

Behavior-Specialized Anomaly Detection for Improving Alert Quality in Provenance-Based Intrusion Detection

Zahra Fazli

Mehdi Kharrazi

kharrazi@sharif.edu

Article

Keywords:

Posted Date: October 22nd, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11142579/v1>

License:  This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

Behavior-Specialized Anomaly Detection for Improving Alert Quality in Provenance-Based Intrusion Detection

Zahra Fazli¹ and Mehdi Kharrazi^{1*}

¹Sharif University of Technology, Tehran, I.R. Iran.

*Corresponding author(s). E-mail(s): kharrazi@sharif.edu;

Abstract

Provenance-based intrusion detection systems (PIDSs) provide a promising approach for detecting sophisticated attacks by modeling the causal relationships among system entities. However, the practical usefulness of their alerts remains limited by poor alert quality. Existing approaches may produce coarse-grained alerts, excessive false positives, or incomplete detection of attack-related entities, increasing the effort required for forensic investigation. These limitations reflect a fundamental trade-off between alert precision and recall: suppressing false alerts may result in missed attack entities, while increasing recall may overwhelm analysts with noisy alerts.

In this paper, we present *Targeter*, a behavior-specialized node-level intrusion detection framework for improving alert quality in provenance graphs. *Targeter* exploits the intrinsic behavioral characteristics of system processes rather than modeling all process behaviors using a single anomaly detector. It partitions processes into behaviorally meaningful categories and trains a dedicated lightweight graph autoencoder for each category. During inference, processes are deterministically routed to their corresponding category-specific models, enabling each model to learn a more homogeneous distribution of normal behavior. We assess alert quality using precision, recall, and F_2 -score.

Experiments on six DARPA Transparent Computing benchmark datasets show that *Targeter* achieves a favorable precision-recall trade-off and F_2 -score than existing node-level PIDSs, resulting in improved alert quality. *Targeter* also maintains low computational overhead, achieving fast inference, low GPU memory consumption, and practical execution on commodity CPUs.

Keywords: Provenance Graph, Intrusion Detection, Behavioral Nature, Mixture of Experts, Anomaly Detection

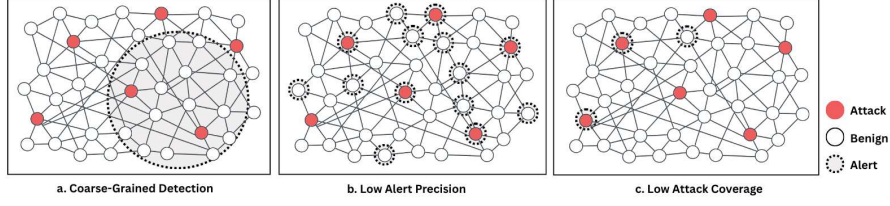


Fig. 1 Limitations of existing PIDS approaches

1 Introduction

Cyberattacks are rapidly increasing in complexity and stealth, this has created serious challenges for traditional detection methods, which often generate alerts without considering the context of events. The inability to detect advanced attacks, such as APTs, as well as zero-day attacks, has led to the development of new detection approaches based on provenance graphs [1–9]. Provenance graphs are directed acyclic graphs that model system behavior and the relationships between different entities. In these graphs, nodes represent system entities like processes, files, and sockets, while edges show interactions such as reading, writing, and executing. This representation helps preserve the underlying cause-and-effect structure. The use of provenance graphs in intrusion detection has attracted significant attention in both rule-based [9–14] and anomaly-based [3–6, 15–17] approaches, collectively referred to as Provenance-based Intrusion Detection Systems (PIDS).

Despite their improved ability to detect complex attacks, PIDSs have not yet seen widespread adoption. Deploying these systems faces multiple challenges, ranging from high processing overheads to limited detection accuracy, as addressed by several works [3–5, 8, 18]. However, an equally important aspect is the quality of the alerts produced by a PIDS and the effort required to analyze them. An alert is useful if it provides sufficiently accurate information about suspicious entities while retaining enough of the attack-related activity to support further investigation, which is influenced by three key factors:

Coarse-Grained Detection: Early provenance-based approaches often generate alerts at the level of subgraphs [3, 16, 19] or execution paths [8], rather than identifying individual malicious entities. As illustrated in Figure 1(a), such alerts usually contain both benign and malicious nodes, forcing analysts to manually identify the truly compromised entities. Dong et al. [20] report that Morse [10] and Holmes [9] require analysts to inspect hundreds of nodes and edges per alert, while Unicorn [16] may generate alert subgraphs containing up to 176,000 nodes and 2.8 million edges. Such coarse-grained alerts substantially increase the effort required for forensic investigation.

Low Alert Precision: To overcome coarse-grained detection, recent PIDSs have shifted toward node-level anomaly detection [4–6, 17, 21]. Although these approaches improve localization of malicious entities, they often produce a large number of false alerts, as illustrated in Figure 1(b). The resulting alert volume significantly increases the effort required to distinguish genuine attack nodes from benign ones. As reported

in [6], methods such as THREATTRACE [4], MAGIC [17], and FLASH [5] suffer from very low precision due to excessive false positives.

Low Attack Coverage: Improving precision, however, can come at the cost of missing attack-related entities. As illustrated in Figure 1(c), a detector that reports only a small subset of the attack may produce fewer false alerts while providing incomplete attack coverage. Kairos [15], ORTHRUS [6] and Velox[18], reduce false alerts, but their reported results show that only a small fraction of attack-related entities may be identified. For example, on the DARPA CADETS-E5 dataset [22], ORTHRUS identifies only 2 out of 124 attack nodes. Low attack coverage requires analysts to perform additional investigation to identify the undetected parts of the attack. In our evaluation, attack coverage is quantified using recall.

Consequently, from a forensic perspective, the quality of generated alerts depends on two complementary properties: how accurately the alerts identify attack-related entities and how completely they expose the attack. We refer to this as *alert quality*. High-quality alerts should therefore be node-level and provide sufficient precision to limit unnecessary investigation while maintaining sufficient recall to avoid leaving substantial portions of an attack undetected. This perspective suggests that evaluating a PIDS system solely by its precision or recall can provide an incomplete picture of its practical usefulness.

To capture this trade-off, we require an evaluation metric that considers both alert precision and recall rather than optimizing either of them in isolation. In the context of PIDSs, however, the importance of the two objectives is not necessarily equal. For example, missing half of the attack-related entities can be more detrimental to forensic investigation than incorrectly reporting half of the generated alerts. Therefore, while alert precision remains important for controlling the investigation burden, we place greater emphasis on attack recall. This motivates the use of the F_2 -score as the primary combined metric in our evaluation, as it jointly considers precision and recall while assigning greater weight to recall.

This perspective raises an important question: *how can a PIDS system improve alert quality?* One possible direction is to employ increasingly complex detection models. However, increasing model complexity does not necessarily address how normal system behavior is represented. Different categories of system processes naturally exhibit different behavioral patterns during normal operation. For example, web browsers, command-line shells and system services interact with files, processes, network resources, and other system entities in different ways. Modeling these heterogeneous behaviors within a single detection mechanism may make it more difficult to distinguish benign behavioral variations from anomalous activities.

Motivated by this observation, we investigate whether behavioral specialization can improve the alert quality. Rather than modeling all processes using the same detection mechanism, we group processes according to their behavioral characteristics and model the resulting categories separately. Our hypothesis is that detectors trained on more behaviorally consistent groups can characterize normal activity within each group more accurately, thereby improving recall while maintaining sufficient precision.

Based on this idea, we introduce *Targeter*, a behavior-specialized framework for node-level provenance-based intrusion detection. Targeter categorizes processes

according to their behavioral characteristics and deterministically routes each process to a category-specific lightweight anomaly detector. Each detector is therefore trained on a behaviorally consistent subset of processes rather than on the complete process population. Targeter follows the general principle of model specialization used in mixture-of-experts(MoE) approaches[23–27], but does not employ a learned gating mechanism or expert competition. Instead, it uses deterministic routing based on pre-defined process categories, allowing us to study the effect of behavioral specialization without introducing an additional learned routing mechanism.

We evaluate Targeter on the DARPA Transparent Computing benchmark, including both Engagement 3 [28] and the more challenging Engagement 5 [22]. We evaluate the proposed approach in terms of precision, recall, and F_2 -score, and conduct controlled experiments and ablation studies to examine the contribution of behavioral specialization. We also analyze the computational overhead of the framework to determine whether the benefits of specialization can be achieved without substantial execution costs.

The main contributions of this work are summarized as follows:

- We highlight the precision–recall trade-off as an important consideration in evaluating node-level alert quality and motivate the use of the F_2 -score when attack coverage is considered more important than precision.
- We investigate behavioral specialization as a mechanism for improving alert quality by modeling different categories of system processes separately rather than relying on a single detector for heterogeneous process behaviors.
- We conduct an extensive evaluation on the DARPA Transparent Computing datasets and analyze the effectiveness of behavioral specialization in terms of precision, recall, F_2 -score, and execution overhead.

2 Related Work

The use of provenance graphs has become a powerful tool for attack detection, thanks to their ability to effectively represent system behavior. Existing studies can be broadly categorized into three directions according to their primary objective: rule-based intrusion detection, learning-based intrusion detection, and attack investigation. While the first two categories aim to identify malicious activities from provenance data, the third focuses on reconstructing and understanding attacks after suspicious activities have been detected. In the following, we review these research directions and discuss their respective strengths and limitations.

2.1 Rule-Based PIDSs

Rule-based approaches utilize predefined signatures or expert-crafted rules to flag suspicious activity. These systems often encode known attack patterns or behaviors using rule templates derived from threat intelligence sources such as the MITRE ATT&CK framework[29]. In general, the main advantage of rule-based methods is their low runtime overhead. However, their inability to detect zero-day attacks and the manual

definition of rules, which may be incomplete, poses significant challenges for these approaches.

SLEUTH [11] and its successors Morse[10], Conan[30], Aptshield[31], CAPTAIN[14], generate initial tags by matching real-time provenance data against handcrafted rules. The detection process is further continued by propagating these tags across the provenance graph, allowing context-aware correlation and state updates for involved entities and final alert generation. To reduce the dependency on manual rule creation, Trec [13] proposes an automated rule generation approach. Trec extracts detection rules directly from structured attack knowledge by analyzing synthetic attack graphs built on top of the MITRE ATT&CK model, offering a scalable and less labor-intensive alternative to manual rule generation.

2.2 Learning-Based PIDSs

To overcome the limitations of handcrafted rules, learning-based approaches attempt to model normal system behavior directly from provenance data and identify deviations as potential attacks. According to the provenance object on which anomaly detection is performed, existing learning-based PIDSs can be broadly categorized into four groups:

- **Subgraph-Based Detection:** Some methods like Stream-Spot [19], Unicorn[16], TBDetector[32] and Prographer [3] aim to identify anomalous topological structures within provenance graphs that may correspond to malicious behavior. These methods suffer from coarse-grained detection.
- **Path-Based Detection:** Path-level approaches detect attacks by analyzing anomalous execution paths within provenance graphs rather than individual entities. ProvDetector [8] extracts causal paths from provenance graphs and learns normal path representations using path embedding techniques. During inference, execution paths that deviate from the learned patterns are identified as suspicious. More recently, Sentient [33] argues that a system entity may participate in both benign and malicious activities simultaneously. Instead of relying solely on node-level information, it jointly models multiple execution paths traversing each entity to better capture its role within an attack campaign.
- **Node-Based Detection:** Approaches such as THREATTRACE [4], FLASH [5], and NODLINK [21] identify suspicious entities (e.g., files, processes, sockets) that deviate from typical behavior patterns. THREATTRACE initializes node embeddings using the number and types of incoming and outgoing edges, and then applies GraphSAGE[34] to learn node representations for anomaly detection. FLASH follows a similar pipeline but employs Word2Vec-based initialization and incorporates temporal semantics in its model. MAGIC[17] employs a graph autoencoder with a masking mechanism and learns to reconstruct masked node features. A common challenge reported for these approaches is the generation of false positives, which can reduce detection precision.
- **Edge-Based Detection:** Techniques such as SHADEWATCHER [1], Kairos [15], ORTHRUS [6], and VELOX [18] detect anomalous interactions or data flows between entities, which can then be used to infer compromised nodes. ORTHRUS,

for instance, trains an autoencoder to predict edge types (e.g., read, write and execute), where inconsistencies between the predicted and observed edge types indicate anomalous interactions. Since provenance graphs typically contain substantially more edges than nodes, edges-level approaches may incur higher computational costs. VELOX demonstrates that such costs can be substantially reduced through a lightweight feature-engineering pipeline rather than complex graph neural models.

Despite significant progress, existing anomaly-based approaches continue to face important limitations. Current methods still struggle to balance recall and precision, leading either to excessive false positives that increase analysts’ investigation effort or to low recall that leaves parts of the attack undetected.

2.3 Provenance-Based Attack Investigation

Attack investigation methods generally fall into two broad categories. The first category includes methods that build upon initial alerts, aiming to expand and reconstruct the full scope of an attack. This includes seed expansion techniques, such as those used in Back-Propagating[35] and Atlas[36], which start from known malicious nodes (seeds) and iteratively grow the attack subgraph by following provenance relationships. Similarly, neighborhood aggregation approaches, such as Distdet [37], Airtag [38], and NODLINK [21], connect additional attack-relevant nodes based on their proximity and interaction patterns within the graph. These methods improve detection accuracy by post-processing the generated alerts.

The second category consists of methods designed to organize and summarize system activity, often independent of any specific alert. These approaches aim to reduce analyst workload by grouping related events into meaningful units. For example, clustering-based systems like Watson [39] and DepComm [2] automatically identify clusters of correlated events and generate human-readable reports, helping analysts make sense of large volumes of provenance data even in the absence of predefined attack seeds.

Although attack investigation methods can substantially facilitate forensic analysis by expanding, organizing, and summarizing alerts, they are inherently dependent on the outputs of the underlying intrusion detection system. Inaccurate or incomplete detection inevitably propagates to the investigation stage, limiting the effectiveness of subsequent attack reconstruction. Therefore, improving the quality of node-level detection remains a prerequisite for effective provenance-based attack investigation.

3 MOTIVATION & PRELIMINARY

In this section, we first present a motivational example to highlight the importance of the behavioral Specialization of system entities, then briefly introduce the Mixture-of-Experts (MoE) paradigm as a general model-specialization principle and clarify how Targeter differs from conventional MoE architectures.

Since foundational concepts such as provenance graphs [40], as well as common techniques in graph neural networks (e.g., GCN [41]), are already well-established in the provenance based intrusion detection literature, we do not revisit them here.

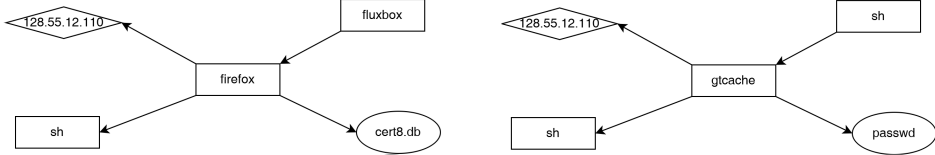
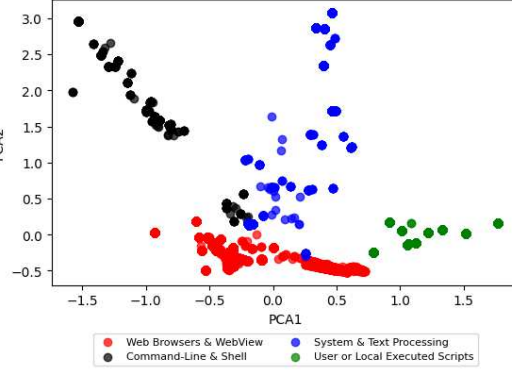


Fig. 2 Left image corresponds to a benign firefox process, and right one depicts a malicious process named gtcache.

Fig. 3 Two-dimensional PCA projection of process node features from the DARPA THEIA-E3, showing four representative categories.



3.1 Motivation

The motivation for behavior specialization stems from a simple observation: the meaning of an observed system action depends on the behavioral context of the process performing that action. An interaction that is common during the normal execution of one type of process may be unusual for another. Therefore, identifying whether a process is behaving anomalously requires not only considering the observed interactions, but also understanding the behavioral characteristics of the process itself.

Figure 2 illustrates this issue using two provenance subgraphs extracted from the DARPA THEIA-E3 dataset. The left subgraph corresponds to a benign execution of *Firefox*, which belongs to the *Web Browsers & WebView* category. The browser is launched by the window manager *fluxbox*, establishes a network connection to the remote host *128.55.12.110*, and accesses its internal certificate database (*cert8.db*). These interactions are consistent with the normal behavior of a web browser.

In contrast, the right subgraph corresponds to a malicious process named *gtcache*, which belongs to the *User or Local Executed Scripts* category. Its executable is created by *Firefox*, launched through *sh*, communicates with the same remote host, and attempts to access the sensitive system file */etc/passwd*.

Although these two subgraphs exhibit similar local graph structures, their behavioral contexts are fundamentally different. Existing graph-based detection methods primarily rely on structural patterns learned from provenance graphs and therefore fail to distinguish these two cases effectively. For example, MAGIC [17] incorrectly identifies both subgraphs as malicious, whereas FLASH [5] and THREATTRACE [4] fail to detect either of them. The example suggests that structural information alone may

not always provide sufficient context for distinguishing benign and malicious process behavior.

This observation raises a more general question: *do different process categories exhibit sufficiently distinct behavioral characteristics to justify specialized detection models?* To investigate this question, we analyze process instances from the DARPA THEIA-E3 dataset. For each process instance, we extract a provenance-derived feature vector that captures both semantic information, such as command-line arguments, file paths, and network addresses, and structural characteristics describing interactions with other system entities. We then apply Principal Component Analysis (PCA) [42] to project the resulting high-dimensional feature vectors into a two-dimensional space for visualization.

Figure 3 presents the resulting projection for four representative process categories: *Web Browsers & WebView*, *Command-Line & Shell*, *System & Text Processing*, and *User or Local Executed Scripts*. Although PCA is used solely for visualization, the figure provides a qualitative visualization of the distribution of process categories. This observation suggests that provenance-derived features capture meaningful behavioral characteristics that naturally distinguish different classes of system processes. The complete behavioral categorization adopted in this work is summarized in Table 2, while the classification methodology is described in Section 5.3.

The example and dataset-level analysis together motivate our central hypothesis: *processes with different behavioral characteristics may require different models of normal activity*. Instead of requiring a single anomaly detector to characterize the behavior of all processes, specializing detection models according to process behavior may provide a more appropriate representation of normal activity within each group. In the remainder of this work, we investigate whether such behavioral specialization can improve the alert quality in node-level PIDS.

3.2 Mixture of Experts

Mixture of Experts is a model architecture that combines multiple expert models, with a routing mechanism determining which expert or experts process each input. Its central idea is to allow different experts to specialize in different subsets of the input space rather than requiring a single model to capture all patterns. Early MoE architectures used learned gating mechanisms to assign inputs to experts [24], while subsequent approaches introduced sparse routing strategies that activate only a subset of available experts for each input [25, 27]. Routing mechanisms can also exploit structural information when inputs are represented as graphs; for example, Wang et al. [26] use local graph information to determine expert assignment.

Targeter is inspired by the general principle of model specialization in MoE, but it does not implement a conventional learned MoE architecture. Specifically, Targeter does not learn a gating function, perform expert competition, or aggregate the outputs of multiple experts. Instead, each process is deterministically assigned to a single category according to its predefined behavioral class, and the corresponding category-specific anomaly detector produces the detection result. Thus, Targeter does not perform learned expert selection or output aggregation. This distinction is important

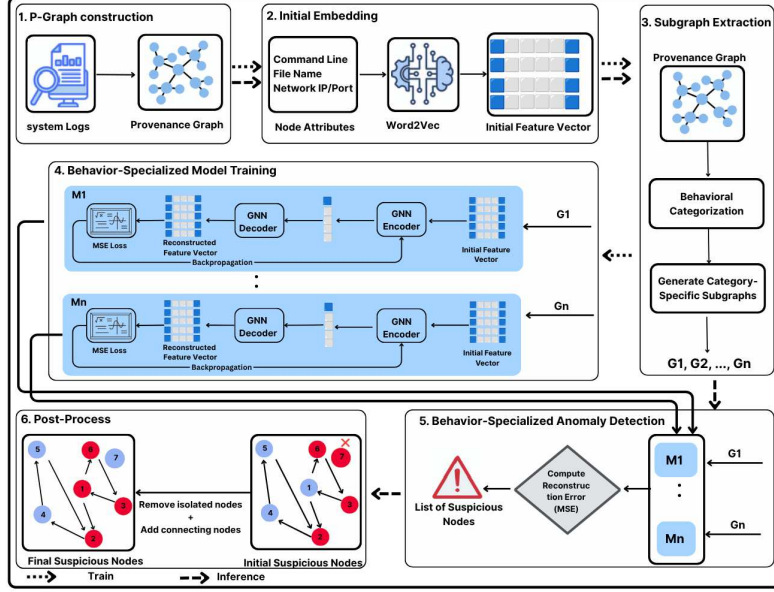


Fig. 4 Overview of Targeter’s framework.

because our objective is to isolate and evaluate the effect of behavioral specialization rather than the effect of a learned gating mechanism.

4 Threat Model & Assumptions

In Targeter, we assume that the integrity of the system logs is preserved and that neither the logs nor the derived provenance graph have been altered or forged by an adversary. Addressing scenarios in which attackers tamper with audit logs or inject false provenance data requires a separate line of defense and is beyond the scope of this work. Furthermore, our method is designed specifically for detecting attacks within a single host environment. The challenges associated with identifying coordinated or distributed attacks across multiple systems, where inter-host communication patterns must be analyzed, are not considered in this study and remain as future work.

5 Design

Targeter is a behavior-specialized, node-level anomaly detection framework for provenance-based intrusion detection which is designed to identify attack-related entities while improving the quality of generated alerts. The key idea of Targeter is to explicitly incorporate the intrinsic behavioral characteristics of system processes into the anomaly detection pipeline. Instead of training a single detector to model all process behaviors, Targeter first groups processes according to their behavioral semantics and then learns an independent behavior-specialized anomaly detection model for each

category. During inference, each process is evaluated only against the model corresponding to its behavioral category, and significant deviations from the learned normal behavior are reported as anomalies.

Figure 4 presents the overall architecture of Targeter, which consists of the following six components:

- **P-Graph Construction:** System audit logs are transformed into a provenance graph, where nodes represent system entities (e.g., processes, files, and sockets) and edges capture their interactions.
- **Initial Embedding:** Each node is assigned an initial feature vector by applying the Word2Vec embedding technique to its associated attributes.
- **Subgraph Extraction:** The provenance graph is decomposed into multiple category-specific subgraphs according to the behavioral category of their process nodes.
- **Behavior-Specialized Model Training:** An independent graph autoencoder is trained for each behavioral category to learn its normal behavioral patterns.
- **Behavior-Specialized Anomaly Detection:** During inference, each node is evaluated using the model associated with its behavioral category. A node is identified as suspicious when its reconstruction error exceeds a predefined threshold.
- **Post-Processing:** The raw suspicious nodes produced by the anomaly detectors are further refined by removing isolated alerts and adding connecting nodes between suspicious entities.

The following subsections describe each component of the proposed framework in detail.

5.1 P-Graph Construction

Similar to previous approaches [4, 5, 9, 15, 16, 43], the proposed framework analyzes system audit logs to detect potential attacks. In the first step, audit logs are collected and preprocessed to extract the information required for subsequent analysis and transform raw log records into a structured representation. The system’s behavior is then modeled as a provenance graph, where nodes represent system entities (i.e., processes, files, or network connections), and edges represent interactions between them. Each node in the graph is associated with three attributes: an identifier, a label and a feature.

- The identifier is a unique ID assigned to each entity when it first appears in the audit logs.
- The label indicates the type of the node, which can be one of *file*, *process*, or *network*.
- The feature encodes node-specific attributes which is file path for file nodes, the command line (cmdline) for process nodes, and the source/destination IP addresses and port numbers for network nodes.

Edges in the provenance graph include the source and destination node identifiers, the type of interaction (e.g., execute, open, read, write, clone, send, and receive), and the timestamp of the event. In system audit logs, identical events may recur at different timestamps, resulting in multiple edges of the same type between the

same source-destination node pair. Such duplicated edges do not contribute additional causal information; instead, they increase graph size and introduce redundancy. Therefore, following the approach adopted in [44] and similarly to the approaches used in [6, 15, 17, 35], we remove duplicate edges while preserving the underlying causal relationships.

Moreover, our proposed framework relies only on the existence and direction of edges in its modeling. As a result, multiple edges of different types between the same pair of nodes do not affect the model’s behavior. Consequently, we retain only the direction of the edge between two nodes, regardless of the interaction type. This abstraction preserves the structural connectivity between entities while reducing graph size and computational overhead.

5.2 Initial Embedding

At this stage, each node is assigned a feature vector that represents its intrinsic characteristics. These vectors are derived from node-specific attributes, including the name and path for files, the command line for processes and the IP address and port number for source and destination for network entities. The initial embeddings are then generated based on these attributes using Word2Vec [45], as outlined in the following steps.

- **Data Cleaning:** Non-semantic components such as numeric directories in system paths (e.g., process IDs like `/proc/320/`) are removed to reduce embedding noise and improve model generalization.
- **Tokenization:** Feature values are partitioned into discrete tokens. Each token serves as a linguistic unit, and the complete feature value is treated as a sentence.
- **Model Training:** A Word2Vec model is trained on all tokenized feature values from the training dataset to learn meaningful semantic representations of tokens.
- **Embedding Calculation:** The initial node embedding $h_v^{(1)}$ is obtained by computing the normalized sum of its constituent token embeddings, as defined in Equation 1, where $h_v^{(1)}$ denotes the initial embedding vector of node v , S represents the set of semantic tokens derived from the node’s feature value, and v_w denotes the Word2Vec embedding corresponding to token w .

$$h_v^1 = \frac{1}{|S|} \sum_{w \in S} v_w \quad (1)$$

This approach ensures that nodes with similar features, such as files with identical paths or processes with analogous command lines, receive similar vector representations. Moreover, by removing non-semantic components such as process identifiers (e.g., 320 in `/proc/320/`) and transient numerical elements in system paths, it enhances robustness against numeric variations in the raw data. These initial embeddings are subsequently refined by the behavior-specialized graph autoencoders through neighborhood aggregation, enabling the model to capture structural dependencies within each subgraph.

5.3 Subgraph Extraction

Subgraph extraction is the step through which Targeter separates heterogeneous process behaviors before model training. Process nodes are assigned to predefined behavioral categories, while file and network nodes are included in the corresponding process neighborhoods without being assigned separate behavioral categories.

Process categorization is performed using command-line-derived behavioral semantics through a deterministic rule-based strategy. The categorization procedure considers the functional role of a process together with its observed interactions, including accessed entities and invoked system calls. Each process is deterministically assigned to one of the predefined behavioral categories. If a process does not clearly match any predefined category, it is assigned to a generic category. This allows previously unseen or uncommon processes to remain within the detection pipeline rather than being excluded.

Importantly, process categories are not used as detection labels; rather, they serve solely to route processes to their corresponding behavior-specialized anomaly detection models. Consequently, detection remains fully anomaly-driven and unsupervised, independent of the categorization strategy.

After process categorization, the provenance graph is decomposed into category-specific subgraphs. Let $\mathcal{C} = \{C_1, \dots, C_n\}$ denote the set of process categories, where C_i contains the processes assigned to category i . Because the provenance graph is directed, we define the one-hop incoming neighborhood of a node v as

$$\mathcal{N}_v^1 = \{u \mid (u, v) \in E\}. \quad (2)$$

The k -hop incoming neighborhood is defined recursively as

$$\mathcal{N}_v^k = \mathcal{N}_v^{k-1} \cup \{u \mid \exists w \in \mathcal{N}_v^{k-1}, (u, w) \in E\}. \quad (3)$$

Thus, $\mathcal{N}_v^k$ contains nodes that can reach v through a directed path of length at most k . For each process category C_i , Targeter constructs a subgraph

$$G_i = (V_i, E_i), \quad (4)$$

where V_i contains all processes belonging to C_i together with their k -hop incoming neighbors:

$$V_i = C_i \cup \bigcup_{v \in C_i} \mathcal{N}_v^k. \quad (5)$$

The edge set is defined as

$$E_i = \{(u, v) \in E \mid u \in V_i, v \in V_i\}. \quad (6)$$

Note that subgraphs corresponding to different process categories could be disconnected or may overlap, as nodes within the k -hop neighborhood of multiple categories can be included in multiple subgraphs.

5.4 Behavior-Specialized Model Training

As described in the previous subsection, the provenance graph is decomposed into behavior-specific subgraphs according to the behavioral category of their process nodes. Targeter then trains an independent anomaly detection model for each behavioral category. Since each model is trained using only the subgraphs associated with a single behavioral category, it learns the normal behavioral characteristics of that category without being influenced by the heterogeneous behaviors of unrelated processes.

All behavior-specialized models share the same architecture and hyperparameters, while being trained independently on different behavioral categories. This design isolates the learning process of each category and enables every model to capture category-specific normal behavioral patterns using only unlabeled benign data.

Each behavior-specialized model is implemented as a graph autoencoder. The encoder consists of three Graph Convolutional Network (GCN) layers [41], which iteratively aggregate information from the local neighborhood of each node while progressively compressing its feature representation.

$$\mathbf{h}_v^{(l)} = \phi \left(\sum_{u \in \mathcal{N}(v) \cup \{v\}} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)} \right), \quad \mathbf{W}^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$$

$$\mathbf{h}_v^{(l)} \in \mathbb{R}^{d_l}, \quad \text{where } d_l = \frac{d_{l-1}}{2} \quad (7)$$

where ϕ is the ReLU activation function, d_l is the dimensionality of the vector at each layer and $\mathbf{W}^{(l)}$ is the learnable weight matrix for layer l . Furthermore, $\mathbf{h}_v^{(1)}$ denotes the initial feature vector of each node, which is derived from word2vec embeddings.

The node representations are updated layer by layer and passed through the encoder sequentially, which inherently performs multi-hop message passing and embedding learning together. The final output $\mathbf{z}_v = \mathbf{h}_v^{(L)}$ is the learned low-dimensional embedding for node v , where L is the number of model layers. The decoder then reconstructs the input features using fully connected layers, expanding the vector back to d_1 dimensions.

$$\hat{\mathbf{x}} = \mathbf{W}_{dec}^{(L)} \phi \left(\mathbf{W}_{dec}^{(L-1)} \phi \left(\dots \phi \left(\mathbf{W}_{dec}^{(1)} \mathbf{z} \right) \right) \right) \quad (8)$$

The training objective is to minimize the reconstruction error using the Mean Squared Error (MSE) loss, where $\|\mathbf{x}_i - \hat{\mathbf{x}}_i\|$ denotes the Euclidean distance between the original feature vector $\mathbf{x}_i$ and its reconstructed counterpart $\hat{\mathbf{x}}_i$.

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N} \sum_{i=1}^N \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2, \quad (9)$$

where $\mathbf{x}_i$ is the original feature of node i and $\hat{\mathbf{x}}_i$ is the reconstructed feature. N is the number of nodes in subgraphs. This unsupervised reconstruction objective

enables each behavior-specialized model to learn compact representations that jointly capture the structural and semantic characteristics of normal system behavior within its corresponding behavioral category.

5.5 Behavior-Specialized Anomaly Detection

When it comes to analyzing unseen samples, the input graph is first decomposed into n subgraphs based on the categories assigned to each process. The nodes in subgraph G_i either belong to category i or lie within the k -hop neighborhood of nodes in category i . Each generated subgraph is evaluated using the behavior-specialized model corresponding to its behavioral category.

Since subgraphs may overlap, a node can appear in multiple subgraphs. Nevertheless, anomaly assessment is performed only with respect to the node’s own behavioral category. Consequently, although neighboring information from other categories contributes to representation learning, the anomaly score of each process is computed only by the model associated with its assigned behavioral category. Specifically, for each subgraph associated with category i , we compute the reconstruction error $\|\hat{\mathbf{x}}_v - \mathbf{x}_v\|$ exclusively for nodes that belong to category i , and use this value as the anomaly score.

Larger anomaly scores indicate that the node features cannot be well reconstructed by the model, implying a deviation from the learned benign patterns; such nodes are thus considered suspicious. Accordingly, a node v in subgraph G_i is classified as suspicious if it satisfies two conditions: (i) it belongs to category i , and (ii) its anomaly score in G_i , denoted by S_v^i , exceeds the predefined threshold for that category. A category-specific threshold is determined from the validation data as the maximum reconstruction error observed for that behavioral category.

$$S_v^i = \|\hat{\mathbf{x}}_v^i - \mathbf{x}_v\|, \quad v \in G_i \quad (10)$$

$$\text{Suspicious}(v) = \begin{cases} 1, & \text{if } S_v^i > \tau_i \text{ and } v \in \text{category}(i) \\ 0, & \text{otherwise} \end{cases} \quad (11)$$

5.6 Post-Process

Finally, the suspicious nodes detected from all behavior-specialized models are aggregated and further refined through a lightweight post-processing stage. The refinement consists of two complementary operations: removing nodes that are less likely to be part of an attack and adding nodes that were previously missed.

Remove Isolated Nodes: Nodes that have no anomalous neighbors within two hops are removed from the list of suspicious nodes. This helps filter out a large number of false positives that may have been incorrectly flagged as part of an attack. This is illustrated in part F of Figure 4, where node number 7 is removed.

Add Connecting Nodes: Some nodes that lie along paths connecting anomalous nodes are added to the list of suspicious nodes. This allows for the inclusion of passive entities, file and network nodes, which may not have been detected in the previous step. This is illustrated in part F of Figure 4, where node number 1 is added.

Table 1 Dataset partitions used for training, validation, and testing.

Dataset	Train		Test		Validation		Attack Nodes
	Nodes	Edges	Nodes	Edges	Nodes	Edges	
THEIA-E3	403,603	18,178,809	735,605	19,405,071	54,498	1,136,036	118
THEIA-E5	716,396	4,477,652	340,954	3,591,469	107,638	554,865	70
CLEARSCOPE-E3	65,534	2,309,734	80,929	3,215,117	19,059	730,453	41
CLEARSCOPE-E5	53,962	24,092,129	71,230	36,911,961	5,944	575,667	22
CADETS-E3	493,607	22,723,596	214,844	8,401,651	21,061	128,916	72
CADETS-E5	2,522,378	195,218,537	1,148,210	61,755,070	1,033,408	11,172,298	124

In the following section, we evaluate Targeter on the DARPA Transparent Computing datasets and compare it with state-of-the-art provenance intrusion detection systems in terms of attack coverage, alert precision, and execution overhead.

6 Evaluation

This section evaluates Targeter from four complementary perspectives. Specifically, we seek to answer the following research questions:

RQ1: How does Targeter compare with existing node-level PIDSs in terms of alert quality?

RQ2: How much does behavior-specialized modeling contribute to the overall detection performance?

RQ3: How does post-processing affect the alert quality?

RQ4: What computational overhead does Targeter introduce compared with existing PIDSs, and is it practical for deployment on commodity CPUs?

All evaluations were conducted on a system running Windows 11, equipped with an Intel Core i9-12900K processor (3.2 GHz, 16 cores), with 64 GB of memory and an NVIDIA GeForce RTX 4070 Ti GPU with 12 GB of memory.

Dataset: We evaluate Targeter using the DARPA Transparent Computing Engagements 3[28] and Engagements 5[22], focusing on three datasets in each: THEIA, CLEARSCOPE, and CADETS, which capture system activity on Ubuntu Linux, Android, and FreeBSD, respectively. For each dataset, we first identified the attack-related sections based on DARPA’s official documentation. These sections were used to define the test sets, while separate portions of the data were allocated for training and validation. The exact configurations and dataset partitions are detailed in the code repository. In general, the number of nodes (representing system entities) and edges (representing system events) varies across subsets; Table 1 provides a summary of these statistics for the training, validation, and test sets, including the number of nodes associated with attack activity in each case.

DARPA datasets include scenario-level descriptions of attack campaigns, but they do not provide fine-grained ground truth labels; specifically, they do not identify which

individual processes, files, or network connections are malicious or involved in the attack paths. This lack of detailed labeling presents challenges for evaluating detection methods, requiring researchers to rely on external annotations or manual inspection to construct meaningful labels. Among the publicly available annotations[4, 6, 46], the ground truth released by ORTHRUS[6] provides the most comprehensive node-level labels and has also been adopted by several recent studies. Therefore, we use the ORTHRUS annotations throughout our evaluation.

Table 2 Functional Categorization of System Processes in Unix-Based and Android Operating Systems

Category	Sub-Category	Samples	Unix	Android
System Services & Administration	System & Core Services	accounts-daemon, syslogd, system_server	✓	✓
	Security & Monitoring Tools	sudo, su, openssl	✓	✓
	System Monitoring & Environment Tools	who, whoami, hostname	✓	
	Time, Randomness, & Expression Utilities	fortune, date, expr	✓	
User Interaction & Communication	Hardware & Communication Services	com.android.bluetooth		✓
	Command-Line Application & Shell	sh, cmd , bash, vim	✓	✓
	Messaging Tools	write, msgs	✓	
Network Tools	Network Services	dhclient, sshd	✓	
	Network Command-Line Tools	ifconfig, netstat, wget	✓	
File & System Management Tools	Package Management & Installation	dpkg, apt-config, com.android.packageinstaller	✓	✓
	File Management & Archiving Utilities	cp, unlink, mkdir	✓	
	System & Text Processing Tools	grep, sort, cat	✓	
GUI Based	User Interface & Desktop Environment	xterm, com.android.systemui	✓	✓
	Web Browsers & WebView	Firefox, com.kk.browser	✓	✓
	Desktop Notifications & Indicators	indicator-session-service, indicator-sound-service	✓	
General Applications	Database Services	Postgres	✓	
	Scripting & Programming Languages	python, perl	✓	
	Development & Build Tools	make, md5	✓	
	Games, Media	com.tveazy.online		✓
	Local Executed Scripts	-	✓	
	Misc Apps	com.bhanu.torch		✓

Category Schema: Table 2 presents the functional categories used across the datasets. The categorization is provided in two variants: one for Unix-based systems and another for Android. Specifically, the CADETS and THEIA datasets adopt the Unix-based categorization, whereas the Clearscope dataset employs the Android categorization.

Training Setup: We use the Adam optimizer [47] with a learning rate of 0.001 and a weight decay of 5×10^{-4} . Dropout with a probability of 0.2 is applied after each non-linear activation to prevent over-fitting. To ensure deterministic training and reproducibility, all relevant random number generators (including those in PyTorch, NumPy, and CUDA) are seeded. The initial feature vector has a dimension of 128, and the encoder produces a final embedding of dimension 16. Each node is analyzed within a three-hop neighborhood.

Evaluation Metrics: All methods are evaluated at the node level. Each system entity is classified as either benign or malicious according to the adopted ground truth. Consequently, true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) correspond to correctly or incorrectly classified nodes.

Unlike several previous studies[4, 5], we do not consider a detected node as correct merely because it lies within the neighborhood of an attack node. Such relaxed matching artificially inflates detection performance while increasing the amount of manual investigation required from security analysts. Therefore, all reported results are based on exact node-level matching.

To assess the practical usefulness and quality of node-level alerts, we report both precision and recall. Precision measures the fraction of reported alerts that correspond to attack-related entities and therefore reflects the amount of noise introduced into the alert set. Recall measures the fraction of attack-related entities that are successfully identified and therefore reflects the completeness of attack detection. We use these two complementary measures to operationalize alert quality: a high-quality alert set should be sufficiently precise to limit unnecessary investigation while maintaining sufficiently high recall to expose a substantial portion of the attack.

We analyze precision and recall jointly and use the F_2 -score as the primary combined metric for alert quality. The F_2 -score incorporates both precision and recall while assigning greater importance to recall, reflecting our view that missing attack-related entities is more costly than reporting additional false alerts. The formula used to calculate the F_β -score [48] is defined as follows, with $\beta = 2$:

$$F_\beta = (1 + \beta^2) \times \frac{\text{Precision} \times \text{Recall}}{(\beta^2 \times \text{Precision}) + \text{Recall}}. \quad (12)$$

Comparable Works: Since this work focuses on node-level attack localization, we compare Targeter with recent node-level provenance intrusion detection systems. Coarse-grained approaches[3, 16, 19], which generate alerts at the sub-graph or path level, address a different problem setting and are therefore not included in the quantitative comparison. Instead, we compare our method with techniques that directly aim to identify attack nodes. The selected baseline methods include THREATTRACE[4], FLASH[5], MAGIC[17], and ORTHRUS[6]. We considered NODLINK[21] but excluded it because it generates alerts only for processes, whereas

Table 3 Comparison of Node-level Detection Accuracy in Targeter with previous works on darpa Engagement 3(E3). P stands for precision, and R stands for recall.

Dataset	Method	TP	FP	TN	FN	P	R	F2
THEIA	Targeter-raw	59	271	735k	59	0.18	0.50	0.37
	Targeter	63	295	735k	55	0.18	0.53	0.38
	ORTHRUS	19	9	735k	99	0.68	0.16	0.19
	MAGIC	118	229k	505k	0	0.00	1.00	0.00
	FLASH	9	186k	549k	109	0.00	0.08	0.00
	THREATRACE1	15	735k	87	0.67	0.26	0.30	
CLEARSCOPE	Targeter-raw	1	221	80k	40	0.00	0.02	0.01
	Targeter	1	221	80k	40	0.00	0.02	0.01
	ORTHRUS	1	499	80k	40	0.00	0.02	0.01
	MAGIC	41	55k	26k	0	0.00	1.00	0.00
	FLASH	12	32k	49k	29	0.00	0.29	0.00
	THREATRACE1	28k	52k	0	0.00	1.00	0.01	
CADETS	Targeter-raw	18	573	214k	54	0.03	0.25	0.10
	Targeter	29	843	214k	43	0.03	0.40	0.12
	ORTHRUS	26	21	215k	46	0.55	0.36	0.39
	MAGIC	72	117k	97k	0	0.00	1.00	0.00
	FLASH	11	2k	213k	61	0.01	0.15	0.03
	THREATRACE2	458	214k	70	0.00	0.03	0.01	

our evaluation considers all system entities. All baseline methods were re-executed under the same dataset partitions and evaluation protocol

RQ1: Alert Quality

Tables 3 and 4 compare Targeter with recent node-level provenance intrusion detection systems on the six benchmark datasets from DARPA Engagement 3 and Engagement 5. Since the primary objective of this work is to improve alert quality for forensic investigation, we evaluate the methods jointly in terms of precision, recall, and F_2 -score.

For completeness, the tables also report the results of *Targeter-Raw*, which corresponds to the output of the behavior-specialized anomaly detection stage before applying the post-processing described in Section 5. We use Targeter-Raw only to isolate the contribution of post-processing; its detailed analysis is presented in RQ3.

Overall, existing methods tend to optimize one objective at the expense of the other. ORTHRUS prioritizes alert precision by producing very few false positives, but consequently misses a large portion of attack-related entities. In contrast, MAGIC achieves almost perfect recall on most datasets but generates an overwhelming number of false alerts, making manual investigation impractical. FLASH and THREATRACE

Table 4 Comparison of Node-level Detection Accuracy in Targeter with previous works on darpa Engagement 5(E5).P stands for precision, and R stands for recall

Dataset	Method	TP	FP	TN	FN	P	R	F2
THEIA	Targeter-raw	30	404	40k	40	0.07	0.43	0.21
	Targeter	43	529	340k	27	0.08	0.61	0.25
	ORTHRUS	12	1	341k	58	0.92	0.17	0.20
	MAGIC	67	140k	200k	3	0.00	0.96	0.00
	FLASH	24	10k	331k	46	0.00	0.34	0.01
	THREATRACE	26	129k	212k	44	0.00	0.37	0.00
CLEARSCOPE	Targeter-raw	8	184	71k	14	0.04	0.36	0.14
	Targeter	10	238	71k	12	0.04	0.45	0.15
	ORTHRUS	2	5	71k	20	0.29	0.09	0.11
	MAGIC	22	40k	31k	0	0.00	1.00	0.00
	FLASH	19	70k	669	3	0.00	0.86	0.00
	THREATRACE	0	2k	69k	22	0.00	0.00	0.00
CADETS	Targeter-raw	61	1k	1M	63	0.05	0.49	0.18
	Targeter	61	875	1M	63	0.07	0.49	0.21
	ORTHRUS	2	10	1M	122	0.17	0.02	0.02
	MAGIC	124	1M	13k	0	0.00	1.00	0.00
	FLASH	2	6k	1M	122	0.00	0.02	0.00
	THREATRACE	7	171k	977k	117	0.00	0.06	0.00

also suffer from excessive false positives and unstable detection performance across different datasets.

Overall, Targeter provides a more favorable balance between precision and recall across most evaluated datasets. Compared with ORTHRUS, it substantially improves recall while maintaining acceptable alert precision. Compared with MAGIC, FLASH, and THREATRACE, it dramatically reduces the number of false positives while still detecting a considerably larger fraction of attack entities. This results in higher F_2 -scores and indicates that Targeter provides a more favorable alert quality under the evaluation criterion adopted in this work.

The only exception is the CADETS-E3 dataset, where Targeter achieves higher recall but a lower F_2 -score than ORTHRUS. CADETS-E3 contains a relatively small number of benign samples for several behavioral categories, limiting the ability of the corresponding behavior-specialized models to accurately learn normal behavior. Nevertheless, this limitation is not observed on the larger Engagement 5 datasets, where sufficient training samples are available for each category.

Another notable observation is the robustness of Targeter on the Engagement 5 datasets. Compared with Engagement 3, Engagement 5 contains substantially more sophisticated, stealthy, and multi-stage attack scenarios. While the performance of methods such as THREATRACE and FLASH noticeably deteriorates under these more challenging attacks, Targeter preserves a favorable balance between alert

Table 5 Effect of Behavior Specialization on Node-Level Detection

Dataset	No Behavioral Specialization		Random Categorization		Behavior-Specialized (Targeter)	
	TP	FP	TP	FP	TP	FP
THEIA-E3	0	1	1	47	63	295
THEIA-E5	1	30	9	652	43	529
CLEARSCOPE-E3	0	0	1	336	1	221
CLEARSCOPE-E5	2	21	9	485	10	238
CADETS-E3	3	958	16	1157	29	843
CADETS-E5	52	2360	59	4263	61	875

precision and recall across both engagements. This behavior suggests that behavior-specialized modeling may be more robust to the increased complexity of the evaluated Engagement 5 scenarios.

RQ2: Contribution of Behavior-Specialized Modeling

To assess the contribution of behavior-specialized modeling, we compare Targeter with two alternative configurations. In the first configuration, all processes are assigned to a single category, and a single anomaly detector is trained for the entire provenance graph, effectively removing behavioral specialization. In the second configuration, the same number of categories is retained, but processes are assigned to categories randomly rather than according to their behavioral characteristics. All other components of the framework remain unchanged.

Table 5 summarizes the results. When behavioral specialization is removed, the number of correctly detected attack nodes generally decreases across the evaluated datasets. This suggests that modeling all process behaviors using a single detector makes it more difficult to capture the diversity of normal system behavior.

Random categorization provides some improvement over the single-model configuration; however, its performance remains consistently below that of the proposed behavior-based categorization. Since randomly formed groups do not necessarily share similar behavioral characteristics, each model is still required to learn relatively heterogeneous behavior patterns.

In contrast, the proposed behavior-based categorization generally provides a better detection trade-off across the evaluated datasets. For instance, on THEIA-E3, the number of correctly detected attack nodes increases from 0 in the single-model configuration and 1 with random categorization to 63 using behavior-specialized models. At the same time, behavior-based categorization may also reduce false positives. On CADETS-E5, the number of false positives decreases from 2360 for the single-model configuration and 4263 for random categorization to 875 with behavior-based categorization.

Overall, these results suggest that the improvement achieved by Targeter is not solely due to employing multiple models. Rather, organizing processes according to their behavioral characteristics appears to provide more suitable training data for each anomaly detector, leading to improved node-level attack detection.

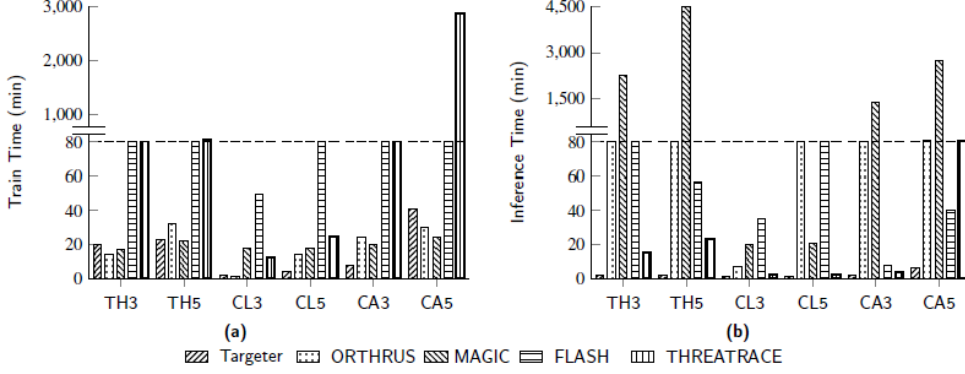


Fig. 5 Training and inference time of **Targeter**, **Orthrus**, **Magic**, **Flash** is shown. (a) Training time; (b) inference time.

RQ3: Effect of Post-Processing

To isolate the contribution of the post-processing stage, we compare the raw output of Targeter (Targeter-Raw) with the final output of Targeter. Table 3 and Table 4 report the corresponding precision, recall, and F_2 -score.

Overall, post-processing increases the number of correctly identified attack-related entities while also introducing additional alerts. With the exception of Clearscope-E3, Targeter achieves better F_2 -score after post-processing. The increase in TP leads to higher recall, while the additional alerts may affect precision depending on the dataset. Importantly, the resulting increase in recall is sufficient to improve the overall F_2 -score across the evaluated datasets.

The improvement in recall is primarily attributable to the second stage of post-processing, which adds connecting nodes between detected suspicious entities. These nodes may not be directly identified by the anomaly detectors but can represent intermediate entities involved in the attack path. At the same time, removing isolated suspicious nodes helps suppress alerts that lack local support from other suspicious entities.

These results indicate that post-processing complements rather than replaces the behavior-specialized detection stage. The anomaly detectors provide the initial suspicious entities, while post-processing refines and expands these detections based on their structural relationships in the provenance graph.

RQ4: Computational Efficiency & Deployment Feasibility

Beyond detection performance, practical deployment also requires low computational overhead. We therefore evaluate Targeter in terms of training time, inference time, GPU utilization, GPU memory consumption, and CPU-only execution.

Figure 5 compares the training and inference times of all evaluated methods. Overall, Targeter achieves one of the shortest training times while consistently providing the fastest inference across the evaluated datasets. FLASH and THREATTRACE exhibit

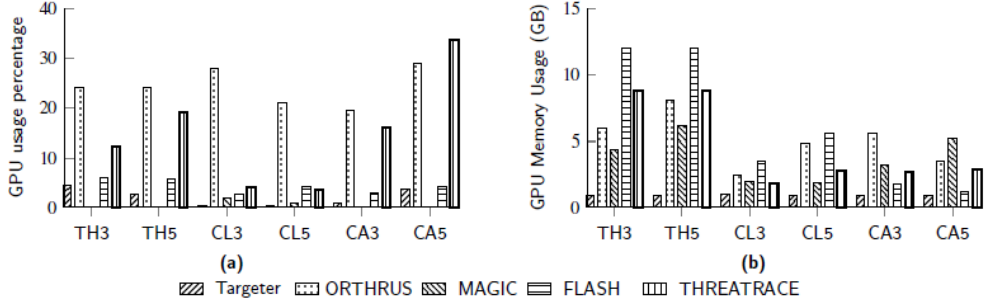


Fig. 6 Comparison of GPU resource consumption at test time by **Targeter**, **ORTHURUS**, **MAGIC**, **FLASH** and **THREATRACE** is shown in terms of (a) GPU usage percentage and (b) GPU memory usage.

substantially longer training times, whereas ORTHRUS and MAGIC require training times comparable to Targeter.

The advantage of Targeter becomes more evident during inference. Across all datasets, Targeter consistently requires less inference time than the competing methods. Although THREATRACE performs competitively on smaller datasets, its inference time increases considerably on larger workloads, particularly on CADETS-E5. In contrast, the inference time of Targeter scales more gracefully with dataset size, indicating better computational scalability.

Figure 6 illustrates GPU utilization and GPU memory consumption during inference. The average GPU utilization of Targeter is considerably lower than that of most other methods, with MAGIC being the main exception. For example, on THEIA-E5, Targeter uses only 2.66% GPU utilization, compared with 24%, 19%, and 5.8% for ORTHRUS, THREATRACE, and FLASH, respectively. Targeter also demonstrates the lowest average GPU memory consumption among the evaluated methods and maintains a relatively stable memory footprint as dataset size increases.

While GPU-based evaluation demonstrates the computational efficiency of Targeter, practical deployment may require execution on commodity servers without dedicated GPUs. We therefore further evaluate Targeter under CPU-only execution.

Table 6 summarizes the CPU execution results. Targeter remains computationally practical across all evaluated datasets even without GPU acceleration. For example, processing the THEIA-E3 dataset, which contains approximately three days of system activity, requires 214 seconds of detection time while utilizing an average CPU usage of 51% and 1.8 GB of memory. Similar resource consumption is observed across the remaining datasets.

To provide a more comprehensive assessment of runtime overhead, Table 6 separately reports preprocessing time and detection time. The preprocessing stage consists of parsing raw audit logs and transforming them into the intermediate representation required by the detection pipeline. The reported detection time includes all remaining stages of the framework, including provenance graph construction, initial feature generation, subgraph extraction, behavior-specialized anomaly detection, and post-processing, rather than only the neural network inference stage.

Table 6 CPU Resource Consumption of Targeter

Dataset	Preprocess Time	Test Time	CPU Usage	Memory Usage	Data Period
THEIA-E3	8min	214s	51%	1.8GB	3 days , 4 hours
THEIA-E5	11min	135s	44%	1.3GB	23 hours
CLEARSCOPE-E3	16min	27s	52%	0.7GB	21 hours
CLEARSCOPE-E5	30min	49s	48%	0.9GB	2 days, 18 hours
CADETS-E3	10min	252s	55%	1.2GB	3days, 23 hours
CADETS-E5	84min	391s	53%	3.4GB	1 day, 1 hour

Overall, these results demonstrate that the behavioral specialization adopted by Targeter does not introduce prohibitive computational overhead despite employing multiple category-specific models. Targeter achieves competitive training time, fast inference, low GPU memory consumption, and remains practical under CPU-only execution, supporting its feasibility for deployment on commodity computing platforms.

7 Discussion and Limitations

The experimental results demonstrate that behavior-specialized anomaly detection improves alert quality while maintaining low execution overhead. Beyond these quantitative findings, several design choices of Targeter warrant further discussion, along with the current limitations of the proposed framework.

7.1 Behavior-Specialized Modeling

A key design principle of Targeter is to explicitly incorporate behavioral semantics into provenance-based anomaly detection. Rather than training a single model to capture the behaviors of all system processes, Targeter partitions processes into behaviorally meaningful categories and trains an independent anomaly detector for each category. This behavior-specialized design enables each detector to model a more homogeneous distribution of normal activities while reducing interference among fundamentally different process behaviors.

7.2 Benefits of the Modular Architecture

Beyond improving detection performance, the modular architecture of Targeter provides several practical advantages.

- **Adaptation to Normality Shift.** Normal system behavior naturally evolves over time due to software updates, configuration changes, and workload variations. Such normality shifts often require anomaly detection models to be retrained. OWAD [49] shows that ignoring normality shift can significantly degrade performance, reporting a 32% F1-score drop for DeepLog [43] on the BGL dataset [50] over 42 days. Since Targeter consists of multiple independent behavior-specialized models, changes affecting a particular behavioral category may require retraining only the corresponding model rather than the entire detection system. This modularity has the potential to reduce adaptation cost in dynamic environments.

- **Category-Specific Detection Policies.** The modular architecture also enables category-specific detection policies. Previous threat reports indicate that certain process categories, such as command interpreters and scripting engines, are disproportionately targeted during real-world attacks [51]. Consequently, different anomaly thresholds can be assigned to different behavioral categories according to their operational risk, providing additional flexibility for practical deployments.
- **Extensibility.** Since each behavioral category is modeled independently, additional categories can be incorporated without redesigning the overall framework. This modular design facilitates incremental extension as new classes of system behaviors emerge.

7.3 Current Limitations

Although the proposed framework demonstrates promising results, several limitations remain. First, the current implementation relies on manually defined behavioral categories. Although the experimental results demonstrate the effectiveness of behavior-specialized anomaly detection, automatically discovering behavioral categories remains an open research problem and an important direction for future work.

Second, the current framework specializes only process entities. Files and network entities are modeled through their interactions with processes but are not assigned dedicated behavioral categories. Extending behavioral specialization to additional entity types may further improve attack localization.

Finally, the evaluation focuses on benchmark provenance datasets under a single-host setting. Assessing the proposed framework under larger-scale deployments and continuously evolving workloads remains an important direction for future investigation.

8 Conclusion

In this paper, we presented Targeter, a behavior-specialized anomaly detection framework for provenance-based intrusion detection. Targeter leverages the intrinsic behavioral characteristics of system processes by categorizing them into behaviorally meaningful groups and learning an independent anomaly detection model for each category. This behavior-specialized design enables normal behaviors to be modeled more accurately while preserving the semantic differences among heterogeneous process types.

Experimental results on the DARPA Engagement 3 and Engagement 5 datasets demonstrate that behavioral specialization provides a better balance between alert precision and recall than existing node-level provenance intrusion detection systems. In addition, the proposed framework achieves this improvement while maintaining low computational overhead.

Data Availability

The DARPA Transparent Computing datasets used in this study are publicly available from the DARPA Transparent Computing program (engagement 3[28], engagement

5[22]). The ground-truth annotations used in this study are available through the ORTHRUS resources ([Zenodo record](#)[6]). The exact train/validation/test partitions and preprocessing scripts are available in our [GitHub repository](#).

Code Availability

The source code used to implement Targeter, generate the experimental results, and reproduce the reported experiments is available at [GitHub](#).

References

- [1] Zengy, J., Wang, X., Liu, J., Chen, Y., Liang, Z., Chua, T.-S., Chua, Z.L.: Shadewatcher: Recommendation-guided cyber threat analysis using system audit records. In: 2022 IEEE Symposium on Security and Privacy (SP), pp. 489–506 (2022). IEEE
- [2] Xu, Z., Fang, P., Liu, C., Xiao, X., Wen, Y., Meng, D.: Depcomm: Graph summarization on system audit logs for attack investigation. In: 2022 IEEE Symposium on Security and Privacy (SP), pp. 540–557 (2022). IEEE
- [3] Yang, F., Xu, J., Xiong, C., Li, Z., Zhang, K.: Prographer: An anomaly detection system based on provenance graph embedding. In: 32nd USENIX Security Symposium (USENIX Security 23), pp. 4355–4372 (2023)
- [4] Wang, S., Wang, Z., Zhou, T., Yin, X., Han, D., Zhang, H., Sun, H., Shi, X., Yang, J.: Threatrace: Detecting and tracing host-based threats in node level through provenance graph learning. *IEEE Transactions on Information Forensics and Security* (2022)
- [5] Rehman, M.U., Ahmadi, H., Hassan, W.U.: Flash: A comprehensive approach to intrusion detection via provenance graph representation learning. In: 2024 IEEE Symposium on Security and Privacy (SP), pp. 139–139 (2024). IEEE Computer Society
- [6] Jiang, B., Bilot, T., El Madhoun, N., Al Agha, K., Zouaoui, A., Iqbal, S., Han, X., Pasquier, T.: Orthrus: Achieving high quality of attribution in provenance-based intrusion detection systems. In: Security Symposium (USENIX Sec’25). USENIX (2025)
- [7] Han, D., Wang, Z., Feng, R., Jin, M., Chen, W., Wang, K., Wang, S., Yang, J., Shi, X., Yin, X., *et al.*: Rules refine the riddle: Global explanation for deep learning-based anomaly detection in security applications. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, pp. 4509–4523 (2024)
- [8] Wang, Q., Hassan, W.U., Li, D., Jee, K., Yu, X., Zou, K., Rhee, J., Chen, Z., Cheng, W., Gunter, C.A., *et al.*: You are what you do: Hunting stealthy malware

via data provenance analysis. In: NDSS (2020)

- [9] Milajerdi, S.M., Gjomemo, R., Eshete, B., Sekar, R., Venkatakrishnan, V.: Holmes: real-time apt detection through correlation of suspicious information flows. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 1137–1152 (2019). IEEE
- [10] Hossain, M.N., Sheikhi, S., Sekar, R.: Combating dependence explosion in forensic analysis using alternative tag propagation semantics. In: 2020 IEEE Symposium on Security and Privacy (SP), pp. 1139–1155 (2020). IEEE
- [11] Hossain, M.N., Milajerdi, S.M., Wang, J., Eshete, B., Gjomemo, R., Sekar, R., Stoller, S., Venkatakrishnan, V.: {SLEUTH}: Real-time attack scenario reconstruction from {COTS} audit data. In: 26th USENIX Security Symposium (USENIX Security 17), pp. 487–504 (2017)
- [12] Li, Z., Wei, Y., Shen, X., Wang, L., Chen, Y., Xu, H., Ji, S., Zhang, F., Hou, L., Liu, W., et al.: Marlin: Knowledge-driven analysis of provenance graphs for efficient and robust detection of cyber attacks. arXiv preprint arXiv:2403.12541 (2024)
- [13] Lv, M., Gao, H., Qiu, X., Chen, T., Zhu, T., Chen, J., Ji, S.: Trec: Apt tactic/technique recognition via few-shot provenance subgraph learning. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, pp. 139–152 (2024)
- [14] Wang, L., Shen, X., Li, W., Li, Z., Sekar, R., Liu, H., Chen, Y.: Incorporating gradients to rules: Towards lightweight, adaptive provenance-based intrusion detection. In: NDSS (2025)
- [15] Cheng, Z., Lv, Q., Liang, J., Wang, Y., Sun, D., Pasquier, T., Han, X.: Kairos: Practical intrusion detection and investigation using whole-system provenance. In: 2024 IEEE Symposium on Security and Privacy (SP), pp. 3533–3551 (2024). IEEE
- [16] Han, X., Pasquier, T., Bates, A., Mickens, J., Seltzer, M.: Unicorn: Runtime provenance-based detector for advanced persistent threats. NDSS (2020)
- [17] Jia, Z., Xiong, Y., Nan, Y., Zhang, Y., Zhao, J., Wen, M.: {MAGIC}: Detecting advanced persistent threats via masked graph representation learning. In: 33rd USENIX Security Symposium (USENIX Security 24), pp. 5197–5214 (2024)
- [18] Bilot, T., Jiang, B., Li, Z., El Madhoun, N., Al Agha, K., Zouaoui, A., Pasquier, T.: Sometimes simpler is better: A comprehensive analysis of state-of-the-art provenance-based intrusion detection systems. In: 34th USENIX Security Symposium (USENIX Security 25), pp. 7193–7212 (2025)

- [19] Manzoor, E., Milajerdi, S.M., Akoglu, L.: Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1035–1044 (2016)
- [20] Dong, F., Li, S., Jiang, P., Li, D., Wang, H., Huang, L., Xiao, X., Chen, J., Luo, X., Guo, Y., *et al.*: Are we there yet? an industrial viewpoint on provenance-based endpoint detection and response tools. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, pp. 2396–2410 (2023)
- [21] Li, S., Dong, F., Xiao, X., Wang, H., Shao, F., Chen, J., Guo, Y., Chen, X., Li, D.: Nodlink: An online system for fine-grained apt attack detection and investigation. NDSS (2024)
- [22] DARPA: DARPA Transparent Computing Engagement 5. <https://github.com/darpa-i2o/Transparent-Computing/blob/master/README.md>. Accessed: 2024-11-11 (2019)
- [23] Chen, Z., Deng, Y., Wu, Y., Gu, Q., Li, Y.: Towards understanding the mixture-of-experts layer in deep learning. Advances in neural information processing systems **35**, 23049–23062 (2022)
- [24] Jacobs, R.A., Jordan, M.I., Nowlan, S.J., Hinton, G.E.: Adaptive mixtures of local experts. Neural computation **3**(1), 79–87 (1991)
- [25] Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., Dean, J.: Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. International Conference on Learning Representations(ICLR) (2017)
- [26] Wang, H., Jiang, Z., You, Y., Han, Y., Liu, G., Srinivasa, J., Kompella, R., Wang, Z., *et al.*: Graph mixture of experts: Learning on large-scale graphs with explicit diversity modeling. Advances in Neural Information Processing Systems **36**, 50825–50837 (2023)
- [27] Fedus, W., Zoph, B., Shazeer, N.: Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. Journal of Machine Learning Research **23**(120), 1–39 (2022)
- [28] DARPA: DARPA Transparent Computing Engagement 3. <https://github.com/darpa-i2o/Transparent-Computing/blob/master/README-E3.md>. Accessed: 2024-11-11 (2018)
- [29] The MITRE Corporation: MITRE ATT&CK. <https://attack.mitre.org/>. Accessed: 2024-02-26 (2024)
- [30] Xiong, C., Zhu, T., Dong, W., Ruan, L., Yang, R., Cheng, Y., Chen, Y., Cheng, S., Chen, X.: Conan: A practical real-time apt detection system with high accuracy

- and efficiency. *IEEE Transactions on Dependable and Secure Computing* **19**(1), 551–565 (2020)
- [31] Zhu, T., Yu, J., Xiong, C., Cheng, W., Yuan, Q., Ying, J., Chen, T., Zhang, J., Lv, M., Chen, Y., et al.: Aptshield: A stable, efficient and real-time apt detection system for linux hosts. *IEEE Transactions on Dependable and Secure Computing* (2023)
 - [32] Wang, N., Wen, X., Zhang, D., Zhao, X., Ma, J., Luo, M., Nie, S., Wu, S., Liu, J.: Tbdetector: Transformer-based detector for advanced persistent threats with provenance graph. *arxiv 2023. arXiv preprint arXiv:2304.02838*
 - [33] Yan, W., An, N., Qiao, W., Wu, W., Jiang, B., Liu, Y., Lu, Z., Liu, J.: Sentient: Multi-scenario behavioral intent analysis for advanced persistent threat detection. *arXiv preprint arXiv:2502.06521* (2025)
 - [34] Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs. In: *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30 (2017)
 - [35] Fang, P., Gao, P., Liu, C., Ayday, E., Jee, K., Wang, T., Ye, Y.F., Liu, Z., Xiao, X.: {Back-Propagating} system dependency impact for attack investigation. In: *31st USENIX Security Symposium (USENIX Security 22)*, pp. 2461–2478 (2022)
 - [36] Alsaheel, A., Nan, Y., Ma, S., Yu, L., Walkup, G., Celik, Z.B., Zhang, X., Xu, D.: Atlas: A sequence-based learning approach for attack investigation. In: *USENIX Security Symposium*, pp. 3005–3022 (2021)
 - [37] Dong, F., Wang, L., Nie, X., Shao, F., Wang, H., Li, D., Luo, X., Xiao, X.: {DISTDET}: A {Cost-Effective} distributed cyber threat detection system. In: *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 6575–6592 (2023)
 - [38] Ding, H., Zhai, J., Nan, Y., Ma, S.: {AIRTAG}: Towards automated attack investigation by unsupervised learning with log texts. In: *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 373–390 (2023)
 - [39] Zeng, J., Chua, Z.L., Chen, Y., Ji, K., Liang, Z., Mao, J.: Watson: Abstracting behaviors from audit logs via aggregation of contextual semantics. In: *NDSS* (2021)
 - [40] Michael, N., Mink, J., Liu, J., Gaur, S., Hassan, W.U., Bates, A.: On the forensic validity of approximated audit logs. In: *Proceedings of the 36th Annual Computer Security Applications Conference*, pp. 189–202 (2020)
 - [41] Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: *Proceedings of the 5th International Conference on Learning Representations (ICLR)* (2017)

- [42] Abdi, H., Williams, L.J.: Principal component analysis. Wiley interdisciplinary reviews: computational statistics **2**(4), 433–459 (2010)
- [43] Du, M., Li, F., Zheng, G., Srikumar, V.: Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 1285–1298 (2017). ACM
- [44] Xu, Z., Wu, Z., Li, Z., Jee, K., Rhee, J., Xiao, X., Xu, F., Wang, H., Jiang, G.: High fidelity data reduction for big data security dependency analyses. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 504–516 (2016)
- [45] Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. In: 1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings (2013)
- [46] Liu, J., Inam, A., Goyal, A., Westfall, K., Riddle, A., Bates, A.: Reapr: Recovery Every Attack Process. <https://bitbucket.org/sts-lab/reapr-ground-truth>. Accessed: 2025-07-31 (2023)
- [47] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
- [48] Christen, P., Hand, D.J., Kirielle, N.: A review of the f-measure: its history, properties, criticism, and alternatives. ACM Computing Surveys **56**(3), 1–24 (2023)
- [49] Han, D., Wang, Z., Chen, W., Wang, K., Yu, R., Wang, S., Zhang, H., Wang, Z., Jin, M., Yang, J., *et al.*: Anomaly detection in the open world: Normality shift detection, explanation, and adaptation. In: NDSS (2023)
- [50] Oliner, A., Stearley, J.: What supercomputers say: A study of five system logs. IEEE International Conference on Dependable Systems and Networks (DSN), 575–584 (2007)
- [51] Logpoint Security Research Team: Top Ten MITRE ATT&CK Techniques. Accessed: 2025-05-29 (2023). <https://www.logpoint.com/en/blog/logpoints-top-ten-mitre-attck-techniques/>

Funding

This work was partially supported by the Sharif University of Technology (SUT) Research Grant AI4030849.

Author Contributions

M.K. supervised the research and provided guidance throughout the study. Z.F. conducted the implementation, experiments, data analysis, and manuscript preparation. Both authors reviewed and approved the final manuscript.

Additional Information

Competing Interests

The authors declare no competing interests.