Sometimes Simpler is Better: A Comprehensive Analysis of State-of-the-Art Provenance-Based Intrusion Detection Systems

Tristan Bilot[1 2 3†], Baoxiang Jiang[4†], Zefeng Li[5], Nour El Madhoun[2],
Khaldoun Al Agha[1], Anis Zouaoui[3], Thomas Pasquier[5]
[1]*Université Paris-Saclay,* [2]*LISITE, Isep,* [3]*Iriguard,*
[4]*Xi'an Jiaotong University,* [5]*University of British Columbia*

Abstract

Provenance-based intrusion detection systems (PIDSs) have garnered significant attention from the research community over the past decade. Although recent studies report near-perfect detection performance, we show that these systems are not viable for practical deployment. We implemented eight state-of-the-art systems within a unified framework and identified nine key shortcomings that hinder their practical adoption. Through extensive experiments, we quantify the impact of these shortcomings using cybersecurity-oriented metrics and propose solutions to address them for real-world applicability. Building on these insights, we demonstrate that most existing systems add unnecessary complexity, whereas a simple neural network reaches state-of-the-art detection on eight of nine DARPA datasets while offering a lighter, faster, and real-time detection solution. Finally, we highlight critical open research challenges that remain unaddressed in the current literature, paving the way for future advancements. To support research, we open-source our framework and provide pre-processed datasets with ground truth to support consistent evaluation.

1 Introduction

Provenance-based security techniques have gained popularity within the security community over the last decade (see Fig. 1). Various works have focused on capturing and representing a system execution as graphs [1–25], analyzing those graphs to identify malicious behavior [26–55], and detecting anomalies to build Provenance-based Intrusion Detection Systems (PIDSs) [56–80]. In this paper, we perform an in-depth study of this latter category.

While earlier systems [56, 71, 72] relied on relatively simple techniques to detect changes in the statistical properties of graphs to identify anomalies, Graph Neural Networks (GNNs) have seen broad adoption in more recent works [57–67, 80]. Although these systems adopt similar architectures (see Fig. 2), they often introduce multiple complex modifications. Furthermore, evaluation methodologies across studies are often inconsistent, with ablation studies rarely performed [80]. As a result, extracting actionable insights to develop practical PIDS solutions is challenging and faces significant barriers to real-world adoption [81]. Through this study, we aim to shed light on the technical barriers hindering wider adoption.

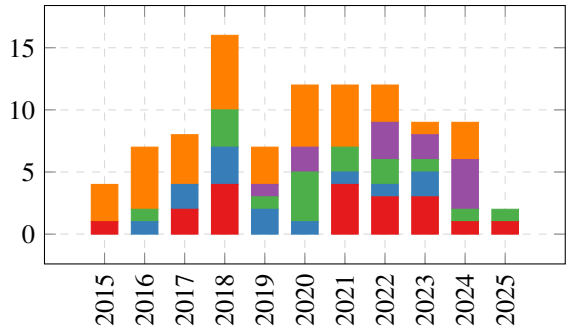


Figure 1: Provenance-security papers published over the last decade (at the time of submission) in ■ USENIX Security [4–8, 34–37, 58, 61, 65, 69, 80, 82–84], ■ ACM CCS [10–12, 38, 39, 85–88], ■ NDSS [2, 3, 27–33, 56, 57, 60, 68, 89], ■ IEEE S&P [13, 26, 40–42, 62, 64, 66, 67, 90, 91] and ■ other venues [18–25, 44–55, 59, 70–79, 92–96].

We identify eight state-of-the-art PIDSs with publicly available source code (§2.2) and conduct a thorough analysis of their architecture, design, and evaluation strategies [1]. Through experiments (§4), we identify nine key shortcomings, pinpoint the systems affected by each, and propose solutions. We perform an in-depth evaluation of all eight systems and identify future research direction.

Our results challenge the prevailing reliance on complex

[†]Work partially completed while at the University of British Columbia.

[1]For transparency, we acknowledge that our own works – SIGL [58], Kairos [62], and Orthrus [80] – are not exempt from these shortcomings and are included in our study.

GNN architectures in the literature, showing that a simple neural network not only surpasses these systems in eight of nine benchmarks but also achieves this with significantly lower computational cost. This highlights that the performance gain attributed to different GNN architectures in the literature is significantly overestimated. We aim for this study to provide a foundation for the development of improved evaluation protocols and to facilitate the adoption of practical PIDSs.

Contributions

- We systematically study eight state-of-the-art PIDSs and identify the differences in their design.
- We conduct a number of experiments, leveraging standard benchmark datasets, to identify and highlight nine shortcomings.
- Building on the insights from those experiments, we build VELOX, a lightweight PIDS, able to outperform state-of-the-art systems in eight out of nine well-established benchmarks.
- We discuss the implication of those findings and highlight open research challenges.
- We release VELOX and a framework implementing eight state-of-the-art PIDSs [58–62, 66, 67, 80] to serve as evaluation baselines in future work (see §3 and §8).

2 Background

Provenance graphs are an essential tool for monitoring and analyzing system behavior. They capture interactions between operating system entities like processes, files, and sockets. These graphs evolve dynamically as the system operates, providing a temporal record of execution that reflects the causal relationships between entities.

By representing interactions as directed edges with rich attributes like process commands, file paths and IP addresses, provenance graphs naturally lend themselves to machine learning analysis [97]. GNNs have notably emerged as a leading approach for learning complex attack patterns for Advanced Persistent Threat (APT) detection [80, 98]. Typically, streams of system events are converted into fixed-duration graphs, which are then processed by GNN models. Due to the scarcity of labeled attack data, most PIDSs rely on anomaly-based, self-supervised methods. These systems train exclusively on benign activity, learning to predict graph properties such as node or edge types. During inference, models assign low anomaly scores to benign behaviors and high scores to suspicious ones. Final decisions are made using thresholds or clustering techniques at the node, neighborhood, or graph level.

2.1 Study Motivation

PIDSs have shown significant promise as an effective and innovative tool for host-based intrusion detection. Recent works (Fig. 1) have consistently demonstrated their capability to detect advanced attacks and threats that evade traditional detection methods. Despite these satisfactory results, a gap exists between academic research and industrial requirements for existing systems.

In a previous study on provenance-based endpoint detection and response (P-EDR) systems [81], all 10 interviewed technical managers identified high client-side overhead as a significant challenge. Additionally, 7 out of 10 reported an excessive number of alarms, despite unanimous agreement (10/10) that provenance-based EDRs are more effective than traditional EDRs due to the superior interpretability of provenance graphs and their ability to be abstracted to higher-level representations [99, 100].

Since then, the size and complexity of systems in the literature have steadily grown, favoring anomaly detection methods based on complex GNNs. While these systems now report fewer false alarms [80], their increasing complexity has compromised practicality in several other aspects. This study experimentally identifies the key shortcomings of state-of-the-art PIDSs that hinder their practical adoption, representing, to our knowledge, the first study of its kind.

2.2 Considered Systems

We reviewed anomaly-based PIDSs from the past decade and identified a recent architectural shift from traditional to GNN-based systems. Among these systems, common components were systematized and integrated into a framework (§3) to support our experiments. We selected recent systems sharing compatible architectures, prioritizing those with open-source implementations or sufficient details for re-implementation. The selected PIDSs are detailed in Table 1, alongside their modularized architectures. We excluded other candidate systems, such as SHADEWATCHER [64] and PROGRAPHER [65], due to the absence or partial availability of their source code. Additionally, CAPTAIN [63] was excluded as its architecture is incompatible with the modular components outlined in Table 1.

The remainder of this section reviews the key contributions of the PIDSs considered in this study, six of which have been published in the last year.

SIGL (USENIX Sec'21) [58]. Formalizes malicious software installation detection as a malware detection task based on provenance graphs. Proposes a Graph LSTM-based autoencoder architecture aiming to reconstruct process nodes' features, utilizing the loss as anomaly score for downstream thresholding.

THREATRACE (IEEE TIFS'22) [59]. Detects attacks at the level of node neighborhoods using a multi-model framework based on GraphSAGE. Flags a neighborhood as malicious if it is misclassified by any of the sub-models.

NODLINK (NDSS'24) [60]. Aims to improve APT detection granularity and reduce false positives by detecting attacks

System	Feature Extraction	Graph Transformation	Featurization	Encoding	Decoding	Optimization	Detection	Open-Source
SIGL	*process*: path; *file*: path; *netflow*: IP		word2vec + random walks + alacarte	Graph LSTM	2-layer MLP	Node features reconstruction	Validation set based threshold	✗
THREATRACE	*nodes*: node type+distribution of edge types			GraphSAGE		Node type prediction + retraining	Fixed threshold	✔
NODLINK	*process*: cmd line; *file*: path; *netflow*: IP + port	Undirected graph	FastText	Weighted sum of neighbors	Variational Auto Encoder (VAE) + 3-layer MLP	Node features reconstruction	Validation set based threshold (90th percentile)	✔
MAGIC	*nodes*: node type; *edges*: edge type	Simple graph with no redundant edges		GAT	GAT + 2-layer MLP	Masked node embedding reconstruction + edge prediction	Outlier detection with K-D tree + fixed threshold	✔
KAIROS	*process*: path; *file*: path; *netflow*: IP + port		Hierarchical feature hashing (HFH)	TGN+graph attention	4-layer MLP	Edge type prediction	Fixed threshold	✔
FLASH	*process*: cmd line, path; *file*: path; *netflow*: IP + port		Word2vec + positional encoding	GraphSAGE	XGBoost	Node type prediction	Fixed threshold	✔
R-CAID	*process*: path+process name	Pseudo-graph with connected root cause	Doc2vec	GAT		Node type prediction	K-Means + Mean Absolute Deviation (MAD)	✗
ORTHRUS	*process*: type+path+cmd line; *file*: type+path; *netflow*: type+IP+port		word2vec	TGN w/o memory+graph attention	3-layer MLP	Edge type prediction	Validation set based threshold + K-Means	✔

Table 1: Systemization of PIDSs selected in this study. All systems were modularized into components within a unified framework to enable efficient experimentation.

at the node level, modeling APT detection as a Steiner Tree Problem (STP).

MAGIC (USENIX Sec'24) [61]. Learns node embeddings using an hybrid loss function and performs neighborhood-level APT detection by measuring the similarity of embeddings to their neighbors, producing an anomaly score for thresholding.

KAIROS (IEEE S&P'24) [62]. Captures the temporal dynamics of system events using a Temporal Graph Network (TGN) on dynamic provenance graphs. Detects APTs at the batch level by computing anomaly scores for edges via edge type prediction.

FLASH (IEEE S&P'24) [66]. Performs neighborhood-level attack detection by computing node-level anomaly scores from text embeddings using a lightweight encoder/decoder to predict node types.

R-CAID (IEEE S&P'24) [67]. Transforms provenance graphs into a pseudo-graph, connecting each node to its root cause, leveraged during embedding. Detects anomalies by measuring the deviation of nodes to clusters.

ORTHRUS (USENIX Sec'25) [80]. Improves detection granularity by detecting attacks at the node level using a lightweight TGN variant. Reduces false positives by applying a K-Means clustering on top of thresholded nodes.

3 Experimental Framework & Setup

Framework. We developed a unified framework that consolidates the eight PIDSs outlined in Table 1 into a single codebase. This framework allows for extensive customization by enabling the integration of components from different systems for complex analyses and ablation studies. To accomplish this, the original PIDS codebases were modularized into seven distinct components, ranging from "Feature Extraction" to "Detection", and adapted into an abstract architecture compatible with the framework. Wherever possible, the original code was optimized with GPU-accelerated operations and refactored following coding best practices to improve maintainability. Each system is configured via its own YAML file, specifying system-specific parameters. Given the significant computational demands of the extensive experiments presented in this study, efficient resource usage was critical, especially under limited CPU and GPU availability. To minimize redundancy, the framework utilizes a pipeline system (Fig. 2) that automatically reuses previously computed components and leverages existing results whenever possible. The framework was implemented primarily using PyTorch and PyTorch Geometric, consisting of 82 Python files and a total of 14,811 lines of code.

Datasets. The experiments in this study utilize the widely adopted DARPA E3 [101] and DARPA E5 [102] datasets, with ground truth provided by Jiang et al. [80]. For the final evaluation (§5), we also benchmark systems on the DARPA OpTC [103] dataset, following the same labeling methodology outlined by Jiang et al. [80]. Additional details and statistics about these datasets can be found in Appendix A.

Re-implementation. The re-implementation of systems within the framework primarily reuses components from their original public repositories. Common steps—such as snapshot construction and training loops—are largely shared across systems and thus abstracted without system-specific logic. For the specialized components listed in Table 1, we integrate the original code modularly. We allow easy swapping through YAML configuration.

For SIGL and R-CAID, which are not open source, we re-implemented components based on their papers. Consequently, any issues observed in our re-implementations may not reflect limitations of the original systems, but rather stem from misunderstandings or implementation details not disclosed in the corresponding papers.

Reproduction Efforts. We provide details of our reproduction efforts in Appendix B. The experiments follow the original evaluation methodology and use the DARPA datasets referenced in the respective papers. Reproducibility issues primarily stemmed from missing code, data, or labels. When all these artifacts were available, our reproduction attempts were generally successful. We point interested readers to a full reproducibility study [104].

Hardware setup. All training and evaluation were performed

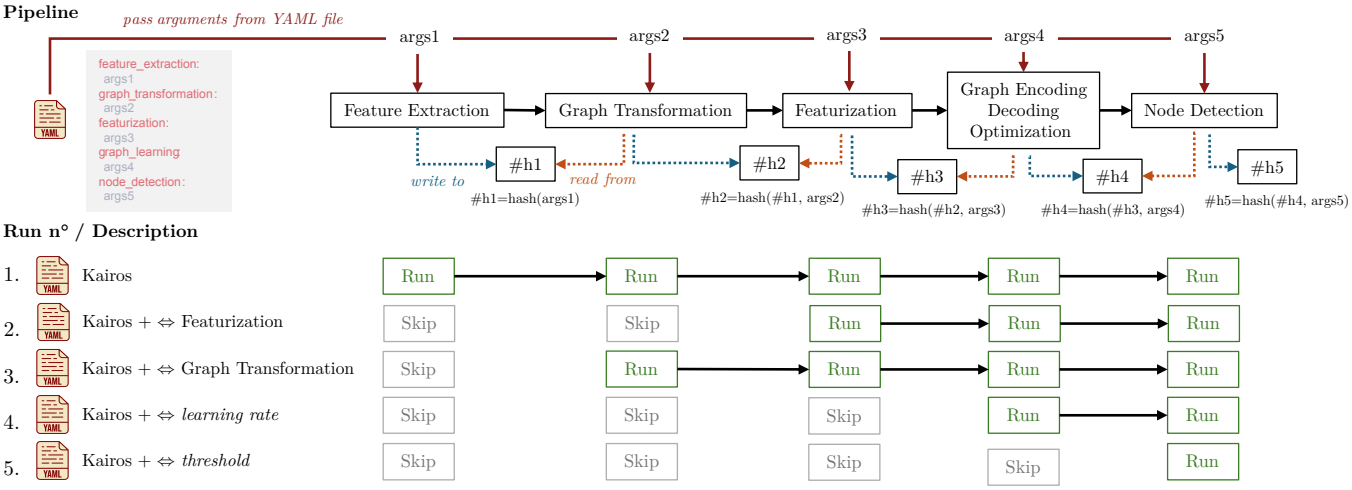


Figure 2: PIDS execution pipeline. Each PIDS is divided into tasks with configurable arguments specified in a YAML file. Task outputs are saved in uniquely hashed folders, with the hash derived from the task's arguments and the preceding task's hash. Changing an argument thus updates the hash and restarts the pipeline from the modified task onward. This approach minimizes redundant computations and supports efficient experimentation with time-intensive PIDSs. "⇔" indicates a modification in a task or in a hyperparameter value, "#i" refers to the hashed folder of a task.

System	Detection Granularity	Level
SIGL	Graph	Low ✗
THREATRACE	Neighborhood	Low ✗
MAGIC	Neighborhood	Low ✗
KAIROS	Batch	Low ✗
FLASH	Neighborhood	Low ✗
R-CAID	Descendant process nodes	Low ✗
NODLINK	Process nodes	High ✔
ORTHRUS	Node	High ✔

Table 2: Detection granularity of studied systems.

on three servers running Ubuntu 22.04, each equipped with a 3.2GHz 64-core AMD EPYC 7343 CPU, 1TB of RAM, and an NVIDIA GA100 GPU featuring 80GB of memory. The experiments in this paper required 453 days of computation.

4 Shortcomings

In this section, we identify nine shortcomings that prevent the practical deployment of state-of-the-art systems. We illustrate those shortcomings through experiments on the eight systems in §2.2 and perform a comprehensive evaluation in §5.

SC1: Insufficient Detection Granularity

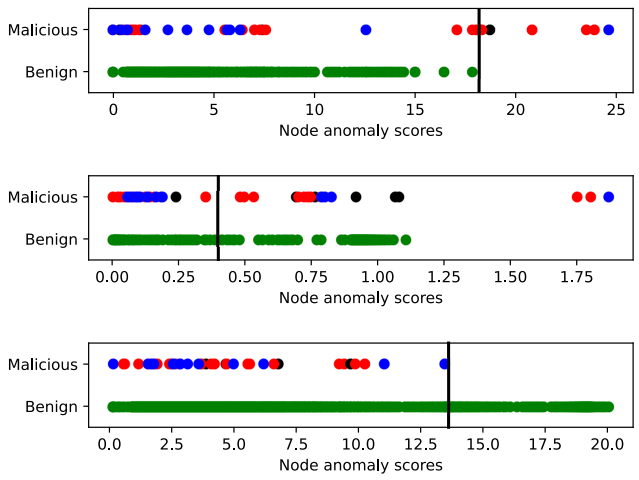
Most systems detect attacks at the granularity of entire graphs or node neighborhoods, often overwhelming analysts with thousands of elements to review [80], leading to burnout and reduced practicality [81, 105].

6/8 systems

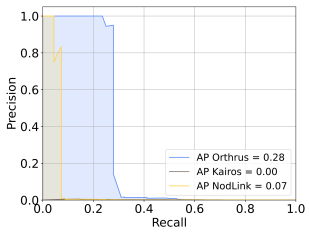
A recent survey of industry professionals indicates that analysts typically expect to investigate between 10 and 100 nodes when analyzing a malicious provenance graph [81]. However, Table 2 shows that most recent systems detect attacks at the level of neighborhoods, batches or entire graphs, resulting in an overwhelming number of nodes for analysts to examine [80].

This outcome is largely driven by the evaluation methods and labeling strategies employed by these systems, which inflate performance by labeling an excessive number of nodes as malicious, rather than accurately pinpointing the nodes directly involved in the attack. This approach leads to the development of impractical systems [80]. For example, the neighborhood-based ground-truth labeling used by THREATRACE —and adopted by other systems such as MAGIC and FLASH —assumes that all nodes within a 2-hop neighborhood of a known malicious node contribute to the attack. This includes cases where unrelated nodes, such as a process loading the same shared library as a malicious process, are also considered malicious. This artificially inflates the number of nodes labeled as malicious. Similarly, batch-based and descendant-based approaches overestimate the number of malicious nodes by orders of magnitude. Jiang et al. [80] investigate this issue in more detail.

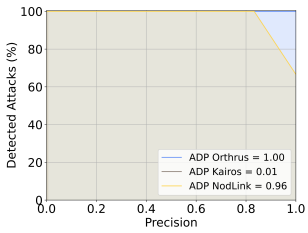
In contrast, node-level detection, as performed by ORTHRUS and NODLINK, identifies a limited set of anomalous nodes. A smaller anomaly set simplifies analysis and facilitates attack reconstruction to uncover additional malicious patterns in the graph [37]. This approach represents a practical evaluation methodology aligned with security analysts' needs [81]. In the rest of this study, we conduct the experi-



(a) Predicted node anomaly scores of ■ ORTHRUS (top), ■ NODLINK (middle) and ■ KAIROS (bottom). Red, black, and blue nodes refer to malicious nodes from distinct attacks, whereas green nodes are benign. Horizontal lines represent thresholds computed by each system.



(b) Area under the precision-recall curve (AP).



(c) Area under the Attack Detection Precision curve (ADP).

Figure 3: Comparison of AP and ADP on the predicted scores of ■ ORTHRUS, ■ NODLINK and ■ KAIROS on E3-CADETS.

ments with node-level detection granularity.

Recommendations. As outlined by Jiang et al. [80], authors should consider adopting node-level or edge-level detection when evaluating PIDSs to ensure results are actionable and reduce analyst fatigue. They should avoid evaluation strategies that artificially inflate performance, such as labeling entire neighborhoods or descendant nodes as malicious.

SC2: Missing Metric to Measure Attack Detection

Most evaluations use post-threshold metrics like precision and recall, which overlook predictive power, misattribute poor performance, and are unsuitable for multi-attack detection.

7/8 systems

State-of-the-art PIDSs detect malicious elements by first assigning it an anomaly score, then separate these elements in two classes: benign and anomaly, based on their scores. Some systems, such as ■ SIGL, ■ THREATRACE, ■ NODLINK, ■ KAIROS, and ■ FLASH, employ thresholding methods, labeling elements as malicious if their scores exceed a predefined threshold. In contrast, other systems, including ■ MAGIC, ■ R-CAID, and ■ ORTHRUS, utilize more sophisticated relative thresholds, flagging elements as anomalous if their scores deviate significantly from clusters.

Past evaluations report performance using post-threshold metrics such as precision, recall, F1-score, and accuracy. However, these metrics have two key limitations when comparing detection systems: (1) they assess performance at a single decision threshold, which can bias results depending on the thresholding strategy used; and (2) they treat all true positives equally, failing to account for the presence of multiple distinct attacks, as observed in the DARPA TC and OpTC datasets.

Fig. 3a presents the anomaly scores generated by ■ ORTHRUS, ■ NODLINK and ■ KAIROS, where each "dot" represents the score of a node in the graph. ■ ORTHRUS separates some nodes corresponding to all three attacks (the rightmost blue, red and black nodes), ■ NODLINK separates nodes from two of the three attacks (red and blue nodes), and ■ KAIROS cannot do so. The insights revealed by such visualizations are easily overlooked when relying solely on post-threshold metrics. They depend on a fixed threshold (represented here as the horizontal line) and do not capture the distribution of anomaly scores, biasing results toward systems with good threshold selection. For example, ■ ORTHRUS' threshold is remarkably well-chosen as it precisely includes one node per attack while excluding the highest benign ones (we discuss why in SC3). In contrast, ■ NODLINK's threshold is too low, resulting in many false positives and metrics that do not reflect the potential ability of the system. In short, relying exclusively on metrics such as precision and recall conflates detection quality with the separate and complex problem of threshold calibration, a process that often requires dedicated optimization [106].

Metrics such as AUC-ROC and Average Precision (AP) address limitation (1) by using the full distribution of anomaly scores, avoiding the need for a fixed threshold. However, they do not address limitation (2), as they are not designed for multi-attack scenarios, which are common in intrusion detection datasets. A PIDS should prioritize detecting each attack and ensuring alerts correspond to real attacks (precision), rather than trying to identify every malicious node (recall) [81]. Indeed, this reduces alert fatigue [81, 105] and past works have shown that attacks can be reconstructed from a few correctly detected nodes [28, 37, 80, 91].

In addition, AUC-ROC can give an overly optimistic view in highly imbalanced datasets [107], which is typical in intrusion detection. AP also tends to emphasize recall over precision, as systems with low recall will consistently yield low area under the precision-recall curve and thus low AP. This can be misleading in scenarios where precision is the primary concern. As illustrated in Fig. 3b, ■ NODLINK receives a near-zero AP despite its ability to clearly distinguish cer-

tain attacks from benign behavior. In short, standard metrics can misrepresent performance when precise attack detection matters more than identifying every malicious node. This is particularly true in heavily imbalanced settings.

Attack Detection Precision (ADP). Building on requirements (1) and (2), we propose the Attack Detection–Precision (ADP) curve, which plots the percentage of attacks detected (y-axis) against node-level precision (x-axis). Fig. 3c shows the ADP curve for NODLINK, derived from Fig. 3a. It indicates that when detecting 66.6% of attacks (2 out of 3 on E3-CADETS), NODLINK achieves 100% precision. However, detecting all attacks (100%), including the rightmost black node, reduces the best-case precision to approximately 80%, as some benign green nodes must also be included.

We distill this curve into a single scalar metric to facilitate comparison of systems. We compute the area under this curve, referred to as ADP throughout the paper.

$$\text{ADP} = \int_0^1 D(p)\,dp, \tag{1}$$

where $D(p)$ computes the percentage of detected attacks at precision p, defined by:

$$D(p) = \frac{|\{A_i \mid A_i \cap R(p) \neq \emptyset\}|}{k}, \tag{2}$$

where A_i is the set of nodes in attack i among the total k attacks and $R(p)$ is the set of nodes flagged at precision p. The ADP metric directly measures the predictive power of PIDSs across multiple attacks, independent of post-processing or thresholding, thus addressing limitations (1) and (2). In practice, KAIROS achieves an ADP score of 0.01, NODLINK 0.96, and ORTHRUS 1.00. These results are consistent with the observations in Fig. 3a. ADP offers a useful way to evaluate detection potential without relying on threshold selection.

Recommendations. Authors should measure and report the predictive power of detection pipelines independently of thresholding, as demonstrated in Fig. 3. We propose ADP to fulfill this objective, and encourage the community to explore this issue further.

SC3: Impractical Thresholding Methods

Certain systems employ thresholding methods unsuited to practical settings. Some rely on fixed, arbirtraty and manually set thresholds that are hard to adapt to different environments, while others use clustering techniques that make assumptions about attack presence and depend on future data, leading to data snooping [108].

5/8 systems

Determining an effective thresholding method is challenging, as it requires separating benign and anomalous instances within the anomaly-score space (Fig. 3a) to maximize true positives while minimizing false positives.

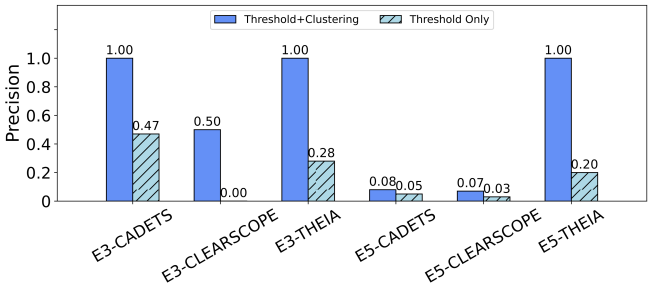


Figure 4: Precision of ORTHRUS with its original validation-based thresholding with clustering (data snooping) vs. thresholding only (non-snooped).

Systems such as THREATRACE, MAGIC, KAIROS, and FLASH rely on manually determined fixed thresholds for detection. This approach is fundamentally flawed for two reasons: (1) identifying the optimal threshold involves iterative "trial and error" adjustments on the test set, leading to data snooping [108, 109], where information that would be unavailable in realistic deployment scenarios is utilized, and (2) fixed thresholds lack adaptability, fail to generalize in dynamic environments and fail against concept drift [110].

Alternatively, ORTHRUS identifies outliers during inference by applying both a threshold and a clustering-based method directly to predicted scores. However, such techniques are also problematic: (1) they require full knowledge of the test data, as clustering is typically performed post-inference on all or part of the test set, also leading to data snooping, and (2) they assume the presence of malicious elements, as clustering methods inherently identify outliers in the anomaly score space.

In contrary, R-CAID avoids data snooping by applying K-Means clustering to the training data only, with node scores during inference assigned using Median Absolute Deviation (MAD). Similarly, SIGL and NODLINK prevent data snooping by employing a thresholding method computed using the validation set, which contains no labeled examples, ensuring the separation of test data from training. Fig. 4 shows that applying such thresholding to ORTHRUS significantly lowers precision, even reaching 0% on E3-CLEARSCOPE.

Recommendations. Authors should ensure that the threshold selection methods do not rely on data unavailable in realistic deployment scenarios (i.e., avoid data snooping [108]). Additionally, they should consider using threshold-independent metrics, such as ADP.

SC4: Unfair Comparison with Baselines

Studies commonly omit hyperparameter tuning for baseline systems, despite the sensitivity of PIDSs to these settings, relying instead on default hyperparameters often obtained in a very different exper-

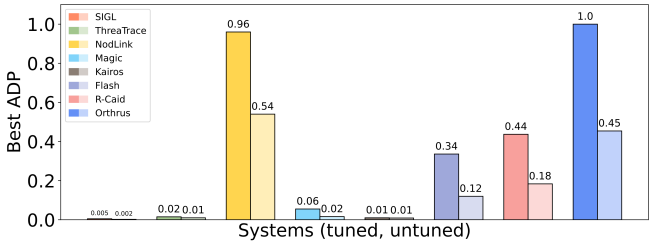


Figure 5: Comparison of best ADP (across five iterations) for tuned systems (original color) and untuned systems (lighter color) on E3-CADETS.

imental setup. This practice leads to unfair comparison between well-tuned and untuned systems.

8/8 systems

Hyperparameters play a key role in the performance of PIDSs. They are tightly linked to the underlying topology and density of graphs. For instance, the size of a graph batch of fixed duration depends on the volume of system events recorded on the host. On E3-CADETS and E5-CADETS, the average number of edges for a 15-minute batch is 5620 and 161,849, respectively (see Appendix A). However, graph size significantly influences the learning task, as larger graphs can lead to *over-squashing*, where the GNN encoder struggles to compress excessive information into a node embedding [111]. Likewise, increased training data requires the model to encode more information, which further affects embedding size. Since hyperparameters are sensitive to varying characteristics, thorough tuning is essential. This practice should extend to baselines as well as the authors' proposed systems. In the literature, none of the eight studied systems mention tuning baselines before evaluation. Fig. 5 shows the impact overlooking tuning has on attack detection performance.

In this study, we ensure a fair and accurate evaluation by consistently tuning all benchmarked systems. However, hyperparameter tuning can be particularly resource-intensive, as it requires running each experiment multiple times – a process further complicated by the inherent instability of PIDSs (see SC5). In the PIDS literature, hyperparameter tuning often relies on test data, meaning reported values reflect system performance under idealized conditions. To the best of our knowledge, tuning PIDSs without prior knowledge of the attacks remains an unresolved challenge.

Recommendations. As emphasized in prior work [108], all evaluated systems should be properly tuned for the specific task under evaluation. The effort invested in hyperparameter selection for baseline systems should match that applied to the authors' own approach to ensure fair and meaningful comparisons.

SC5: Not Measuring Instability

Our experiments show that all systems exhibit predictive instability [112,113] under identical configurations, despite the need for consistent IDS decisions. This phenomenon, driven by random factors like weight initialization, is largely overlooked in existing studies.

8/8 systems

Assessing the instability of a PIDS is crucial for evaluating the consistency of its predictions and the reliability of the system. High instability leads to widely varying predictions from the same system on identical data, undermining analyst confidence and real-world adoption. Despite its importance, the instability is not measured in any past study.

To quantify system instability, we adapt the uncertainty measurement method from Lakshminarayanan et al. [114] to PIDSs. This involves running a well-tuned system with identical configuration and input across multiple iterations and measuring the ADP for each iteration. Experiments are run for $T = 5$ iterations with different seeds. Each experiment starts from the "Featurization" task of the pipeline to ensure that both the text embeddings and GNN embeddings are trained with a different initialization.

To measure instability, we introduce the relative ADP standard deviation, noted $\tilde{\sigma}_{\text{ADP}}$, which quantifies the variability of ADP scores from the mean across all iterations.

$$\tilde{\sigma}_{\text{ADP}} = \frac{\sigma_{\text{ADP}}}{\overline{\text{ADP}}} \times 100, \tag{3}$$

where σ_{ADP} is the standard deviation of ADP across the T iterations and $\overline{\text{ADP}}$ is the mean of these scores.

We run each system on all datasets for five iterations and report the min, max and mean ADP scores in Fig. 6, along with $\tilde{\sigma}_{\text{ADP}}$ as per Equation 3 in Fig. 7. The results reveal significant variability in ADP scores across most systems when identical experiments are repeated, with deviations from the mean often approaching 100%. For instance, running ■ ORTHRUS on E3-THEIA yields ADP values ranging from 1.00 in the best case to below 0.1 in the worst case.

We hypothesize that this instability stems from the random initialization of weights and other sources of variation across iterations [113]. Enhancing system robustness to such instability poses significant challenges, requiring further research to improve existing methodologies [115,116]. Potential research directions are detailed in §6.

Instability is critical during both the research and production phases of PIDS development. Accordingly, we recommend averaging metrics over multiple iterations, a practice we adopt throughout this paper.

Recommendations. The authors should account for the inherent instability of their system by repeating the entire evaluation pipeline multiple times and reporting the average and standard deviation of metrics, as is standard practice in the machine learning literature [117, 118].

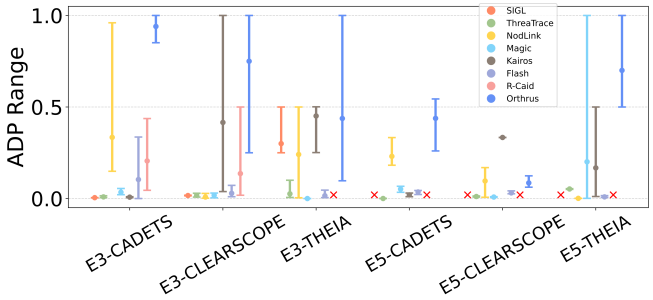


Figure 6: ADP range of tuned systems across five iterations. A bar spans the minimum to maximum ADP, with a dot indicating the mean. Runtime and memory errors are marked with a red cross.

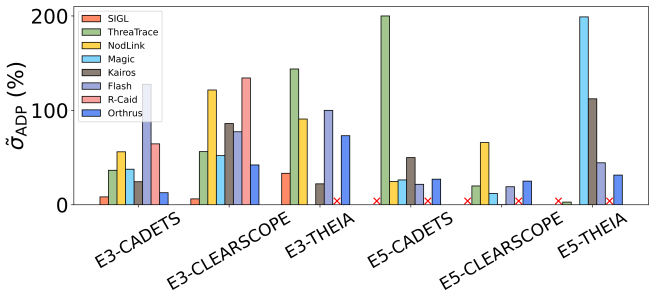


Figure 7: $\tilde{\sigma}_{ADP}$ across five iterations on all datasets. Runtime and memory errors are marked with a red cross.

SC6: Featurization Methods Trained on Test Data

Some systems use text embeddings that cannot handle unseen words during inference. Mitigation involves either incorporating test data, leading to data snooping [108], or substituting unseen words with zero-vectors, whose impact is unclear.

1/8 systems

Text embedding-based featurization methods are widely employed in PIDSs, as evidenced by six out of the eight systems analyzed in this study. Understanding their limitations and evaluating their necessity for detection are important.

Inductive methods like word2vec+alacarte (■ SIGL), FastText (■ NODLINK), and HFH (■ KAIROS) are trained on a defined vocabulary and can infer embeddings for Out-Of-Vocabulary (OOV) words. Conversely, transductive methods like word2vec (■ FLASH, ■ ORTHRUS) and Doc2vec (■ R-CAID) either learn embeddings from the entire dataset to avoid OOVs (■ ORTHRUS) or substitute OOV words with zero-vectors (■ FLASH, ■ R-CAID). However, training on test data introduces data snooping.

To understand the impact of training on training data versus the full dataset, we evaluate the detection performance of different featurization methods under both scenarios in Fig. 8.

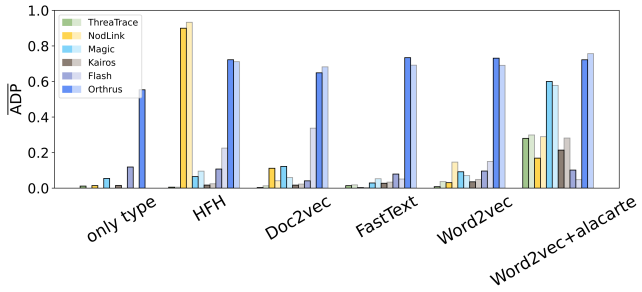


Figure 8: Comparison of $\overline{ADP}$ for tuned systems using different featurization methods trained on training set (original color) and full dataset (light color). For a fair comparison of methods, all experiments use ■ ORTHRUS' text features (Table 1, "Feature Extraction") as input. Each experiment runs with embedding sizes of 32, 128, and 256, reporting the mean ADP of the best size across five iterations.

■ FLASH's word2vec with positional encoding is excluded due to similar results to standard word2vec but much higher runtime.

How important are text embeddings for detection? To address this, we evaluate each system without text embeddings, using one-hot encoded system entity types as node features (the *only type* method in Fig. 8). The consistently lower scores relative to text embedding methods highlight the critical role of capturing textual semantics from file paths, process command lines, and network flows. Notably, systems like ■ THREATRACE and ■ MAGIC, which original design lack text embeddings, demonstrate significant detection improvements when certain text embedding methods are applied.

What is the effect of training on test data? Intuitively, training text embeddings on test data might inflate performance, as the test distribution is incorporated and all words are included in the vocabulary. However, as shown in Fig. 8, training on test data provides minimal advantage in top-performing experiments, even for transductive methods. Notably, these methods perform well when trained solely on training data, replacing unknown words with zero-vectors. Computationally expensive methods to handle OOV words appear unnecessary.

Recommendations. Authors should ensure that text embedding strategies do not introduce data snooping [108]. Embedding models should be trained exclusively on training data, and the handling of out-of-vocabulary tokens should be discussed and empirically validated.

SC7: Overly Complex Architectures

PIDSs are complex, with multiple interacting components, yet without ablation studies, it is unclear if simpler architectures could achieve similar performance.

8/8 systems

The absence of detailed ablation studies introduces un-

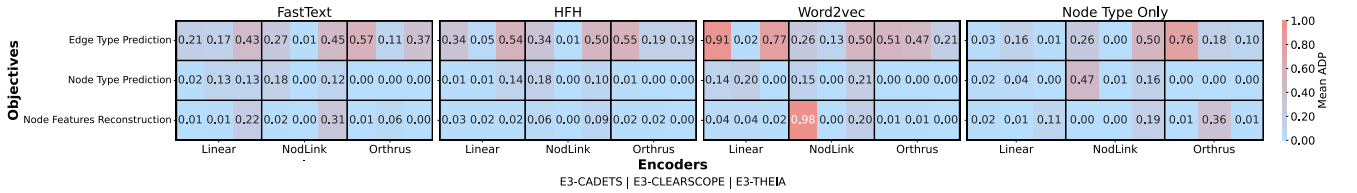


Figure 9: Grid-search ablation study conducted from ■ ORTHRUS with four different Featurization methods and various encoders and objectives. Reported numbers are the mean ADP across five iterations. Each cell comprises the values of an experiment on datasets `E3-CADETS`, `E3-CLEARSCOPE` and `E3-THEIA`.

certainty about the essential components of an architecture. Overly complex designs increase the risk of overfitting, as well as runtime and memory overhead, attack surface, and deployment costs [108]. Moreover, ablation studies play a key role in guiding future research by identifying the most valuable components for reuse in empirical work.

To identify the critical components of high-performing systems and develop a more streamlined model, we conduct a series of ablation experiments, as illustrated in Fig. 9. Beginning with the components of the state-of-the-art system, ■ ORTHRUS, we systematically substitute individual components across all possible Objective/Encoder pair combinations using a grid search approach applied to multiple featurization methods. To ensure robust evaluation across diverse scenarios, each ablation is tested on three E3 datasets, with the mean ADP calculated over five iterations to account for instability, as described in SC5. Additionally, to avoid data snooping, as outlined in SC6, all featurization methods are trained exclusively on the training data.

Objective functions. The objective functions analyzed include edge type prediction, node type prediction, and node feature reconstruction. Those are all the functions used in the literature (Table 1).

Encoders. For the encoders, we focus on those utilized in ■ ORTHRUS and ■ NODLINK, chosen for their demonstrated superior performance in prior experiments (Fig. 6). ■ KAIROS' encoder is excluded due to its architectural similarity to ■ ORTHRUS and its consistently lower performance. Additionally, we introduce a new encoder, referred to as "Linear," which consists of a single linear layer. This addition is intended to assess the necessity and effectiveness of more complex GNN-based encoders in achieving high performance.

Featurization methods. Finally, the experiments are conducted using four different featurization methods. FastText (■ NODLINK), Word2Vec (■ ORTHRUS), and HFH (■ KAIROS) are selected for their optimal balance between runtime efficiency and detection performance. Additionally, node types are used as standalone features to assess the impact of text embeddings on detection accuracy.

Edge type prediction performs better. As illustrated in Fig. 9, edge type prediction consistently outperforms other objective functions on average. This superiority can be attributed to the higher number of edges relative to nodes in all datasets (Appendix A), enabling more backpropagation steps. Furthermore, edge-level decoding inherently models pairwise node interactions, offering richer semantic learning than node-level objectives.

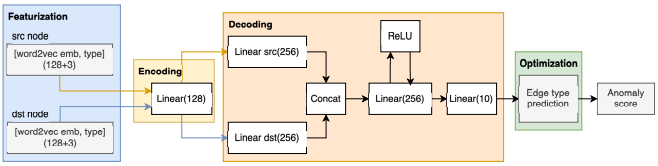
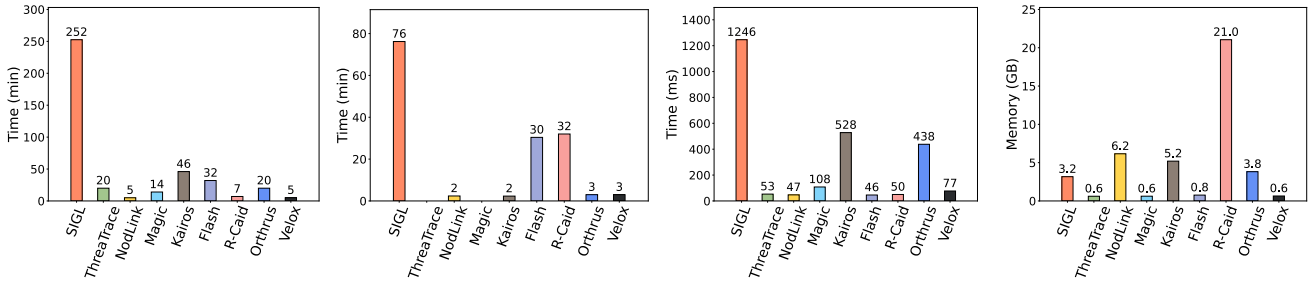


Figure 10: Neural network architecture of ■ VELOX.

A simple linear layer suffices as an encoder. Combining a linear encoder with Word2Vec achieves the highest ADP on two out of three datasets, with mean ADPs of 0.91 and 0.77 for `E3-CADETS` and `E3-THEIA`, respectively. Furthermore, pairing the linear encoder with other text embedding methods also proves effective for attack detection. This unexpected result suggests that graph-based encoders are not essential for detecting attacks in these datasets.

Graph structure encodes attack-related semantics. While text embedding methods achieve higher average ADP scores, ■ ORTHRUS' and ■ NODLINK's encoders, relying solely on node types as features, still successfully detect attacks. This demonstrates that GNN-based encoders leverage attack-related semantics directly from the graph structure.

Designing VELOX. Building on the experiments described above, we propose ■ VELOX, a new system derived from ■ ORTHRUS' architecture. ■ VELOX replaces the complex TGN-based encoder with a simple linear model, resulting in a lightweight and efficient neural network (see Fig. 10). Text embeddings are generated using word2Vec, following the methodology outlined in ■ ORTHRUS while addressing SC6. For each incoming system event, the embeddings of its two end nodes are retrieved, with OOV tokens replaced by a zero vector. These embeddings are then passed into the neural network. First, the end-node embeddings are processed through a linear layer, which functions as the "Encoding" component. The subsequent steps align with ■ ORTHRUS' "Decoding" component: source and destination entity embeddings are processed through separate linear layers to capture



(a) Encoder training time (b) Featurization training time (c) Time per batch (inference) (d) Peak memory (inference)

Figure 11: Runtime and memory consumption of systems on E3-CADETS with 15 min-batches.

orientation-specific semantics, reflecting the distinct roles of emitting and receiving nodes in provenance graphs. The processed embeddings are concatenated and fed into a two-layer MLP with non-linear activation, enabling the model to learn complex patterns. The anomaly score is computed as the cross entropy between the predicted and actual types of the input event, evaluated across ten different system calls.

Recommendations. Authors should ensure that any increase in architectural complexity, and its associated computational cost, is justified by substantial performance gains. They should conduct and report ablation studies to isolate the contribution of individual components and validate their impact on detection performance and computational cost.

SC8: Insufficient Scalability

Some systems have long training times, high inference costs, or excessive memory use, limiting their practicality in real-world settings where frequent retraining and low memory footprints (<160MB) are necessary [81].

8/8 systems

Training runtime. The results in Fig. 11a reveal substantial variability in encoder training times, with SIGL's RNN incurring particularly high training costs. Similarly, Fig. 11b highlights disparities in featurization training times. Word2Vec (ORTHRUS, VELOX), FastText (NODLINK), and HFH (KAIROS) are relatively low-cost featurization techniques. In contrast, SIGL and FLASH use word2Vec variants that come at significantly higher costs, as does R-CAID with Doc2Vec. Interestingly, the choice of embedding technique had relatively little impact on ORTHRUS' performance, as shown in Fig. 8. Minimizing training time is especially critical for practical IDSs, as they require frequent retraining to address concept drift [81].

Inference runtime. During inference (Fig. 11c), node-level systems (NODLINK, FLASH, R-CAID, and THREATRACE) are generally faster than edge-level systems (ORTHRUS, KAIROS, and VELOX). This is primarily due to the datasets described in Appendix A, including E3-CADETS, which contain significantly more edges than nodes per 15-minute batch. The complexity of edge-level systems scales with E (edges) rather than N (nodes), resulting in increased runtime. However, as shown in SC7, this additional cost is justified by improved performance. Notably, despite using an edge-level objective, VELOX achieves efficiency comparable to the fastest node-level systems. It is 5.7x and 6.8x faster than ORTHRUS and KAIROS, respectively, due to its simpler architecture.

Memory. Fig. 11d highlights substantial variation in peak memory usage across the evaluated systems. R-CAID exhibits particularly high memory consumption due to its pseudo-graph transformation process.[2] Systems such as R-CAID, NODLINK, SIGL, KAIROS, and ORTHRUS employ relatively complex GNN architectures. For instance, the TGN-based encoders in KAIROS and ORTHRUS require large GPU tensors to facilitate efficient indexing, contributing to their higher memory usage. In contrast, lighter node-level encoders like those in THREATRACE, FLASH, VELOX, and MAGIC exhibit significantly lower memory requirements.

All evaluated systems surpass the maximum memory footprint of 160 MB suggested in past work [81]. While what constitutes reasonable performance upper-bound can be debated, we note that VELOX achieves significantly lower memory usage during real-time detection scenarios, peaking at only 5.7 MB, as detailed in §5.3.

Recommendations. Authors should measure and report the computational costs of both training and inference. These metrics should be discussed in the context of the system's intended deployment model to assess its practicality in real-world settings.

SC9: Lacking Real-Time Detection

Current systems necessitate completing graph construction prior to inference, restricting detec-

[2]This task is currently performed post-capture. A mechanism similar to CamQuery [87] could potentially perform the graph transformation during capture with minimal overhead. However, to the best of our knowledge, no such implementation currently exists.

System	SC1	SC2	SC3	SC4	SC5	SC6	SC7	SC8	SC9
SIGL			✔			✔			
THREATRACE						✔			
NODLINK	✔		✔			✔			
MAGIC						✔			
KAIROS						✔			
FLASH						✔			
R-CAID			✔			✔			
ORTHRUS	✔	✔							
VELOX	✔	✔	✔	✔	✔	✔	✔	✔	✔

Table 3: Shortcomings addressed by systems.

> tion to fixed intervals of x minutes. A practical PIDS should enable near real-time detection.
>
> 8/8 systems

In state-of-the-art PIDSs, provenance events are collected from a data stream, and a graph is constructed every time a predefined criterion–such as the number of edges or elapsed time–is met. This graph is then utilized during training and inference. While effective for training, this approach introduces delays during inference, as detection is performed on data batches rather than in real-time. For instance, in ■ KAIROS, detection is conducted at fixed intervals, such as every 15 minutes, requiring the batch to be fully constructed before inference, which limits its real-time detection capabilities.

An ideal PIDS should have the ability to raise alerts in real-time as events occur [81]. Achieving this with current state-of-the-art systems would require constructing a graph for each newly recorded event. However, the significant computational overhead of such systems makes this approach impractical. Despite its critical importance in industry, the application of GNN-based systems for real-time detection remains underexplored. In contrast, ■ VELOX processes events individually, enabling efficient real-time detection. As demonstrated in §5, ■ VELOX achieves strong detection performance across most datasets at a fraction of the computational cost. This marks a departure from the prevailing trend in the literature, which often prioritizes increasingly complex and resource-intensive architectures.

Recommendations. Authors should state the detection strategy used by the system (e.g., real-time vs. batched). They should evaluate and report performance within the context of that strategy to ensure alignment with practical deployment requirements.

Summary. Building on the preceding analyses and experiments, Table 3 provides a summary of the shortcomings addressed by each system.

Addressing shortcomings with VELOX. At the design level, ■ VELOX resolves SC3 by calculating a threshold from the maximum score encountered in the validation set. SC6 is addressed by training text embeddings solely on training data. SC1, SC4, SC5 and SC7 do not necessarily have a clear solution, we identify and quantify them explicitly through experiments. Finally, SC8 and SC9 are addressed by the intrinsic lightweight and real-time nature of ■ VELOX.

System	ADP Mean	ADP Min	ADP Best	$\bar{\sigma}_{ADP}$	Precision Mean	Precision Min	Precision Best	TP Best	FP Best
E3-CADETS									
SIGL	0.00	0.00	0.01	8%	0.00	0.00	0.00	0	45
THREATRACE	0.01	0.01	0.01	36%	0.00	0.00	0.00	26	17k
NODLINK	0.33	0.15	0.96	56%	0.00	0.00	0.00	50	55k
MAGIC	0.03	0.02	0.06	37%	0.00	0.00	0.00	68	144k
KAIROS	0.01	0.01	0.01	24%	0.00	0.00	0.00	0	8
FLASH	0.10	0.00	0.34	127%	0.05	0.00	0.25	1	3
R-CAID	0.21	0.04	0.44	64%	0.21	0.00	1.00	2	0
ORTHRUS	0.94	0.85	1.00	12%	0.66	0.40	1.00	10	0
VELOX	0.94	0.77	1.00	9%	0.85	0.44	1.00	8	0
E3-CLEARSCOPE									
SIGL	0.02	0.01	0.02	6%	0.00	0.00	0.01	3	391
THREATRACE	0.02	0.01	0.03	56%	0.00	0.00	0.00	41	35k
NODLINK	0.01	0.00	0.03	121%	0.00	0.00	0.00	0	0
MAGIC	0.02	0.00	0.03	52%	0.00	0.00	0.00	0	0
KAIROS	0.42	0.04	1.00	86%	0.00	0.00	0.00	0	0
FLASH	0.03	0.01	0.07	77%	0.00	0.00	0.00	35	23k
R-CAID	0.14	0.02	0.50	134%	0.03	0.00	0.08	15	169
ORTHRUS	0.75	0.25	1.00	42%	0.00	0.00	0.00	1	399
VELOX	0.30	0.02	1.00	116%	0.00	0.00	0.00	1	913
E3-THEIA									
SIGL	0.30	0.25	0.50	33%	0.18	0.10	0.33	1	2
THREATRACE	0.03	0.01	0.10	143%	0.00	0.00	0.00	118	701k
NODLINK	0.24	0.00	0.50	90%	0.00	0.00	0.00	24	199k
MAGIC	0.00	0.00	0.00	0%	0.00	0.00	0.00	74	196k
KAIROS	0.45	0.25	0.50	22%	0.70	0.50	1.00	2	0
FLASH	0.02	0.01	0.05	99%	0.00	0.00	0.00	3	8.2k
R-CAID									
ORTHRUS	0.44	0.10	1.00	73%	0.24	0.07	1.00	8	0
VELOX	0.72	0.42	0.97	30%	0.53	0.03	0.91	10	1

Table 4: Detection results on E3 datasets. Rows in gray indicate timed-out or failed experiments.

5 Systems Comparison

In this section, we compare all eight systems while considering the shortcomings presented in §4.

5.1 Detection Performance

We perform node-level detection, consistent with SC1. Table 4, Table 5, and Table 6 show the results. We note that most systems were not designed with node-level detection in mind (■ NODLINK and ■ ORTHRUS are the only systems designed for such a task).

Experimental setup. To ensure fair evaluation, both training and thresholding exclude test data as per SC3 and SC6. For ■ ORTHRUS, this involves training word2Vec on training data and using a validation-based threshold solely for detection. As in previous experiments, all systems were tuned following the guidelines detailed in SC4. To account for and quantify instability, each experiment was repeated five times, as outlined in SC5. We report the mean, max and min ADP to evaluate the predictive capabilities of the systems. As post-thresholding metrics, we report the mean, minimum, and best precision across all iterations. Additionally, true positives and false positives are calculated for the instance with the highest precision to represent attribution quality in a best-case scenario. Some experiments involving ■ SIGL and ■ R-CAID were excluded due to excessive training time–exceeding two days–or OOM

System	ADP Mean	ADP Min	ADP Best	$\tilde{\sigma}_{ADP}$	Precision Mean	Precision Min	Precision Best	TP Best	FP Best
E5-CADETS									
SIGL									
THREATRACE	0.00	0.00	0.00	199%	0.00	0.00	0.00	98	3.1M
NODLINK	0.23	0.18	0.33	24%	0.00	0.00	0.00	76	708k
MAGIC	0.05	0.03	0.07	26%	0.00	0.00	0.00	118	2.6M
KAIROS	0.02	0.01	0.03	49%	0.00	0.00	0.00	0	6
FLASH	0.04	0.02	0.04	21%	0.00	0.00	0.00	6	24k
R-CAID									
ORTHRUS	0.44	0.26	0.54	27%	0.43	0.25	0.50	1	1
VELOX	0.31	0.10	0.50	51%	0.12	0.10	0.14	11	68
E5-CLEARSCOPE									
SIGL									
THREATRACE	0.01	0.01	0.01	19%	0.00	0.00	0.00	42	64k
NODLINK	0.10	0.01	0.17	65%	0.00	0.00	0.00	33	38k
MAGIC	0.01	0.01	0.01	11%	0.00	0.00	0.00	50	105k
KAIROS	0.33	0.33	0.33	0%	0.12	0.08	0.25	1	3
FLASH	0.03	0.03	0.04	19%	0.00	0.00	0.00	24	24k
R-CAID									
ORTHRUS	0.09	0.06	0.12	25%	0.04	0.00	0.10	1	9
VELOX	0.45	0.25	0.61	31%	0.34	0.21	0.38	8	13
E5-THEIA									
SIGL									
THREATRACE	0.05	0.05	0.05	2%	0.00	0.00	0.00	64	722k
NODLINK	0.00	0.00	0.00	0%	0.00	0.00	0.00	36	38k
MAGIC	0.20	0.00	1.00	199%	0.00	0.00	0.00	68	445k
KAIROS	0.17	0.01	0.50	112%	0.03	0.00	0.08	2	23
FLASH	0.01	0.01	0.02	44%	0.00	0.00	0.00	31	311k
R-CAID									
ORTHRUS	0.70	0.50	1.00	31%	0.39	0.33	0.50	1	1
VELOX	1.00	1.00	1.00	0%	1.00	1.00	1.00	2	0

Table 5: Detection results on E5 datasets. Rows in gray indicate timed-out or failed experiments.

System	ADP Mean	ADP Min	ADP Best	$\tilde{\sigma}_{ADP}$	Precision Mean	Precision Min	Precision Best	TP Best	FP Best
H051-OpTC									
SIGL									
THREATRACE	0.57	0.03	1.00	66%	0.00	0.00	0.00	114	1.5M
NODLINK	0.20	0.00	1.00	197%	0.00	0.00	0.00	20	102k
MAGIC	0.37	0.20	0.50	30%	0.00	0.00	0.00	102	259k
KAIROS	0.50	0.11	1.00	81%	0.14	0.00	0.33	1	2
FLASH	0.14	0.03	0.50	130%	0.00	0.00	0.00	2	9.2k
R-CAID									
ORTHRUS	0.68	0.06	1.00	59%	0.08	0.00	0.20	1	4
VELOX	0.57	0.11	1.00	64%	0.12	0.00	0.50	1	1
H201-OpTC									
SIGL									
THREATRACE	0.13	0.12	0.17	12%	0.00	0.00	0.00	2.9k	1.4M
NODLINK	0.32	0.08	1.00	108%	0.02	0.02	0.02	1.4k	78k
MAGIC	0.87	0.33	1.00	30%	0.00	0.00	0.00	1.1k	1.2M
KAIROS	0.75	0.25	1.00	42%	0.38	0.00	1.00	1	0
FLASH	1.00	1.00	1.00	0%	0.00	0.00	0.01	1.5k	239k
R-CAID									
ORTHRUS	0.48	0.23	1.00	57%	0.14	0.00	0.25	1	3
VELOX	0.90	0.50	1.00	22%	0.18	0.11	0.25	1	3
H501-OpTC									
SIGL									
THREATRACE	0.62	0.25	1.00	52%	0.01	0.00	0.05	2	42
NODLINK	0.03	0.01	0.08	82%	0.00	0.00	0.00	355	102k
MAGIC	1.00	1.00	1.00	0%	0.00	0.00	0.00	185	260k
KAIROS	0.38	0.09	1.00	82%	0.12	0.00	0.33	1	2
FLASH	0.58	0.14	1.00	62%	0.00	0.00	0.00	222	215k
R-CAID									
ORTHRUS	0.25	0.14	0.33	29%	0.11	0.04	0.20	1	4
VELOX	0.39	0.20	0.50	34%	0.13	0.00	0.20	1	4

Table 6: Detection results on OpTC datasets. Rows in gray indicate timed-out or failed experiments.

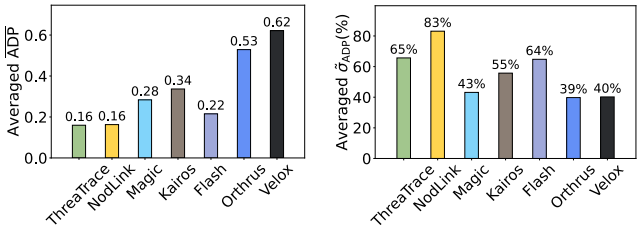
errors. As discussed in §3, this could be attributed to our own re-implementation.

Predictive capability and post-thresholding performance. While the precision scores reported in Table 4 and Table 5 align with those in the most recent study by Jiang et al. [80], the ADP scores are notably higher than one would anticipate. The underwhelming precision scores of certain systems can be attributed to suboptimal threshold selection. Better threshold selection should be a focus of future study.

VELOX's surprising performance. The simple linear layer used by ■ VELOX achieves remarkable performance, matching or exceeding the best ADP scores of significantly more complex systems on eight out of nine datasets (excluding `H501-OpTC`). With best-case ADP scores of 0.5 or higher across all datasets, ■ VELOX demonstrates that detecting the attacks in these datasets does not require modeling graph patterns—a simple text embedding fed into a neural network is sufficient. This unexpected result challenges the prevailing trend in the literature, which often favors increasingly complex GNN architectures.

Inconsistent performance across datasets. Fig. 12a highlights ■ ORTHRUS and ■ VELOX as clear leaders in predictive capabilities across all datasets. However, they underperform in certain scenarios, which poses challenges for practical deployment. Specifically, we ignore the conditions under which these systems may fail to perform optimally. The results presented in Table 6 are particularly concerning for the community, especially given the infrequent use of the DARPA OpTC dataset in past evaluations. This issue is further discussed in §6.

High instability. The significant instability exhibited by all



(a) Overall averaged $\overline{ADP}$. (b) Overall averaged $\tilde{\sigma}_{ADP}$.

Figure 12: $\overline{ADP}$ and $\tilde{\sigma}_{ADP}$ average across five iterations for all datasets.

systems (Fig. 12b) presents a major obstacle to the practical deployment of PIDSs. This issue is likely rooted in the unsupervised learning techniques currently prevalent in the literature. We hypothesize that incorporating methods like negative sampling could potentially improve performance. However, our experience indicates that applying such techniques in a naive manner results in degraded predictive capabilities. This highlights the need for further exploration and refinement of learning methodologies.

Computational cost VS detection performance. Fig. 13 shows $\overline{ADP}$ as a function of different metrics representing computational cost. ■ VELOX shows the best cost/performance trade-off on most datasets. We think such figures are particularly important to assess the effectiveness and practicality of PIDSs and encourage their inclusion in future work.

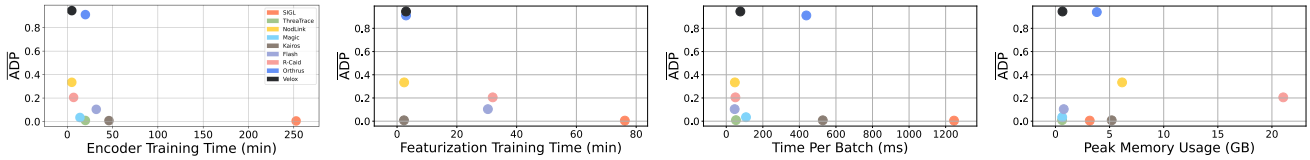
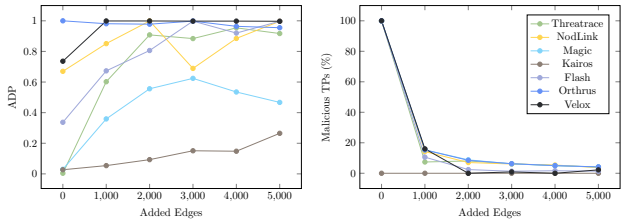


Figure 13: Evolution of $\overline{\text{ADP}}$ with respect to runtime and memory metrics on E3-CADETS.



(a) ADP score against adversarial attack.

(b) Percentage of TPs directly related to attack activities.

Figure 14: Performance of tuned systems against the adversarial attack on E3-CADETS.

5.2 Adversarial Robustness

Previous work [62, 66, 80] evaluates the robustness of detection systems using the methodology developed by Goya et al. [89]. Leveraging the publicly available code, we manipulate the graph to execute an evasion attack on PIDSs. The graph manipulation inserts graph elements to create deceptive similarities between the node neighborhood distributions in the attack subgraph and those in benign provenance subgraphs. The results of these experiments are presented in Fig. 14.

A comparison of Fig. 14a and Fig. 14b reveals key insights on how the attacks proposed by Goya et al. affects PIDSs. Most systems successfully detect the spurious activity introduced by the attacker-controlled process, as evidenced by a stable or rising ADP in Fig. 14a. However, post-thresholding alerts are frequently dominated (see Fig. 14b) by these spurious, benign-looking activities, which could potentially mislead cyber analysts. Despite this, the attack fails to suppress alerts and, in some cases, even increases their frequency. While the attack proposed by Goya et al. has been shown to successfully target older PIDSs [56, 57, 71], state-of-the-art systems exhibit at least partial robustness against this type of attack.

5.3 Achieving Real-Time Detection

■ VELOX is inherently optimized for real-time detection, as it requires only the text embeddings of the two endpoint nodes and the edge type for inference on incoming edges. This design not only enables ■ VELOX to function as a real-time PIDS but also significantly reduces memory consumption by eliminating the need to store entire graphs or deep network layers in memory. The lightweight architecture of ■ VELOX allows it to operate efficiently on CPUs, unlike other systems that rely on GPUs for effective inference.[3] This advantage enhances ■ VELOX's practicality and deployability on client devices with limited computational resources.

To assess the feasibility of deploying ■ VELOX in practical scenarios on a host device, we conduct an evaluation in a real-time detection setting and show the results in Fig. 15.

Memory usage. As illustrated in Fig. 15a, the memory usage on the host device remains constant during activity peaks and do not exceed 5.7MB, significantly below the 160MB maximum memory footprint reported by Dong et al. [81]. The memory consumption remains indeed stable because the input size, represented by the event edge, does not vary. The overall limited memory footprint is primarily due to the need to store only the neural network weights, the two node embeddings, and general process-related variables.

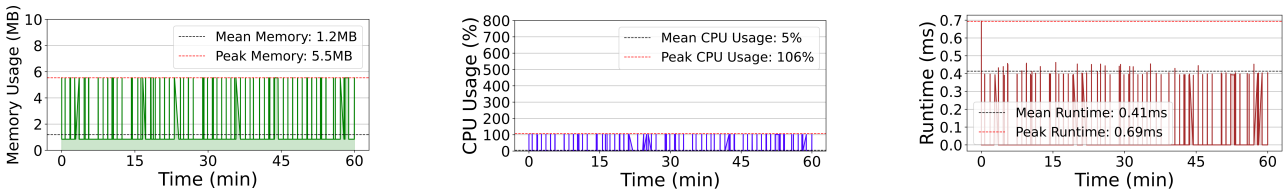
Runtime overhead. As shown in Fig. 15b, the CPU usage peaks at 107% indicating that a single core is sufficient during inference activity. With modern multi-core CPU architectures, ■ VELOX's operations leave most cores available for other user tasks—even in the worst-case scenario during inference. Further, when considering idle periods, ■ VELOX utilizes only 5% of a single CPU. This satisfies the average maximum of 5.2% overhead requirements reported by Dong et al. [81].

Inference time. Fig. 15c shows that ■ VELOX achieves an inference runtime of approximately 41 ms per edge, or roughly 2,400 edges/s. For comparison, for all datasets used in this study the peak is reached on E5-CADETS with 1,832 edges per second, significantly below ■ VELOX's limit. This leaves sufficient capacity for post-processing tasks such as real-time analysis and attack reconstruction.

6 Discussion & Future Work

Sometimes simpler is better. While security researchers do not routinely preregister [119] their experiments, had we done so our final results would not have matched our initial hypothesis. We initially hypothesized that combining various "tricks" and optimizations from the literature would surpass state-of-the-art systems. However, our findings reveal that a relatively simple neural network outperforms most existing

[3] We note that all evaluations in this paper have been performed on machines with high-end NVIDIA GA100 GPUs. All GNN-based systems will see their performance significantly degraded on lower-end machines.



(a) RAM usage during inference (b) CPU usage (8-core CPU) (c) Inference time

Figure 15: Resource consumption of ■ VELOX in a real-time detection setting on E3-CADETS.

methods. Better understanding the conditions under which (and why) this simpler design can outperform graph-aware detection techniques need further exploration.

The need for interpretability. We were often left speculating about the reasons behind a given system's performance, which fundamentally stems from the lack of interpretability in current PIDSs. While the decisions made by VELOX are relatively straightforward to interpret—anomalies can be traced back to pairwise interactions between system objects (e.g., files, processes, sockets, etc.)—the same cannot be said for GNN-based systems. GNNs depend on complex message passing and network effects, making their interpretability a significant challenge [120]. Moreover, developing interpretable models could simplify security analysts' work [121].

Taking a closer look at capture mechanisms. Tables 4, 5, and 6 highlight that the performance of a given system can vary significantly across datasets (e.g., ■ KAIROS exhibits low ADP on E3-CADETS but performs relatively well on E3-CLEARSCOPE and E3-THEIA). At least four dimensions differ across datasets: (1) the operating systems being monitored, (2) the benign activities[4], (3) the malicious activities, and (4) how the provenance graph represents system execution[5]. The community has long suspected [123] that the way a capture mechanism represents activity impacts downstream detection performance. However, current datasets do not allow for disentangling the effects of these four dimensions. We hypothesize that co-designing capture and detection mechanisms could significantly improve performance.

Robustness to adversarial manipulation. Goya et al. [89] were the first to attempt evading PIDSs. While effective against older systems, their approach falls short against state-of-the-art systems (see §5.2). We attempted to evaluate robustness using ProvNinja [84] but could not due to its incomplete open-source release. Key datasets are missing, and the documentation is limited. For instance, generating conspicuous events involves selecting the k least regular events, but neither the paper nor the code specifies how k is determined. Further, the authors indicate that the DARPA TC dataset has significant limitations, including short duration, scripted activities, and a limited selection of user-facing applications. Exploring evasion techniques against PIDSs remains an open challenge.

Beware of computational cost. Training complex GNN models is expensive. Even when minimizing extraneous computation (see §3), the experiments backing our finding required 453 days of compute time. Chief contributor to this cost are hyperparameter tuning cost (see SC4) and instability measurement (see SC5). The research cost may be prohibitive outside of well funded labs. For the sustainability and equitable access of the PIDS subfield, we echo Jiang et al. call for agreed upon ground truth and standardized evaluation methodology [80]. Focus should be on comparing recent systems with optimal detection performance and cost trade-offs.

7 Conclusion

To the best of our knowledge, this paper represents the most comprehensive study of provenance-based intrusion detection systems to date, encompassing a total runtime of 453 days. We identified key shortcomings in the existing literature and demonstrated that a simple neural network can outperform state-of-the-art systems in eight out of nine well-established benchmarks. We aim to inspire efforts in designing more effective and practical PIDSs and hope that our findings contribute to improving evaluation methodologies in future research.

8 Compliance with the Open Science Policy

We open-source our framework (see §3), including VELOX and the code to run all baselines (except for SIGL). The artifact's permanent link is https://zenodo.org/records/15603122. The latest version of the framework is available at: https://github.com/ubc-provenance/PIDSMaker. All code, ground truth data, and instructions for reproducing the experiments presented in this paper are available at: https://github.com/ubc-provenance/PIDSMaker/tree/velox.

9 Ethics Considerations

We have considered the risks and benefits of our research [124] and, to the best of our knowledge, it raises no

[4]The extent to which the quality of benign activities was considered during dataset design remains unclear. Repetitive or unrealistic benign behavior can inflate detection performance.

[5]Chan et al. [122] demonstrated that different capture systems can represent the same event in significantly different ways.

ethical concerns. All experiments use publicly available, ethically collected datasets that contain no sensitive information.

Acknowledgments

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. Research reported in this publication was supported by an Amazon Research Award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of Amazon.

References

[1] Kyu Hyung Lee, X. Zhang, and Dongyan Xu. High Accuracy Attack Provenance via Binary-based Execution Partition. In *Network and Distributed System Security Symposium (NDSS'13)*. The Internet Society, 2013.

[2] Shiqing Ma, X. Zhang, and Dongyan Xu. ProTracer: Towards Practical Provenance Tracing by Alternating Between Logging and Tainting. In *Network and Distributed System Security Symposium (NDSS'16)*. The Internet Society, 2016.

[3] Azadeh Tabiban, Heyang Zhao, et al. ProvTalk: Towards Interpretable Multi-level Provenance Analysis in Networking Functions Virtualization (NFV). In *Network and Distributed System Security Symposium (NDSS'22)*. The Internet Society, 2022.

[4] Adam Bates, Dave Jing Tian, et al. Trustworthy Whole-System Provenance for the Linux Kernel. In *Security Symposium (USENIX Sec'15)*. USENIX, 2015.

[5] Yang Ji, Sangho Lee, et al. Enabling Refinable Cross-Host Attack Investigation with Efficient Data Flow Tagging and Tracking. In *Security Symposium (USENIX Sec'18)*. USENIX, 2018.

[6] Xutong Chen, Hassaan Irshad, et al. CLARION: Sound and Clear Provenance Tracking for Microservice Deployments. In *Security Symposium (USENIX Sec'21)*. USENIX, 2021.

[7] Pubali Datta, Isaac Polinsky, et al. ALASTOR: Reconstructing the Provenance of Serverless Intrusions. In *Security Symposium (USENIX Sec'22)*. USENIX, 2022.

[8] Viet Tung Hoang, Cong Wu, and Xin Yuan. Faster Yet Safer: Logging System Via Fixed-Key Blockcipher. In *Security Symposium (USENIX Sec'22)*. USENIX, 2022.

[9] Heng Yin, Dawn Xiaodong Song, et al. Panorama: Capturing System-wide Information Flow for Malware Detection and Analysis. In *Conference on Computer and Communications Security (CCS'07)*. ACM, 2007.

[10] Yang Ji, Sangho Lee, et al. RAIN: Refinable Attack Investigation with On-demand Inter-Process Information Flow Tracking. *Conference on Computer and Communications Security (CCS'17)*, 2017.

[11] Riccardo Paccagnella, Kevin Liao, et al. Logging to the Danger Zone: Race Condition Attacks and Defenses on System Audit Frameworks. In *Conference on Computer and Communications Security (CCS'20)*. ACM, 2020.

[12] Jun Zeng, Chuqi Zhang, and Zhenkai Liang. PalanTír: Optimizing Attack Provenance with Hardware-enhanced System Observability. In *Conference on Computer and Communications Security (CCS'22)*. ACM, 2022.

[13] R. C. 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, 2024.

[14] Kiran-Kumar Muniswamy-Reddy, David A. Holland, et al. Provenance-aware storage systems. In *Annual Technical Conference (ATC'06)*. USENIX, 2006.

[15] Kiran-Kumar Muniswamy-Reddy, Uri Braun, et al. Layering in provenance systems. In *Annual Technical Conference (ATC'09)*. USENIX, 2009.

[16] Ashish Gehani and Dawood Tariq. Spade: Support for provenance auditing in distributed environments. In *International Middleware Conference (Middleware'12)*. USENIX/ACM/IFIP, 2012.

[17] Devin J. Pohly, Stephen E. McLaughlin, et al. Hi-fi: collecting high-fidelity whole-system provenance. In *Asia-Pacific Computer Systems Architecture Conference (ACSAC'12)*, 2012.

[18] Manolis Stamatogiannakis, Paul T. Groth, and Herbert Bos. Looking inside the black-box: Capturing data provenance using dynamic instrumentation. In *International Provenance and Annotation Workshop (IPAW'14)*. Springer, 2014.

[19] Manolis Stamatogiannakis, Paul T. Groth, and Herbert Bos. Decoupling provenance capture and analysis from execution. In *Workshop on the Theory and Practice of Provenance (TaPP'15)*. USENIX, 2015.

[20] Yang Ji, Sangho Lee, and Wenke Lee. RecProv: Towards Provenance-Aware User Space Record and Replay. In *International Provenance and Annotation Workshop (IPAW'16)*. Springer, 2016.

[21] Jörg Thalheim, Pramod Bhatotia, and Christof Fetzer. INSPECTOR: Data Provenance Using Intel Processor Trace (PT). In *International Conference on Distributed Computing Systems (ICDCS'16)*. IEEE, 2016.

[22] Thomas Pasquier, Xueyuan Han, et al. Practical whole-system provenance capture. In *Symposium on Cloud Computing (SoCC'17)*. ACM, 2017.

[23] Fei Wang, Yonghwi Kwon, et al. Lprov: Practical library-aware provenance tracing. In *Annual Computer Security Applications Conference (ACSAC'18)*, 2018.

[24] Sheung Chi Chan, James Cheney, et al. ProvMark: A Provenance Expressiveness Benchmarking System. In *International Middleware Conference (Middleware'19)*. USENIX/ACM/IFIP, 2019.

[25] Soo Yee Lim, Bogdan Stelea, et al. Secure Namespaced Kernel Audit for Containers. In *Symposium on Cloud Computing (SoCC'21)*. ACM, 2021.

[26] Sadegh M Milajerdi, Rigel Gjomemo, et al. HOLMES: Real-time APT Detection through Correlation of Suspicious Information Flows. In *Symposium on Security and Privacy (S&P'19)*. IEEE, 2019.

[27] Yonghwi Kwon, Fei Wang, et al. MCI : Modeling-based Causality Inference in Audit Logging for Attack Investigation. In *Network and Distributed System Security Symposium (NDSS'18)*. The Internet Society, 2018.

[28] Yushan Liu, Mu Zhang, et al. Towards a Timely Causality Analysis for Enterprise Security. In *Network and Distributed System Security Symposium (NDSS'18)*. The Internet Society, 2018.

[29] Wajih Ul Hassan, Shengjian Guo, et al. NoDoze: Combatting Threat Alert Fatigue with Automated Provenance Triage. In *Network and Distributed System Security Symposium (NDSS'19)*. The Internet Society, 2019.

[30] Wajih Ul Hassan, Mohammad A. Noureddine, et al. OmegaLog: High-Fidelity Attack Investigation via Transparent Multi-layer Log Analysis. In *Network and Distributed System Security Symposium (NDSS'20)*. The Internet Society, 2020.

[31] Runqing Yang, Shiqing Ma, et al. UIScope: Accurate, Instrumentation-free, and Visible Attack Investigation for GUI Applications. In *Network and Distributed System Security Symposium (NDSS'20)*. The Internet Society, 2020.

[32] Le Yu, Shiqing Ma, et al. 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.

[33] Jun Zeng, Zheng Leong Chua, et al. WATSON: Abstracting Behaviors from Audit Logs via Aggregation of Contextual Semantics. In *Network and Distributed System Security Symposium (NDSS'21)*. The Internet Society, 2021.

[34] Shiqing Ma, Juan Zhai, et al. MPI: Multiple Perspective Attack Investigation with Semantic Aware Execution Partitioning. In *Security Symposium (USENIX Sec'17)*. USENIX, 2017.

[35] Peng Gao, Xusheng Xiao, et al. Saql: A Stream-based Query System for Real-Time Abnormal System Behavior Detection. In *Security Symposium (USENIX Sec'18)*. USENIX, 2018.

[36] Abdulellah Alsaheel, Yuhong Nan, et al. ATLAS: A Sequence-based Learning Approach for Attack Investigation. In *Security Symposium (USENIX Sec'21)*. USENIX, 2021.

[37] Pengcheng Fang, Peng Gao, et al. Back-Propagating System Dependency Impact for Attack Investigation. In *Security Symposium (USENIX Sec'22)*. USENIX, 2022.

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

[39] Enes Altinisik, Fatih Deniz, and Husrev 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.

[40] Wajih Ul Hassan, Adam Bates, and Daniel Marino. Tactical Provenance Analysis for Endpoint Detection and Response Systems. In *Symposium on Security and Privacy (S&P'20)*. IEEE, 2020.

[41] Thijs van Ede, H. Aghakhani, et al. DEEPCASE: Semi-Supervised Contextual Analysis of Security Events. In *Symposium on Security and Privacy (S&P'22)*. IEEE, 2022.

[42] Mahmood Sharif, Pubali Datta, et al. DrSec: Flexible Distributed Representations for Efficient Endpoint Security. In *Symposium on Security and Privacy (S&P'24)*. IEEE, 2024.

[43] Samuel T. King and Peter M. Chen. Backtracking Intrusions. *ACM Transactions on Computer Systems*, 2003.

[44] Kexin Pei, Zhongshu Gu, et al. HERCULE: Attack Story Reconstruction via Community Discovery on Correlated Log Graph. In *Annual Computer Security Applications Conference (ACSAC'16)*, 2016.

[45] Sadegh M. Milajerdi, Birhanu Eshete, et al. ProPatrol: Attack Investigation via Extracted High-Level Tasks. In *International Conference on Information Systems Security (ICISS'18)*. Springer, 2018.

[46] Peng Gao, Xusheng Xiao, et al. AIQL: Enabling Efficient Attack Investigation from System Monitor-

ing Data. In *Annual Technical Conference (ATC'18)*. USENIX, 2018.

[47] Frank Capobianco, Rahul George, et al. Employing Attack Graphs for Intrusion Detection. In *New Security Paradigms Workshop (NSPW'19)*. ACM, 2019.

[48] Jiaping Gui, Ding Li, et al. APTrace: A Responsive System for Agile Enterprise Level Causality Analysis. In *International Conference on Data Engineering (ICDE'20)*. IEEE, 2020.

[49] Peng Gao, Fei Shao, et al. Enabling Efficient Cyber Threat Hunting With Cyber Threat Intelligence. *International Conference on Data Engineering (ICDE'21)*, 2021.

[50] Yushan Liu, Xiaokui Shu, et al. RAPID: Real-Time Alert Investigation with Context-aware Prioritization for Efficient Threat Discovery. In *Annual Computer Security Applications Conference (ACSAC'22)*, 2022.

[51] Xiaoyan Sun, Jun Dai, et al. Using Bayesian Networks for Probabilistic Identification of Zero-Day Attack Paths. *IEEE Transactions on Information Forensics and Security*, 2018.

[52] Yulai Xie, Yafeng Wu, et al. P-Gaussian: Provenance-Based Gaussian Distribution for Detecting Intrusion Behavior Variants Using High Efficient and Real Time Memory Databases. *IEEE Transactions on Dependable and Secure Computing*, 2021.

[53] Atmane Ayoub Mansour Bahar, Kamel Soaïd Ferrahi, et al. FedHE-Graph: Federated Learning with Hybrid Encryption on Graph Neural Networks for Advanced Persistent Threat Detection. In *International Conference on Availability, Reliability and Security (ARES'24)*. SBA Research, 2024.

[54] Mathieu Barré, Ashish Gehani, and Vinod Yegneswaran. Mining Data Provenance to Detect Advanced Persistent Threats. In *Workshop on the Theory and Practice of Provenance (TaPP'19)*. USENIX, 2019.

[55] Sanjeev Das, Yang Liu, et al. Semantics-Based Online Malware Detection: Towards Efficient Real-Time Protection Against Malware. *IEEE Transactions on Information Forensics and Security*, 2016.

[56] Xueyuan Han, Thomas Pasquier, et al. Unicorn: Runtime Provenance-Based Detector for Advanced Persistent Threats. In *Network and Distributed System Security Symposium (NDSS'20)*. The Internet Society, 2020.

[57] Qi Wang, Wajih Ul Hassan, et al. 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.

[58] Xueyuan Han, Xiao Yu, et al. SIGL: Securing Software Installations Through Deep Graph Learning. In *Security Symposium (USENIX Sec'21)*. USENIX, 2021.

[59] Su Wang, Zhiliang Wang, et al. Threatrace: Detecting and tracing host-based threats in node level through provenance graph learning. *IEEE Transactions on Information Forensics and Security*, 17:3972–3987, 2021.

[60] Shaofei Li, Feng Dong, et al. 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.

[61] Zian Jia, Yun Xiong, et al. MAGIC: Detecting Advanced Persistent Threats via Masked Graph Representation Learning. In *Security Symposium (USENIX Sec'24)*. USENIX, 2024.

[62] Zijun Cheng, Qiujian Lv, et al. Kairos: Practical Intrusion Detection and Investigation using Whole-system Provenance. In *Symposium on Security and Privacy (S&P'24)*. IEEE, 2023.

[63] Lingzhi Wang, Xiangmin Shen, et al. Incorporating Gradients to Rules: Towards Lightweight, Adaptive Provenance-based Intrusion Detection. *arXiv preprint arXiv:2404.14720*, 2024.

[64] Jun Zengy, Xiang Wang, et al. SHADEWATCHER: Recommendation-guided Cyber Threat Analysis using System Audit Records. In *Symposium on Security and Privacy (S&P'22)*. IEEE, 2022.

[65] F. Yang, Jiacen Xu, et al. PROGRAPHER: An Anomaly Detection System based on Provenance Graph Embedding. In *Security Symposium (USENIX Sec'23)*. USENIX, 2023.

[66] 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.

[67] 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.

[68] Isaiah J. King and Huimin Huang. Euler: Detecting Network Lateral Movement via Scalable Temporal Graph Link Prediction. In *Network and Distributed System Security Symposium (NDSS'22)*. The Internet Society, 2022.

[69] Md Nahid Hossain, Sadegh M. Milajerdi, et al. SLEUTH: Real-time Attack Scenario Reconstruction from COTS Audit Data. In *Security Symposium (USENIX Sec'17)*. USENIX, 2017.

[70] Yulai Xie, Dan Feng, et al. Unifying intrusion detection and forensic analysis via provenance awareness. *Future Generation of Computer Systems*, 61:26–36, 2016.

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

[72] Xueyuan Han, Thomas Pasquier, et al. FRAPpuccino: Fault-detection through Runtime Analysis of Provenance. In *Workshop on Hot Topics in Cloud Computing (HotCloud'17)*. USENIX, 2017.

[73] Mark Lemay, Wajih Ul Hassan, et al. Automated Provenance Analytics: A Regular Grammar Based Approach with Applications in Security. In *Workshop on the Theory and Practice of Provenance (TaPP'17)*. USENIX, 2017.

[74] Yulai Xie, Dan Feng, et al. Pagoda: A hybrid approach to enable efficient real-time provenance based intrusion detection in big data environments. *IEEE Transactions on Dependable and Secure Computing*, 17:1283–1296, 2020.

[75] Maya Kapoor, Joshua Melton, et al. PROV-GEM: Automated Provenance Analysis Framework using Graph Embeddings. In *International Conference on Machine Learning and Applications (ICMLA'21)*. IEEE, 2021.

[76] Isaiah J. King, Xiaokui Shu, et al. EdgeTorrent: Real-time Temporal Graph Representations for Intrusion Detection. In *International Symposium on Research in Attacks, Intrusions and Defenses (RAID'23)*, 2023.

[77] Lingxiang Meng, Rongrong Xi, et al. PG-AID: An Anomaly-based Intrusion Detection Method Using Provenance Graph. In *International Conference on Computer Supported Cooperative Work in Design (CSCWD'24)*. IEEE, 2024.

[78] Boyuan Xu, Yiru Gong, et al. ProcSAGE: an efficient host threat detection method based on graph representation learning. *Springer Cybersecurity*, 2024.

[79] Gbadebo Ayoade, Khandakar Ashrafi Akbar, et al. Evolving Advanced Persistent Threat Detection using Provenance Graph and Metric Learning. In *Conference on Communications and Network Security (CNS'20)*. IEEE, 2020.

[80] Baoxiang Jiang, Tristan Bilot, et al. ORTHRUS: Achieving High Quality of Attribution in Provenance-based Intrusion Detection Systems. In *Security Symposium (USENIX Sec'25)*. USENIX, 2025.

[81] Feng Dong, Shaofei Li, et al. 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.

[82] Wajih Ul Hassan, Mark Lemay, et al. Towards Scalable Cluster Auditing through Grammatical Inference over Provenance Graphs. In *Network and Distributed System Security Symposium (NDSS'18)*. The Internet Society, 2018.

[83] Hailun Ding, Juan Zhai, et al. The Case for Learned Provenance Graph Storage Systems. In *Security Symposium (USENIX Sec'23)*. USENIX, 2023.

[84] Kunal Mukherjee, Joshua Wiedemeier, et al. Evading Provenance-Based ML Detectors with Adversarial System Actions. In *Security Symposium (USENIX Sec'23)*. USENIX, 2023.

[85] Dongqi Han, Zhiliang Wang, et al. DeepAID: Interpreting and Improving Deep Learning-based Anomaly Detection in Security Applications. In *Conference on Computer and Communications Security (CCS'21)*. ACM, 2021.

[86] Min Du, Feifei Li, et al. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In *Conference on Computer and Communications Security (CCS'17)*. ACM, 2017.

[87] Thomas Pasquier, Xueyuan Han, et al. Runtime Analysis of Whole-System Provenance. In *Conference on Computer and Communications Security (CCS'18)*. ACM, 2018.

[88] Fucheng Liu, Yu Wen, et al. Log2vec: A Heterogeneous Graph Embedding Based Approach for Detecting Cyber Threats within Enterprise. In *Conference on Computer and Communications Security (CCS'19)*. ACM, 2019.

[89] Akul Goyal, Xueyuan Han, et al. 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.

[90] Md Nahid Hossain, S. Sheikhi, and R. C. Sekar. Combating Dependence Explosion in Forensic Analysis Using Alternative Tag Propagation Semantics. In *Symposium on Security and Privacy (S&P'20)*. IEEE, 2020.

[91] Zhiqiang Xu, Pengcheng Fang, et al. DEPCOMM: Graph Summarization on System Audit Logs for Attack Investigation. In *Symposium on Security and Privacy (S&P'22)*. IEEE, 2022.

[92] Adam Bates, Dave Jing Tian, et al. Taming the costs of trustworthy provenance through policy reduction. *ACM Transactions on Internet Technology (TOIT)*, 17:1 – 21, 2017.

[93] Shiqing Ma, Juan Zhai, et al. Kernel-Supported Cost-Effective Audit Logging for Causality Tracking. In *Annual Technical Conference (ATC'18)*. USENIX, 2018.

[94] Noor Michael, Jaron Mink, et al. On the forensic validity of approximated audit logs. In *Annual Computer Security Applications Conference (ACSAC'20)*, 2020.

[95] Wajih Ul Hassan, Ding Li, et al. 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)*, 2020.

[96] Xingzi Yuan, Omid Setayeshfar, et al. DroidForensics: Accurate Reconstruction of Android Attacks via Multi-layer Forensic Logging. In *Asia Conference on Computer and Communications Security (AsiaCCS'17)*. ACM, 2017.

[97] Muhammad Adil Inam, Yinfang Chen, et al. Sok: History is a vast early warning system: Auditing the provenance of system intrusions. In *Symposium on Security and Privacy (S&P'23)*. IEEE, 2023.

[98] Tristan Bilot, Nour El Madhoun, et al. Graph neural networks for intrusion detection: A survey. *IEEE Access*, 11:49114–49139, 2023.

[99] Pooneh Nikkhah Bahrami, Ali Dehghantanha, et al. Cyber kill chain-based taxonomy of advanced persistent threat actors: Analogy of tactics, techniques, and procedures. *Journal of information processing systems*, 15(4):865–889, 2019.

[100] Fernando Maymí, Robert Bixler, et al. Towards a definition of cyberspace tactics, techniques and procedures. In *2017 IEEE international conference on big data (big data)*, pages 4674–4679. IEEE, 2017.

[101] Transparent Computing Engagement 3 Data Release, Accessed 11th July 2025. https://github.com/darpa-i2o/Transparent-Computing/blob/master/README-E3.md.

[102] Transparent Computing Engagement 5 Data Release, Accessed 11th July 2025. https://github.com/darpa-i2o/Transparent-Computing.

[103] Mike van Opstal and William Arbaugh. Operationally Transparent Cyber (OpTC) Data Release, 2019. https://github.com/FiveDirections/OpTC-data.

[104] Talha Abrar, Ahmad Shamail, et al. On the Reproducibility of Provenance-based Intrusion Detection that uses Deep Learning. In *Conference on Reproducibility and Replicability (REP'25)*. ACM, 2025.

[105] Sathya Chandran Sundaramurthy, Alexandru G Bardas, et al. A human capital model for mitigating security analyst burnout. In *Symposium on Usable Privacy and Security (SOUPS'15)*. USENIX, 2015.

[106] Aron Laszka, Waseem Abbas, et al. Optimal thresholds for intrusion detection systems. In *Symposium and Bootcamp on the Science of Security (HotSoS'16)*, 2016.

[107] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *International Conference on Machine learning (ICML'06)*, pages 233–240, 2006.

[108] Daniel Arp, Erwin Quiring, et al. Dos and Don'ts of Machine Learning in Computer Security. In *Security Symposium (USENIX Sec'22)*. USENIX, 2022.

[109] Vladimir Cherkassky and Filip M Mulier. *Learning from data: concepts, theory, and methods*. John Wiley & Sons, 2007.

[110] Roberto Jordaney, Kumar Sharad, et al. Transcend: Detecting Concept Drift in Malware Classification Models. In *Security Symposium (USENIX Sec'17)*. USENIX, 2017.

[111] Jake Topping, Francesco Di Giovanni, et al. Understanding over-squashing and bottlenecks on graphs via curvature. In *International Conference on Learning Representations (ICLR'22)*, 2022.

[112] Srinadh Bhojanapalli, Kimberly Wilber, et al. On the Reproducibility of Neural Network Predictions. *arXiv preprint arXiv:2102.03349*, 2021.

[113] Cecilia Summers and Michael J Dinneen. Nondeterminism and Instability in Neural Network Optimization. In *International Conference on Machine Learning (ICML'21)*, 2021.

[114] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. *Conference on Neural Information Processing System (NeurIPS'17)*, 2017.

[115] Haodong Li, Guosheng Xu, et al. MalCertain: Enhancing Deep Neural Network Based Android Malware Detection by Tackling Prediction Uncertainty. In *International Conference on Software Engineering (ICSE'24)*. IEEE/ACM, 2024.

[116] Deqiang Li, Tian Qiu, et al. Can We Leverage Predictive Uncertainty to Detect Dataset Shift and Adversarial Examples in Android Malware Detection? In *Annual Computer Security Applications Conference (ACSAC'21)*, 2021.

[117] Thomas N Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR'17)*, 2017.

[118] Emanuele Rossi, Ben Chamberlain, et al. Temporal Graph Networks for Deep Learning on Dynamic Graphs. *arXiv preprint arXiv:2006.10637*, 2020.

Statistics	E3-CADETS	E3-THEIA	E3-CLEARSCOPE
# Nodes	643,486	1,248,267	291,347
# Edges	5,688,416	11,651,551	1,398,858
# Attacks	3	2	1
# Malicious nodes per attack	(8 - 43 - 24)	(61 - 58)	(41)
# Malicious edges per attack	(1526 - 31,352 - 3672)	(66,028 - 80,997)	(24,133)
Days of train set	2018-04-**02~05, 07~09**	2018-04-**02~08**	2018-04-**03~05, 07~10**
Days of val set	2018-04-**10**	2018-04-**09**	2018-04-**02**
Days of test set	2018-04-**06, 11~13**	2018-04-**10, 12,13**	2018-04-**11, 12**
Unused days	None	2018-04-**11**	2018-04-**06, 13**
$\bar{x}$ In/Out-degree	8.84	9.33	4.80
$\bar{x}$ Nodes per 15-min	1276.77	4657.46	537.51
$\bar{x}$ Edges per 15-min	5620.96	22,025.62	1693.53

Table 7: DARPA TC E3 dataset statistics.

Statistics	E5-CADETS	E5-THEIA	E5-CLEARSCOPE
# Nodes	12,573,595	2,138,087	381,479
# Edges	151,490,722	45,075,615	54,188,800
# Attacks	2	1	3
# Malicious nodes per attack	(19 - 107)	(70)	(26 - 17 - 8)
# Malicious edges per attack	(59,323 - 8,708,508)	(35,564)	(225,870 - 83 - 38)
Days of train set	2019-05-**08, 09, 11**	2019-05-**08~10**	2019-05-**08~12**
Days of val set	2019-05-**12**	2019-05-**11**	2019-05-**13**
Days of test set	2019-05-**16,17**	2019-05-**14, 15**	2019-05-**14, 15, 17**
Unused days	2019-05-**10, 13~15**	2019-05-**12, 13, 16, 17**	2019-05-**16**
$\bar{x}$ In/Out-degree	12.05	21.08	142.05
$\bar{x}$ Nodes per 15-min	34,479.60	19,260.10	1806.13
$\bar{x}$ Edges per 15-min	161,849.06	52,352.63	58,645.89

Table 8: DARPA TC E5 dataset statistics.

[119] Brian A Nosek, Charles R Ebersole, et al. The preregistration revolution. *Proceedings of the National Academy of Sciences*, 2018.

[120] Yongqiang Chen, Yatao Bian, et al. How Interpretable Are Interpretable Graph Neural Networks? In *International Conference on Machine Learning (ICML'24)*, 2024.

[121] Azqa Nadeem, Daniël Vos, et al. SoK: Explainable Machine Learning for Computer Security Applications. In *European Symposium on Security and Privacy (EuroS&P'23)*. IEEE, 2023.

[122] Sheung Chi Chan, James Cheney, et al. ProvMark: A Provenance Expressiveness Benchmarking System. In *International Middleware Conference*. ACM/USENIX/IFIP, 2019.

[123] Xueyuan Han, James Mickens, et al. Xanthus: Push-button orchestration of host provenance data collection. In *International Workshop on Practical Reproducible Evaluation of Computer Systems*. ACM, 2020.

[124] Tadayoshi Kohno, Yasemin Acar, and Wulf Loh. Ethical Frameworks and Computer Security Trolley Problems: Foundations for Conversations. In *Security Symposium (USENIX Sec'23)*. USENIX, 2023.

A Datasets

Tables 7, 8, and 9 present statistics for the DARPA TC E3, E5, and OpTC datasets. The TC dataset, created during the Red Team vs. Blue Team competition, includes the publicly available E3 and E5 portions. The Red Team simulated three attacker types—"Nation State," "Common Threat," and "Metasploit"—with distinct strategies. E3 spanned two weeks, and the E5 nine days. OpTC records benign activity from 500 Windows hosts over seven days, followed by three days of both benign and APT activity. Each day features an attack targeting a specific host (201, 501 or 051), and all events associated with that host are collected to form each of the three datasets.

Statistics	OPTC-H201	OPTC-H501	OPTC-H051
# Nodes	5,268,452	3,964,099	4,615,935
# Edges	19,605,225	15,783,742	17,760,252
# Attacks	1	1	1
# Malicious nodes per attack	(2,905)	(749)	(114)
# Malicious edges per attack	(300,655)	(324,284)	(383,667)
Days of train set	2019-09-**19~21**	2019-09-**19~21**	2019-09-**19~21**
Days of val set	2019-09-**22**	2019-09-**22**	2019-09-**22**
Days of test set	2019-09-**23 ~25**	2019-09-**23 ~25**	2019-09-**23 ~25**
Unused days	2019-09-**16 ~18**	2019-09-**16 ~18**	2019-09-**16 ~18**
$\bar{x}$ In/Out-degree	3.72	3.98	3.85
$\bar{x}$ Nodes per 15-min	9,702.71	9,288.17	9,384.92
$\bar{x}$ Edges per 15-min	24,629.68	24,778.24	24,196.53

Table 9: OPTC dataset statistics.

	THREATRACE	NODLINK	MAGIC	KAIROS	FLASH	ORTHRUS
Reproduced (pre-trained)	3/8	N/A	3/3	3/7	7/7	N/A
Reproduced (full)	4/8	0/3	3/3	2/7	0/7	5/6
Contacted Authors	Yes	Yes	Yes	Yes	Yes	Yes

Table 10: Results of our reproduction on the DARPA datasets (E3, E5, and OpTC) using pre-trained models (N/A where unavailable) and models trained from scratch. We report the number of succesful reproductions over the total number of DARPA datasets used in the original evaluation.

B Reproduction Details

We provide details of our attempt to reproduce past results (see Table 10) on DARPA datasets (E3, E5, and OpTC) using authors' original evaluation methodologies. We consider a reproduction successful if the results are within 5% of the original. We do not consider a reproduction successful if we are unable to confirm the original system's behavior (e.g., due to missing code or ground truth labels). Overall, reproduction attempts were succesful when all data, code, and labels were available. All authors were contacted.

THREATRACE. The code to reproduce E5 experiments was missing. We could reproduce E3 experiments. The system was not originally evaluated against OpTC.

NODLINK. The code for the E3 experiments was missing. However, we note that we could reproduce results on the publicly released authors' dataset. The system was not originally evaluated against E5 and OpTC.

MAGIC. We could reproduce results for the E3 experiments. The system was not originally evaluated on E5 and OpTC.

KAIROS. We could reproduce the E3 results. The authors' original pipeline errored when generating ground truth files for the E5 datasets. The labels for OpTC were missing.

FLASH. We could reproduce all results using pre-trained models. We obtained lower results when training from scratch. We confirm the issue is present in the original authors' publicly released code.

ORTHRUS. Reproduced results are consistent with the paper with the exception of `E3-CLEARSCOPE` results.