On the Reproducibility of Provenance-based Intrusion Detection that uses Deep Learning

Talha Abrar
University of Arizona
Tucson, Arizona, USA
tabrar@arizona.edu

Ahmad Shamail
University of Arizona
Tucson, Arizona, USA
shamail@arizona.edu

Mohammad Jaffer Iqbal
University of Waterloo
Waterloo, Canada
j2iqbal@uwaterloo.ca

Amaan Ahmed
Lahore University of Management Sciences
Lahore, Pakistan
aman.ahmed259002@gmail.com

Muhammad Abdullah
Lahore University of Management Sciences
Lahore, Pakistan
abdullahastro07@gmail.com

Muhammad Shayan
Lahore University of Management Sciences
Lahore, Pakistan
shayanhanif73@gmail.com

Fareed Zaffar
Lahore University of Management Sciences
Lahore, Pakistan
fareed.zaffar@lums.edu.pk

Thomas Pasquier
University of British Columbia
Vancouver, Canada
tfjmp@cs.ubc.ca

David Eyers
University of Otago
Dunedin, New Zealand
dme@cs.otago.ac.nz

Ashish Gehani*
SRI
Menlo Park, California
USA
ashish.gehani@sri.com

Abstract

As cyber-threats grow in scale and sophistication, intrusion detection systems that incorporate system provenance and deep learning have emerged as a promising direction for detecting advanced persistent threats (APTs). We endeavor to reproduce the experimental results from eight such systems published over the past four years in top-tier research venues. We encountered numerous challenges that obstruct reproducibility, including incomplete or non-functional source code releases, missing documentation, unavailability of datasets or detailed preprocessing steps, and unclear or inconsistent descriptions of experimental procedures. We detail and categorize these challenges to demonstrate the obstacles researchers may encounter when reproducing studies in this domain. Our findings highlight gaps in reaching the ideals of open science in this area of intrusion detection research.

CCS Concepts

• **Security and privacy** → **Intrusion detection systems**; • **Computing methodologies** → *Neural networks.*

Keywords

Intrusion Detection Systems, Deep Learning, Provenance-based Intrusion Detection, Reproducibility

*Talha was responsible for overall coordination. Ahmad, and Jaffer contributed equally as Editors, alongside Talha. Amaan, Abdullah, and Shayan served equally as Evaluators, as defined in Section 3.1.

ACM REP '25, Vancouver, BC, Canada

ACM ISBN 979-8-4007-1958-5/25/07
https://doi.org/10.1145/3736731.3746140

ACM Reference Format:
Talha Abrar, Ahmad Shamail, Mohammad Jaffer Iqbal, Amaan Ahmed, Muhammad Abdullah, Muhammad Shayan, Fareed Zaffar, Thomas Pasquier, David Eyers, and Ashish Gehani. 2025. On the Reproducibility of Provenance-based Intrusion Detection that uses Deep Learning. In *ACM Conference on Reproducibility and Replicability (ACM REP '25), July 29–31, 2025, Vancouver, BC, Canada.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3736731.3746140

1 Introduction

The need for advanced and adaptable intrusion detection systems has grown increasingly urgent in response to escalating threats such as probing, infiltration, lateral movement, data exfiltration, and backdoor insertion by nation-state and economically motivated actors. Stealthy attacks that unfold over extended periods within complex and high-throughput systems present a particular challenge. Detecting such advanced persistent threats (APTs) was a central focus of DARPA's Transparent Computing (TC) program. In the five years since the program's conclusion, the threat landscape has further evolved, with fileless malware increasingly supplanted by sophisticated living-off-the-land (LoTL) techniques [8, 55].

To enhance system observability, DARPA's Transparent Computing (TC) program [15] demonstrated the value of elevating low-level audit event streams into intermediate-level provenance graphs [34]. These graphs explicitly capture causal relationships among temporally distant yet logically connected entities, such as users, processes, and system artifacts. This approach led to the emergence of a new class of provenance-based intrusion detection systems (PIDS), which has since gained considerable traction in the research community (see dedicated surveys of the field [30, 79]). In parallel, industrial deployments of PIDS have shown markedly improved performance over conventional endpoint detection and response (EDR) systems [18].

Recent work has explored the use of deep learning techniques to manage the substantial volumes of graph-structured data generated when monitoring activity across one or more hosts in a network [10, 25, 77]. Although heuristic in nature, these methods offer attractive scalability by eliminating the significant manual effort previously required to explicitly specify correct system behaviors or to formally encode complex security properties. While such specifications were possible before the advent of deep learning–based approaches, they demanded substantial domain expertise and manual effort, limiting their practical scalability [26].

Over the past three years, top-tier security conferences have featured several systems that leverage deep learning over provenance graphs to detect intrusions [5, 10, 21, 25, 35, 70, 73, 77]. The first generation of these systems was designed primarily for offline forensic analysis—that is, they assume full availability of provenance data prior to the start of analysis. For example, SIGL [25] operates on the software installation graph generated after a package manager installs a new application; Atlas [5] begins analyzing logs (from which provenance is inferred) only after a compromise has been detected; ShadeWatcher [77] relies on computationally intensive backward lineage queries during a preprocessing phase, which hinders its ability to process provenance data as it is produced. In scenarios where detection latency is high, the system under attack may suffer confidentiality, integrity, or availability losses before a response can be initiated.

Second-generation systems aim to support online intrusion detection, but many face significant scalability challenges. Systems such as ThreaTrace [73], Kairos [10], and R-CAID [21] maintain in-memory neural embeddings that grow proportionally with the size of the provenance graph, which itself increases with the duration of system monitoring. R-CAID [21], in particular, also relies on lineage queries, similar to earlier offline approaches. Other designs achieve scalability by introducing abstractions that reduce robustness, making them susceptible to adversarial manipulation [20, 53]. For instance, EdgeTorrent [35] constructs sketches over fixed temporal windows, limiting detection granularity to the size of those windows. Flash [70] reuses existing embeddings when new elements differ only slightly from prior ones, potentially allowing a stealthy adversary's subtle deviations to evade detection.

A growing demand for effective solutions in the field of Provenance based Intrusion Detection using Deep Learning (PIDDL) underscores the importance of continued research in this area. Progress depends not only on innovation but also on the ability to meaningfully compare new approaches with prior work. It makes the reproducibility of existing results essential. To this end, we undertook a systematic effort to reproduce the findings of eight recent and influential PIDDL studies published at top-tier security conferences, workshops, and journals. Our investigation reveals significant challenges, including inconsistent levels of transparency and support from original authors, as well as substantial variation in implementation details, platforms, and datasets.

Contributions

- We reproduce the experimental results of eight PIDDL systems and quantify the discrepancies between reported and reproduced outcomes.
- We identify domain-specific properties relevant to assessing reproducibility in PIDDL research.
- We categorize the primary challenges that hinder reproducibility across these systems.
- We highlight instances where engagement with original authors facilitated the resolution of reproducibility issues.

Table 1: Salient aspects of the Provenance Graph Construction stage.

Salient Aspect	System
Stage is Omitted	AirTag
Memory Cache System	ThreaTrace, NodLink
Batch Log Processing	ThreaTrace, NodLink, Flash, Kairos

Table 2: Salient aspects of the Graph Representation and Learning stage.

Salient Aspect		System
Non-GNN Based		Atlas, AirTag, NodLink
GNN-Based	Knowledge Graph	ShadeWatcher
	Multi-Model Framework	ThreaTrace
	Temporal Graph Network	Kairos
	Selective Graph Traversal	Flash
	Masked Graph Learning	Magic

Table 3: Salient aspects of the Detection stage.

Salient Aspect	System
Online Detection Capability	ThreaTrace, NodLink, Flash, Kairos
Captures Long-Running Attacks	NodLink, Kairos
Model Adaptation Feature	ShadeWatcher, Magic
Distinctive Graph Generation	NodLink, Kairos
Requires Attack Symptom	Atlas

2 Typical PIDDL Workflow

Provenance-based intrusion detection systems that incorporate deep learning typically follow a common pipeline comprising three key stages: (1) provenance graph construction, (2) graph representation and learning, and (3) detection. Figure 1 illustrates this pipeline, while Tables 1, 2, and 3 highlight how specific systems diverge from this typical workflow.

Provenance Graph Construction. Systems begin by ingesting provenance data and constructing corresponding graphs. The input format varies depending on the host operating system and the provenance capture tool employed [9]. These graphs serve to normalize heterogeneous input data into a unified representation, allowing subsequent components to operate over a consistent structure. To build the graph, systems map provenance events to kernel

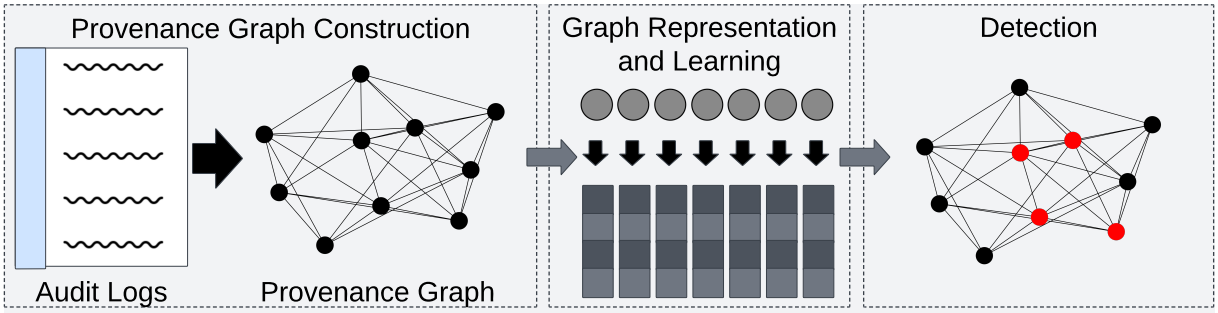

Figure 1: Three key stages in a PIDDL pipeline

Table 4: Summary of findings. *[pre-trained, scratch models]/total datasets; ✗= no pre-trained models provided. **✓✓= Both train/test and CPU/memory reported; ✓= Only test/memory reported; ✗= Neither reported. †Atlas and NodLink evaluate each dataset at two granularities. AirTag evaluates each host in an MDataset separately. Flash evaluates three attacks in the OpTC dataset separately. We only add a count if percentage differences are applicable to all sub-evaluations of a dataset. ‡AirTag's use of DepImpact datasets is not included in these counts.

Property	ShadeWatcher	Atlas†	ThreaTrace	AirTag†‡	NodLink†	Flash†	Kairos	Magic
End-to-End Execution	✗	✓	✓	✓	✓	✓	✓	✓
Exact Result Reproducibility*	[✗,0]/2	[0,0]/10	[0,0]/11	[0,0]/16	[✗,0]/6	[1,0]/8	[3,2]/8	[2,0]/5
Minor Result Difference (<2%)*	[✗,0]/2	[2,3]/10	[1,2]/11	[8,2]/16	[✗,0]/6	[6,1]/8	[3,3]/8	[4,3]/5
Data Availability	1/2	10/10	11/11	16/16	4/6	8/8	8/8	5/5
Label Availability	0/2	10/10	7/11	16/16	1/6	7/8	7/8	5/5
Ready-to-run Scripts	0/2	0/10	6/11	10/16	0/6	7/8	7/8	4/5
Trained Model Presence	0/2	10/10	6/11	16/16	0/6	7/8	8/8	5/5
CPU/Mem Overhead Discussion**	✓✓	✗	✓✓	✗	✗	✓✓	✓✓	✓
Train/Test Times Presence**	✓✓	✓✓	✓	✓	✓	✓	✓	✓✓

objects and their interactions. The number of objects and interactions represented varies across systems, directly influencing the graph's size and memory footprint. As the graph grows, it may incur higher storage costs but also offers richer structural complexity, which can improve downstream feature discrimination. By structuring input data as provenance graphs, these systems effectively capture the causal relationships among kernel-level entities.

Graph Representation and Learning. Once provenance graphs are constructed, PIDDL systems transform them into representations suitable for machine learning–based anomaly detection. Deep learning methods, particularly graph neural networks (GNNs), are commonly used to generate neural embeddings that capture the structural and contextual relationships among nodes and edges. These embeddings enable models to learn complex patterns inherent in system behavior. Additionally, some systems incorporate techniques such as Word2Vec [52] to embed node and edge attributes, enriching the representation with semantic information that enhances the model's ability to discriminate between benign and anomalous behavior.

Detection. In the final stage, the learned embeddings are used to identify malicious activity. Detection mechanisms vary across systems, depending on the desired level of granularity. A common approach is anomaly-based detection, wherein models are trained on patterns of benign behavior, and deviations from these learned patterns are flagged as potential anomalies.

3 Methodology

We initially identified 14 representative intrusion detection systems that leverage provenance information and incorporate deep learning techniques in their pipelines, focusing on work published since 2020. These systems appeared in top-tier security venues, including USENIX Security [5, 17, 25, 31, 75] (5/14), NDSS [40, 72, 76] (3/14), IEEE S&P [10, 21, 70, 77] (4/14), ACM RAID [35] (1/14), and IEEE TIFS [73] (1/14). From these, we selected eight systems [5, 10, 17, 31, 40, 70, 73, 77] with publicly available open-source repositories, excluding those lacking accessible code. We assigned the selected systems to three undergraduate students (referred to as evaluators) and three graduate students (referred to as editors). This division of roles was intended to ensure an unbiased evaluation process: evaluators independently attempted to reproduce results without prior assumptions, while editors cross-checked their findings and provided methodological guidance.

3.1 Evaluators & Editors

Our methodology consists of two stages. In the first stage, evaluators independently attempt to reproduce the results reported in the original papers, documenting any technical issues encountered, such as code errors or missing dependencies. These issues are then reported to the editors. In the second stage, editors triage the reported problems, identifying which issues to escalate to the original authors via email or GitHub and proposing fixes for resolvable problems (e.g., configuration adjustments). Authors are

contacted (by either editors or evaluators) to clarify ambiguities and minimize the risk of false-negative conclusions. When contacted via email, all the authors of the paper were included. Following iterative re-evaluation, editors assess whether the discrepancies between the reproduced and reported results have been accurately identified and appropriately addressed.

3.2 Evaluation Properties

To systematically assess the reproducibility of the eight selected systems, we define a set of key properties (Table 4) that encompass critical aspects of software execution, data availability, and experimental validation.

End-to-End Execution. We check whether all parts of the system, from log ingestion through to detection, are included, can be built, and run successfully.

Exact Result Reproducibility. All systems include a prototype evaluated on one or more datasets to assess performance. This property captures the extent to which the reported evaluation results are exactly reproducible. Specifically, we check whether the reproduced outcomes match the figures presented in the paper across all reported metrics. When a dataset is evaluated at multiple levels of granularity, reproducibility requires that results match at each level. In cases where multiple models are used to evaluate the same dataset, we consider the results reproducible if the reproduced outcomes match those reported for at least one of the models. For example, Flash is deemed reproducible if the results align when using its GNN for downstream classification, even if they differ from those obtained using the XGBoost model, assuming both are reported in the paper.

Minor Result Difference. This property relaxes the criteria for exact result reproducibility by allowing small deviations. Specifically, it verifies that the reproduced F1 scores differ from the reported values by less than 2%.

Data Availability. This property records the number of datasets used in the paper that are publicly accessible.

Label Availability. For each publicly available dataset, we assess whether the corresponding ground truth labels are also publicly accessible. It is important to note that detection granularity varies across systems [32]. Therefore, a general ground truth (e.g., that provided with the DARPA TC E3 datasets) may not be sufficient to consider a dataset as having usable ground truth labels.

Ready-to-Run Scripts. This property evaluates how many dataset evaluations can be executed with minimal effort from the evaluator. In cases where both pre-trained models and models trained from scratch are evaluated, a dataset is considered successful under this criterion if either approach runs without requiring manual intervention.

Trained Model Availability. To support reproducibility, researchers may release trained models for the datasets used in their evaluations. This property records how many of those datasets have corresponding trained models publicly available.

CPU and Memory Overhead Discussion. This property evaluates whether the paper discusses memory consumption and CPU usage under varying system hyperparameters. Such analysis is critical for assessing diverse deployment environments feasibility.

Training and Testing Time Reporting. This property assesses whether the paper reports model training and testing times for at least one dataset used in the evaluation. Providing this information supports fair performance comparisons with future systems and helps set realistic expectations for researchers attempting to reproduce the results.

4 System Case Studies

We present detailed case studies of our efforts to reproduce each of the eight systems, highlighting the primary challenges encountered during the process. These challenges are summarized in Table 5. We report the key quantitative discrepancies observed, comparing the published results against those obtained using the best-performing model from our reproduction (either the available pre-trained model or a model trained from scratch). We reproduce evaluations on all available datasets with one exception: we exclude AirTag's evaluation on the DepImpact datasets [19], as this forms part of an ablation study exploring model behavior under varying conditions, rather than constituting the core evaluation of the system.

4.1 Shadewatcher

ShadeWatcher [77] introduces a novel approach to cyber threat detection by recasting it as a recommendation problem. It predicts the likelihood of interaction between kernel objects, akin to inferring user-item interactions in traditional recommendation systems. It adopts Watson's [76] strategy to construct behavior subgraphs rooted at data objects, transforms them into bipartite graphs representing system-entity interactions, and integrates these with a noise-reduced provenance graph to form a *knowledge graph (KG)*. An unsupervised GNN [37] is then applied to the KG to learn both direct and semantic relationships between system entities, enabling the detection of anomalous entities based on unlikely interactions.

Evaluation Datasets. 2 datasets: a closed-source custom dataset simulating six different cyber attacks, and the TRACE team's open-source DARPA TC Engagement 3 dataset [14].

Evaluator Experience

Missing Code. The ShadeWatcher codebase lacks the implementation of the interaction extraction component, including the construction of bipartite graphs, which constitutes a core contribution of the paper. Manual inspection confirms that this functionality is entirely absent. As reported by Kairos [10], the authors have stated that this component is proprietary. After parsing the provenance graph, the parser fails to generate any of the output files required by the recommendation model, such as the knowledge graph, extracted interactions, or entity metadata. Without these artifacts, the model cannot be trained or evaluated, precluding an end-to-end execution of ShadeWatcher. Although the model can be run using example text files provided in the repository, its evaluation logic assumes all inputs are benign and therefore only reports true negatives and false positives.

Investigating Workarounds for the Missing Code. Because ShadeWatcher's interaction extraction component relies on Watson's strategy—and Watson is not open source—the evaluator was unable to reimplement the missing functionality. However, a framework called ProvNinja [53] includes an evaluation of ShadeWatcher.

Table 5: Key Reproducibility Challenges. A ✗indicates that the issue is present in the corresponding system. *We add a cross here if the issue could not be resolved even after trying to trace the file names from the code. **Deployment-environment-specific code or debugging code left in.

Reproducibility Challenge	ShadeWatcher	Atlas	ThreaTrace	AirTag	NodLink	Flash	Kairos	Magic
Missing Code Components	✗	–	–	–	–	✗	–	–
Dependency Issues	✗	–	✗	✗	✗	–	✗	–
Absent Required Files	✗	–	✗	✗	✗	✗	✗	–
Unspecified Train/Test Files*	✗	–	✗	–	✗	✗	–	–
Incomplete Documentation	–	✗	–	–	✗	✗	–	–
Software Bugs	–	✗	–	✗	✗	✗	✗	✗
Unintended Code Retained**	–	–	✗	✗	–	–	✗	–
Paper-Code Inconsistency	–	✗	–	–	✗	✗	–	–
Pre-Trained/Scratch Model Variance	–	✗	✗	✗	–	✗	✗	✗

To understand how its authors conducted this evaluation, we manually inspected their public fork of the ShadeWatcher codebase. While this fork included modifications to enable file output required by the recommendation model, it did not reimplement the missing interaction extraction logic. Instead, it approximated interactions by treating direct edges between kernel objects in the provenance graph as interactions—an approach that deviates from the methodology described in the original ShadeWatcher paper. Although the fork remained accessible through late 2024, it has since been removed from its original GitHub link. We contacted the authors to request access but did not receive a response.

Reproduction Results

ShadeWatcher's codebase lacks the implementation of the interaction extraction component, including the bi-partite graph creation, which is essential to the system. This prevents any end-to-end execution of ShadeWatcher.

4.2 Atlas

Atlas [5] introduces a framework for reconstructing complete "attack stories" (temporal sequences of attack steps—from audit logs) starting from a known attack symptom system-entity node. The core idea is to convert a noise-reduced provenance graph into lemmatized *sequences*, each representing a temporally ordered set of events within an entity's *neighborhood*. To construct a balanced dataset, Atlas under-samples benign sequences and over-samples attack sequences. These sequences are then mapped to a generalized vocabulary, vectorized, and used to train an LSTM model to distinguish between attack and non-attack patterns. During investigation, Atlas classifies each sequence as benign or malicious and uses the flagged sequences to identify suspicious system entities and events.

Evaluation Datasets. 10 custom datasets: 4 from a single-host environment (SDatasets) and 6 from a multi-host environment (MDatasets).

Evaluator Experience

Ambiguous README Instructions. During the testing phase, Atlas outputs detected attack entities and their prediction scores to the console. Users are expected to manually extract this output, clean it, and format it into a JSON file required by the evaluation script. This process involves subjective decisions, such as removing redundant entities (e.g., repeated process instances referencing the same file) and adding "obviously related" entities, without any explicit criteria. The absence of clear guidelines results in inconsistent and potentially non-reproducible evaluation outcomes. We raised a GitHub issue [1] and contacted the authors via email to verify our cleaned entity set, but did not receive a response. Additionally, the README omits critical instructions: users must manually modify hard-coded flags in `atlas.py` to generate resampled files and adjust them again to load those files for training. Failure to do so results in a `FileNotFoundError` during training. We sought clarification from the authors regarding this workflow but received no reply.

Inconsistency Between Paper and Shared Code. A critical step in Atlas's pipeline is *Selective Sequence Sampling*, where benign sequences are under-sampled and attack sequences are over-sampled to produce a balanced training dataset. The paper describes a mutation-based oversampling technique that randomly replaces vocabulary tokens in lemmatized sequences with other tokens of the same semantic type (e.g., a `system_process` may only be substituted with another process-related word, not a file-related one). However, manual inspection of the code reveals that this strategy is not implemented. Instead, the code performs oversampling by simply duplicating and appending malicious samples [58]. We reported this discrepancy to the authors via email but did not receive a response.

Runtime Error. The evaluator encountered a runtime error in `graph_reader.py`, caused by improperly formatted node names and attributes in the `.DOT` file generated by `graph_generator.py`. Specifically, missing quotation marks around node identifiers and IP addresses led the `NetworkX` library's `read_dot()` function to misinterpret these entries, resulting in a parsing failure. To address the issue, `graph_generator.py` was modified to enclose node names and IP attributes in quotation marks and to ensure proper formatting before writing the `.dot` file.

Impact of Noise Reduction. Atlas's paper reports entity-based investigation results prior to applying noise reduction, whereas the released code computes results after noise has been removed from the provenance graph. This substantially reduces the number of entities considered. Models trained by the evaluator exhibited notable discrepancies in entity-level evaluation, yielding significantly lower F1 scores for several scenarios: S2 (0.51 vs. 0.92), S4 (0.72 vs. 0.89), M1 (0.41 vs. 0.94), M2 (0.46 vs. 0.93), M3 (0.63 vs. 0.97), and M4 (0.62 vs. 0.92). We contacted the authors via email for clarification

and guidance regarding this inconsistency but did not receive a response.

Reproduction Results

F1 scores for the S1 and S2 datasets matched the reported values at the event-level exactly. Entity-level S1, S3, M3, M5, and event-level S3, S4, M1, M2, M3, M4, M5, and M6 results differed by $< \pm 2\%$ from the reported values. Entity-level S2, M1, M2, M4, and M6 results differed from the reported values by $\pm 2 - 10\%$, while S4 showed a difference of $> \pm 10\%$.

4.3 ThreaTrace

ThreaTrace [73] is an anomaly-based, node-level detector that differentiates itself from prior graph-based approaches (e.g., StreamSpot [48], Unicorn [24]) and path-based detectors (e.g., ProvDetector [72]) by employing multiple GraphSAGE [23] models to learn the roles of benign nodes in a provenance graph. To reduce both false positives and false negatives, ThreaTrace trains multiple sub-models iteratively, masking confidently classified nodes and forwarding only ambiguous ones to subsequent models. During detection, a node is classified as benign if at least one sub-model correctly identifies its type (e.g., *file*, *process*, *socket*); otherwise, it is flagged as abnormal. As ThreaTrace operates in a streaming setting, flagged nodes are held in a queue for a time window T to capture their evolving context. If a node remains flagged as abnormal within this window, it is deemed anomalous. An alert is raised when the number of such anomalous nodes exceeds a predefined tolerance $\hat{T}$.
Evaluation Datasets. 11 datasets: (i) the StreamSpot dataset [61], (ii) the Unicorn SC datasets (consisting of the SC-1 [49] and SC-2 [50]) and (iii) the DARPA TC datasets (*Theia*, *TRACE*, *CADETS* and *Five Directions* from Engagement 3 and Engagement 5) [15].

Evaluator Experience

Variability Across Training Runs. The evaluator observed that training the models from scratch on the Unicorn and DARPA TC E3 datasets resulted in performance variations exceeding 5% across runs, indicating high variability in the training process. This issue persisted despite attempts to contact the authors for clarification. Consequently, all reported results are averaged over three independent training runs to ensure consistency.
Use of Testing Data in Training. While debugging ThreaTrace, the evaluator identified three instances of potential overlap between validation and testing data, constituting a *data snooping* issue [7]. In the StreamSpot training script, validation and testing graphs are selected randomly, without explicitly preventing the same graph from appearing in both sets [68]. In the Unicorn training script, one validation set (*validateSetA*) includes 10 attack graphs randomly sampled from a pool of 25, all of which also appear in the test set. The other validation set (*validateSetB*) contains 50 benign graphs, which may overlap with the 25 benign test graphs due to sampling without exclusion [69]. In the DARPA TC Engagement 3 script, each model is evaluated on the test set during training, and models with poor performance are subsequently discarded [67]. Disabling this step results in a significant performance drop, with F1 scores falling to 0.44 for Theia, 0.00 for TRACE, 0.54 for CADETS, and 0.40 for Five Directions. These issues were reported via a GitHub issue [47] and subsequently through email. The author acknowledged the oversight in the DARPA TC E3 script and clarified that this validation step was originally introduced for debugging, but was inadvertently left in the published code.
Issues with Pre-trained Models. The pre-trained StreamSpot model misclassified both benign and attack graphs as benign, while the TRACE model produced near-zero precision, recall, and F1 scores. Further, a required file (`threshold_unicorn.txt`) needed to run the pre-trained Unicorn model, was missing from the repository. The evaluator reported these issues via GitHub [44] and email. The author acknowledged the problems over email. However, the missing threshold file was not provided.
Missing Script and Ground Truth. ThreaTrace evaluates on the DARPA TC Engagement 5 (E5) datasets; however, unlike for E3, it does not provide dedicated training or evaluation scripts for E5. The existing E3 script contains hardcoded file names, making it unsuitable for E5 without modification. As a result, the training and testing files for E5 remain unspecified. We raised these concerns via GitHub [41, 42]. Additionally, the ground truth labels for E5 are absent from the repository. Upon contacting the authors by email, we were informed that the source code and ground truth for E5 were lost due to a server error.

Reproduction Results

ThreaTrace's F1 scores differed by less than $\pm 2\%$ for the StreamSpot, Unicorn SC-1 and DARPA TC E3 CADETS datasets. The F1 scores differed within ± 2-10% for Unicorn SC-2, DARPA TC E3 Theia, TRACE and Five Directions. The repository is missing necessary resource files required to evaluate the the DARPA TC E5 datasets, making them non-reproducible.

4.4 AirTag

AirTag [17] introduces a novel approach by bypassing the traditional PIDDL step of constructing a provenance graph. Instead, it demonstrates that training a deep learning model directly on raw event logs yields superior detection performance. AirTag is evaluated against Atlas [5], which addresses class imbalance through selective sampling. In contrast, AirTag employs a one-class support vector machine (OC-SVM) [62], trained exclusively on benign data. The OC-SVM estimates a decision function that defines a boundary around the benign data; any sample falling outside this boundary is classified as malicious. For attack investigation, AirTag first embeds log entries using BERT [16], and then uses the OC-SVM to classify each entry as benign or malicious. Malicious entries are subsequently used to construct a provenance graph that visualizes the inferred attack.
Evaluation Datasets. 20 datasets: ten sourced from Atlas (SDatasets and MDatasets, section 4.2), four from DepImpact [19], and six created by the AirTag authors (UDatasets). UDatasets contain logs collected from single-host environments where the attacks are forced to fail to evaluate its effectiveness in scenarios where the attack chain is incomplete.

Evaluator Experience

Missing Dataset Resources. Unlike the SDatasets and MDatasets, no scripts were provided to reproduce results for the UDatasets in an end-to-end manner. Since both SDatasets and UDatasets were

simulated in single-host environments, we identified the SDataset scripts as a potential substitute for evaluating the UDatasets. However, the untokenized training files required for the false-positive filtering step were missing. This step relies on word frequency analysis to ensure that malicious classifications are both rare and robust, and it requires the dataset to follow a specific format. After raising this issue on GitHub [65], the authors uploaded additional files. Unfortunately, these files corresponded to the SDatasets, not the UDatasets. Despite further follow-up, no additional response was received.

Problematic Deployment Details. AirTag's codebase includes deployment-specific configurations that hinder reproducibility. Although the repository's README recommends executing a Bash script for end-to-end evaluation on the SDatasets and MDatasets, GPU identifiers are hardcoded to match the original deployment environment. This limitation is undocumented, not configurable via command-line arguments, and only apparent upon manual code inspection. As a result, users following the provided instructions without modification are likely to encounter significantly increased training times, as the model silently defaults to CPU execution when the specified GPU is unavailable.

Runtime Error. The evaluator encountered the same graph construction issue in AirTag as previously observed in Atlas. This issue was independently confirmed with the authors [64].

Inaccurate File Labels. An inconsistency was initially observed between AirTag's paper and code during the false-positive filtering step. The paper indicates that word frequencies used for the rareness check are computed from testing files; however, for three of the four SDatasets, the code instead used training files. Upon raising this issue via email, the authors clarified that mislabeled filenames were the source of confusion. A manual review of Atlas 's preprocessing confirmed that dataset labels vary across testing phases—for example, "training S1" in one phase may be labeled "testing S1" in another. Because AirTag 's authors used data from a specific phase, the filenames appeared inconsistent, although the actual data usage was correct.

Reproduction Results

F1 scores of the reproduced results differed by $< \pm 2\%$ from the reported values for all SDatasets except S2, and all MDatasets except M4 (both Host 1 and Host 2). For both the S2 and M4 (both hosts) datasets, F1 scores differed within $\pm 2 - 10\%$ of the reported values. The repository is missing necessary resource files required to evaluate the UDatasets, making them non-reproducible.

4.5 NodLink

NodLink [40] detects advanced persistent threats (APTs) by formulating the problem as an instance of the Steiner Tree Problem (STP) [28, 29], a classical graph theory problem that seeks a minimum-weight subgraph connecting a specified set of terminal nodes. NodLink adapts this formulation to provenance-based intrusion detection by first identifying indicators of compromise, process nodes flagged as anomalous by a variational autoencoder [36], and then attempting to connect them. Unlike traditional STP approaches that rely on shortest-path algorithms, NodLink constructs size-bounded subgraphs around each anomalous node, expanding nodes based on an importance score that combines anomaly score, node degree, and distance. This process operates over all system events within rolling 10-second windows. Anomaly scores are aggregated per subgraph and updated incrementally as subgraph are merged via a cache. Subgraph with unusually high aggregate scores are flagged using Grubbs' Test [22].

Evaluation Datasets. 6 datasets: (i) DARPA TC E3 CADETS, Theia, and TRACE, (ii) an attack dataset simulated in an industrial environment (Industrial Arena dataset), (iii) an attack dataset simulated in a lab environment (In-Lab Arena dataset), and (iv) an open world dataset containing real client data from a security company (Open World).

Evaluator Experience

Paper-Code Inconsistency. The paper states that only graphs triggering alerts are counted as graph-level true positives (TPs), and only process nodes within those graphs are considered node-level TPs. However, code inspection revealed a deviation from this description: all process nodes in cached graphs are counted as node-level TPs, regardless of whether their corresponding graphs triggered alerts. Additionally, while the paper describes a cache eviction policy based on graph *energy* (a function of anomaly score and update time) intended to limit memory usage, the code instead retains the top 20 graphs per time window based solely on anomaly score. We raised these discrepancies with the authors [66], but received no response. Since NodLink only reports node-level recall, we inferred additional metrics by analyzing the codebase. Given that node-level recall is computed across all cached graphs, we treated attack-relevant process nodes in those graphs as node-level TPs. Process nodes in the cache that are not attack-relevant were treated as node-level false positives (FPs), allowing us to calculate node-level precision. Similarly, cached graphs without any attack-relevant nodes were considered graph-level FPs, while those containing such nodes were treated as graph-level TPs, enabling computation of graph-level precision. However, because the code does not support identifying graph-level false negatives under the published methodology, we were unable to compute graph-level recall.

Missing Script and Ground Truth. NodLink's repository includes scripts only for the In-Lab Arena dataset, which rely on hardcoded file names that exists only within that dataset. Consequently, evaluating NodLink on other datasets requires prior knowledge of which files to use for training and testing. Furthermore, the provided code is tailored to the format of the In-Lab Arena logs, necessitating separate parsing scripts for the E3 datasets. Ground-truth labels are also essential for evaluation, but were not included. We contacted the authors via GitHub [2] and email, and were referred to an external repository containing a parsing script. However, no ground-truth labels or guidance on training/testing file selection was provided, and our follow-up request received no response.

Undocumented Structural Modification in Code. While the NodLink paper reports evaluation metrics for the full In-Lab Arena dataset, the accompanying code splits the dataset into three subsets without documenting how results are aggregated. Upon contacting the authors [3], we confirmed that the reported metrics are computed by summing the results across all subsets before calculating dataset-level values.

Reproduction Results

For the In-Lab Arena dataset, the node-level F1 score of the reproduced result differed by $> \pm 10\%$ from the reported value, while the graph-level F1 score could not be calculated as the graph-level recall could not be determined from the code. The Industrial Arena and Open World datasets are not available publicly, and hence, results on these could not reproduced. Similarly, data processing scripts and ground truth labels for the E3 CADETS, Theia and TRACE datasets are missing, making their evaluations non-reproducible.

4.6 Flash

Flash [70] introduces efficient system-level anomaly detection by combining graph-based representation learning (capturing patterns in graphs of system events) and semantic attribute encoding (transforming system attributes with context into rich vector representations). A key innovation is its *embedding recycling database*, which caches pre-computed graph embeddings for reuse during inference, significantly reducing latency without compromising accuracy. To suppress noise, Flash employs selective edge traversal and attribute abstraction, emphasizing causally meaningful relationships. It supports both batch and real-time log processing, incorporates positional encoding to model temporal dependencies, and integrates lightweight classifiers (e.g., XGBoost, SVM, Random Forest) for fallback anomaly detection. Additionally, Flash constructs *attack evolution graphs* to visualize and analyze detected anomalies.

Evaluation Datasets. 8 datasets: (i) DARPA TC E3 Theia, TRACE, CADETS, Five Directions, (ii) DARPA OpTC [56], StreamSpot, and (iii) Unicorn SC-1 and SC-2.

Evaluator Experience

Inconsistent and Missing Code. The provided notebooks lacked clear instructions for conducting evaluations, and several key components were either missing or inconsistent with the paper. For instance, while a Word2Vec model was included in the code, it was never utilized; instead, a pre-trained model was imported from an external library. To proceed, we modified the pipeline to incorporate our own custom-trained Word2Vec model. Although the paper reports results using XGBoost, no implementation was provided for training or evaluating it. Finally, only SC-2 of the Unicorn dataset was supported in the code, despite SC-1 being discussed in the paper; no code was available for evaluating SC-1.

Inconsistent Dataset Sizes. The aggregate results reported in the paper do not align with the number of samples observed during our reproduction runs using the provided notebooks. Since each notebook independently fetches and parses the dataset using its own embedded logic (without any modification on our part) the discrepancies in sample counts between our runs, the published results, and those used for the pre-trained models suggest inconsistencies in data processing. Furthermore, several dataset-specific notebooks omit the reporting of true negatives, which was the metric with the largest deviation from the values presented in the paper.

Dataset Parsing Failures. Parsing the StreamSpot dataset failed due to incorrect directory handling in the provided parser, which caused downstream scripts to break. We corrected the parser to enable successful reproduction of the pre-trained results.

Training File Paths. For the OpTC dataset, the absence of documentation regarding training file paths resulted in errors during retraining. Although the authors shared a link to benign files following our outreach, they did not specify which files were required or how they should be integrated into the workflow. Additional follow-up inquiries did not receive a response.

Inefficient Code. Model retraining on the *Unicorn* dataset was prohibitively slow due to inefficient implementation. Although a suggested fix [4] improved execution time, the retrained models produced highly variable results, occasionally deviating significantly from the pre-trained outputs without any consistent pattern. This variability raises concerns about the stability and reproducibility of the training process.

Reproduction Results

The reproduced F1 score for the Unicorn SC-2 dataset matched the reported value exaclty. For all other datasets except for OpTC (Attack 2) and Unicorn SC-1, F1 scores differed by $< \pm 2\%$ from the original results. The F1 score of OpTC (Attack 2) differed within $\pm 2-10\%$ of the reported value while the code required for evaluating the Unicorn SC-1 dataset was missing, making its evaluation non-reproducible.

4.7 Kairos

Kairos [10] introduces a novel framework for graph learning in PIDDL systems by modeling both temporal and structural relationships within provenance graphs. It adopts an encoder-decoder architecture designed to operate on a stream of edges, dynamically updating the graph representation as it evolves over time. The encoder leverages a temporal graph network (TGN) [59] composed of a unified message passing (UniMP) model [63] and a gated recurrent unit (GRU) [11]. UniMP embeds each edge based on the *state* of its neighborhood, while the GRU updates node states in response to new edges. The decoder applies a multilayer perceptron to predict edge types. The difference between the predicted and actual edge type defines the edge's *reconstruction error*. Kairos processes system events in time windows, queuing overlapping windows based on shared suspicious nodes. Suspicious nodes are identified using a combination of edge reconstruction error and frequency. If the aggregated anomaly score within a queue exceeds a predefined threshold, the associated time windows are flagged as malicious.

Evaluation Datasets. 8 datasets: (i) StreamSpot, (ii) DARPA TC Theia, CADETS, and ClearScope from both Engagement 3 and 5, and (iii) Darpa OpTC.

Evaluator Experience

File Generation Bug. During testing, Kairos uses an `attack_list` to label malicious subgraphs, assigning a ground truth label of 1 if a matching name is found, and 0 otherwise. These lists are dataset-specific and are expected to be automatically generated as part of the evaluation workflow. However, for the E5 CADETS, Theia, and ClearScope datasets, the required files are not generated. This results in a runtime error for E5 CADETS and E5 Theia due to missing dictionary keys corresponding to attack graph names. In the case of E5 ClearScope, the differing code structure avoids a crash, but the issue causes no graphs to be labeled as malicious, leading

to zero true positives and zero false negatives being reported. We reported this issue to the authors [43, 45], but did not receive a response.

Notebook Issues. Several issues were encountered in the provided notebooks. In the E5 Theia notebook, the saved and loaded GNN model filenames were inconsistent, requiring manual correction. The OpTC preprocessing notebook, unlike others, did not specify training and testing files, necessitating manual identification. Furthermore, ground truth labels for OpTC were missing, resulting in a runtime error. This issue was reported to the authors [46], but no response was received.

Reproduction Results

F1 scores of the reproduced results for StreamSpot, E3 CADETS, and E3 ClearScope matched the reported values exactly, while they differed by $< \pm 2\%$ for E3 Theia. F1 scores for E5 ClearScope, Theia, and CADETS could not be calculated due to the file generation bug outlined above. Ground-truth labels for the OpTC dataset are also missing, making its evaluation non-reproducible.

4.8 Magic

Magic [31] detects stealthy attacks by applying graph representation learning to system logs. Its core innovation is an *edge merging technique* that reduces redundancy while preserving critical information, thereby simplifying the graph structure without compromising fidelity. Magic further employs a *graph masked auto-encoder (GMAE)* [27] to learn robust node embeddings by reconstructing masked node relationships, enhancing its ability to detect previously unseen attack patterns. For anomaly detection, it uses a k-nearest neighbors (KNN) approach, leveraging a K-D Tree built from embeddings of benign system behavior. Anomaly scores are computed based on distances to the nearest benign neighbors, and entities exceeding a defined threshold are flagged as suspicious. An adaptive learning mechanism continuously refines the model using system feedback, improving its capacity to detect advanced persistent threats (APTs) while maintaining a low false positive rate.

Evaluation Datasets. 5 datasets: (i) StreamSpot, (ii) Unicorn Wget [51], and (iii) DARPA E3 TRACE, Theia, and CADETS.

Evaluator Experience

Labeling and Data Organization Errors. The Wget dataset exhibited inconsistencies due to misordered files during preprocessing. The labeling script incorrectly assumed that the first 25 graphs represented attacks and the remainder were benign. However, because file ordering was arbitrary, this assumption led to mislabeled samples. This issue had been previously raised in an open GitHub issue [33], where the authors clarified that the labeling logic relied on file order rather than explicit metadata. To address this, the evaluator manually enforced a deterministic file order, which improved the results but did not fully eliminate discrepancies. Further investigation revealed additional inconsistencies caused by non-deterministic training conditions. Although a fixed seed was expected for reproducibility, it was overridden in multiple locations within the training scripts, leading to variations in output across runs. After correcting the seed settings, classification results became more stable, though minor variations still persisted.

Reproduction Results

F1 scores for E3 Theia and CADETS matched the reported values exactly. They differed by $< \pm 2\%$ for StreamSpot and E3 TRACE and by $\pm 2 - 10\%$ for Wget.

5 Interactions with authors

In the process of reproducing results from various research papers, we contacted the original authors via GitHub (by opening issues) and sent an email to all authors requesting clarifications and to resolve technical challenges. When initiating contact through GitHub, we allowed five business days for a response before sending a follow-up email. Authors were deemed unresponsive if no reply was received within seven business days following the email. Table 6 summarizes the number of issues raised, responses received, and associated response times.

We define an issue as a distinct query; for example, a request for dependency clarification and a request for a missing file are treated as separate issues. Requests made via GitHub or email may contain multiple queries so one reply may lead to the same 'time to respond' for all the included queries. If a query applies to multiple datasets but requires distinct responses (e.g., missing ground truth data for each dataset), each instance is counted as a separate issue. Conversely, if a single response resolves a shared problem across multiple datasets (e.g., a common bug), it is counted as one issue.

These interactions reveal varying levels of engagement across research groups, reflecting differences in accessibility and willingness to support reproducibility efforts. Groups such as ThreaTrace, AirTag, and NodLink responded to most queries, indicating active maintenance and a commitment to supporting external validation. In contrast, ShadeWatcher, Kairos, and Atlas authors did not respond at all. Such inconsistency poses a substantial barrier to reproducibility, as the absence of author engagement can effectively block attempts to validate published results. In some cases, non-responsiveness may be due to practical constraints, such as maintainers transitioning to new roles or institutions. Ideally, repositories should clearly indicate whether they are actively maintained, enabling the research community to set appropriate expectations and plan accordingly.

Table 6 also reports average response times for each system, highlighting both delayed replies and complete non-responsiveness. While some authors responded within minutes, others took several days or even weeks, as observed with ThreaTrace. In contrast, the authors of Flash replied within an impressive 38 minutes (0.63 hours), demonstrating strong responsiveness. However, the column indicating issues resolved reveals that prompt replies do not always equate to effective support. Indeed, several issues were acknowledged but left unresolved. For instance, although ThreaTrace provided multiple responses, many failed to address the underlying concerns. This distinction underscores an important point: timely or frequent responses alone are not sufficient. Reproducibility remains hindered when issues are inadequately addressed or left unresolved.

Overall, our experience highlights that while author responsiveness is valuable, it must be accompanied by substantive, solution-oriented communication to meaningfully support reproducibility.

Table 6: Issue Response and Resolution Metrics Across Systems

System	# Issues	# Responded	#Resolved	Time to Respond (Hours)	Avg. Time/Issue (Hours)
ShadeWatcher	3	0	0	N/A	N/A
Atlas	21	0	0	N/A	N/A
ThreaTrace	20	17	2	[10×226.02, 4×477.15, 3×13.4]	247.59
AirTag	15	9	3	[6×13.7, 1×8.02, 1×34.25, 1×6.42]	15.54
NodLink	12	10	2	[9×19.78, 1×0.48]	17.85
Flash	3	1	1	[1×0.63]	0.63
Kairos	5	0	0	N/A	N/A
Magic	2	1	1	[1×1.75]	1.75

Although not always expected, clear indications of a project's maintenance status are highly beneficial and contribute significantly to fostering a more reproducible and collaborative research ecosystem.

6 Related Work

We review prior work on reproducibility in order of increasing domain specificity. We begin with studies that assess reproducibility across different computer science domain, then focusing on security systems that incorporate machine learning, and finally narrowing to intrusion detection systems.

Reproducibility in Computer Science. A decade ago, a series of studies examined whether code accompanying ACM systems papers could be successfully built and executed. The initial investigation [12] reviewed approximately 600 papers to determine whether they were supported by code and whether a student could build and run the systems within 30 minutes. The study found that only 30% of code-backed papers could be built successfully. This was followed by a follow-up study by the original authors [13] (with relaxed constraints, even so only about 50% of the systems could be successfully built) and an independent replication effort [39] is ongoing. A study in image processing [38] assessed code availability and found that none of the 15 papers examined provided code. A subsequent, larger study [71] extended this analysis to 134 papers, revealing that only 9% had code available. Similarly, a study in computational linguistics [74] examined approximately 400 papers to contact authors of those without data and code, and attempted to reproduce results from 10 of them. Of those, only 6 were reproducible, and just one yielded results identical to those reported in the original publication. Our work builds upon and extends these efforts by conducting a comprehensive review that directly addresses each of these aspects within the context of provenance-based intrusion detection systems.

Machine Learning Security. The integration of machine learning into security systems is increasingly seen as inevitable [6], and the reproducibility of research in this space has drawn growing attention. A study of 750 machine learning papers published in security conferences [54] found that 60% did not provide any code to reproduce their experiments. Among those that did, only 20% yielded results consistent with the original publications. Other studies have highlighted broader issues in the machine learning for security literature. For example, an analysis of 30 papers revealed that each exhibited at least three of several identified methodological pitfalls [7]. Similarly, a review of machine learning-based location privacy-preserving mechanisms [57] found that these approaches often failed to protect user data when tested against real-world distributions. While comprehensive reproducibility analyses exist for general machine learning security research [54], our work focuses specifically on a narrower and more specialized domain: provenance-based intrusion detection systems. Our methodology also differs in key respects. Rather than halting at the first sign of technical obstacles, we conduct a detailed artifact evaluation, analyzing each system component and incorporating feedback from the original authors to troubleshoot issues. Additionally, given the smaller number of systems under review, we document the evaluator experience for each case, offering future researchers a clear sense of the effort and expectations involved in replicating these systems.

Intrusion Detection. To the best of our knowledge, the reproducibility of PIDDL systems has not been systematically evaluated prior to this work. However, prior studies have identified reproducibility-related concerns in general intrusion detection systems. A recent survey [60] found that only 2 out of 21 host-based IDSes evaluated system efficiency in terms of CPU and memory overhead (in our study 4 out of 8 systems do so, see Table 4). This is particularly concerning given the growing popularity of graph-based intrusion detection techniques, which, while enabling rich feature extraction, are often computationally expensive [78, 79]. The same study also reported that only 6 of the evaluated systems measured any of the key performance metrics—detection time, testing time, classification time, and inference time—which are essential for assessing real-time detection capabilities (in this study 8/8 report test time performance, see Table 4). While these prior findings highlight indirect reproducibility challenges in the broader IDS literature, to the best of our knowledge our work is the first to directly examine reproducibility of detection performance.

7 Conclusion

This paper presents a comprehensive reproducibility study of eight Provenance-based Intrusion Detection using Deep Learning (PIDDL) systems. It evaluates whether these systems can be reproduced from publicly available artifacts, identifying critical gaps in code completeness, data availability, documentation, and evaluation consistency. The study reveals that no system is fully reproducible end-to-end, with flaws including missing components, configuration issues, and undocumented behaviors. We make the following recommendations to the community:

Provide Complete and Self-Contained Codebases. Ensure all core components are included, avoiding reliance on proprietary or missing modules. It is best to host versioned, archived repositories.
Standardize Dataset Interfaces and Label Formats. Use consistent file formats, naming conventions, and include clearly documented ground-truth labels and splits.
Automate Evaluation Pipelines with Configuration Interfaces. Replace hardcoded parameters with configuration options and provide end-to-end scripts with clear usage instructions.
Ensure Deterministic and Transparent Training. Fix random seeds, avoid nondeterministic operations, and log environment details to reduce variance in the results produced.

Acknowledgments

This material is based upon work supported in part by the Intelligence Advanced Research Projects Activity (IARPA) and Army Research Office (ARO) under Contract No. W911NF-20-C-0038. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Intelligence Advanced Research Projects Agency (IARPA) and Army Research Office (ARO).

References

[1] abdullahasif07. 2024. *Clarification needed for manually cleaning entities.* GitHub. https://github.com/purseclab/ATLAS/issues/23 Created: 2024-11-20.
[2] abdullahasif07. 2024. *Request for DARPA Datasets and Parsing Code.* GitHub. https://github.com/PKU-ASAL/Simulated-Data/issues/12 Created: 2024-12-01.
[3] abdullahasif07. 2025. *Clarification on Result Aggregation for In-Lab Arena Dataset.* GitHub. https://github.com/PKU-ASAL/Simulated-Data/issues/15 Created: 2025-03-06.
[4] abouelkhair5. 2024. *Suggestion to improve efficiency in the unicorn notebook.* GitHub. https://github.com/DART-Laboratory/Flash-IDS/issues/2 Created: 2024-04-09.
[5] Abdulellah Alsaheel, Yuhong Nan, Shiqing Ma, Le Yu, Gregory Walkup, Berkay Celik, Xiangyu Zhang, and Dongyan Xu. 2021. ATLAS: A Sequence-based Learning Approach for Attack Investigation. *30th USENIX Security Symposium* (2021).
[6] Giovanni Apruzzese, Pavel Laskov, Edgardo Montes de Oca, Wissam Mallouli, Luis Brdalo Rapa, Athanasios Vasileios Grammatopoulos, and Fabio Di Franco. 2023. The Role of Machine Learning in Cybersecurity. *Digital Threats* (2023).
[7] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. 2022. Dos and Don'ts of Machine Learning in Computer Security. *31st USENIX Security Symposium* (2022).
[8] Frederick Barr-Smith, Xabier Ugarte-Pedrero, Mariano Graziano, Riccardo Spolaor, and Ivan Martinovic. 2021. Survivalism: Systematic Analysis of Windows Malware Living-Off-The-Land. *42nd IEEE Symposium on Security and Privacy* (2021).
[9] Sheung Chi Chan, James Cheney, Pramod Bhatotia, Thomas Pasquier, Ashish Gehani, Hassaan Irshad, Lucian Carata, and Margo Seltzer. 2019. ProvMark: A Provenance Expressiveness Benchmarking System. *International Middleware Conference* (2019).
[10] Zijun Cheng, Qiujian Lv, Jinyuan Liang, Yan Wang, Degang Sun, Thomas Pasquier, and Xueyuan Han. 2024. Kairos: Practical Intrusion Detection and Investigation using Whole-system Provenance. *45th IEEE Symposium on Security and Privacy* (2024).
[11] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation. *Empirical Methods in Natural Language Processing* (2014).
[12] Christian Collberg, Todd Proebsting, Gina Moraila, Akash Shankaran, Zuoming Shi, and Alex M Warren. 2014. Measuring Reproducibility in Computer Systems Research. *Technical Report (University of Arizona)* (2014).
[13] Christian Collberg, Todd Proebsting, and Alex M. Warren. 2015. Repeatability and Benefaction in Computer Systems Research. *Technical Report (University of Arizona)* (2015).
[14] DARPA Transparent Computing. 2018. *DARPA Engagement 3 TRACE Dataset.* https://drive.google.com/drive/folders/1EvIyZI7eCiON8xT8D8G4CENI_vXVshb2
[15] DARPA Transparent Computing. 2020. *DARPA Transparent Computing Dataset.* https://github.com/darpa-i2o/Transparent-Computing
[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (2019).
[17] Hailun Ding, Juan Zhai, Yuhong Nan, and Shiqing Ma. 2023. AIRTAG: Towards Automated Attack Investigation by Unsupervised Learning with Log Texts. *32nd USENIX Security Symposium* (2023).
[18] Feng Dong, Liu Wang, Xu Nie, Fei Shao, Haoyu Wang, Ding Li, Xiapu Luo, and Xusheng Xiao. 2023. DISTDET: A Cost-Effective Distributed Cyber Threat Detection System. *32nd USENIX Security Symposium* (2023).
[19] Pengcheng Fang, Peng Gao, Changlin Liu, Erman Ayday, Kangkook Jee, Ting Wang, Yanfang (Fanny) Ye, Zhuotao Liu, and Xusheng Xiao. 2022. Back-Propagating System Dependency Impact for Attack Investigation. *31st USENIX Security Symposium* (2022).
[20] Akul Goyal, Xueyuan Han, Gang Wang, and Adam Bates. 2023. Sometimes, You Aren't What You Do: Mimicry Attacks against Provenance Graph Host Intrusion Detection Systems. *30th Annual Network and Distributed System Security Symposium* (2023).
[21] Akul Goyal, Gang Wang, and Adam Bates. 2024. R-CAID: Embedding Root Cause Analysis within Provenance-based Intrusion Detection. *45th IEEE Symposium on Security and Privacy* (2024).
[22] Frank Ephraim Grubbs. 1950. Sample Criteria for Testing Outlying Observations. *The Annals of Mathematical Statistics* (1950).
[23] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *31st International Conference on Neural Information Processing Systems* (2017).
[24] Xueyuan Han, Thomas Pasquier, Adam Bates, James Mickens, and Margo Seltzer. 2020. Unicorn: Runtime Provenance-Based Detector for Advanced Persistent Threats. *27th Annual Network and Distributed System Security Symposium* (2020).
[25] Xueyuan Han, Xiao Yu, Thomas Pasquier, Ding Li, Junghwan Rhee, James Mickens, Margo Seltzer, and Haifeng Chen. 2021. SIGL: Securing Software Installations Through Deep Graph Learning. *30th USENIX Security Symposium* (2021).
[26] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. 2019. NODOZE: Combatting Threat Alert Fatigue with Automated Provenance Triage. *26th Annual Network and Distributed System Security Symposium* (2019).
[27] Zhenyu Hou, Xiao Liu, Yukuo Cen, Yuxiao Dong, Hongxia Yang, Chunjie Wang, and Jie Tang. 2022. GraphMAE: Self-Supervised Masked Graph Autoencoders. *28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (2022).
[28] Frank K Hwang and Dana S Richards. 1992. Steiner tree problems. *Networks* (1992).
[29] Makoto Imase and Bernard M. Waxman. 1991. Dynamic Steiner Tree Problem. *SIAM Journal on Discrete Mathematics* (1991).
[30] Muhammad Adil Inam, Yinfang Chen, Akul Goyal, Jason Liu, Jaron Mink, Noor Michael, Sneha Gaur, Adam Bates, and Wajih Ul Hassan. 2023. SoK: History is a Vast Early Warning System: Auditing the Provenance of System Intrusions. *44th IEEE Symposium on Security and Privacy* (2023).
[31] Zian Jia, Yun Xiong, Yuhong Nan, Yao Zhang, Jinjing Zhao, and Mi Wen. 2024. MAGIC: Detecting Advanced Persistent Threats via Masked Graph Representation Learning. *33rd USENIX Security Symposium* (2024).
[32] Baoxiang Jiang, Tristan Bilot, Nour El Madhoun, Khaldoun Al Agha, Anis Zouaoui, Shahrear Iqbal, Xueyuan Han, and Thomas Pasquier. 2025. ORTHRUS: Achieving High Quality of Attribution in Provenance-based Intrusion Detection Systems. *Security Symposium (USENIX Sec'25)* (2025).
[33] jiangdie666. 2024. *Question about wget dataset results.* GitHub. https://github.com/FDUDSDE/MAGIC/issues/13 Created: 2024-05-23.
[34] Joud Khoury, Timothy Upthegrove, Armando Caro, Brett Benyo, and Derrick Kong. 2020. An Event-based Data Model for granular information flow tracking. *12th International Workshop on Theory and Practice of Provenance* (2020).
[35] Isaiah J. King, Xiaokui Shu, Jiyong Jang, Kevin Eykholt, Taesung Lee, and H. Howie Huang. 2023. EdgeTorrent: Real-time Temporal Graph Representations for Intrusion Detection. *26th International Symposium on Research in Attacks, Intrusions and Defenses* (2023).
[36] Diederik P. Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. *International Conference on Learning Representations* (2014).
[37] Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations* (2017).
[38] Jelena Kovacevic. 2007. How to encourage and publish reproducible research. *2007 IEEE International Conference on Acoustics, Speech and Signal Processing* (2007).
[39] Shriram Krishnamurthi. 2023. Examining "Reproducibility in Computer Science". https://cs.brown.edu/~sk/Memos/Examining-Reproducibility/ Memo.
[40] Shaofei Li, Feng Dong, Xusheng Xiao, Haoyu Wang, Fei Shao, Jiedong Chen, Yao Guo, Xiangqun Chen, and Ding Li. 2024. NODLINK: An Online System for Fine-Grained APT Attack Detection and Investigation. *31st Annual Network and*

Distributed System Security Symposium (2024).
[41] m shayan73. 2024. *Code for Evaluating DARPA TC E5.* GitHub. https://github.com/threaTrace-detector/threaTrace/issues/21 Created: 2024-11-22.
[42] m shayan73. 2024. *DARPA TC E5 Files Used.* GitHub. https://github.com/threaTrace-detector/threaTrace/issues/20 Created: 2024-11-22.
[43] m shayan73. 2024. *E5 Clearscope Results.* GitHub. https://github.com/ubc-provenance/kairos/issues/18 Created: 2024-11-21.
[44] m shayan73. 2024. *Issues with pre-trained models.* GitHub. https://github.com/threaTrace-detector/threaTrace/issues/17 Created: 2024-10-26.
[45] m shayan73. 2024. *Missing graph files in testing.* GitHub. https://github.com/ubc-provenance/kairos/issues/16 Created: 2024-11-21.
[46] m shayan73. 2024. *OpTC Files Used and Missing Groundtruth.* GitHub. https://github.com/ubc-provenance/kairos/issues/19 Created: 2024-11-21.
[47] m shayan73. 2024. *Use of testing data in training phase.* GitHub. https://github.com/threaTrace-detector/threaTrace/issues/18 Created: 2024-10-29.
[48] Emaad Manzoor, Sadegh M. Milajerdi, and Leman Akoglu. 2016. Fast Memory-efficient Anomaly Detection in Streaming Heterogeneous Graphs. *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016).
[49] margoseltzer. 2019. *Unicorn SC-1 Dataset.* https://github.com/margoseltzer/camflow-apt
[50] margoseltzer. 2019. *Unicorn SC-2 Dataset.* https://github.com/margoseltzer/shellshock-apt
[51] margoseltzer. 2019. *Unicorn Wget Dataset.* https://github.com/margoseltzer/wget-apt
[52] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. *27th International Conference on Neural Information Processing Systems* (2013).
[53] Kunal Mukherjee, Joshua Wiedemeier, Tianhao Wang, James Wei, Feng Chen, Muhyun Kim, Murat Kantarcioglu, and Kangkook Jee. 2023. Evading Provenance-Based ML Detectors with Adversarial System Actions. *32nd USENIX Security Symposium* (2023).
[54] Daniel Olszewski, Allison Lu, Carson Stillman, Kevin Warren, Cole Kitroser, Alejandro Pascual, Divyajyoti Ukirde, Kevin Butler, and Patrick Traynor. 2023. "Get in Researchers; We're Measuring Reproducibility": A Reproducibility Study of Machine Learning Papers in Tier 1 Security Conferences. *ACM SIGSAC Conference on Computer and Communications Security* (2023).
[55] Talha Ongun, Jack W. Stokes, Jonathan Bar Or, Ke Tian, Farid Tajaddodianfar, Joshua Neil, Christian Seifert, Alina Oprea, and John C. Platt. 2021. Living-Off-The-Land Command Detection Using Active Learning. *24th International Symposium on Research in Attacks, Intrusions and Defenses* (2021).
[56] DARPA OpTC-data. 2020. *DARPA Operationally Transparent Cyber Dataset.* https://github.com/FiveDirections/OpTC-data
[57] Simon Oya, Carmela Troncoso, and Fernando Pérez-González. 2019. Rethinking Location Privacy for Unknown Mobility Behaviors. *2019 IEEE European Symposium on Security and Privacy* (2019).
[58] purseclab. 2020. *Atlas's oversampling code.* https://github.com/purseclab/ATLAS/blob/main/atlas.py#L94
[59] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. 2020. Temporal Graph Networks for Deep Learning on Dynamic Graphs. *ICML Workshop on Graph Representation Learning and Beyond* (2020).
[60] Hami Satilmiş, Sedat Akleylek, and Zaliha Yüce Tok. 2024. A Systematic Literature Review on Host-Based Intrusion Detection Systems. *IEEE Access* (2024).
[61] sbustreamspot. 2016. *StreamSpot Dataset.* https://github.com/sbustreamspot/sbustreamspot-data
[62] Bernhard Schölkopf, John C. Platt, John Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. 2001. Estimating the Support of a High-Dimensional Distribution. *Neural Computation* (2001).
[63] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. 2021. Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification. *30th International Joint Conference onArtificial Intelligence* (2021).
[64] tallyAB. 2024. *Syntax error in generated DOT file.* GitHub. https://github.com/dhl123/Airtag-2023/issues/7 Created: 2024-11-10.
[65] tallyAB. 2024. *UDatasets preprocessed, non-tokenized log files are missing.* GitHub. https://github.com/dhl123/Airtag-2023/issues/3 Created: 2024-11-9.
[66] tallyAB. 2025. *Clarification on True Positive Counting and Cache Eviction Logic.* GitHub. https://github.com/PKU-ASAL/Simulated-Data/issues/16 Created: 2025-03-26.
[67] threaTrace detector. 2023. *ThreaTrace's DARPA TC E3 Training Script validate() function.* https://github.com/threaTrace-detector/threaTrace/blob/master/scripts/train_darpatc.py#L138
[68] threaTrace detector. 2023. *ThreaTrace's Streamspot Training Script validate() function.* https://github.com/threaTrace-detector/threaTrace/blob/master/scripts/train_streamspot.py#L295
[69] threaTrace detector. 2023. *ThreaTrace's Unicorn Training Script splitDataset() function.* https://github.com/threaTrace-detector/threaTrace/blob/master/scripts/train_unicornsc.py#L292
[70] Mati Ur Rehman, Hadi Ahmadi, and Wajih Ul Hassan. 2024. Flash: A Comprehensive Approach to Intrusion Detection via Provenance Graph Representation Learning . *45th IEEE Symposium on Security and Privacy* (2024), 3552–3570.
[71] Patrick Vandewalle, Jelena Kovacevic, and Martin Vetterli. 2009. Reproducible research in signal processing. *IEEE Signal Processing Magazine* (2009).
[72] Qi Wang, Wajih Ul Hassan, Ding Li, Kangkook Jee, Xiao Yu, Kexuan Zou, Junghwan Rhee, Zhengzhang Chen, Wei Cheng, Carl Gunter, and Haifeng Chen. 2020. You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis. *27th Annual Network and Distributed System Security Symposium* (2020).
[73] Su Wang, Zhiliang Wang, Tao Zhou, Hongbin Sun, Xia Yin, Dongqi Han, Han Zhang, Xingang Shi, and Jiahai Yang. 2022. THREATRACE: Detecting and Tracing Host-Based Threats in Node Level Through Provenance Graph Learning. *IEEE Transactions on Information Forensics and Security* (2022).
[74] Martijn Wieling, Josine Rawee, and Gertjan van Noord. 2018. Reproducibility in Computational Linguistics: Are We Willing to Share? *Computational Linguistics* (2018).
[75] Fan Yang, Jiacen Xu, Chunlin Xiong, Zhou Li, and Kehuan Zhang. 2023. PROGRAPHER: An Anomaly Detection System based on Provenance Graph Embedding. *32nd USENIX Security Symposium* (2023).
[76] Jun Zeng, Zheng Leong Chua, Yinfang Chen, Kaihang Ji, Zhenkai Liang, and Jian Mao. 2021. WATSON: Abstracting Behaviors from Audit Logs via Aggregation of Contextual Semantics. *28th Annual Network and Distributed System Security Symposium* (2021).
[77] Jun Zeng, Xiang Wang, Jiahao Liu, Yinfang Chen, Zhenkai Liang, Tat-Seng Chua, and Zheng Leong Chua. 2022. Shadewatcher: Recommendation-guided cyber threat analysis using system audit records. *43rd IEEE Symposium on Security and Privacy* (2022).
[78] Meihui Zhong, Mingwei Lin, Chao Zhang, and Zeshui Xu. 2024. A survey on graph neural networks for intrusion detection systems: Methods, trends and challenges. *Computers and Security* (2024).
[79] Michael Zipperle, Florian Gottwalt, Elizabeth Chang, and Tharam Dillon. 2022. Provenance-based Intrusion Detection Systems: A Survey. *Comput. Surveys* (2022).

A Editor-Evaluator Interaction Examples

In the following subsections we describe the sequence of interactions between the evaluators and editors in our team, and the authors of the PIDDL systems that we were examining. For each issue mentioned in the evaluator experience, in cases where the evaluators were unable to resolve the issues on their own, we show below the significant points of interaction.

A.1 ShadeWatcher

Missing Code.
Evaluator: Reported that the code did not generate the data required by the recommendation model.
Editor: Investigated and found that the interaction extraction component (required to generate this data) was missing. Requested the missing code from the authors, but received no response.

Investigating Workarounds for the Missing Code.
Editor: Found Provninja's fork of ShadeWatcher. Recommended that the evaluator examine how it implements the missing interaction extraction component.
Evaluator: Reported that the interaction extraction code in Provninja's fork does not align with the original paper.
Editor: Verified that Provninja's approach is inconsistent with the method described in the original paper.
Editor: Contacted ProvNinja's authors to request access to their fork after its removal. Received no response.

A.2 Atlas

Ambiguous README Instructions.
Evaluator: Reported a FileNotFound error during training, with no context or explanation provided in the README.

Editor: Recommended that the evaluator manually modify the hardcoded flags in `atlas.py` to first generate the resampled files, and later load them for training.
Evaluator: Applied the fix and reported that the error was successfully circumvented.
Evaluator: Reported that the README provided unclear instructions on manually cleaning predicted entities before evaluation, making it difficult to determine which entities to include or exclude.
Editor: Verified evaluator's cleaned entities. Recommended sharing the complete workflow (first generating resampled files and later loading them for training by modifying hardcoded flags) and the cleaned entities with the author for validation.
Evaluator: Shared the workflow and cleaned entities with the author for validation and guidance. Received no response.

Inconsistency Between Paper and Shared Code.
Evaluator: Reported that the oversampling strategy described in the paper is not implemented. Instead, the code duplicates malicious samples.
Editor: Verified the issue and recommended that the evaluator contact the author for guidance.
Evaluator: Reported the inconsistency to the author and requested guidance. Received no reply.

Runtime Error.
Evaluator: Reported a runtime error in `graph_reader.py`.
Editor: Investigated the cause of the error, and recommended that the evaluator modify `graph_generator.py` to ensure that node names and IP attributes are enclosed in quotation marks when written to the `.dot` file, so they are correctly parsed by `graph_reader.py`.
Evaluator: Applied the fix and reported that the error was successfully circumvented.

Impact of Noise Reduction.
Evaluator: Reported a significantly lower number of entities in the results compared to Atlas's reported results in the paper, along with lower F1 scores in the entity-level evaluation.
Editor: Investigated the issue and found that the entity counts reported in Table 4 of the Atlas paper (computed as TP + FP + TN + FN) correspond to the number of entities before noise reduction, as seen in Figure 8. Since the code included noise reduction as part of its pipeline, it was more appropriate to report results after noise reduction. Recommended that the evaluator report the issue to the author.
Evaluator: Reported the issue to the author. Received no reply.

A.3 ThreaTrace

Variability Across Training Run Results.
Evaluator: Reported that ThreaTrace produced inconsistent results, with evaluation results across different training runs varying by more than 5%. Suggested that the authors may have used a fixed random seed for stability.
Editor: Asked the evaluator to verify that no residual data remained between training runs and reach out to the author to see if a specific seed value was used.
Author: Verified that no specific seed value was used.
Evaluator: Verified that no residual data remained between training runs.
Editor: Reported the variability in results to the authors.
Author: Acknowledged the issue but no solution was provided.
Editor: Recommended that the evaluator run (both training and testing) ThreaTrace three times per dataset and average the resulting metrics.

Use of Testing Data In Training.
Evaluator: Noticed that the validation sets used during the training phase had potential overlap with the test data in the Streamspot, Unicorn, and DARPA TC E3 training scripts.
Editor: Verified and asked the evaluator to escalate the issue to the author.
Author: Acknowledged the issue, and provided reasoning for overlap in the DARPA TC E3 training script. No explanation was provided for Streamspot and Unicorn training scripts. No code was provided.
Editor: Suggested that the evaluator remove the code causing the train/test overlap in the DARPA TC E3 training script, and re-run the experiment.
Evaluator: Reported a significant performance drop on the DARPA TC E3 datasets after removing the code causing train/test overlap.
Editor: Verified and reported the issue to the author.
Author: Acknowledged the issue, but no solution was provided.

Issues with Pre-trained Models.
Evaluator: Reported that the provided pre-trained models for StreamSpot and TRACE did not work as expected and produced poor results, and that a `threshold_unicorn.txt` (required for testing the pre-trained model for the Unicorn dataset) was missing from the repository.
Editor: Verified and asked the evaluator to escalate the issue to the author.
Author: Acknowledged the issues, but did not provide the missing threshold file or any fix for the pre-trained models.

Missing Script and Ground Truth.
Evaluator: Reported the absence of a ground truth and training script for DARPA TC E5 datasets (and hence any hard-coded file names for training and testing as in the DARPA TC E3 script).
Editor: Verified and escalated the issue to the author.
Author: Provided reasoning for the missing training script and ground truth, and recommended contacting the authors of Kairos [10] to obtain the ground truth. No training script, file names for training/testing, or ground truth was provided.
Editor: Since Kairos's DARPA TC E5 ground truth is a list of malicious time windows (and hence sub-graphs), it was not directly applicable (without significant effort) to ThreaTrace, which is a node-level detector. Furthermore, without concrete file names for training and testing, evaluation on DARPA TC E5 was not possible.

A.4 AirTag

Missing Dataset Resources.
Evaluator: Reported that, unlike the other datasets, there are no run logs (copies of the system's output provided by the authors as a record of their exact run results) for the UDatasets to compare results to.
Editor: Compared the run log results for the other datasets with the graphs in the paper (which did not provide exact values) and found that the trends matched exactly. Recommended continuing the evaluation of the UDatasets and comparing the results to approximate values derived from the paper's graphs.
Evaluator: Reported that there are no scripts or instructions to evaluate the UDatasets.
Editor: Investigated and identified the SDataset scripts as a suitable alternative for evaluating the UDatasets since both datasets were simulated in single-host environments. Recommended attempting the evaluation using those scripts.
Evaluator: Reported that the untokenized version of the UDatasets was missing which was necessary for evaluation.
Editor: Verified that the files were missing and escalated the issue to the author.
Author: Uploaded new files in response.
Evaluator: Replied that the uploaded files did not correspond to the UDatasets and requested the correct files. The author did not respond.

Problematic Deployment Details.
Evaluator: (At the start of the evaluation effort) Reported that AirTag's evaluation was done in a setup with multiple GPUs and questioned whether running the system in our single-GPU environment would require significant code changes.
Editor: (Based on prior experience with AirTag) Informed the evaluator that modifying the GPU IDs in the bash script to match our environment should be sufficient to get the system running.
Evaluator: Confirmed that the system was running and that results for SDatasets were obtainable. However, reported that the MDatasets were taking too long to run (no results after 5 hours), despite having changed the GPU IDs—it was assumed that the code was using the GPU.
Editor: Investigated and found that the evaluator was using a CPU version for one of the libraries. Concluded that the results for SDatasets were likely produced via a CPU run and the MDatasets failed due to being more compute intensive. Recommended switching to the GPU version of the library and trying again.
Evaluator: Confirmed the MDatasets were now running and results were successfully obtained.

Runtime Error.
Evaluator: Reported a bug in the graph file generated by AirTag.
Editor: Recommended that the evaluator run the graph generation code to check whether the bug persisted.
Evaluator: Confirmed that running the code reproduced the exact same graph file and that the bug persisted.
Editor: Verified the issue and escalated it to the author.
Author: Acknowledged the issue, explained the cause of the bug, and suggested a fix.
Evaluator: Applied the fix but encountered a different bug after resolving the first one. Reported the new issue to the author.
Author: Suggested another fix.
Evaluator: Implemented the fix and confirmed that the issue was resolved.

A.5 NodLink

Paper-Code Inconsistency.
Evaluator: Reported that NodLink was only printing node-level recall. They were able to obtain node-level precision from the code, but graph-level metrics could not be obtained.
Editor: Investigated the issue and found that, based on the definitions of node-level and graph-level results, it should at least be possible to obtain graph-level precision from the code. Explained the reasoning to the evaluator and recommended trying again.
Evaluator: Reported that they were able to obtain graph-level precision successfully.
Editor: Noted during the investigation that NodLink's true positive calculation and cache management technique did not appear to align with the paper's description, and recommended that the evaluator examine this further.
Evaluator: Confirmed that this seemed to be the case.
Editor: Escalated the issue to the authors for verification. We did not receive a response.

Undocumented Structural Modification in Code.
Evaluator: Reported that the repository was evaluating three different datasets and that it was not clear which datasets from the paper these corresponded to. The README did not clarify this.
Editor: Verified the issue and recommended that the evaluator escalate it to the authors for clarification.
Author: Clarified that the three datasets corresponded to the single "In-Lab Arena" dataset from the paper.
Evaluator: Reported they were able to obtain results from the three sub-datasets but the repository README did not specify how these should be aggregated. Noted that averaging the results yielded better performance than the metrics reported in the paper.
Editor: Suggested that the single dataset result in the paper was likely an aggregation—not an average—of the three sub-dataset results. Recommended that the evaluator reach out to the authors with their aggregation results and method for verification.
Author: Confirmed that this was indeed how the results had been aggregated.

A.6 FLASH

Inconsistent and Missing Code.
Evaluator: Noted that although the code contains a function to train a Word2Vec model (`train_word2vec_model`), it is never used. Word2Vec is instead loaded from a pre-trained library.
Editor: Contacted the authors to ask how to run Word2Vec training from scratch, and what should be passed as the `train_file_path` input.
Author: Responded that the benign training logs from the OpTC dataset should be used. Once decompressed, the JSON files can be parsed using the provided `extract_logs` function and passed to

the training function.
Evaluator: Then reported that the GNN and XGBoost training scripts also required training file paths, but the README and code did not specify which OpTC logs to use.
Editor: Followed up with the author asking for clarification on the appropriate benign files for training the GNN and XGBoost models, since only evaluation files (e.g., 0201, 0501, 0051) were clearly indicated. No response was received.

Inefficient Code.
Evaluator: Reported that retraining on the Unicorn dataset was prohibitively slow and referred to a potential fix suggested in a GitHub issue.
Editor: Reviewed the recommendation (noting that we cannot confirm whether or not the user suggesting the resolution on GitHub was one of the authors) and decided to test its effectiveness.
Evaluator: Implemented the fix; while the code ran, the results were inconsistent.

A.7 Kairos

File Generation Bug.
Evaluator: Reported a bug in two different notebooks that they could not resolve. Also reported a significant performance issue in a third notebook, which was unusual since other notebooks with results were close to the paper's reported metrics.
Editor: Investigated the issue and recommended that the evaluator escalate it to the authors. Noted that the issues across the three notebooks were likely caused by the same underlying bug and asked the evaluator to examine this further.
Author: Responded that the students who had worked on the project had graduated.
Evaluator: Confirmed that the bug appeared to be the same across all three notebooks and was manifesting differently due to the variations in code structure.

A.8 Magic

Labeling and Data Organization Errors.
Evaluator: Reported inconsistent results when training on parsed raw logs versus using the provided `graph.pkl` and checkpoint, despite ensuring correct file ordering (first 25 attack logs, next 125 normal).
Editor: Recommended trying to rerun the code on multiple different machines, while ensuring the same setup as the authors. Issues persisted, so reached out to author.
Author: Confirmed the issue was due to unsorted filenames in `wget_parser.py` and shared corrected results after sorting.
Evaluator: Clarified they had already applied sorting, but still observed discrepancies.
Author: Reproduced evaluator's results and identified the root cause as inconsistent random seeds across files.
Evaluator: Applied fix and confirmed improved performance.