, the amount of aggregated information is controlled. This approach enhances scalability and has proven highly effective in inductive settings, which involve generalizing a model to nodes that were not encountered during training [26]. We define the sampled subset as

$$S_{\mathcal{N}}(v) = \text{SAMPLE}(S_t(v), \mathcal{N}, \mathcal{T}, t), \tag{3}$$

where SAMPLE is a function that selects the $\mathcal{N}$ last incoming edges prior to time $t$.

The structural and temporal interactions within these sampled graphs are captured using an attention-based GNN that leverages features. ORTHRUS is the first system to incorporate the types of both system entities and events as features during encoding. Specifically, system call types are encoded as one-hot *edge features*, while *node features* are constructed by concatenating the one-hot encoded system entity type (e.g., file, process, or socket) with the word embedding of that entity, as detailed in §4.2. These combined features enrich the graph with valuable information that the GNN can exploit, along with the structural and temporal properties gathered during the previous temporal sampling step. The encoder computes an embedding for each node by integrating its own features, the features of its neighboring nodes, and the associated edge features:

$$\mathbf{h}_v = \text{GNN}\left(\mathbf{x}_v, \{\mathbf{x}_u, \mathbf{e}_{uv} \mid (u,v) \in S_{\mathcal{N}}(v)\}\right), \tag{4}$$

where $\mathbf{x}_u$ and $\mathbf{x}_v$ represent the feature vectors of the emitting node $u$ and receiving node $v$, respectively, $\mathbf{h}_v$ is the resulting embedding of node $v$, and $\mathbf{e}_{uv}$ denotes the edge features of edge $(u,v)$.

Similarly to KAIROS, we employ a variant of the GraphTransformer layer introduced by Shi et al. [60] (Equations 5 and 6) as our GNN layer. This GraphTransformer variant is particularly well-suited for learning on provenance graphs, as it learns attention coefficients that weigh the importance of each system entity relative to others. This mechanism allows each node to focus on the most relevant neighboring information during aggregation, enabling attention-based GNNs to efficiently capture essential features that distinguish between benign and anomalous instances during detection, thereby enhancing attribution quality. For every edge $(u,v)$, an attention coefficient $\alpha_{u,v}$ is calculated using the features of both the source and destination nodes, along with the edge features, and is normalized with a softmax function across all connected neighbors:

$$\alpha_{u,v} = \text{softmax}\left(\frac{(\mathbf{W}_3\mathbf{x}_v)^\top(\mathbf{W}_4\mathbf{x}_u + \mathbf{W}_5\mathbf{e}_{uv})}{\sqrt{d}}\right), \tag{5}$$

where $d$ represents the hidden size, and $\mathbf{W}_3$, $\mathbf{W}_4$, and $\mathbf{W}_5$ are learnable weight matrices. The normalized attention coefficients are subsequently used in the aggregation step to weigh the learned features of neighboring nodes:

$$\mathbf{h}_v = \mathbf{W}_1\mathbf{x}_v + \sum_{(u,v)\in S_{\mathcal{N}}} \alpha_{u,v}\left(\mathbf{W}_2\mathbf{x}_u + \mathbf{W}_5\mathbf{e}_{uv}\right), \tag{6}$$

where $\mathbf{W}_1$ and $\mathbf{W}_2$ are weight matrices. Our experiments, detailed in §5.4, demonstrate that setting the temporal sampling parameter $\mathcal{N}$ between 10 and 20, with a node embedding size of $z = 32$, yields the best average performance across datasets.

**Decoder.** At this stage, the node embeddings effectively capture the structural and temporal behaviors specific to each entity in the provenance graph. The decoder then learns these embeddings by computing a loss function that optimizes a given objective. Specifically, the decoder is trained to predict the edge type for all connected node pairs. By accurately predicting edge types, the model not only reinforces its understanding of the interactions between system entities but also deepens its comprehension of the relationship dynamics between them. This is essential for distinguishing between benign and potentially malicious events. The integration of

node and edge types as features within the graph also significantly enhances the model's ability to predict an event's type based on its surrounding context.

We have designed a simple yet effective decoder (Equation 7) that distinguishes between source and destination entities by projecting their embeddings through distinct weight matrices. This approach accounts for the differing semantics of nodes that emit and receive information, ensuring these roles are appropriately modeled during training. Formally, the predicted type of an edge $(u, v)$ is calculated based on the embeddings of its end nodes such that:

$$\hat{y}_{uv} = \sigma\left(\mathbf{W}_g\left[\mathbf{W}_s\mathbf{h}_u, \mathbf{W}_d\mathbf{h}_v\right]\right), \tag{7}$$

where $\hat{y}_{uv}$ denotes the predicted edge type, $[.,.]$ is the concatenation operation, $\sigma$ is the Sigmoid activation function and $\mathbf{W}_g$, $\mathbf{W}_s$ and $\mathbf{W}_d$ denote the gate, source and destination weight matrices, respectively. The entire model is trained by deriving a loss value from the predicted edge type $\hat{y}_{uv}$ and the ground truth $y_{uv}$ using the Cross-Entropy (CE) loss across all edge types to predict:

$$\mathcal{L}_{uv} = \mathrm{CE}(\hat{y}_{uv}, y_{uv}). \tag{8}$$

By optimizing this loss for each benign edge in the training set, the weights within the encoder and decoder are updated, allowing ORTHRUS to learn the user's normal activity. During inference, anomalous temporal or structural patterns in the graph will produce embeddings that deviate significantly from those learned during training, resulting in a high reconstruction loss. This high reconstruction loss can then be leveraged in a subsequent detection step to identify anomalies.

## 4.4 Anomaly Detection

ORTHRUS can learn representations of benign and malicious nodes directly in the embedding space. Identifying malicious embeddings with minimal false positives is a crucial challenge that all PIDSs must address. While some state-of-the-art techniques [36, 66] detect anomalies based on scores exceeding a manually set threshold, this approach is impractical in real-world scenarios due to concept drift and changes in underlying data. To address this issue, ORTHRUS employs a two-step anomaly detection method designed to reduce false positives.

**Automatic Anomaly Thresholding.** Suspicious nodes are flagged when their anomaly score exceeds a threshold, which is automatically set to the highest anomaly score observed in the validation set. Typically, an entire day of benign data is reserved as a validation set, ensuring it remains unused during training. In this step, nodes with high anomaly scores—indicating significant deviation from benign activity—can be effectively distinguished from other nodes predicted as benign with lower scores. This approach has proven highly effective in identifying deviations from normal behavior [29].

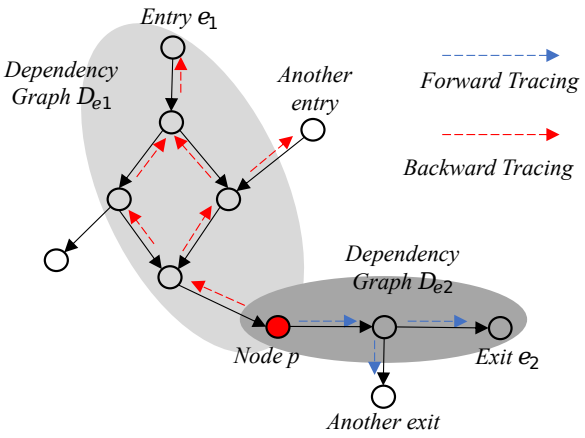



Figure 3: Causality analysis from a detected node $p$.

**Outlier Clustering.** To further differentiate between benign anomalies, which may commonly occur in real-world situations, and truly malicious anomalies, ORTHRUS applies a K-means clustering algorithm [30] with two clusters to the set of suspicious nodes identified through thresholding. This approach isolates outliers with the highest anomaly scores, effectively identifying the cluster of the most suspicious anomalies and thereby enhancing attribution quality. The cluster of the most suspicious anomalies can then be prioritized in subsequent evaluation steps, while the cluster containing less suspicious nodes is treated as benign anomalies and not considered part of an attack. This clustering step significantly reduces false positives and alleviates the workload for analysts (see §5.5).

## 4.5 Attack Reconstruction

ORTHRUS reconstructs the attack associated with nodes flagged as malicious during anomaly detection and provides security analysts with *attack summary graphs* to facilitate investigation. Intuitively, the attack should correspond to events within a small subgraph surrounding a malicious node. This subgraph can be extracted using causality analysis [22, 39]. However, some causal dependencies between benign and malicious nodes also exist (e.g., a *firefox* instance used to download a piece of malware may continue to perform benign actions for hours after the initial incident). ORTHRUS conducts a causality analysis for each node $p$ identified during the anomaly detection phase.

**Dependency Analysis.** When an anomaly is detected, we extract a subgraph corresponding to the time window surrounding the anomalous event. The duration of this time window is adjustable, allowing analysts to tailor it based on preferences, graph size, or the presence of high anomaly scores near the malicious event. Since attack reconstruction is a post-processing step with anomaly scores already stored, analysts can re-run this process using different time window sizes with-

out incurring significant computational overhead. This flexibility facilitates the identification of attack boundaries and the discovery of relevant attack patterns. We set the default time window to 15 minutes, as this duration proved optimal across most datasets, consistent with previous findings [18]. By limiting the graph size analyzed, this approach reduces computational costs. Additionally, the extracted time-window subgraph is transformed into a directed acyclic graph (DAG) to ensure all nodes along a path maintain causal relationships. This is achieved by creating multiple versions of nodes as the state of the corresponding subject or object changes [49]. Next, we identify potential attack *entry* and *exit* nodes. An entry point could be the attacker's IP address used to infiltrate the system by delivering a malicious payload, while an exit point might be a file that has been stealthily relocated to a hidden directory within the system. An entry node, which does not have parents, serves as the root of a path. Conversely, *exit* nodes are without children, marking the end of a path. Entries and exits define the boundaries of an attack path. We identify entry and exit nodes by traversing the graph from $p$ using *backward tracing* and *forward tracing*, as illustrated in Fig. 3.

We define a *dependency graph* as the set of all paths between an entry/exit node and $p$. In Fig. 3, the *dependency graph* $D_{e_1}$ includes all paths generated through tracing between entry $e_1$ and $p$, while the *dependency graph* $D_{e_2}$ includes paths traced between exit $e_2$ and $p$. We repeat this operation until all *dependency graphs* between $p$ and all possible entry/exit are identified.

**Critical Dependency Identification.** After identifying all potential entries and exits, along with the *dependency graphs* connecting them to $p$, we select the most anomalous entry and exit to reconstruct the attack graph. To do this, we assign a *criticality score* $f_C(e_i)$ to each entry/exit $e_i$. The entry and exit with the highest criticality scores are identified as the critical entry and critical exit, respectively. The union of their *dependency graphs* is then mapped back to the original provenance graph and reported as the attack graph to the security analyst.

**Calculating the Criticality Score.** We associate every node in a *dependency graph* with two scores: (1) a *degree score* $f_D(u)$; and (2) an *anomaly score* $f_A(u)$. The *degree score* $f_D(u)$ measures the relevance of a node $u$ to the *dependency graph*:

$$f_D(u) = \text{OutDegree}(u)/\text{InDegree}(u). \tag{9}$$

This approach builds on the insight from Fang et al. [22] that nodes with a high *degree score* exert strong influence on data flow, enabling more accurate identification of key nodes within a *dependency graph*. The *anomaly score* $f_A(u)$ is the average reconstruction loss (from §4.3) of all edges connected to $u$:

$$f_A(u) = \frac{1}{|E_u|} \sum_{uv \in E_u} \mathcal{L}_{uv}. \tag{10}$$

Here, we build on the insight that attack nodes are clustered in highly anomalous regions of the provenance graph (i.e., regions with high reconstruction loss). Finally, the criticality score $f_C(e)$ of an entry/exit $e$ is calculated as the average of the normalized degree and anomaly score of all nodes within the corresponding dependency graph $D_e$:

$$f_C(e) = \frac{1}{|V_e|} \sum_{u \in V_e} (\hat{f_D}(u) + \hat{f_A}(u)), \tag{11}$$

where $V_e$ is the set of nodes in the dependency graph $D_e$, $\hat{f_D}(u)$ and $\hat{f_A}(u)$ denote the degree and anomaly scores normalized with min-max normalization, to eliminate the difference in magnitude between scores.

## 5 Evaluation

ORTHRUS' design fulfills two objectives: (1) detecting all attacks occurring within a system, and (2) minimizing the amount of data that security analysts need to review. Given the significant imbalance in the datasets (with ratios of malicious nodes ranging from 1:10,000 to 1:1,000,000), it is crucial to carefully choose our optimization criteria. As Dong et al. [20] highlight, a practical system must minimize false positives (i.e., ensure high precision) as long as all attacks are detected, rather than focusing on identifying every single node involved in an attack (i.e., high recall). Additionally, the significant dataset imbalance must be considered when selecting evaluation metrics. We assess ORTHRUS' ability to meet these objectives. We begin by introducing the datasets and baselines, we then address the following research questions:
**RQ1:** Is ORTHRUS able to detect all attacks?
**RQ2:** What is the quality of attribution?
**RQ3:** Is ORTHRUS computationally efficient?
**RQ4:** How do hyperparameters influence performance?
**RQ5:** How the different ORTHRUS' components contribute to overall performance?
**RQ6:** How robust is ORTHRUS against mimicry attacks?

All training and evaluation were conducted on a server running Ubuntu 22.04, equipped with a 3.2GHz 16-core AMD EPYC 7343 CPU, 1024 GB of memory, and an NVIDIA GA100 GPU with 80GB of memory.

### Datasets

To the best of our knowledge, the only large datasets that are publicly and widely available are those associated with the DARPA Transparent Computing Program [7]. Other datasets are either small (Manzoor et al. [44] and Han et al. [27]), not publicly available (Zeng et al. [72] and Wang et al. [65]),

or designed primarily for attack matching, lacking sufficient benign behavior to properly train and evaluate anomaly-based systems (Alsaheel et al. [13]).

Consequently, we leverage the publicly available datasets [8, 9] from the DARPA TC program, which are widely used throughout the literature as benchmarks for PIDSs [18, 24, 27, 36, 57, 65, 73]. During the TC program, a team of security experts organized several adversarial engagements simulating real-world attacks on an enterprise network targeting critical service infrastructure (e.g., SSH, email, or web servers). These attacks occurred alongside benign activities such as browsing the internet, checking and responding to emails, and performing SSH logins. The data was captured by three different mechanisms—CADETS, THEIA, and CLEARSCOPE—operating in three distinct environments: FreeBSD, Linux, and Android. These datasets are highly imbalanced, as indicated by the low prevalence of malicious nodes outlined in the dataset statistics in Table 1. We give further details about how we split the datasets in Appendix A.

### Evaluated Systems

We select five state-of-the-art PIDSs [18, 29, 36, 57, 66] as baselines. To the best of our knowledge, this is the most extensive comparison performed to date, focusing on recent (2021–2024) anomaly-based systems. We exclude older systems that cannot provide node-level anomaly scores (e.g., UNICORN [27] and Streamspot [44]). We note that our selected baselines were evaluated against those approaches and outperform them. We also exclude systems that require attack samples and/or descriptions (e.g., ATLAS [13] and ProvG-Searcher [14]), as they cannot be fairly compared to anomaly-based systems. Furthermore, we exclude systems that were not fully open-sourced, did not function, or could not be re-implemented based on the paper (e.g., ProvDetector [65], ShadeWatcher [73], and R-CAID [25]).[2] We ensure that our comparison includes all three open-source state-of-the-art anomaly detection systems published in 2024 (i.e., Kairos [18], MAGIC [36], and Flash [57]). Further details on past evaluations are provided in Appendix B.

To ensure consistent evaluation across all baseline methods, we use the original code for each system whenever possible, modifying only the final evaluation step to use our evaluation strategy. We follow the original papers' instructions and parameterize the systems to achieve the best possible outcome under our evaluation methodology.

**Kairos [18].** Kairos, based on a TGN encoder [58], is trained to predict edge types using node state vectors and a 4-layer MLP decoder. We use the original Kairos code [3], modifying only its time window-level evaluation to a node-level evaluation (corresponding to the output described in §4.3.1 of their paper).

[2]This should not be seen as a reflection on the quality of these systems.

| Dataset | System | E3 | E5 |
|---|---|---|---|
| CADETS | ORTHRUS | ✓ 3/3 | ✓ 2/2 |
| | Kairos | ✗ 0/3 | ✗ 0/2 |
| | ThreaTrace | ✓ 3/3 | ✓ 2/2 |
| | SIGL | ✗ 0/3 | ✗ 0/2 |
| | MAGIC | ✓ 3/3 | ✓ 2/2 |
| | Flash | ✓ 3/3 | ✓ 2/2 |
| THEIA | ORTHRUS | ✓ 2/2 | ✓ 1/1 |
| | Kairos | ~ 1/2 | ✗ 0/1 |
| | ThreaTrace | ✓ 2/2 | ✓ 1/1 |
| | SIGL | ~ 1/2 | ✗ 0/1 |
| | MAGIC | ✓ 2/2 | ✓ 1/1 |
| | Flash | ✓ 2/2 | ✓ 1/1 |
| CLEARSCOPE | ORTHRUS | ✓ 1/1 | ✓ 3/3 |
| | Kairos | ✗ 0/1 | ~ 1/3 |
| | ThreaTrace | ✓ 1/1 | ✓ 3/3 |
| | SIGL | ✓ 1/1 | ~ 2/3 |
| | MAGIC | ✓ 1/1 | ✓ 3/3 |
| | Flash | ✗ 0/1 | ✓ 3/3 |

Table 3: Attack detection performance on DARPA datasets.

**ThreaTrace [66].** ThreaTrace trains a GraphSAGE [26] model to predict node type and reports misclassified nodes as anomalous. We use the publicly available ThreaTrace's source code [6].

**SIGL [29].** SIGL detects malicious graphs by assigning a *normality score* to each node (see §4.5 in their paper) using a Graph-LSTM architecture [55]. Since SIGL is not open-sourced, we re-implemented it following prior work [24].

**MAGIC [36].** MAGIC employs a Graph Attention Network [64] based autoencoder to embed nodes and performs outlier detection to identify anomalies We use the publicly available MAGIC code [5] and their proposed thresholding method, which relies on ground truth labels.[3]

**Flash [57].** Flash utilizes a GNN model with positional encoding [63] to detect anomalous nodes based on type prediction. We use the publicly available Flash code [2], which specifies a confidence threshold for each dataset. For datasets that Flash did not evaluate, we follow the paper's guidelines to determine the appropriate threshold.

**ORTHRUS-full.** We run ORTHRUS from the provenance graph construction step to the attack reconstruction step (§4.1~§4.5).

**ORTHRUS-ano.** We run ORTHRUS without the attack reconstruction step described in §4.5.

## 5.1 Attack Detection Performance

First, we evaluate ORTHRUS' ability to detect all attacks within each dataset and compare its performance with the selected baselines. An attack is considered detected if the system flags any node directly involved in the attack as malicious. The results presented in Table 3 demonstrate ORTHRUS' ability to detect all attacks. When other systems fail to detect

[3]This approach may present potential issues, but it is the method followed by the authors.

| Dataset | System | TP | FP | TN | FN | Precision | MCC |
|---|---|---|---|---|---|---|---|
| E3-CADETS | ORTHRUS-full | 25 | 23 | 268k | 43 | 0.52 | **0.44** |
| | ORTHRUS-ano | 10 | 0 | 268k | 58 | **1.00** | 0.38 |
| | Kairos | 0 | 9 | 268k | 68 | 0.00 | 0.00 |
| | Threatrace | 61 | 252k | 16k | 7 | 0.00 | 0.00 |
| | SIGL | 0 | 80 | 268k | 68 | 0.00 | 0.00 |
| | MAGIC | 63 | 80k | 188k | 5 | 0.00 | 0.02 |
| | Flash | 13 | 2.4k | 266k | 55 | 0.01 | 0.03 |
| E3-THEIA | ORTHRUS-full | 48 | 11 | 699k | 70 | 0.81 | **0.57** |
| | ORTHRUS-ano | 8 | 0 | 699k | 110 | **1.00** | 0.26 |
| | Kairos | 4 | 0 | 699k | 114 | **1.00** | 0.18 |
| | Threatrace | 88 | 672k | 27k | 30 | 0.00 | -0.01 |
| | SIGL | 1 | 29 | 699k | 117 | 0.03 | 0.02 |
| | MAGIC | 115 | 395k | 304k | 3 | 0.00 | 0.01 |
| | Flash | 22 | 32k | 667k | 96 | 0.00 | 0.01 |
| E3-CLEARSCOPE | ORTHRUS-full | 2 | 6 | 111,347 | 39 | 0.25 | **0.11** |
| | ORTHRUS-ano | 1 | 1 | 111k | 40 | **0.50** | **0.11** |
| | Kairos | 0 | 7 | 111k | 41 | 0.00 | 0.00 |
| | Threatrace | 41 | 88k | 24k | 0 | 0.00 | 0.01 |
| | SIGL | 1 | 11k | 100k | 40 | 0.00 | 0.00 |
| | MAGIC | 40 | 102k | 9.6k | 1 | 0.00 | 0.00 |
| | Flash | 0 | 15k | 96k | 41 | 0.00 | -0.01 |

Table 4: Comparison of node-level detection performance on DARPA E3 datasets. The raw data is available in Appendix F.

| Dataset | System | TP | FP | TN | FN | Precision | MCC |
|---|---|---|---|---|---|---|---|
| E5-CADETS | ORTHRUS-full | 2 | 10 | 3M | 121 | **0.17** | **0.05** |
| | ORTHRUS-ano | 1 | 5 | 3M | 122 | **0.17** | 0.04 |
| | Kairos | 0 | 6 | 3M | 123 | 0.00 | 0.00 |
| | Threatrace | 91 | 3M | 7k | 32 | 0.00 | -0.03 |
| | SIGL | 0 | 66 | 3M | 123 | 0.00 | 0.00 |
| | MAGIC | 123 | 3M | 541 | 0 | 0.00 | 0.00 |
| | Flash | 45 | 34k | 3M | 78 | 0.00 | 0.02 |
| E5-THEIA | ORTHRUS-full | 13 | 2 | 747k | 56 | 0.87 | 0.4 |
| | ORTHRUS-ano | 2 | 0 | 747k | 67 | **1.00** | **0.17** |
| | Kairos | 0 | 2 | 747k | 69 | 0.00 | 0.00 |
| | Threatrace | 66 | 739k | 8k | 3 | 0.00 | 0.00 |
| | SIGL | 0 | 23 | 747k | 69 | 0.00 | 0.00 |
| | MAGIC | 1 | 297k | 451k | 68 | 0.00 | -0.01 |
| | Flash | 43 | 296k | 452k | 26 | 0.00 | 0.00 |
| E5-CLEARSCOPE | ORTHRUS-full | 4 | 8 | 151k | 47 | **0.33** | **0.16** |
| | ORTHRUS-ano | 2 | 7 | 151k | 49 | 0.22 | 0.09 |
| | Kairos | 1 | 3 | 151k | 50 | 0.25 | 0.07 |
| | Threatrace | 41 | 142k | 8k | 10 | 0.00 | -0.01 |
| | SIGL | 10 | 63 | 151k | 41 | 0.14 | 0.16 |
| | MAGIC | 51 | 139k | 11k | 0 | 0.00 | 0.01 |
| | Flash | 15 | 4.6k | 146k | 36 | 0.00 | 0.03 |

Table 5: Comparison of node-level detection performance on DARPA E5 datasets. The raw data is available in Appendix F.

attacks, they typically flag nodes in the attack's vicinity rather than nodes directly involved. This aligns with §2 hypothesis.

## 5.2 Attribution Quality

In RQ1, we observed that most state-of-the-art detection systems can successfully identify the majority of attacks. Next, we aim to assess the quality of attribution. Using E3-CADETS as an example, as described in Table 1, we identified 68 nodes that are directly relevant to the attacks. As shown in Table 4, ORTHRUS-full flagged a total of 48 nodes (25 true positives, 23 false positives) as malicious, while Flash flagged 2,394 nodes (13 true positives, 2,381 false positives).

Given the imbalanced nature of our datasets (i.e., attacks constitute a very small portion of system activity) and our objectives, *precision* is the most relevant metric, provided that all attacks are detected. Precision measures the probability that a node flagged as malicious is actually part of an attack. ORTHRUS achieves a precision of 52%, compared to less than 1% for Flash.

Metrics like accuracy, ROC-AUC, and F1 score are commonly used in binary classification tasks but can yield overly optimistic results, especially with imbalanced datasets (e.g., a classifier that always predicts negatives could achieve over 99% accuracy). In contrast, the Matthews correlation coefficient (MCC) [45] (see §5.2) is a more reliable measure [56], as it produces high scores only when predictions perform well across all four categories of the confusion matrix: True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN).

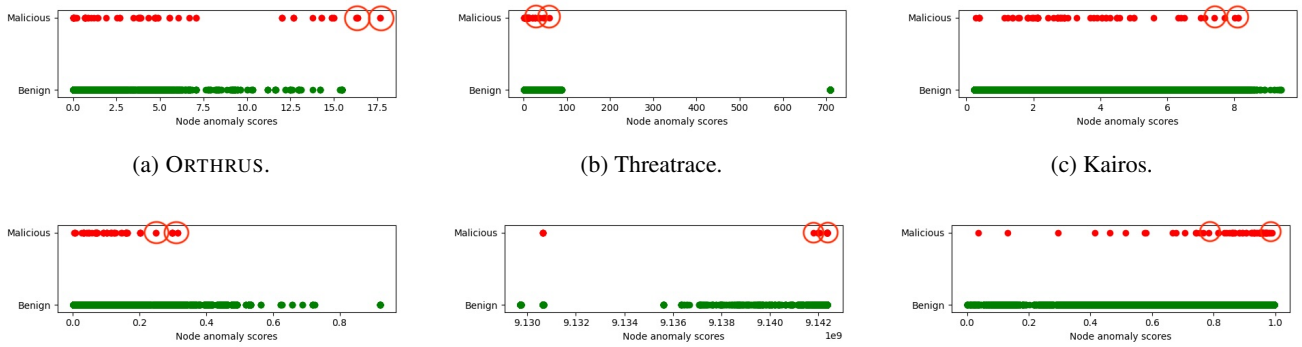
By accounting for both positive and negative elements proportionally, MCC is particularly well-suited for evaluating binary classifiers on highly imbalanced datasets. An MCC value of 1 indicates perfect classification, 0 reflects random guessing, and -1 signifies complete misclassification, where all negative samples are predicted as positive and vice versa. However, MCC is dataset-dependent (i.e., an MCC of 0.5 on two different datasets does not imply equal performance). It should only be used to compare the relative performance of two models on the same dataset. Tables 4 and 5 show that ORTHRUS outperforms all other baselines in both MCC and precision.

These results could be attributed to poor threshold selection by state-of-the-art systems. However, Fig. 4 and our analysis suggest a different conclusion. Fig. 4 shows that previous systems struggle to distinguish between malicious and benign nodes, often assigning them similar anomaly scores. In contrast, ORTHRUS is capable of clearly identifying a few malicious nodes as outliers (see Fig. 4a). Further analysis reveals that ThreaTrace, Flash, and MAGIC tend to flag large areas of the graph containing and surrounding anomalies, creating an unnecessary volume of data for analysts. SIGL and Kairos flag nodes in the vicinity of an attack but not necessarily within it. In particular, Kairos identifies nodes occurring concurrently with attacks, aligning with their evaluation strategy. This supports our hypothesis that the choice of evaluation approach contributes to system report quality.

## 5.3 Computational Efficiency

When evaluating the practicality of a PIDS, training runtime and memory consumption are crucial metrics [20]. Figures 5, 6 and 7 show the efficiency differences between the baselines. **ORTHRUS** often outperforms the other baselines in both runtime and GPU memory usage due to its lightweight architecture designed to avoid costly operations.

In contrast, many other models rely on resource-intensive architectures that negatively impact training time, testing time, and memory consumption. For instance, **Kairos** exhibits longer training time (see Tables 10 and 11) due to its




(a) ORTHRUS. (b) Threatrace. (c) Kairos.

(d) SIGL. (e) Magic. (f) Flash.

Figure 4: Anomaly detection systems distinguish between benign and anomalous elements by attributing a reconstruction error to each element.A benign element's reconstruction error should be close to zero, while an anomalous element's error should be large. The discriminative power of an anomaly detection system is defined by its ability to clearly separate benign and malicious categories. This figures display node reconstruction errors on the E3-CADETS dataset, with benign elements in green and malicious ones in red. Larger gaps between anomalous and benign nodes indicates better systems. Red circles indicate elements with the highest anomaly score within each attack.

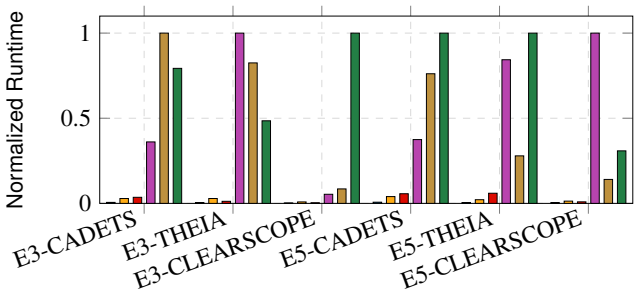



Figure 5: Comparison of normalized training time (the lower the better) between Orthrus, Kairos, Threatrace, SIGL, MAGIC, and Flash. Normalization is performed within each dataset by dividing each system's training time by the maximum training time for that dataset (raw data in Appendix F).

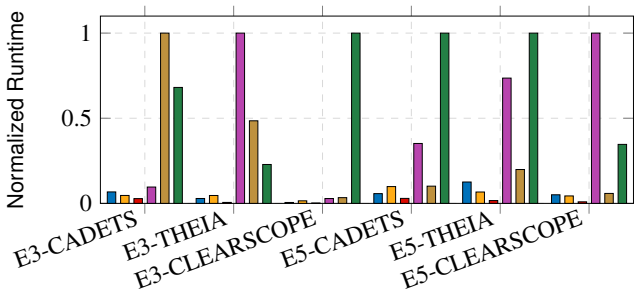



Figure 6: Comparison of normalized testing times (the lower the better) between Orthrus, Kairos, Threatrace, SIGL, MAGIC, and Flash. Normalization is performed within each dataset by dividing each system's testing time by the maximum testing time for that dataset (raw data in Appendix F).

use of RNNs to update node memory vectors—an operation that is computationally expensive and difficult to parallelize on GPUs. Similarly, **SIGL** applies RNN operations to every node within each time window, leading to high training times. The use of Graph Attention Networks [64] by **MAGIC** in both the encoder and decoder results in high computational complexity due to the need for calculating attention weights. Likewise, **Flash** requires substantial memory as it maintains feature vector lists for every edge for positional encoding, limiting its training to small batches. Finally, although **ThreaTrace** achieves the best testing time performance, it continues training until a True-Negative-based metric meets a specified threshold, requiring a high number of training epochs and thus extending training time.

## 5.4 Hyperparameters Study

Hyperparameters are critical to the success of any machine learning model. Fig. 8 illustrates the impact of different hyperparameters on the number of detected attacks, including a breakdown of true positives (TPs) and false positives (FPs). The overall detection performance is summarized by the MCC score, and the evolution of memory consumption is also shown.

**Node Dropout:** Regularization through node dropout significantly enhances detection performance during ORTHRUS' training. While most dropout rates yield similar results, we observe an increase in true positives when the dropout rate is set to 25%.

**Learning Rate:** Choosing the right learning rate is essential for smooth training. Our experiments show that a learning

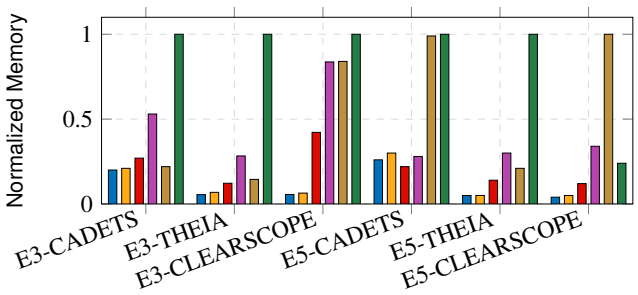



Figure 7: Comparison of normalized memory usage (the lower the better) between Orthrus, Kairos, Threatrace, SIGL, MAGIC, and Flash. Normalization is performed within each dataset by dividing each system's GPU memory footprint by the maximum GPU memory footprint for that dataset (raw data in Appendix F).

rate of $lr = 1e-4$ delivers optimal results across datasets.

**Word2vec Embedding Size ($x$):** The Word2vec embedding size must be large enough to capture the semantics of tokens representing file paths, command lines, or IP addresses, without being so large that it hinders generalization. For example, embedding sizes of $x = 32$ or $x = 64$ result in 10 true positives with no false positives, while increasing the size to $x = 128$ or $x = 256$ decreases true positives and increases false positives. This highlights the critical role of the Word2vec embedding. Additionally, we observe a significant increase in memory usage as the embedding size increases.

**GNN Embedding Size ($z$):** The GNN embedding size is essential for capturing the structural and temporal features of the graph through message passing. A size of $z = 32$ performs well on small datasets, while $z = 64$ typically yields better results on larger datasets.

**Neighborhood Size ($\mathcal{N}$):** The neighborhood size controls the amount of information aggregated during each message-passing step. A small neighborhood (e.g., $\mathcal{N} = 5$) detects 0 TP, while larger values ($10 \leq \mathcal{N} \leq 50$) detect between 6 and 10 TPs.

These experiments demonstrate that ORTHRUS can detect all three attacks in this dataset in most cases, even with significant changes in hyperparameter values. This highlights ORTHRUS' robust detection capability, which is not heavily reliant on hyperparameter tuning. Our findings suggest that these hyperparameter trends are consistent across datasets, enabling ORTHRUS to effectively detect attacks on diverse datasets with minimal tuning.

## 5.5 Ablation Study

We conduct an ablation study to evaluate the individual contributions of each component in ORTHRUS. We systematically replace or remove one component at a time. The specific ablations are detailed in Table 6, with the results of these

| Component | With Component (✓) | Without Component (✗) |
|---|---|---|
| **Featurization** | ORTHRUS' Word2vec embedding (§4.2) | Hierarchical hashing as in Kairos |
| **Encoding** | ORTHRUS' encoder (§4.3) | Kairos' TGN encoder |
| **Clustering** | ORTHRUS' anomaly detection algorithm (§4.4) | Automatic anomaly thresholding only |
| **Reconstruction** | ORTHRUS' attack reconstruction algorithm (§4.5) | No tracing algorithm used |

Table 6: Description of the ablations performed.

| Dataset | Featurization | Encoding | Clustering | Reconstruction | TP | FP | Precision | Memory |
|---|---|---|---|---|---|---|---|---|
| E3-THEIA | ✗ | ✓ | ✓ | ✓ | 51 | 13 | 0.79 | 2.03GB |
| | ✓ | ✗ | ✓ | ✓ | 41 | 772 | 0.05 | 5.75GB |
| | ✓ | ✓ | ✗ | ✓ | 48 | 11 | 0.81 | 2.03GB |
| | ✓ | ✓ | ✓ | ✗ | 8 | 0 | 1.00 | 2.03GB |
| | ✓ | ✓ | ✓ | ✓ | 48 | 11 | 0.81 | 2.03GB |
| E5-THEIA | ✗ | ✓ | ✓ | ✓ | 0 | 155 | 0.00 | 4.23GB |
| | ✓ | ✗ | ✓ | ✓ | 13 | 53 | 0.20 | 11.10GB |
| | ✓ | ✓ | ✗ | ✓ | 20 | 11,420 | 0.00 | 4.23GB |
| | ✓ | ✓ | ✓ | ✗ | 2 | 0 | 1.00 | 4.23GB |
| | ✓ | ✓ | ✓ | ✓ | 13 | 2 | 0.87 | 4.23GB |

Table 7: Ablation results. The darker the precision score, the more important a component is.

experiments on the E3-THEIA and E5-THEIA datasets presented in Table 7.

First, replacing ORTHRUS' Word2vec *Featurization* component has minimal impact on the smaller E3-THEIA dataset but is essential for E5-THEIA, which features larger and denser provenance graphs. Similarly, removing ORTHRUS' *Clustering* component has no effect on E3-THEIA but is crucial for E5-THEIA, where standalone thresholding detects only two true positives. Substituting ORTHRUS' encoder with the TGN encoder from Kairos negatively affects performance on both datasets, as seen by the significant increase in false positives (this corroborates the lower detection quality of Kairos seen in Fig. 4). Additionally, using a memory vector for each node with TGN results in higher memory consumption. Lastly, while the *Reconstruction* component in ORTHRUS-full generates more false positives than ORTHRUS-ano, it helps analysts by identifying significantly more true positives. The use of this component remains optional, depending on the specific needs of the security analyst.

## 5.6 Robustness Against Mimicry Attacks

To evaluate ORTHRUS' robustness against adversarial attacks, we conducted adversarial mimicry attacks on PIDSs, following the methodology outlined by Goyal et al. [24]. Their research highlights the vulnerability of granular PIDSes, such as StreamSpot [44], Unicorn [27], ProvDetector [65], and SIGL [29], to adversarial attacks. This strategy involves manipulating the distributional graph encoding to execute an evasion attack on PIDSs, aiming to create deceptive similarities between the node neighborhood distributions in the attack graph and those in benign provenance graphs. Following on previous work [18, 57] footsteps, we use the publicly available code from Goyal et al. to manipulate the E3-CADETS dataset. We show the effect it has on ORTHRUS's detection

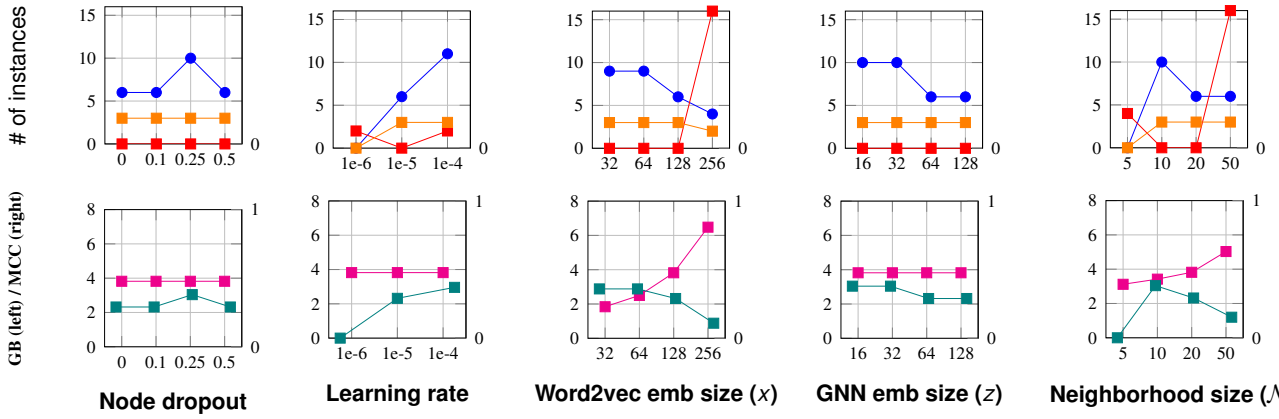



Figure 8: Hyperparameters study of ORTHRUS on E3-CADETS. We show TP, FP, detected attacks, memory consumption (GB), and MCC.

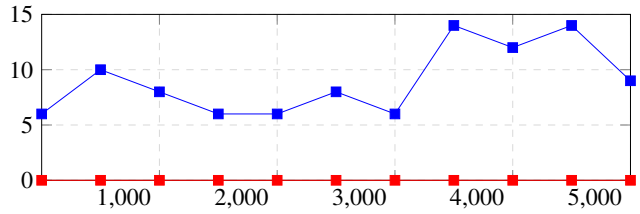



Figure 9: Effect of adversary mimicry attacks [24] against our system. We show the number of edges added to the attack graph (x-axis) and the number of TP and FP (y-axis).

performance in Fig. 9. ORTHRUS proves robust against such manipulation. This confirms recent findings from Cheng et al. [18] and Rehman et al. [57] that more complex GNN-based detection strategies are resistant to such attacks.

## 6 Discussion, Limitations & Future Work

**Open-source Availability.** While the literature presents a wealth of innovative PIDSs, not all related code has been publicly released, often for legitimate reasons (e.g., ShadeWatcher [73] relies on proprietary software). Sadly, re-implementing a system based solely on the descriptions provided in a paper is often challenging, limiting the scope of evaluations that can be conducted. We strongly encourage the community to adopt open-sourcing as the default practice. In line with this, we have made our own artifacts publicly available (see §9).

**Analysis of False Positives.** On certain datasets, ORTHRUS produces false positives. Examining their positions in the graph reveals patterns that inform potential improvements. We observe that false positives frequently occur within the one-hop neighborhood of malicious nodes. For example, in E5-CLEARSCOPE, 5 of the 7 false positives detected by ORTHRUS-ano are netflow nodes sharing the same IP address as attack nodes.[4] The two others are temporally distant and lack direct connections. The false positives within the surrounding of attacks arise because GNN message passing propagates malicious signals to neighboring nodes via outgoing edges, a phenomenon previously noted in THREATRACE [66]. A common mitigation is to adopt an evaluation strategy that labels nodes within the two-hop neighborhood of malicious nodes as malicious. However, as discussed in §2, this is simply ignoring the core issue and directly leads to systems with reduced attribution quality. Addressing this limitation through better false-positive mitigation strategies remains an open and critical research direction.

**Impact of the Capture Mechanism.** In Table 5, we observe poor performance of ORTHRUS and other systems on E5-CADETS. Several factors may explain this: (1) each capture system (CADETS, THEIA, CLEARSCOPE) represents system execution differently and reports varying information (e.g., CADETS omits the paths of most files and processes, while CLEARSCOPE lacks process paths and remote addresses of netflows); (2) the attacks in each dataset differ, with some being more stealthy than others; and (3) some graph elements appear to be missing, possibly due to crashes reported in DARPA's documentation or flaws in the capture mechanism, as identified by Sekar et al. [59]. Understanding the influence of capture mechanisms on downstream intrusion detection performance is an important research question, beyond the scope of this paper.

**The Importance of Training Time.** Discussions with colleagues in industry have emphasized the importance of regularly retraining intrusion detection models to address concept drift. As new software is deployed or updates are rolled out,

[4] This could be benign activity from the attacker machine and/or attack steps not appearing in the DARPA textual ground truth (see Appendix C).

the number of false positives tends to increase, necessitating retraining. While machine learning techniques exist to manage these issues, they have not been thoroughly explored in the context of PIDS and cannot be readily applied [18].

As shown in §5.3, ORTHRUS is the fastest PIDS among the systems we evaluated, making it well-suited for regular retraining. However, a comprehensive evaluation of this problem, along with various approaches to address it, remains difficult due to the lack of dedicated datasets [18]. We leave such exploration for future work.

## 7 Related Work

For a comprehensive overview of provenance-based security, we refer readers to the SoK by Inam et al. [35].

**Anomaly Detection.** Early anomaly-based works [27, 44, 65] identify and report *coarse-grained* anomalies by clustering statistical graph summaries. More recent approaches [18, 29, 36, 66, 73] have leveraged advances in learning graph-structured data to model fine-grained, contextualized relationships between *nodes* in provenance graphs. ThreaTrace [66] and Flash [57] train GraphSAGE [26] and GNN models, respectively, to predict node types and report misclassified nodes as anomalous. ThreaTrace removes correctly predicted nodes and re-trains on misclassified nodes until none remain. Flash incorporates positional encoding from Transformers [63] into the node embedding step to capture temporal information from the sequential order of tokens. MAGIC [36] leverages a masked auto-encoder, utilizing a Graph Attention Network [64] in both the encoder and decoder to embed node features, identifying anomalies using a K-D tree search technique. Kairos [18] models long-term temporal relationships of system entities by assigning them state vectors using Temporal Graph Networks (TGNs) [58], then detects malicious time windows when their scores exceed a specified threshold. Similarly, SIGL [29] captures temporal dependencies of entities using a Graph Long Short-Term Memory (Graph LSTM) [55] model to detect malicious software installations. R-CAID [25] combines GNN and Root Cause Analysis (RCA) techniques to learn global causal relationships between nodes and their root causes. While state-of-the-art methods have made significant progress in fine-grained anomaly detection, they often overlook the attribution quality of detection signals. ORTHRUS is the first to achieve meaningful node-level detection without overwhelming false positives.

**Attack Investigation.** Attack investigation is the oldest application of provenance in security [39]. OmegaLog [34] and ALchemist [71] combine system-layer provenance data with higher-layer application logs to provide more semantic context for event investigation. UIScope [70] focuses on reconstructing Graphical User Interface-related attacks by correlating provenance data with User Interface events. Recent alert investigation systems based purely on provenance data, such as DEPIMPACT [22], NoDoze [31], Swift [33], and ProTracker [42], generate alerts by assigning scores to events based on their causal context. NoDoze and Swift assign anomaly scores to each event in the provenance graph based on the frequency of related events. ProTracker computes priority scores for system events based on their rarity and fanout, while DEPIMPACT calculates edge scores using multiple features, including timing, data flow amount, and node degree. They then aggregate the scores along neighboring edges to reconstruct the attack graph. ORTHRUS tightly integrates anomaly detection and attack investigation into a single system.

## 8 Conclusion

We identified limitations in the evaluation strategy of provenance-based intrusion detection systems, showing that this has led to the design of systems with poor *attribution quality*, which directly contributes to security analysts' alert fatigue. We proposed an alternative, conservative evaluation strategy that favors systems generating fewer alerts. We then designed a lightweight anomaly detection pipeline, ORTHRUS, and demonstrated its superiority over state-of-the-art systems on well-established benchmark datasets.

## 9 Compliance with the Open Science Policy

**Datasets Availability.** The DARPA datasets are publicly available [8, 9] and come with textual description of the attacks. We manually analyzed the datasets and annotated nodes (as benign or part of an attack). The annotations are publicly available at https://github.com/ubc-provenance/ground-truth. To the best of our knowledge this is the first time such annotations are publicly released.

**Software Artifacts Availability.** The source code is publicly available at https://github.com/ubc-provenance/orthrus. We also provide detailed instructions to reproduce the results presented in §5.

## 10 Ethics Considerations

To the best of our knowledge this work does not raise any ethical issues. All experiments have been performed on publicly available datasets that have been acquired in an ethical manners and not contain any sensitive information.

## Acknowledgments

We would like to thank USENIX Security 2025 reviewers from their helpful feedback. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC). Nous remercions le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG) de son soutien. We thank Shreya Gangopadhyay for her assistance

in re-implementing SIGL. Her contribution was supported by the Mitacs Globalink Research Internships program. This work was partially supported by research funding from the National Research Council Canada (NRC). Research reported in this publication was partially supported by an Amazon Research Award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of Amazon.

## References

[1] Event Tracing for Windows (ETW), Accessed 14th January 2025. https://docs.microsoft.com/en-us/windows-hardware/drivers/devtest/event-tracing-for-windows--etw-.

[2] Flash code, Accessed 14th January 2025. https://github.com/DART-Laboratory/Flash-IDS.

[3] Kairos code, Accessed 14th January 2025. https://github.com/ProvenanceAnalytics/kairos.

[4] Linux System Audit (auditd), Accessed 14th January 2025. https://man7.org/linux/man-pages/man8/auditd.8.html.

[5] MAGIC code, Accessed 14th January 2025. https://github.com/FDUDSDE/MAGIC.

[6] ThreaTrace code, Accessed 14th January 2025. https://github.com/threaTrace-detector/threaTrace.

[7] Transparent Computing, Accessed 14th January 2025. https://www.darpa.mil/program/transparent-computing.

[8] Transparent Computing Engagement 3 Data Release, Accessed 14th January 2025. https://github.com/darpa-i2o/Transparent-Computing/blob/master/README-E3.md.

[9] Transparent Computing Engagement 5 Data Release, Accessed 14th January 2025. https://github.com/darpa-i2o/Transparent-Computing.

[10] Jaime D. Acevedo-Viloria, Luisa Roa, Soji Adeshina, Cesar Charalla Olazo, Andrés Rodríguez-Rey, Jose Alberto Ramos, and Alejandro Correa Bahnsen. Relational Graph Neural Networks for Fraud Detection in a Super-App environment. In *KDD Workshop on Machine Learning in Finance*. ACM, 2023.

[11] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph-based Anomaly Detection and Description: A Survey. *Data Mining and Knowledge Discovery*, 2015.

[12] Bushra A Alahmadi, Louise Axon, and Ivan Martinovic. 99% False Positives: A Qualitative Study of SOC Analysts' Perspectives on Security Alarms. In *Security Symposium (USENIX Sec'22)*. USENIX, 2022.

[13] Abdulellah Alsaheel, Yuhong Nan, Shiqing Ma, Le Yu, Gregory Walkup, Z Berkay Celik, Xiangyu Zhang, and Dongyan Xu. ATLAS: A Sequence-based Learning Approach for Attack Investigation. In *Security Symposium (USENIX Sec'21)*. USENIX, 2021.

[14] Enes Altinisik, Fatih Deniz, and Hüsrev Taha Sencar. ProvG-Searcher: A Graph Representation Learning Approach for Efficient Provenance Graph Search. In *Conference on Computer and Communications Security (CCS'23)*. ACM, 2023.

[15] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. Dos and Don'ts of Machine Learning in Computer Security. In *Security Symposium (USENIX Sec'22)*. USENIX, 2022.

[16] Adam Bates, Dave Jing Tian, Kevin RB Butler, and Thomas Moyer. Trustworthy Whole-System Provenance for the Linux Kernel. In *Security Symposium (USENIX Sec'15)*. USENIX, 2015.

[17] Zijun Cheng, Qiujian Lv, Jinyuan Liang, Yan Wang, Degang Sun, Thomas Pasquier, and Xueyuan Han. KAIROS: Practical Intrusion Detection and Investigation using Whole-system Provenance (Supplementary Material), 2023. https://tfjmp.org/publications/2024-sp-supp.pdf.

[18] Zijun Cheng, Qiujian Lv, Jinyuan Liang, Yan Wang, Degang Sun, Thomas Pasquier, and Xueyuan Han. Kairos: Practical Intrusion Detection and Investigation using Whole-system Provenance. In *Symposium on Security and Privacy (S&P'24)*. IEEE, 2024.

[19] Kaize Ding, Jundong Li, Nitin Agarwal, and Huan Liu. Inductive Anomaly Detection on Attributed Networks. In *International Joint Conference on Artificial Intelligence (IJCAI'20)*, 2020.

[20] Feng Dong, Shaofei Li, Peng Jiang, Ding Li, Haoyu Wang, Liangyi Huang, Xusheng Xiao, Jiedong Chen, Xiapu Luo, Yao Guo, and Xiangqun Chen. Are we there yet? An Industrial Viewpoint on Provenance-based Endpoint Detection and Response Tools. In *Conference on Computer and Communications Security (CCS'23)*. ACM, 2023.

[21] Dongsheng Duan, Lingling Tong, Yangxi Li, Jie Lu, Lei Shi, and Cheng Zhang. AANE: Anomaly Aware Network Embedding For Anomalous Link Detection. In

*International Conference on Data Mining (ICDM'20)*. IEEE, 2020.

[22] Pengcheng Fang, Peng Gao, Changlin Liu, Erman Ayday, Kangkook Jee, Ting Wang, Yanfang Fanny Ye, Zhuotao Liu, and Xusheng Xiao. Back-Propagating System Dependency Impact for Attack Investigation. In *Security Symposium (USENIX Sec'22)*. USENIX, 2022.

[23] Ashish Gehani and Dawood Tariq. SPADE: Support for Provenance Auditing in Distributed Environments. In *International Conference on Distributed Systems Platforms and Open Distributed Processing*. ACM/IFIP/USENIX, 2012.

[24] Akul Goyal, Xueyuan Han, Gang Wang, and Adam Bates. Sometimes, You Aren't What You Do: Mimicry Attacks against Provenance Graph Host Intrusion Detection Systems. In *Network and Distributed System Security Symposium, (NDSS'23)*. The Internet Society, 2023.

[25] Akul Goyal, Gang Wang, and Adam Bates. R-CAID: Embedding Root Cause Analysis within Provenance-based Intrusion Detection. In *Symposium on Security and Privacy (S&P'24)*. IEEE, 2024.

[26] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. In *Annual Conference on Neural Information Processing Systems (NeurIPS'17)*, 2017.

[27] Xueyuan Han, Thomas Pasquier, Adam Bates, James Mickens, and Margo I. Seltzer. Unicorn: Runtime Provenance-Based Detector for Advanced Persistent Threats. In *Network and Distributed System Security Symposium (NDSS'20)*. The Internet Society, 2020.

[28] Xueyuan Han, Thomas Pasquier, Tanvi Ranjan, Mark Goldstein, and Margo I. Seltzer. FRAPpuccino: Fault-detection through Runtime Analysis of Provenance. In *Workshop on Hot Topics in Cloud Computing (HotCloud'17)*. USENIX, 2017.

[29] Xueyuan Han, Xiao Yu, Thomas Pasquier, Ding Li, Junghwan Rhee, James W. Mickens, Margo I. Seltzer, and Haifeng Chen. SIGL: Securing Software Installations Through Deep Graph Learning. In *Security Symposium (USENIX Sec'21)*. USENIX, 2021.

[30] John A Hartigan and Manchek A Wong. Algorithm AS 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 1979.

[31] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. NoDoze: Combatting Threat Alert Fatigue with Automated Provenance Triage. In *Network and Distributed System Security Symposium (NDSS'19)*. The Internet Society, 2019.

[32] Wajih Ul Hassan, Mark Lemay, Nuraini Aguse, Adam Bates, and Thomas Moyer. Towards Scalable Cluster Auditing through Grammatical Inference over Provenance Graphs. In *Network and Distributed System Security Symposium (NDSS'18)*. The Internet Society, 2018.

[33] Wajih Ul Hassan, Ding Li, Kangkook Jee, Xiao Yu, Kexuan Zou, Dawei Wang, Zhengzhang Chen, Zhichun Li, Junghwan Rhee, Jiaping Gui, and Adam Bates. This is Why We Can't Cache Nice Things: Lightning-Fast Threat Hunting using Suspicion-Based Hierarchical Storage. In *Annual Computer Security Applications Conference (ACSAC'20)*. ACM, 2020.

[34] Wajih Ul Hassan, Mohammad Ali Noureddine, Pubali Datta, and Adam Bates. Omegalog: High-fidelity attack investigation via transparent multi-layer log analysis. In *Network and Distributed System Security Symposium (NDSS'20)*. The Internet Society, 2020.

[35] Muhammad Adil Inam, Yinfang Chen, Akul Goyal, Jason Liu, Jason Mink, Noor Michael, Sneha Gaur, Adam Bates, and Wajih Ul Hassan. SoK: History is a Vast Early Warning System: Auditing the Provenance of System Intrusions. In *Symposium on Security and Privacy (S&P'22)*. IEEE, 2022.

[36] Zian Jia, Yun Xiong, Yuhong Nan, Yao Zhang, Jinjing Zhao, and Mi Wen. MAGIC: Detecting Advanced Persistent Threats via Masked Graph Representation Learning. In *Security Symposium (USENIX Sec'24)*. USENIX, 2024.

[37] Maya Kapoor, Joshua Melton, Michael Ridenhour, Siddharth Krishnan, and Thomas Moyer. PROV-GEM: Automated provenance analysis framework using graph embeddings. In *International Conference on Machine Learning and Applications (ICMLA'21)*. IEEE, 2021.

[38] Isaiah J King, Xiaokui Shu, Jiyong Jang, Kevin Eykholt, Taesung Lee, and H Howie Huang. EdgeTorrent: Real-time Temporal Graph Representations for Intrusion Detection. In *International Symposium on Research in Attacks, Intrusions and Defenses (RAID'23)*. ACM, 2023.

[39] Samuel T King and Peter M Chen. Backtracking intrusions. In *Symposium on Operating systems Principles (SOSP'03)*. ACM, 2003.

[40] Shaofei Li, Feng Dong, Xusheng Xiao, Haoyu Wang, Fei Shao, Jiedong Chen, Yao Guo, Xiangqun Chen, and Ding Li. NODLINK: An Online System for Fine-Grained APT Attack Detection and Investigation. In

*Network and Distributed System Security Symposium, (NDSS'24)*. The Internet Society, 2024.

[41] Zhenyuan Li, Qi Alfred Chen, Runqing Yang, Yan Chen, and Wei Ruan. Threat detection and investigation with system-level provenance graphs: A survey. *Elsevier Computers & Security*, 2021.

[42] Yushan Liu, Mu Zhang, Ding Li, Kangkook Jee, Zhichun Li, Zhenyu Wu, Junghwan Rhee, and Prateek Mittal. Towards a timely causality analysis for enterprise security. In *Network and Distributed System Security Symposium (NDSS'18)*. The Internet Society, 2018.

[43] Xiaoxiao Ma, Jia Wu, Shan Xue, Jian Yang, Chuan Zhou, Quan Z Sheng, Hui Xiong, and Leman Akoglu. A Comprehensive Survey on Graph Anomaly Detection With Deep Learning. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[44] Emaad Manzoor, Sadegh Momeni, Venkat Venkatakrishnan, and Leman Akoglu. Fast Memory-efficient Anomaly Detection in Streaming Heterogeneous Graphs. In *International Conference on Knowledge Discovery and Data Mining (KDD'16)*. ACM, 2016.

[45] Brian W Matthews. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Elsevier Biochimica et Biophysica Acta (BBA)-Protein Structure*, 1975.

[46] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781*, 2013.

[47] Sadegh M. Milajerdi, Birhanu Eshete, Rigel Gjomemo, and V. N. Venkatakrishnan. POIROT: Aligning Attack Behavior with Kernel Audit Records for Cyber Threat Hunting. In *Conference on Computer and Communications Security (CCS'19)*. ACM, 2019.

[48] Sadegh Momeni Milajerdi, Rigel Gjomemo, Birhanu Eshete, R. Sekar, and V. N. Venkatakrishnan. HOLMES: real-time APT detection through correlation of suspicious information flows. In *Symposium on Security and Privacy, (S&P'19)*. IEEE, 2019.

[49] Kiran-Kumar Muniswamy-Reddy, David Holland, Uri Braun, and Margo Seltzer. Provenance-aware Storage Systems. In *Annual Technical Conference (ATC'06)*. USENIX, 2006.

[50] Riccardo Paccagnella, Pubali Datta, Wajih Ul Hassan, Adam Bates, Christopher W. Fletcher, Andrew Miller, and Dave Tian. Custos: Practical Tamper-Evident Auditing of Operating Systems Using Trusted Execution. In *Network and Distributed System Security Symposium (NDSS'20)*. The Internet Society, 2020.

[51] Riccardo Paccagnella, Kevin Liao, Dave Tian, and Adam Bates. Logging to the Danger Zone: Race Condition Attacks and Defenses on System Audit Frameworks. In *Conference on Computer and Communications (CCS'20)*. ACM, 2020.

[52] Thomas Pasquier, Xueyuan Han, Mark Goldstein, Thomas Moyer, David M. Eyers, Margo I. Seltzer, and Jean Bacon. Practical Whole-system Provenance Capture. In *Symposium on Cloud Computing (SoCC'17)*. ACM, 2017.

[53] Thomas Pasquier, Xueyuan Han, Thomas Moyer, Adam Bates, Olivier Hermant, David M. Eyers, Jean Bacon, and Margo Seltzer. Runtime Analysis of Whole-System Provenance. In *Conference on Computer and Communications Security (CCS'18)*. ACM, 2018.

[54] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time. In *Security Symposium (USENIX Sec'19)*. USENIX, 2019.

[55] Nanyun Peng, Hoifung Poon, Chris Quirk, Kristina Toutanova, and Wen-tau Yih. Cross-Sentence *N*-ary Relation Extraction with Graph LSTMs. *MIT Press Transactions of the Association for Computational Linguistics*, 2017.

[56] David MW Powers. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061*, 2020.

[57] Mati Ur Rehman, Hadi Ahmadi, and Wajih Ul Hassan. FLASH: A Comprehensive Approach to Intrusion Detection via Provenance Graph Representation Learning. In *Symposium on Security and Privacy (S&P'24)*. IEEE, 2024.

[58] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal Graph Networks for Deep Learning on Dynamic Graphs. In *International Conference on Machine Learning (ICML'20)*. PMLR, 2020.

[59] R Sekar, Hanke Kimm, and Rohit Aich. eAudit: A fast, Scalable and Deployable Audit Data Collection System. In *Symposium on Security and Privacy (S&P'24)*. IEEE, 2023.

[60] Yunsheng Shi, Zhengjie Huang, Wenjin Wang, Hui Zhong, Shikun Feng, and Yu Sun. Masked Label Prediction: Unified Massage Passing Model for Semi-Supervised Classification. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2021.

[61] Sathya Chandran Sundaramurthy, Alexandru G Bardas, Jacob Case, Xinming Ou, Michael Wesch, John McHugh, and S Raj Rajagopalan. A Human Capital Model for Mitigating Security Analyst Burnout. In *Symposium On Usable Privacy and Security (SOUPS'15)*. USENIX, 2015.

[62] Jianheng Tang, Jiajin Li, Ziqi Gao, and Jia Li. Rethinking Graph Neural Networks for Anomaly Detection. In *International Conference on Machine Learning (ICML'22)*. PMLR, 2022.

[63] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *Advances in Neural Information Processing Systems (NeurIPS'17)*, 2017.

[64] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph Attention Networks. In *International Conference on Learning Representations (ICLR'18)*, 2018.

[65] Qi Wang, Wajih Ul Hassan, Ding Li, Kangkook Jee, Xiao Yu, Kexuan Zou, Junghwan Rhee, Zhengzhang Chen, Wei Cheng, Carl A. Gunter, and Haifeng Chen. You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis. In *Network and Distributed System Security Symposium (NDSS'20)*. The Internet Society, 2020.

[66] Su Wang, Zhiliang Wang, Tao Zhou, Xia Yin, Dongqi Han, Han Zhang, Hongbin Sun, Xingang Shi, and Jiahai Yang. THREATRACE: Detecting and Tracing Host-Based Threats in Node Level Through Provenance Graph Learning. *IEEE Transactions on Information Forensics and Security*, 2022.

[67] Ziyu Wang, Nanqing Luo, and Pan Zhou. GuardHealth: Blockchain Empowered Secure Data Management and Graph Convolutional Network Enabled Anomaly Detection in Smart Healthcare. *Journal of Parallel and Distributed Computing*, 2020.

[68] Zhang Xu, Zhenyu Wu, Zhichun Li, Kangkook Jee, Junghwan Rhee, Xusheng Xiao, Fengyuan Xu, Haining Wang, and Guofei Jiang. High Fidelity Data Reduction for Big Data Security Dependency Analyses. In *Conference on Computer and Communications Security (CCS'16)*. ACM, 2016.

[69] Fan Yang, Jiacen Xu, Chunlin Xiong, Zhou Li, and Kehuan Zhang. PROGRAPHER: An Anomaly Detection System based on Provenance Graph Embedding. In *Security Symposium (USENIX Sec'23)*. USENIX, 2023.

[70] Runqing Yang, Shiqing Ma, Haitao Xu, Xiangyu Zhang, and Yan Chen. UIScope: Accurate, Instrumentation-free, and Visible Attack Investigation for GUI Applications. In *Network and Distributed System Security Symposium (NDSS'20)*. The Internet Society, 2020.

[71] Le Yu, Shiqing Ma, Zhuo Zhang, Guanhong Tao, Xiangyu Zhang, Dongyan Xu, Vincent E Urias, Han Wei Lin, Gabriela F Ciocarlie, Vinod Yegneswaran, and Ashish Gehani. ALchemist: Fusing Application and Audit Logs for Precise Attack Provenance without Instrumentation. In *Network and Distributed System Security Symposium (NDSS'21)*. The Internet Society, 2021.

[72] Jun Zeng, Zheng Leong Chua, Yinfang Chen, Kaihang Ji, Zhenkai Liang, and Jian Mao. WATSON: Abstracting Behaviors from Audit Logs via Aggregation of Contextual Semantics. In *Annual Network and Distributed System Security Symposium (NDSS'21)*. Internet Society, 2021.

[73] Jun Zengy, Xiang Wang, Jiahao Liu, Yinfang Chen, Zhenkai Liang, Tat-Seng Chua, and Zheng Leong Chua. ShadeWatcher: Recommendation-guided Cyber Threat Analysis using System Audit Records. In *Symposium on Security and Privacy (S&P'22)*. IEEE, 2022.

[74] Ge Zhang, Zhao Li, Jiaming Huang, Jia Wu, Chuan Zhou, Jian Yang, and Jianliang Gao. eFraudCom: An E-commerce Fraud Detection System via Competitive Graph Neural Networks. *ACM Transactions on Information Systems*, 2022.

[75] Zhaoqi Zhang, Panpan Qi, and Wei Wang. Dynamic malware analysis with feature engineering and feature learning. *AAAI Conference on Artificial Intelligence*, 2021.

[76] Michael Zipperle, Florian Gottwalt, Elizabeth Chang, and Tharam Dillon. Provenance-based intrusion detection systems: A survey. *ACM Computing Surveys (CSUR)*, 2022.

## A Datasets Details

We summarize in Table 8 how we split the datasets for training, validation, and detection.

## B Past Evaluations

Table 9 shows anomaly detection systems we select to compare against ORTHRUS. We use the same node-level detection performance with the same evaluation approach to ensure a fair comparison. Based on our knowledge, this is the most comprehensive comparison to date.

| Datasets | Training Data (yyyy-mm-dd) | Validation Data (yyyy-mm-dd) | Test Data (yyyy-mm-dd) |
|---|---|---|---|
| E3-CADETS | 2018-04-03/04/05/07/08/09/10 | 2018-04-02 | **2018-04-06**<br>2018-04-11<br>**2018-04-12**<br>**2018-04-13** |
| E3-THEIA | 2018-04-02/03/04/05/06/07/08 | 2018-04-09 | **2018-04-10**<br>**2018-04-12**<br>2018-04-13 |
| E3-CLEARSCOPE | 2018-04-03/04/05/07/08/09/10 | 2018-04-02 | **2018-04-11**<br>2018-04-12 |
| E5-CADETS | 2019-05-08/09/11 | 2019-05-12 | **2019-05-16**<br>**2019-05-17** |
| E5-THEIA | 2019-05-08/09/10 | 2019-05-11 | 2019-05-14<br>**2019-05-15** |
| E5-CLEARSCOPE | 2019-05-08/09 | 2019-05-11 | 2019-05-14<br>**2019-05-15**<br>**2019-05-17** |

Table 8: DARPA data used for training, validation, and test. **Bold** days refer to as attack days in which both benign and attack activities exist. The remaining days are benign days with only benign activities.

| | | ThreaTrace [66] | SIGL [29] | Kairos [18] | MAGIC [36] | Flash [57] | Orthrus |
|---|---|---|---|---|---|---|---|
| **2016** | **Steamspot** [44] | ✓ | ✓ | | ✓ | | |
| **2017** | **Frappuccino** [28] | | ✓ | | | | |
| **2020** | **Unicorn** [27] | ✓ | | ✓ | ✓ | ✓ | |
| | **ProvDetector** [65] | ✓ | | | | | |
| **2021** | **Prov-Gem** [37] | | | | ✓ | | |
| **2022** | **SIGL** [29] | | | | | | ✓ |
| | **ThreaTrace** [66] | | | ✓ | ✓ | ✓ | ✓ |
| **2024** | **Kairos** [18] | | | | | | ✓ |
| | **MAGIC** [36] | | | | | | ✓ |
| | **Flash** [57] | | | | | | ✓ |

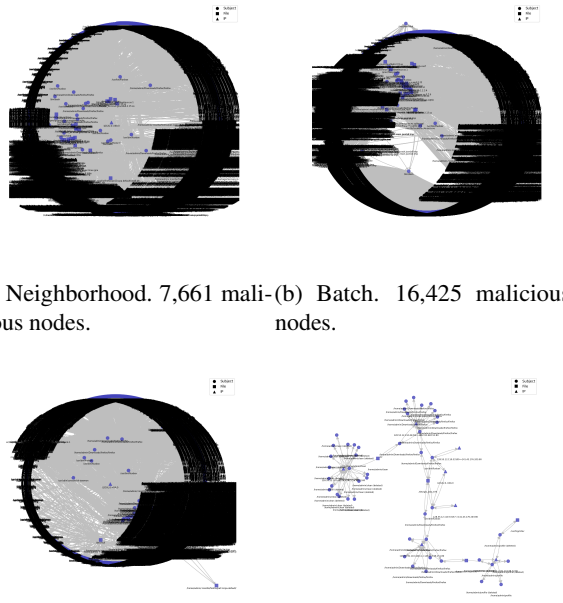
Table 9: Comparison of baselines used in past publications. Horizontal – systems used as baselines in past evaluations. Vertical – ORTHRUS and past publications used as baselines in this paper.

## C Ground Truth Construction

We constructed our ground truth by loading the DARPA datasets into a database. Using the textual ground truth descriptions provided by the DARPA team [8, 9], we queried nodes based on name and timestamps. The results were manually analyzed to remove incorrect matches (e.g., excluding instances of software unrelated to the attacks when multiple instances were running).

The textual descriptions occasionally omitted nodes, causing attacks to be fragmented into multiple subgraphs. To address this, we retrieved the shortest paths connecting nodes across these subgraphs, merging them into a single, larger attack graph. Finally, we manually reviewed the results again, removing any erroneous nodes or edges.

When uncertain about a graph element, we erred on the side of caution and excluded it (this occured only for a few nodes on two datasets). In the worst-case scenario, this may lead to underestimating a system's detection performance. However, as discussed in §2, we believe this approach is preferable to overestimating performance, as it encourages systems evaluated against such baselines to achieve better attribution quality. Once the methodology was established, the process required only a few days to complete. Researchers applying this methodology should be able to generate similar ground truth results. To support reproducibility, we have made the ground truth publicly available; see §9.



(a) Neighborhood. 7,661 malicious nodes.

(b) Batch. 16,425 malicious nodes.

(c) Source. 5,248 malicious nodes.

(d) Ours. 58 malicious nodes.

Figure 10: Visualized ground truth for attack *Firefox Backdoor w/ Drakon In-Memory* on E3-THEIA. We note that the only Fig. 10d is legible.

## D Example Ground Truth Vizualiation

Fig. 10 shows the visualization of an E3-THEIA attack for four approaches above. Three current evaluation approaches label too many irrelevant nodes, making it difficult to visually present and analyze reported attacks, even when a system performs well in detecting and reconstructing them.

## E Matthews Correlation Coefficient

The Matthews Correlation Coefficient [45] is used in Machine Learning to measure of the quality of binary classifications.

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{12}$$

## F Raw Results

Table 10 and Table 11 show the raw data presented in Table 4, Table 5, Fig. 5, and Fig. 7.

| Dataset | System | TP | FP | TN | FN | Precision | MCC | Training Time | Testing Time | GPU Memory |
|---|---|---|---|---|---|---|---|---|---|---|
| E3-CADETS | ORTHRUS-full | 25 | 23 | 268,062 | 43 | 0.52 | **0.44** | **4min40** | 52min31 | **3.82GB** |
| | ORTHRUS-ano | 10 | 0 | 268,085 | 58 | **1.00** | 0.38 | | | |
| | Kairos | 0 | 9 | 268,076 | 68 | 0.00 | 0.00 | 22min49 | 36min26 | 3.93GB |
| | Threatrace | 61 | 252,117 | 15,968 | 7 | 0.00 | 0.00 | 28min28 | **21min51** | 5.22GB |
| | SIGL | 0 | 80 | 268,005 | 68 | 0.00 | 0.00 | 4h48 | 1h15 | 10.07GB |
| | MAGIC | 63 | 79,766 | 188,319 | 5 | 0.00 | 0.02 | 13h18 | 13h01 | 4.22GB |
| | Flash | 13 | 2,381 | 265,704 | 55 | 0.01 | 0.03 | 10h33 | 8h52 | 19.18GB |
| E3-THEIA | ORTHRUS-full | 48 | 11 | 699,166 | 70 | 0.81 | **0.57** | **3min58** | 41min39 | **2.03GB** |
| | ORTHRUS-ano | 8 | 0 | 699,177 | 110 | **1.00** | 0.26 | | | |
| | Kairos | 4 | 0 | 699,177 | 114 | **1.00** | 0.18 | 24min21 | 1h07 | 2.53GB |
| | Threatrace | 88 | 671,883 | 27,294 | 30 | 0.00 | -0.01 | 10min19 | **8min17** | 4.51GB |
| | SIGL | 1 | 29 | 699,148 | 117 | 0.03 | 0.02 | 14h07 | 24h04 | 10.44GB |
| | MAGIC | 115 | 394,906 | 304,271 | 3 | 0.00 | 0.01 | 11h39 | 11h41 | 5.35GB |
| | Flash | 22 | 32,082 | 667,095 | 96 | 0.00 | 0.01 | 6h51 | 5h30 | 36.93GB |
| E3-CLEARSCOPE | ORTHRUS-full | 2 | 6 | 111,347 | 39 | 0.25 | **0.11** | **2min50** | 5min56 | **0.65GB** |
| | ORTHRUS-ano | 1 | 1 | 111,352 | 40 | **0.50** | **0.11** | | | |
| | Kairos | 0 | 7 | 111,346 | 41 | 0.00 | 0.00 | 9min52 | 16min25 | 0.74GB |
| | Threatrace | 41 | 87,501 | 23,852 | 0 | 0.00 | 0.01 | 3min55 | **2min04** | 4.90GB |
| | SIGL | 1 | 11,372 | 99,981 | 40 | 0.00 | 0.00 | 1h01 | 31min17 | 9.71GB |
| | MAGIC | 40 | 101,737 | 9,616 | 1 | 0.00 | 0.00 | 1h37 | 36min56 | 9.75GB |
| | Flash | 0 | 15,137 | 96,216 | 41 | 0.00 | -0.01 | 19h01 | 18h16 | 11.60GB |

Table 10: Comparison of node-level detection performance on DARPA E3 datasets.

| Dataset | System | TP | FP | TN | FN | Precision | MCC | Training Time | Testing Time | GPU Memory |
|---|---|---|---|---|---|---|---|---|---|---|
| E5-CADETS | ORTHRUS-full | 2 | 10 | 3,111,245 | 121 | **0.17** | **0.05** | **42min35** | 6h01 | 21.10GB |
| | ORTHRUS-ano | 1 | 5 | 3,111,250 | 122 | **0.17** | 0.04 | | | |
| | Kairos | 0 | 6 | 3,111,249 | 123 | 0.00 | 0.00 | 4h03 | 10h22 | 23.85GB |
| | Threatrace | 91 | 3,104,018 | 7,237 | 32 | 0.00 | -0.03 | 5h45 | **3h05** | **17.31GB** |
| | SIGL | 0 | 66 | 3,111,189 | 123 | 0.00 | 0.00 | 38h00 | 36h58 | 22.72GB |
| | MAGIC | 123 | 3,110,714 | 541 | 0 | 0.00 | 0.00 | 77h13 | 10h38 | 79.36GB |
| | Flash | 45 | 33,941 | 3,077,314 | 78 | 0.00 | 0.02 | 101h26 | 105h06 | 80.19GB |
| E5-THEIA | ORTHRUS-full | 13 | 2 | 747,381 | 56 | 0.87 | 0.4 | **14min30** | 6h29 | 4.23GB |
| | ORTHRUS-ano | 2 | 0 | 747,383 | 67 | **1.00** | **0.17** | | | |
| | Kairos | 0 | 2 | 747,381 | 69 | 0.00 | 0.00 | 1h02 | 3h27 | **4.16GB** |
| | Threatrace | 66 | 739,322 | 8,061 | 3 | 0.00 | 0.00 | 2h51 | **51min13** | 11.59GB |
| | SIGL | 0 | 23 | 747,360 | 69 | 0.00 | 0.00 | 40h20 | 37h59 | 24.44GB |
| | MAGIC | 1 | 296,554 | 450,829 | 68 | 0.00 | -0.01 | 13h21 | 10h16 | 16.95GB |
| | Flash | 43 | 295,729 | 451,654 | 26 | 0.00 | 0.00 | 47h50 | 51h37 | 80.18GB |
| E5-CLEARSCOPE | ORTHRUS-full | 4 | 8 | 150,666 | 47 | **0.33** | **0.16** | **22min19** | 3h30 | **1.72GB** |
| | ORTHRUS-ano | 2 | 7 | 150,667 | 49 | 0.22 | 0.09 | | | |
| | Kairos | 1 | 3 | 150,671 | 50 | 0.25 | 0.07 | 1h06 | 3h02 | 2.26GB |
| | Threatrace | 41 | 142,487 | 8,187 | 10 | 0.00 | -0.01 | 44min53 | **37min46** | 5.94GB |
| | SIGL | 10 | 63 | 150,610 | 41 | 0.14 | 0.16 | 82h50 | 69h16 | 16.38GB |
| | MAGIC | 51 | 139,385 | 11,289 | 0 | 0.00 | 0.01 | 11h39 | 4h03 | 48.24GB |
| | Flash | 15 | 4,552 | 146,122 | 36 | 0.00 | 0.03 | 25h34 | 24h00 | 11.60GB |

Table 11: Comparison of node-level detection performance on DARPA E5 datasets.