USENIX

THE ADVANCED COMPUTING
SYSTEMS ASSOCIATION

TAPAS: An Efficient Online APT Detection with Task-guided Process Provenance Graph Segmentation and Analysis

Bo Zhang, *Nanjing University of Science and Technology;* Yansong Gao, *The University of Western Australia;* Changlong Yu and Boyu Kuang, *Nanjing University of Science and Technology;* Zhi Zhang, *The University of Western Australia;* Hyoungshick Kim, *Sungkyunkwan University;* Anmin Fu, *Nanjing University of Science and Technology*

https://www.usenix.org/conference/usenixsecurity25/presentation/zhang-bo-tapas

This paper is included in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

TAPAS: An Efficient Online APT Detection with Task-guided Process Provenance Graph Segmentation and Analysis

Bo Zhang[1,*], Yansong Gao[2,*], Changlong Yu[1], Boyu Kuang[1], Zhi Zhang[2], Hyoungshick Kim[3], Anmin Fu[1,†]

[1]Nanjing University of Science and Technology
[2]The University of Western Australia
[3]Sungkyunkwan University,
{zhangbo07, yu_cl, kuang, fuam}@njust.edu.cn, {garrison.gao, zhi.zhang}@uwa.edu.au, hyoung@skku.edu

Abstract

Advanced Persistent Threats (APTs) pose critical security challenges to institutions and enterprises through sophisticated, long-duration attack campaigns. While recent APT detection methods primarily leverage *provenance graphs* constructed from kernel-level audit logs to reveal attack patterns, they face severe scalability limitations in production environments. The provenance graphs grow rapidly (several GB per day) and require long-term maintenance to capture APT campaigns that span months, creating prohibitive storage and computational overhead for real-time detection.

To address these challenges, we propose TAPAS, an efficient online APT detection framework that reduces graph dimensionality in both spatial and temporal spaces. For spatial dimensionality, TAPAS focuses on the backbone of the provenance graph, which is often large-scale but sparse. Specifically, TAPAS constructs stacked LSTM-GRU models that iteratively update the representations of the backbone nodes based on relevant redundant nodes, replacing direct storage and computation of these redundancies. For temporal dimensionality, TAPAS designs a task-guided backbone graph segmentation algorithm that identifies active subgraphs as objects to be detected in real-time, reducing structural redundancy in the temporal space.

Evaluation in widely used benchmark datasets, DARPA TC and OpTC, demonstrates TAPAS's effectiveness in providing fast, low-overhead online detection while maintaining similar detection accuracy to state-of-the-art methods. Our results show that TAPAS reduces storage requirements by up to 1806× and achieves 99.99% accuracy with an average detection time of 12.78 seconds per GB of audit data, validating its practicality for enterprise deployment with throughputs well above the enterprise requirement of 10^4KB/s.

*The authors contribute equally to this paper.
†Corresponding author.

1 Introduction

Advanced Persistent Threats (APTs) launched by skilled attackers have become a significant threat to cybersecurity [72]. They are a customized combination of multiple attack techniques characterized by long latency and extreme stealthiness. APT attacks have caused severe damage to institutions and enterprises [14, 41] and are becoming increasingly prevalent [29, 31]. APT attackers use deliberate tactics to evade detection, and many attacks are associated with 0-day vulnerabilities, making APTs difficult to detect.

Provenance analysis plays an important role in the detection of APT [76]. It organizes kernel-level operating system audit records into a provenance graph, where nodes represent system objects (i.e., processes, files, and network connections), and directed edges represent interactions from processes to system object nodes. The provenance graph provides a structured description of the system's execution history. Provenance analysis facilitates causal relationships and traces provenance between historical behaviors in the system, providing a rich and valuable context for identifying malicious events [66, 75].

Consequently, recent APT detection works [21, 28, 64, 71] mainly build on provenance analysis, focusing on studying various approaches to identify suspicious structures in the provenance graph. Early works [20, 39] take a snapshot of the *entire provenance graph* as a representation of the current system state and discover the compromised system states via detecting the snapshots. However, attacks can be hard to identify in snapshots taken at specific times, and a few attack behaviors can be obscured by a large number of benign behaviors, thus risking detection evasion [16]. Some recent work [21, 71] captures and represents the neighborhood structure of nodes/edges in the provenance graph as features of nodes/edges and detects malicious entities/actions by assessing the plausibility of nodes/edges. However, their detection is over fine-grained and thus faces serious overhead problems. The state-of-the-art (SOTA) work [28, 64] periodically generates provenance graphs based on audit logs received within

the time windows and then detects attacks based on graph representation learning. However, they run the risk of missing slow-evolving APT attacks. These attack actions are spread over multiple time windows in small amounts, so they do not cause significant anomalies in single time windows.

While SOTA APT detection methods achieve strong accuracy, they face significant implementation challenges in industrial settings due to excessive memory and storage requirements, creating a substantial gap between academic research and industry deployment needs. The core challenge stems from the reliance of these methods on maintaining extensive provenance graphs for threat detection, which grow continuously and unbounded over time. As documented in multiple studies [10,22,33,57], this unconstrained growth creates mounting performance overhead that impedes effective real-time detection in production environments.

Our research addresses the challenge of prohibitively high overhead in APT attack detection. Upon our pilot study (see §3), we first give insights about the following two key issues that affect the performance of provenance-based APT detection methods: 1) Enormous scale of provenance graphs. APT attacks often span months or even years. Detection methods maintain a provenance graph that describes fine-grained historical behavior over time to provide a complete view of APT attacks. However, the provenance graph can grow rapidly (several GB per day [33,61]) to huge sizes. As a result, APT detection faces severe pressure on both provenance graph storage and computation. 2) Redundant computation of the invariant part within the whole provenance graph. APT detection systems periodically detect the current provenance graph, repeating the feature representation and detection over the entire graph area each time. However, compared to the last audited graph, the current provenance graph contains many invariant structures because many processes have been terminated or are inactive. Representing and inspecting these invariant parts leads to redundant computations.

We address these two critical issues by using provenance graph backbone and changed-localization detection techniques. We reduce the graph dimensionality in spatial (i.e., leveraging backbone) and temporal space (i.e., removing invariant parts in the time-axis).

Spatial Space Dimension Reduction. Process nodes can be regarded as the backbone of the provenance graph since all directed edges in the provenance graph start from process nodes, representing system executions started by processes. And other nodes can be regarded as attachments to the process nodes that point to them. We retain only the process graph, which serves as the backbone to preserve the structural semantics of the provenance graph. In addition, we generate a representation vector for each process node to embed its attached node relationships. Thus, we replace the long-term storage and graph computation of the numerous attached nodes and their connectivity relationships by maintaining only the process representation vectors.

Temporal Space Dimension Reduction. In temporal space, we perform graph partitioning based on the tasks to which the processes belong. A task is a single program run in the operating system, consisting of processes/threads that initiate system execution with a common purpose and cooperate [34,65]. We partition processes executing the same task into a subgraph based on the structure of the relationships between them and detect only the *changing* subgraphs to avoid redundant computation on numerous invariant subgraphs.

Building upon our key insights of spatial and temporal dimensional reduction in the APT provenance graph, we propose an efficient online APT detection framework `TAPAS`, which achieves a low computational cost of detection to improve usability in real-world environments while exhibiting high detection performance.

`TAPAS` first constructs an active process graph from audit records, which stores the backbone structural information of the provenance graph. For the history of process interactions, `TAPAS` designs an iteratively updated process representation using a stacked LSTM-GRU model [38,59], leveraging its strength in capturing sequential patterns to efficiently encode historical process behaviors that may indicate APT activity. The stacked LSTM-GRU architecture captures the long-term relationships of sequences in the LSTM model and uses the GRU layer to further reduce the error by eliminating the overfitting problem. Our method enables a low-dimensional representation that embeds the history of process behavioral events and dramatically improves storage efficiency by replacing the direct storage and computation of these long-term events, thereby reducing the storage and computation overheads of long-term provenance tracking.

`TAPAS` then develops a task segmentation algorithm that segments the active process graph based on system tasks, which is crucial for APT detection for several reasons: First, a single task achieves an end goal of that task by activating multiple processes that cooperate with each other, and these processes are inherently goal-consistent. Second, by analyzing the behaviors of processes in the context of a system task, we can summarize the task goal of the processes as a whole, and thus distinguish more clearly between legitimate activities and potential APT behaviors that abuse or misuse normal system operations. Third, task-level segmentation allows us to capture the temporal and causal relationships between different components of an APT campaign while maintaining computational efficiency through focused analysis of relevant subgraphs.

We implement `TAPAS` and evaluate its effectiveness and performance on the publicly available APT attack dataset, the DARPA Transparent Computing (DARPA TC) E3 datasets [48], generated during the Red-Blue confrontation and the OpTC dataset [4]. We use four public sub-datasets of DARPA TC, collected separately by different teams on different platforms. And we comprehensively compare our `TAPAS` with SOTA APT detection methods, including FLASH [55]

(IEEE S&P '2024) and Magic [28] (USENIX Security '2024). The evaluation demonstrates that TAPAS achieves significant improvements in time overhead, CPU usage, and memory consumption while maintaining comparable or superior accuracy in attack identification.

In summary, we make the following contributions*:

- We propose TAPAS, a low-overhead online task-level APT detection framework that significantly facilitates detecting APT attacks in the real world.
- We design a process representation generation method that provides a refreshed perspective of history interaction embedding, which can reduce the enormous long-term memory consumption in provenance analysis by scaling down the provenance graph from the spatial space.
- We design a real-time task-based process subgraph segmentation algorithm that can efficiently and promptly identify subgraphs attributing to new records, which helps reduce the detection overhead by excluding invariant subgraphs from the temporal space.
- We evaluate TAPAS on widely-used public datasets and compare it with SOTA studies. The results show that TAPAS can significantly reduce detection overhead while accurately detecting attacks.

2 Background and Related Work

We begin by presenting essential background on data provenance, a widely adopted technique for APT detection. Subsequently, we introduce the LSTM/GRU model utilized in this study. We also provide a brief overview of community discovery, as it shares the subgraph-based concept relevant to our approach. Finally, we review related work in the domain of provenance-based APT detection, under which our proposed method can be categorized.

2.1 Data Provenance

System Audit. The system audit is a comprehensive record of all OS-level system calls made during the observation period [24]. Commonly used system audit tools include Endpoint Detection and Response (EDR) tools on Windows systems [5] and audit tools on Linux systems, such as Auditd [51, 70]. System auditing captures various types of runtime information that describe all events and entities involved in the system's history. Consequently, system auditing can be leveraged to monitor runtime behavior, enabling analysts to gain valuable insights into cyber-attacks through data provenance [27, 68].

*Our source code is released at https://doi.org/10.5281/zenodo.15610687

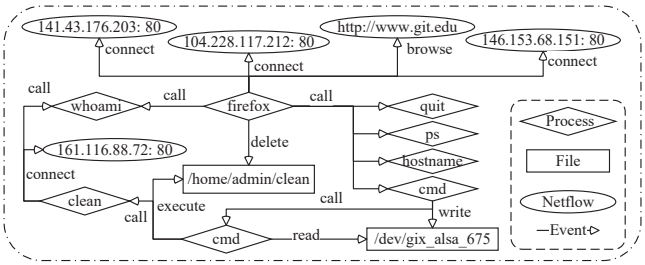

Figure 1: Example of the provenance graph

Provenance Analysis. It, also known as causal analysis, seeks to characterize the provenance and history of entities, data, processes, activities, users, and data involved in a system's life cycle [54]. The usual data source of provenance analysis is system audit [58]. Provenance analysis reconstructs the runtime information flow based on audit records, thereby capturing possible relationships and dependency structures between entities [12, 43]. Specifically, it uses nodes and edges, respectively, to represent system entities (including processes, files, and netflows) and system events in the audit records, thus transforming the independent audit records into a whole directed acyclic graph, called a provenance graph [62].

Figure 1 is an example of a provenance graph showing an APT attack scenario exploiting a Firefox vulnerability. We use different shapes to represent different types of nodes: diamonds for processes, rectangles for files, and ovals for network connections. The figure shows the process of an attacker connecting to a malicious external link and removing traces of the attack.

2.2 LSTM/GRU Model

A Recurrent Neural Network (RNN) is a type of neural network designed for modeling time-series data. It retains historical context through recurrent connections between hidden units and the use of time delays [1], allowing it to capture temporal correlations between events that may be far apart in time [60]. Among RNN variants, the Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) models are particularly effective at capturing temporal dependencies [40]. LSTM networks excel at learning complex long-term dependencies, while GRUs offer a simpler architecture with reduced computational overhead. A stacked LSTM-GRU model combines the strengths of both architectures [44].

2.3 Community Discovery

A community is a cluster of process nodes with a specific regional structure and clustering characteristics. Each community contains only closely related processes that work together to accomplish specific system tasks (e.g., file com-

pression). The goal of community discovery is to partition the provenance graph into a set of disjoint or overlapping communities [46]. Both detection and investigation of APT attacks can be based on community discovery. APT detection based on community discovery [7, 8] partitions a system snapshot into multiple communities by considering nodes closely associated with historical behavioral edges as a community and then detects anomalous communities. It can determine the approximate location of an attack. However, community partitioning requires additional graph consistency computations, which can cause excessive time overhead and is, therefore, more commonly used for attack investigation in forensic analysis. Attack investigation based on community discovery [36, 63, 68] generates a summary graph by partitioning the dependency graph into communities (i.e., subgraphs) and presenting a concise summary for each community.

2.4 Provenance-based APT Detection

Recent APT detection studies leverage provenance analysis to detect attack actions revealed in provenance graphs. The granularity of the detection can be categorized into graph, path, and node/edge levels.

Graph-level Detection. These approaches [46, 64] capture the provenance graph at various times, produce system snapshots, and then detect any malicious ones. A snapshot provides a comprehensive description of the current state of the protected system. Typically, they train a graph learning model to act as a detector that automatically collects features and determines whether the current snapshot is benign or malicious. However, this approach can easily miss attacks because a small number of attacks can be buried in a large number of benign behaviors, making the snapshot as a whole behave benign [10]. Even if a malicious snapshot is detected, it is difficult to determine the exact location of the attack. Another work [7] divides system snapshots into multiple communities through community discovery, and then detects malicious communities. However, community discovery requires the additional task of graph computation, which results in excessive time overhead.

Path-level Detection. Some approaches [2, 30, 67] extract paths from the provenance graph and use neighbourhood structure and causal provenance as features to detect malicious paths. A notable challenge of this approach is how to extract the paths to be detected. Provenance graphs constructed from system execution history are often large and have complex but sparse structures. Obviously, traversing all paths and then discriminating them one by one is impractical. Therefore, path extraction requires the specification of an appropriate strategy to determine the scope of the extraction. In the case of improper path extraction, such as extracting a small neighborhood, it would result in spreading a single attack to be detected into multiple path sequences, resulting in the interruption of the attack trace and affecting the detection accuracy. In addition, this approach still suffers from the problem of attacks being drowned out by benign behaviors.

Node/Edge-level Detection. Recent approaches [21, 28, 55, 71] focus on the plausibility evaluation of process nodes and system event edges, treating nodes with anomalous behavior or node pairs with anomalous interactions as attacking processes. They capture the execution or interaction history of different types of processes from node neighborhoods, which provide rich information to characterize process nodes/event edges. Node/edge level detection has the finest granularity and achieves high detection accuracy compared to the graph and path level methods described above. However, this fine-grained detection can also significantly increase the computation required in the detection process, which can negatively impact performance and cost.

3 Motivation: Overhead Analysis of Provenance Graphs for APT Detection

While provenance analysis based on audit records has demonstrated effectiveness for APT detection, its substantial overhead creates a significant gap between academic solutions and industry requirements [10, 50]. This overhead burden needs urgent attention to make these detection systems viable for real-world deployment.

Our analysis reveals two primary sources of overhead in provenance-based APT detection. The first stems from redundant feature extraction, as existing approaches repeatedly process the entire provenance graph for each detection call, even though many processes remain inactive across multiple detection intervals. This inefficiency is particularly pronounced given the sparse nature of active nodes, resulting in substantial computational waste. The second source relates to the long-term maintenance requirements of provenance graphs, which must store comprehensive historical behavior information including processes, system events, and objects. This continuous accumulation of fine-grained data leads to unbounded graph growth over time.

To validate these insights empirically, we analyzed real-world data from the DARPA datasets (detailed in Section 5.1). Figure 2 presents a provenance graph generated from the DARPA TC dataset spanning 1 hour and 15 minutes of audit records. We examine this data through two key analytical perspectives.

1) There are indeed stable and invariant structures in the provenance graph, such as the circle at the top of the graph, which consists of many interacting edges. The operating system has some procedures that act periodically, such as virtual memory management. These programs create a new process that initiates many interactions for only a short time and then recreates the process the next time it acts. Such interactions are recorded in the provenance graph each time. They must

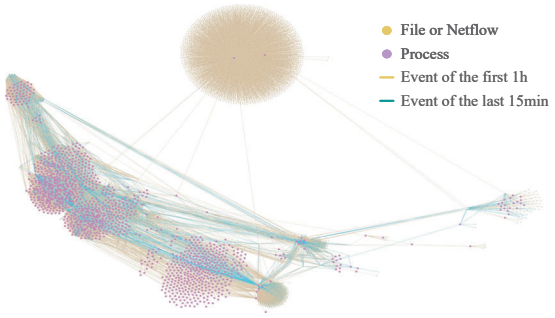

Figure 2: Provenance graph generated from the DARPA datasets. Nodes include 1349 process subjects (labeled in pink) and 5322 files & network connection objects (both labeled in brown). Edges include 13,119 events that occurred in the first hour (labeled in brown) and 2,062 events that occurred in the last fifteen minutes (labeled in blue).

be kept long because provenance analysis cannot determine whether they will be associated with a future event. In addition, some processes may continue to be active in the future but stay dormant for two or more consecutive detections. Reassessment of these dormant regions is unnecessary because of high redundancy.

2) Most of the provenance graph consists of interacting edges and operational object nodes, including file and network nodes. Figure 2 shows 1349 process nodes compared to 5322 object nodes and 15181 edges. This indicates that maintaining a process graph can significantly reduce the graph size compared to maintaining the entire provenance graph because the process size is up to, e.g., 3× and 11× smaller than that of nodes and edges.

Table 1: Audit record statistics in the TC-Theia dataset

Data	Duration	Subject	Object	Event
Benign scenario-1	Avg. 1 minute	59	285	25068
Benign scenario-1	Total (about 1h)	3455	16702	1470908
Benign scenario-2	Avg. 1 minute	26	39	1696
Benign scenario-2	Avg. 1 hour	1546	2324	101731
Benign scenario-2	Total (about 9h)	13916	20919	915579

We also count the amount of audit records, and the results are shown in Table 1. We counted the number of audit records of the subject, object, and event types on an average of every minute, every hour, and a cumulative length of over 9 hours. From a temporal perspective, the subjects (processes) and events active during the period (e.g., 1 hour on average) represent only a small fraction of the total. Therefore, detecting only the part related to active processes can significantly reduce the computational task. From the perspective of data types, the number of subjects (processes) is much smaller than the number of object or event records. (The number of subject records is approximately two orders of magnitude smaller than the sum of the number of object and event records.) As a result, the process graph is much smaller than the entire provenance graph, which contains all of the historical information.

Based on the above insights, we aim to relax the huge burden of APT detection by reducing the size of the entire provenance graph for long-term storage and computation and focusing the detection on the active portion of the graph while preserving the semantic information of provenance. To achieve this, we introduce TAPAS, which leverages the lightweight backbone of a process graph and process representation vectors instead of the large-scale sparse provenance graph to capture historical information. In addition, it avoids extensive redundant computation on stable graph structures by dividing processes that work together to accomplish the same system task into task subgraphs and then detecting only the active subgraphs. Significantly, the processes that work together to accomplish the same system task are partitioned into task subgraphs, and then only the active subgraphs are detected.

4 TAPAS

We first define the threat model used for TAPAS, followed by an overview of TAPAS. Implementation details of each of the four key components in TAPAS are then elaborated.

4.1 Threat Model

Our threat model addresses sophisticated external attackers who target organizational systems to extract sensitive internal information. This model aligns with recent research in provenance-based detection [3, 11, 55]. While attackers may employ advanced techniques to evade detection, their activities necessarily generate traceable evidence within system audit records, enabling detection through careful analysis of system behaviors and interactions.

Our model incorporates several foundational assumptions about system security. First, we consider the operating system and kernel-space auditing frameworks as part of our trusted computing base, ensuring that attackers cannot tamper with the generation, storage, or transmission of audit records. Second, we require that attackers have not obtained full system control privileges prior to the initiation of system auditing. This assumption is supported by existing secure provenance systems and tamper-evident logging techniques [18,45], which provide mechanisms to maintain log integrity and detect malicious interference with provenance logs [47]. Our threat model excludes attack vectors that operate outside the visibility of system auditing mechanisms, such as hardware trojans and side-channel attacks. This focused scope allows us to concentrate on detecting and analyzing threats that manifest through observable system behaviors while acknowledging other attack surfaces requiring different defensive approaches.

4.2 Overview

TAPAS is an efficient APT detection framework. It designs a process historical behavior representation to capture the current state of a process based on its historical evolutions with an embedded vector. This allows TAPAS to maintain process vectors and a process graph representing the inter-process relationships instead of tediously maintaining the entire provenance graph, which records all but often highly redundant historical behaviors, thus spatially reducing the graph structure describing the historical behaviors. It then detects only those process graph areas to which the active processes belong during the detection period, thereby reducing the need for unnecessary repetitive detection of invariant structures in the graph. To accurately extend the graph areas affected by active processes, TAPAS segments the process nodes in the process graph that cooperate to perform a common system task into a single subgraph. Therefore, TAPAS reduces the computational complexity of maintaining system history descriptions and detection in spatial and temporal dimensions, thus providing efficient, low-overhead APT observations.

Figure 3 illustrates the architecture of TAPAS, a system designed for detecting malicious activity within process execution graphs. TAPAS operates in two distinct phases: offline training and online detection.

Offline Training: This phase establishes a normal behavior baseline. Audit logs are parsed to construct the Active Process Graph (APG), capturing historical process execution patterns and parent-child relationships. Historical process behaviors are embedded, generating a sequence of embeddings $E_0, E_1, \cdots, E_n$, which are used to train a sequence model (e.g., GRU/LSTM) for generating corresponding representations $Y_0, Y_1, \cdots, Y_n$. The APG is then segmented into task subgraphs, using these representations as node features. Finally, a classifier is trained on these subgraphs with benign/malicious labels.

Online Detection: This phase uses the trained model for real-time attack detection. Incoming audit records (process descriptions and behaviors) update the APG structure and generate new process representations. The graph segmentation component identifies changed subgraphs due to these updates (structural or representational). The trained classifier then analyzes these changed subgraphs, performing attack detection by summarizing process behaviors.

These phases rely on four core components: active process graph construction, representation generation, graph segmentation, and detection. These components are described in detail below.

Active Process Graph Construction (§4.3). TAPAS begins with parsing the audit records and then constructing an APG according to the parsing results. The audit records consist of process descriptions, process execution event descriptions, and process execution object descriptions. By parsing the audit records, TAPAS obtains the parent-child relationships between active processes from the process descriptions and the processes' historical behaviors by combining the process execution event descriptions and execution object descriptions. The inter-process relationships are used to construct and maintain the APG, where nodes represent active processes and edges represent parent-child relationships. Process historical behavior will be used later to compute process representation vectors.

Representation Generation (§4.4). TAPAS computes a process representation for each node in the APG to embed all historical behavioral events of the process. The process representation is generated based on an iterative update of a feature vector consisting of the historical behavior of the process and associated objects. We build a stacked LSTM-GRU model to generate the process representation. TAPAS replaces the tedious storage of a large number of historical events over the long term with a dimensionality-reduced temporal latent representation. This is beneficial in saving storage overhead and computational effort for APT detection.

Graph Segmentation (§4.5). TAPAS designs an algorithm to segment task subgraphs according to the structure of the APG, with the goal of partitioning processes that collaborate on a common system task into a subgraph. This task-guided graph segmentation method allows TAPAS to aggregate the processes in the subgraph to enhance extraction of common task semantics and to constrain the detection within a sensible domain. When the structure of the APG is updated, the task separation algorithm checks and updates the system task subgraph segmentation along the ancestor direction of the updated location.

Detection (§4.6). Based on the segmented APG and process representation vectors generated by the above components, TAPAS detects the tasks to which active processes belong, i.e., task subgraphs that have been updated (including both structure update and process node representation update). We build a detection model based on graph representation learning, which uses the structural information of the task subgraph and the process representations of the nodes to obtain a task subgraph representation, and then classifies and alerts malicious subgraphs.

4.3 Active Process Graph Construction

The active process graph construction consists of two steps: data parsing and APG construction. TAPAS first parses the received audit records and extracts the attribute fields. Record types include subject, event, and object, which describe processes, process executions, and process execution objects, respectively. Among these, the subject record contains the process information description, which TAPAS is used to obtain processes and their parent-child relationships as nodes and edges in the APG. Event and object records are used to facilitate the subsequent generation of process representations.

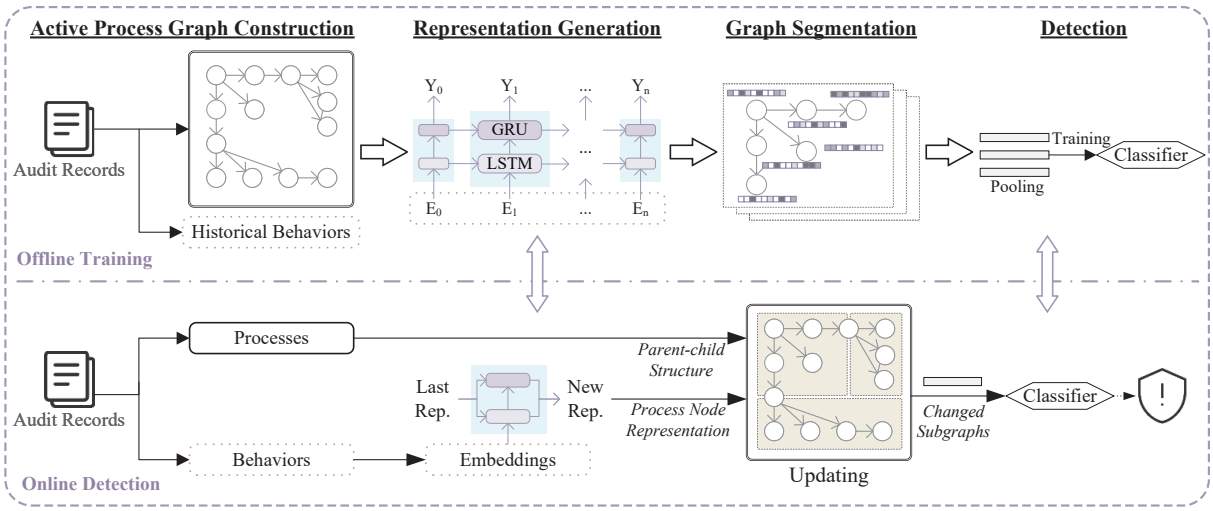

Figure 3: Overview of TAPAS

Table 2: System entities and attributes

Record	Entity	Attributes
Subject	Process	UUID, parent ID, tgid
Event	Event	type
FileObject	File	filepath, suffix, dev
NetflowObject	Netflow	src/dst address, local&remote port

Data Parsing. TAPAS parses the audit records to obtain process descriptions and historical behaviors, and to extract attribute fields. Specifically, TAPAS obtains the system process descriptions, process executions, and process execution object descriptions from the Subject, Event, and FileObject/NetflowObject items in the audit records, and extracts the corresponding attributes as shown in Table 2. The subject item describes the launcher of system executions, i.e., process or thread. We identify whether a subject describes a process or a thread based on the three attributes of its unique subject identifier (*UUID*), parent identifier (*parent ID*), and thread identifier (*tgid*). The identified threads and their behaviors are merged into the process entity to which they belong. Merging threads helps us summarize the historical behavior of processes and avoid thread influence on subsequent graph segmentation. Event items and object items are combined to describe the historical behavior of the processes.

APG Construction. Based on the process descriptions obtained from data parsing, TAPAS constructs an APG to represent the parent-child relationship between processes. Nodes in APG represent system processes, and the inter-process relationship is described by directed edges pointing from the parent to the child. First, TAPAS checks threads based on the following properties: a thread's *tgid* is the same as the process it belongs to, and its *parent ID* is the same as the process's *UUID* [6]. So if a Subject's *parent ID* points to a Subject with the same *tgid*, it must be a thread. For thread-related activities, we merge them into the process to which the thread belongs. Then, based on the *parent ID* of a process, TAPAS constructs an APG that describes the inter-process relationship in terms of pointing from the parent to the child process.

In the online detection phase, newly received process descriptions trigger changes in the corresponding process nodes and inter-process relationship edges, which in turn lead to an update in the structure of the APG. The areas where the structure update occurs are included in the next detection.

4.4 Representation Generation

TAPAS maintains an embedded representation for each process, which provides a summary of the history of actions initiated by the process. The process representations employed by TAPAS permit the saving of the graph structure corresponding to events and objects. Consequently, in contrast to provenance graphs, the graphs that TAPAS is required to maintain over time are considerably smaller, which is highly beneficial in reducing storage and computational overhead. The process representation generation is completed in three steps: embedding extraction for objects, redundancy reduction for events, and constructing a stacked LSTM-GRU model that uses the last embedding representation of the process and new behavioral events to generate a new embedding representation. The detailed descriptions of these steps are provided below.

Embedding Extraction. For files and network entities, TAPAS extracts the attributes and transforms them into a 4-dimensional embedding vector. It is used in the following computation of the process representation. The embedding vector for a file consists of <*file flag, path, file type, dev*>. The *file flag* is a fixed value that distinguishes between files and network entities. We construct a system path hash table

based on the Linux/Windows system file directory structure to map the *path* field of the embedding vector. The *file type* field is obtained based on the file suffix, and *dev* is obtained from the device number field in the audit record, which identifies the driver such as RAM disk, cdrom, and sound card. The embedding vector for a network consists of <*network flag, src/dst address, local port, remote port*>. The *network flag* is also a fixed value. The *src/dst address* provides the distance of the network segment between the communicating side and the local device. *Local port* and *remote port* indicate the port types. The extracted object embeddings are kept separately and looked up when computing the process representation. This approach eliminates the need for TAPAS to maintain graph structures for a large number of objects.

Event Redundancy Reduction. TAPAS receives event audit records and performs an iterative computation process to derive process representations based on the attributes of events and objects. However, the audit records may contain a substantial number of redundant events in two principal instances: 1) The behavior of an event occurring over an extended period may result in the generation of multiple, consecutive, and identical event records. For instance, a single network communication behavior may result in the generation of multiple redundant records. 2) A process may launch the same behavior on the same object on multiple occasions, resulting in the creation of discontinuous, duplicate records. To illustrate, a process may read a file on multiple occasions. It is essential to address these redundancies before the iterative computation of the process representation, as this can markedly accelerate TAPAS's processing of large-scale input data. Specifically, the triple <*subject ID, event type, object ID*> is employed to identify redundancies. Incoming events are converted into triples. For redundant events with the same triple, only one triple entry is retained, and the number of occurrences is counted. Following the reduction of redundancies, events are represented as a six-tuple vector V_e, spliced by *event type*, *event count*, and a four-dimensional *object embedding*.

Representation Model. Next, TAPAS iteratively computes a process representation of the historical events of the process based on the event vector V_e generated after embedding extraction and redundancy reduction. TAPAS constructs a stacked LSTM-GRU model [59] that captures the long-term historical or temporal behavior of processes and takes its output as the process representation. The LSTM-GRU model generates output by utilizing a received external input x and a hidden state h of itself.

The structure of the stacked LTSM-GRU model in TAPAS is shown in Figure 4, which contains an LSTM cell and a GRU cell. Specifically, we take the embedding vector V_e describing the new event at time T and the process representation of the previous time $T-1$ as the input and hidden representation of $Cell_{LSTM}$. $Cell_{LSTM}$ computes and produces the output:

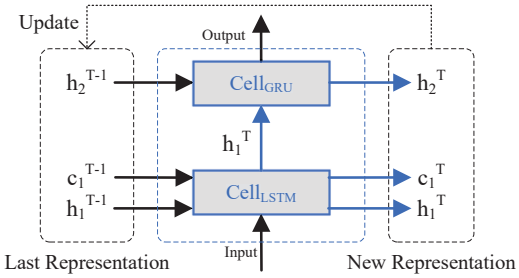

Figure 4: Stacked LSTM-GRU model

$$h_1^T, c_1^T = LSTM(V_e, h_1^{T-1}, c_1^{T-1}).$$

The following layer $Cell_{GRU}$ receives the output of the previous cell as input and the process representation at $T-1$ as hidden representation to generate the outputs:

$$h_2^T = GRU(h_1^T, h_2^{T-1}).$$

We use these outputs (h_1^T, c_1^T, and h_2^T) to update the process representation. In the offline training phase, we use the process representations to predict the next event embedding vector for that process to motivate the model to capture the rich details of the event sequence. In the online detection phase, TAPAS receives the new process behavior and queries the current process representation, sending them into the representation model to update the process representation. Processes with updated representations are recorded and prepared for the next detection.

4.5 Graph Segmentation

Next, TAPAS segments the APG into subgraphs that perform different system tasks based on the graph structure. A system task is a common purpose that corresponds to a process and its set of operations and guides the actions of the process [34, 65]. As a result, malicious processes often show close associations because of the guidance of common tasks [16, 74]. Therefore, we partition the processes collaborating on common system tasks in the APG into a subgraph and summarize the subgraph's task semantics by aggregating the process nodes' features to support detection.

For a process and its parent, if the process generates more than one child process, it usually initiates a new system task, and its children do not cooperate with the parent [63]. We note that the segmentation is supported by the way modern operating systems manage processes. Specifically, when a process spawns multiple child processes, it often reflects the decomposition of a workload into separate system tasks, and these child processes generally operate independently of their parent process [37]. In contrast, to conserve system resources and improve execution efficiency, multithreading is commonly used, where threads within the same process share

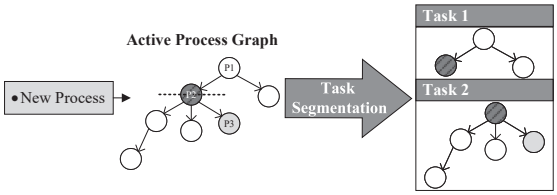

Figure 5: Example of task segmentation

memory and other resources to perform concurrent operations [42]. As a result, a new child process with the allocation of additional system resources—is typically created when initiating a new and distinct task [35].

Therefore, TAPAS separates processes with more than one child from their parent processes into different tasks. To avoid losing the semantics of the tasks belonging to the parent process, we take a node-separation approach, i.e., we assign the segmented process to both its upstream and downstream tasks. Figure 5 shows an example of a task separation process. TAPAS task separation is triggered when a new process node $P3$ joins APG. Since the child process number of $P3$'s parent node $P2$ is exceeded, $P2$ and its child processes are separated with $P2$'s parent process $P1$ into two tasks. To avoid losing the semantics of $task1$, the node $P2$ is still kept in $task1$.

In the offline phase, the graph segmentation algorithm checks the process nodes in the APG one by one. In the online phase, the graph segmentation algorithm is triggered by the APG's structure update. Specifically, TAPAS checks the process nodes affected by the structure update and marks the nodes that need to be segmented. The added segmented nodes then cause TAPAS to further check their ancestor nodes. We describe the graph construction and segmentation procedure in Algorithm 1. Each subject in the audit records has a parent and a process identifier. TAPAS traverses the subjects in the streaming audit data. For each subject s_i, TAPAS first checks if it is a thread based on its thread ID $s_i.tg$ with its parent $s_i.p$ (lines 2-7). If it is a thread, it is merged directly into its parent process. Then, if process s_i is not in the APG before, TAPAS adds it to the APG and checks if the segmentation state needs to be updated along the direction of its ancestors (lines 9-29); if process s_i already exists in the APG, TAPAS updates its parentage in the APG according to the newly recorded parent field $s_i.p$ and checks the segmentation state along the direction of the ancestors of the original and the new parent, respectively (lines 30-45).

Algorithm 1 Graph Construction and Segmentation

Input: Streaming data (subjects), each subject s_i contains a parent p and a tgid tg; Active process graph APG
Output: List of the segmented subject $List_{seg}$;

```
1:  for all s_i do
2:      if s_i.p in APG then
3:          s_ip ← Get s_i.p in APG;
4:          if s_ip.tg = s_i.tg then
5:              MERGE s_i to s_i.p;
6:              CONTINUE;
7:          end if
8:      end if
9:      s_k ← GetParent(s_i) in APG;
10:     if s_i not in APG then
11:         AddNode(s_i), AddEdge(s_i.p → s_i) to APG;
12:         s_k.ChildNum + 1
13:         repeat
14:             children←GetChildNum(s_k);
15:             if children >2 then
16:                 if s_k not in List_seg then
17:                     ADD s_k to List_seg;
18:                     s_k ← GetParent(s_k) in APG;
19:                     s_k.ChildNum − 1
20:                 end if
21:             else if s_k in List_seg then
22:                 DEL s_k from List_seg;
23:                 s_k ← GetParent(s_k) in APG;
24:                 s_k.ChildNum + 1
25:             end if
26:         until List_seg no longer update
27:     else if s_k ≠ s_i.p then
28:         Update s_i in APG;
29:         children←GetChildNum(s_k);
30:         if children = 2 then
31:             Del s_k from List_seg;
32:             Check s_k.ancestry;
33:         end if
34:         children←GetChildNum(s_i.p);
35:         if children >2 then
36:             ADD s_i.p to List_seg;
37:             Check s_i.p.ancestry;
38:         end if
39:     end if
40: end for
```

4.6 Detection

TAPAS detects task subgraphs taken in conjunction with process representation vectors. Specifically, for each task, TAPAS constructs a subgraph, denoted as $G = (V, E)$, where V represents the set of nodes and E represents the set of edges in the graph. The node attributes comprise process representations, which are designated as $\{x_v, \forall v \in V\}$. In the case of processes exhibiting no historical behavior, the x_v values are set to the full zero vector. TAPAS builds a graph representation learning model that collects the neighborhood information of nodes based on the subgraph structure and node attributes to generate node representations, which are pooled to output a graph representation. Subsequently, TAPAS uses a classifier to determine whether the graph representation is malicious.

During online detection, TAPAS only detects the task subgraphs corresponding to processes for which new behavior

has occurred since the last detection. APG structure updates and process representation updates resulting from previous components are recorded to help identify subgraphs with active processes. The detector identifies abnormal subgraphs among them and issues an attack alert. Detecting only these updated task subgraphs limits detection to what is necessary, thus saving computation on the invariant regions of the graph and reducing overhead. This process can be streamed as audit records come in, but we prefer to set a time window for periodic batch detection to avoid the impact of frequent graph operations on system performance.

5 Evaluation

Experimental Setup. We implement TAPAS in Python. We use Pytorch [26] to implement the stacked GRU model for generating process representations and PyG [13] to implement the GNN model for detection.

We evaluate TAPAS's overhead and effectiveness using widely used public datasets (§5.1). All experiments are performed on a computer running Windows 11 with a 2.20GHz Intel(R) Core(TM) i9-13900HX CPU, an RTX 4060 Laptop GPU, and 32GB of memory. When training the representation generation model, lr is 0.1, lr decay factor is 0.1, and the decay rate is 500. The classifier consists of two GraphSAGE layers and one linear layer. The lr of the classifier is 0.001 and the weight decay factor is 5×10^{-4}. To ensure the objectivity of the evaluation, ten repetitions of each experiment are conducted, and the results are averaged. TAPAS's objective is to achieve low overhead APT detection while maintaining detection effectiveness, providing real-world practical online detection capabilities.

Questions. Consequently, our evaluation focuses on answering the following questions:

Q1. How effective is TAPAS as an APT detection framework in different scenarios? (§5.2)

Q2. How efficient is TAPAS's runtime performance? An enterprise system with 500 hosts generates audit data at a rate of about 10^4KB/s [33], can TAPAS's throughput keep up with this rate to meet real-world demand? (§5.3)

Q3. What is the TAPAS's performance in a relationship with increasing data volumes? (§5.4)

Q4. Can TAPAS contribute to reducing the disk storage space required for detection? (§5.5)

5.1 Datasets

We perform comprehensive evaluations and comparisons for TAPAS using the public DARPA TC E3 dataset that have four subdatasets of Cadets, Fivedirections, Theia, Trace [48] and OpTC dataset [4], which serve as standard benchmarks for APT detection methods [3, 8, 28, 55, 64].

The DARPA datasets were collected over a two-week period during adversarial engagements in an enterprise network environment. In these scenarios, a red team executed APT attacks to exfiltrate sensitive information by exploiting various vulnerabilities, while blue teams performed network host auditing and causality analysis to detect these intrusions. Attack footprints are distributed across 17 hosts over 8 days [48]. DARPA datasets are standing out as the longest attack period [9] and are regarded as the closest to real-world adversarial campaigns [39, 58] among known public APT datasets.

Our evaluation encompasses four sub-datasets of DARPA, totaling 351.2GB of audit records. The Theia dataset was collected from Ubuntu 12.04 hosts, the Cadets dataset was obtained from a FreeBSD 11.0 host, the FiveDirections dataset was gathered from a Windows 7 machine, and the Trace dataset was collected from Ubuntu 14.04 hosts. We label malicious tasks using established ground truth [9] and entity labels [53].

The OpTC dataset spans eight days, with attack campaigns executed over a dedicated three-day evaluation period. On each of these days, three distinct types of attacks were carried out: PowerShell Empire, data exfiltration, and malware upgrades. Each attack type was launched on a separate host.

5.2 Effectiveness

We evaluate the effectiveness of TAPAS detection on the DARPA TC and OpTC datasets in comparison to SOTA work. The evaluation metrics include accuracy, precision, recall, and F1 score. TAPAS achieves an average of 99.10% precision, 98.56% recall, 99.33% accuracy, and 98.83% F1 score on the DARPA TC dataset. In particular, TAPAS scores a minimum of 97.32% and a maximum of 99.72% on the F1 score, which represents the comprehensive performance of the method. On the OpTC dataset, the average is 97.88% precision, 96.65% recall, 97.83% accuracy, and 97.19% F1 score. The three attacks in OpTC are independent and exhibit short, bursty durations. Due to these limitations, the detection performance of TAPAS on OpTC is slightly lower than that on the DARPA dataset. It is worth noting that other SOTA approaches [55, 58] also experience performance degradation on OpTC. Despite the challenges posed by OpTC, TAPAS achieves a higher detection recall than the SOTA baselines, demonstrating its strong generalizability.

Table 3 presents a comparison between TAPAS and representative SOTA detection methods. Specifically, we evaluate TAPAS against node/edge-level detection schemes—FLASH [55] and MAGIC [28]—as well as graph-level approaches—UNICORN [20] and ThreaTrace [58]. All the SOTA methods are reproduced using their publicly available code and default parameters under a unified machine setting to ensure fair comparison. For the DARPA datasets, the evaluation is conducted on the complete Theia, CADETS, FiveDirections, and TRACE datasets. Due to the absence of a publicly released, preprocessed version of the FiveDirections dataset by MAGIC, we are unable to include MAGIC's per-

Table 3: Comparison between TAPAS and SOTA methods

Dataset	Sub-dataset	Approach	Prec.	Rec.	Acc.	F1-Score
DARPA	Cadets	FLASH [55]	0.8895	0.9995	0.9954	0.9413
		MAGIC [28]	0.7079	0.9977	0.9912	0.8281
		UNICORN [20]	0.9291	1.0000	0.9313	0.9633
		Threatrace [58]	0.9042	0.9997	0.9981	0.9496
		TAPAS (Ours)	**0.9938**	0.9990	**0.9982**	**0.9963**
	Fivedirections	FLASH [55]	0.6638	0.9153	0.9996	0.7695
		UNICORN [20]	0.8534	0.9900	0.8500	0.9167
		Threatrace [58]	0.6742	0.9153	0.9996	0.7764
		TAPAS (Ours)	**0.9902**	0.9834	0.9875	**0.9866**
	Theia	FLASH [55]	0.8901	0.9977	0.9991	0.9408
		MAGIC [28]	0.9350	1.0000	0.9975	0.9664
		UNICORN [20]	0.8030	1.0000	0.8116	0.8908
		Threatrace [58]	0.8704	0.9974	0.9989	0.9296
		TAPAS (Ours)	**0.9854**	0.9601	0.9875	**0.9732**
	Trace	FLASH [55]	0.9054	0.9884	0.9934	0.9451
		MAGIC [28]	0.9798	0.9998	0.9989	0.9897
		UNICORN [20]	0.7625	0.9839	0.7590	0.8592
		Threatrace [58]	0.7156	1.0000	0.9892	0.8343
		TAPAS (Ours)	**0.9945**	1.0000	**0.9999**	**0.9972**
OpTC	Attack 1	FLASH [55]	0.9048	0.9194	1.0000	0.9120
		Threatrace [58]	0.8413	0.8548	1.0000	0.8480
		TAPAS (Ours)	**0.9729**	**0.9682**	0.9759	**0.9705**
	Attack 2	FLASH [55]	0.9450	0.9220	0.9999	0.9334
		Threatrace [58]	0.8483	0.8732	0.9998	0.8606
		TAPAS (Ours)	**0.9783**	**0.9624**	0.9732	**0.9697**
	Attack 3	FLASH [55]	0.9167	0.9270	0.9998	0.9218
		Threatrace [58]	0.8611	0.8708	0.9997	0.8659
		TAPAS (Ours)	**0.9791**	**0.9671**	0.9811	**0.9729**

formance on that dataset. For the OpTC dataset, since two SOTA schemes [20, 28] do not consider evaluation on OpTC, we only provide a comparison of TAPAS with Flash [55] and Threatrace [58]. Our results demonstrate that TAPAS 's task-level detection approach consistently outperforms both node/edge-level and graph-level detection methods across key performance metrics.

5.3 Performance

We first evaluate the end-to-end performance overhead of TAPAS on each of the four sub-datasets of the DARPA TC and OpTC in terms of four metrics: time overhead, throughput, memory use, and CPU usage.

End-to-End Performance. In terms of comparison, it is difficult to make an end-to-end comparison because previous approaches neglect overhead evaluation or use different datasets and test items. However, comparing data sizes and performance metrics, such as time consumption, CPU usage, and memory usage, can reflect method performance fairly to a large extent. The data size is the main factor that affects the

Table 4: End-to-end time overhead

Dataset	Size	Approach	Duration
Theia	79.3GB	TAPAS	25.75min
		MAGIC	43.37min
Cadets	35.7GB	TAPAS	8.08min
		MAGIC	16.35min
Trace	19.2GB	TAPAS	4.70min
		MAGIC	24.91min
Fivedirections	217.0GB	TAPAS	45.62min
OpTC	4.5GB	TAPAS	1.99min

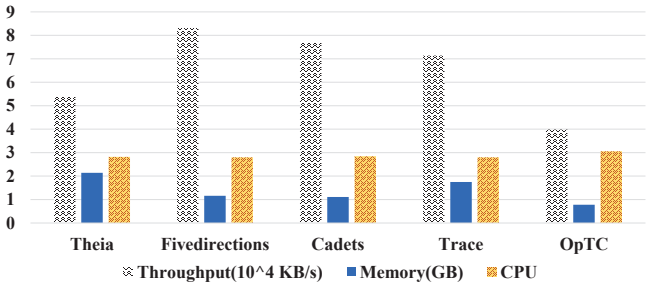

Figure 6: Runtime performance of TAPAS

performance overhead of the detector. Also, the data composition (i.e., the number and proportion of the three types of records: subject, event, and object) can have some impact since the detection is related to the graph computation, so the complexity of the constructed graph impacts the performance.

We show the experimental results of TAPAS in terms of end-to-end time overhead in Table 4. MAGIC outperforms the previous methods in terms of overhead. For instance, Shadewatcher's [71] runtime on the Trace dataset was 7.65 hours, and ProGrapher's [64] detection speed on the DARPA TC dataset averaged 29.31 minutes/GB. In addition, although we have successfully run FLASH and obtained detection effectiveness results, we are not able to calculate its time overhead due to interruptions caused by out of memory during the end-to-end run. In this context, TAPAS runs in about half the time of MAGIC. TAPAS takes an average of 12.61 seconds to process 1 GB of audit data. Even considering a busy system capable of generating 100 GB of audit data per day, TAPAS could complete the inspection of this data in about 20 minutes.

We present the experimental results of TAPAS in terms of throughput, memory usage, and CPU usage, as shown in Figure 6. Previous detection approaches rarely test these metrics or fail to provide accurate results, making them difficult to compare. Regarding throughput, TAPAS achieves an impressive average of 8.13×10^4KB/s across the five datasets. Notably, the DARPA dataset was collected under high-intensity attack scenarios, resulting in a log generation rate that significantly exceeds typical real-world conditions. Empirical studies show that a real-world enterprise system with approximately 500 hosts generates logs at a rate of around

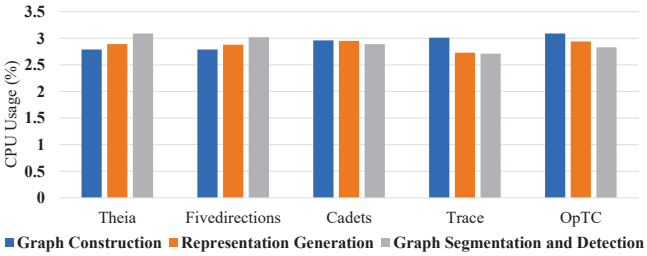

Figure 7: CPU usage of TAPAS's components

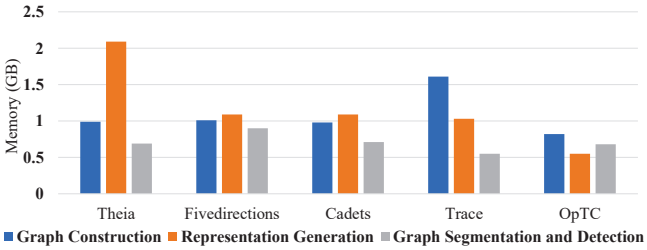

Figure 8: Memory performance of TAPAS's components

10^4KB/s [33]. TAPAS clearly surpasses this real-world requirement, delivering throughput several times higher. In addition, TAPAS's average runtime memory and CPU usage are about 1.4GB and 2.9%, respectively, which are well within the capabilities of even personal computers.

Components Performance. To deeply analyze the contribution of each component to TAPAS's performance overhead, we also run TAPAS's graph construction, representation generation, and anomaly detection components separately and measure their performance overhead.

In terms of CPU usage (as shown in Figure 7), the representation generation component has the highest metric value, with the maximum usage stabilizing at around 3%. This is due to the fact that it iteratively needs to compute new process representations by stacking GRU models multiple times. As shown in Figure 8, the caching of data by the graph construction component leads to higher memory usage. The large-scale task subgraphs in the Fivedirections dataset may be responsible for the increase in memory usage for its anomaly detection. In terms of time overhead, most of TAPAS's time overhead is caused by the graph construction component, as shown in Figure 9. It requires streaming to parse and process large-scale audit logs.

5.4 Overhead Relationship with Data Volume

Next, we conduct scalability testing to evaluate TAPAS's performance characteristics as data volume increases. Using the Fivedirections dataset, we measure the trend of TAPAS performance changes when the number of raw audit records grows from 5 million to 50 million. The results shown in Figure 10 demonstrate TAPAS's efficient resource management. CPU utilization remains stable, fluctuating around 3% regardless of data volume. Both memory usage and processing time exhibit linear growth patterns with increasing data volume, indicating predictable scalability. Specifically, each additional 5 million audit records results in an average increase of 50.89 seconds in processing time and 87.56 MB in memory consumption.

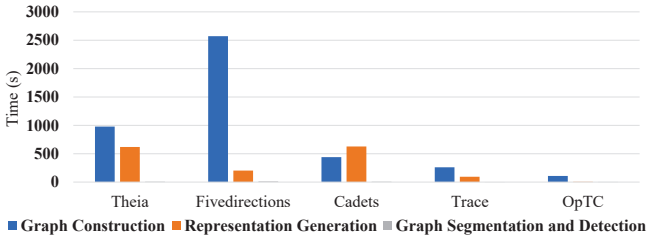

Figure 9: Time performance of TAPAS's components

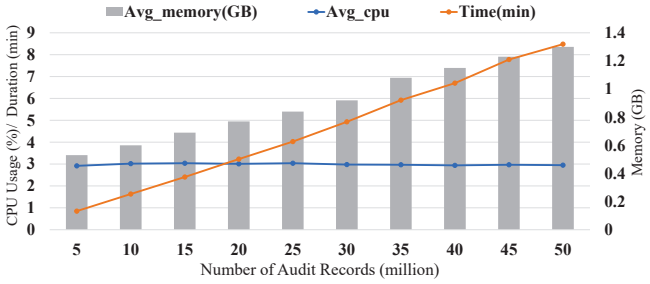

Figure 10: The memory, CPU usage and time of TAPAS vary with data volume

TAPAS maintains a process backbone graph along with a representation vector for each process. As new processes are added, the backbone graph expands and the number of process representations increases, leading to a gradual rise in memory usage. However, this growth is slow because new processes constitute only a small fraction of the overall audit data. TAPAS demonstrates low overall memory consumption, e.g., requiring on average just 1.2 GB to process 50 million audit records. Its scalability can be further improved through mechanisms such as time-based windowing or the removal of inactive (dead) processes to limit memory growth.

5.5 Storage

In addition to runtime overhead, data storage is also an important component of detector performance. Detecting APT attacks requires long-term collection of provenance information, but large amounts of historical data overwhelm disk storage for real-world use. TAPAS uses an active process graph and an object-embedded database as an alternative to long-term audit data storage. We compare the amount of data TAPAS stores to the amount of raw data and present the results in Table 5. The results show that TAPAS reduces the storage space required to audit the five datasets by a hundred or even a thousand times.

Table 5: Storage overhead

Dataset	Size	Storage	Reduce
Theia	79.3GB	81MB	1002×
Fivedirections	217.0GB	123MB	1806×
Cadets	35.7GB	184MB	199×
Trace	19.2GB	208MB	95×
OpTC	4.5GB	35.7MB	130×

Table 6: Model performance comparison for detection

Model	Precision	Recall	Accuracy	F1-Score
LSTM-GRU	0.9945	1.0000	0.9999	0.9972
LSTM	0.9978	0.8571	0.9956	0.9156
GRU	0.9145	0.8560	0.9934	0.8830

5.6 Efficacy of Process Representation

We also validate the effectiveness of the TAPAS process representation model by comparing the effects of the stacked LSTM-GRU model with those of the LSTM and GRU models. Figure 11 shows the training loss versus batch curve for these models. The stacked LSTM-GRU model we used converges significantly faster than the other two models, achieving smaller losses as the loss values stabilize. This means that this model is able to capture the hidden relationships of the process behavior sequences more accurately and is faster to train.

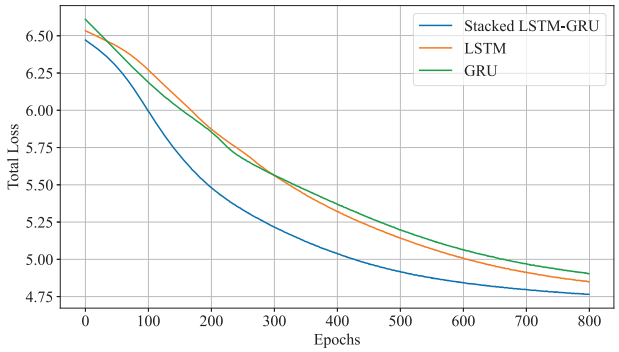

Figure 11: Comparison of training losses for different models

In addition, we test the contribution of these models to detection efficiency. We generated process representations using the stacked LSTM-GRU, LSTM, and GRU models, respectively. The process representations are used for attack detection, and Table 6 shows the corresponding detection results. Compared to the other two models, the stacked LSTM-GRU performs best in almost all four metrics, especially when the recall is significantly higher.

6 Discussion

Object Provenance Capturing. TAPAS captures dependencies on other system entities using a stacked LSTM-GRU model (Section 4.4). This model iteratively incorporates interactions between processes and their dependent entities to update the process representation vector, which encodes the historical behavior of each process. File and network objects, when causally linked to a process, can similarly influence its representation and are therefore captured by TAPAS. Lateral movement across hosts remains a significant challenge in APT detection [12, 20, 27, 28, 58, 64, 71], and TAPAS is no exception. Accurately associating audit information across hosts is inherently difficult, often leading to a loss of provenance information and, consequently, detection failures. However, TAPAS leverages localized process subgraphs for attack identification, which mitigates the impact of provenance disconnection due to cross-host movement—unlike approaches that rely on large-scale provenance graph neighborhoods. TAPAS is specifically designed to efficiently detect attacks at the subgraph level. A key limitation of this design is its inability to detect fine-grained inter-object malicious behaviors at the edge level. For instance, TAPAS cannot directly identify that host A infects host B or that process P reads from a malicious file F. These types of interactions are represented by graph edges and require further fine-grained, manual analysis. Nonetheless, TAPAS helps narrow the scope of such investigations.

Limitation of Available Datasets. There remains a significant lack of publicly available, large-scale, long-term APT datasets collected from real-world environments. While the DARPA dataset is the most representative publicly available benchmark for real-world APT scenarios (as detailed in Section 5.1), it still falls short in key aspects such as attack duration, scale, and the inclusion of novel attack techniques. In contrast to DARPA's relatively constrained scope, real-world APT attacks can span years [32], affect numerous victims, and even compromise entire supply chains [15]. Additionally, new attack techniques and zero-day vulnerabilities continue to emerge [19]. The OpTC dataset exhibits greater shortcomings than DARPA. It includes only three isolated attacks, each targeting a different host on separate days, with an average duration of just five hours in a burst-like manner [56]. These attacks are unrelated and do not reflect the long dwell times and low-frequency patterns typical of real-world APT behavior. Moreover, the attack techniques featured in the OpTC dataset are largely outdated and do not incorporate recent advancements in APT attacks.

Provenance Reduction. The storage burden of provenance data in APT detection systems increases significantly over time. Two common approaches to mitigate this issue are data compression and data pruning, which reduce provenance data from the perspectives of encoding format and content, respectively. Data compression [12, 17, 52] minimizes storage requirements by transforming the encoding format of data [49], and is a widely used technique across many domains. Data pruning [25, 73] eliminates portions of the source data that are deemed unnecessary for generating evidence

chains. Unlike compression, which operates at the encoding level, pruning requires semantic understanding to identify and remove redundant or irrelevant information. Existing data reduction methods often rely on predefined heuristics, such as eliminating repeated events or discarding certain fields. However, data compression is typically more suited to long-term archival, whereas data pruning tends to be coarse-grained and conservative to avoid negatively impacting detection accuracy. TAPAS takes a different approach. By employing a stacked LSTM-GRU model to capture the historical interactions between processes and file/network objects, TAPAS effectively reduces the dimensionality of the provenance graph—both spatially and temporally—while maintaining high detection performance for APT attacks.

Fine-grained Analysis. Fine-Grained Analysis. Fine-grained attack analysis of long-running processes remains an open challenge in APT detection. TAPAS is designed for high-speed, online attack detection and complements existing fine-grained analysis techniques. As an online system, TAPAS preserves the temporal sequence of events, which helps narrow the scope of subsequent manual analysis. Our evaluation on the DARPA dataset shows that the detected subgraphs are small, with an average of only 6.87 nodes, enabling efficient manual inspection. Each process node retains event-level summaries, providing rich contextual information for identifying malicious behavior. Prior studies [23, 66] have demonstrated that execution partitioning techniques can significantly mitigate dependency explosion by identifying application states and combining application-level and audit-level information for fine-grained analysis. Nonetheless, there is room for optimisation in terms of reducing the application limitations of execution partitioning techniques and reducing the overhead of fine-grained analysis [69]. Integrating TAPAS with execution partitioning approaches can further reduce the overall overhead of fine-grained analysis while preserving detection accuracy.

7 Conclusion

We introduce TAPAS, a task-level online APT detection framework designed to provide real-world usability of APT detection systems. TAPAS uses a process graph to describe the correlation relationships between processes and designs the model to update a process representation based on the process iteratively launched events, which is used to summarize the historical actions of a process, reducing the size of the data to be maintained in the long run. In addition, TAPAS detects subgraphs against changes, further reducing the computational complexity of the detection model. Evaluations on five widely used datasets show that TAPAS accurately identifies attacks at high speeds and low runtime loads and can keep up with real-world usage requirements.

Ethics considerations

Our research adheres to strict ethical guidelines, ensuring responsible practices throughout. TAPAS is designed to operate at the system kernel level, enabling real-time APT detection while preserving user privacy and minimizing potential misuse. The datasets used in this study—DARPA TC E3 datasets [48] collected during the Red-Blue confrontation exercises and OpTC dataset—provide standardized benchmarks for evaluating APT detection systems, further ensuring ethical use of data.

Open Science

We are dedicated to upholding the values of the Open Science Policy and strive to encourage transparency, reproducibility, and collaboration in scientific research. We release our source code at https://doi.org/10.5281/zenodo.15610687 as per the conference's requirements.

Acknowledgment

We would like to thank the anonymous reviewers and our shepherd for their detailed and valuable comments. This work is supported by National Natural Science Foundation of China (62372236, 62402223), Qing Lan Project of Jiangsu Province, and Postgraduate Research & Practice Innovation Program of Jiangsu Province (SJCX25_0176).

References

[1] Safwan Mahmood Al-Selwi, Mohd Fadzil Hassan, Said Jadid Abdulkadir, Amgad Muneer, Ebrahim Hamid Sumiea, Alawi Alqushaibi, and Mohammed Gamal Ragab. Rnn-lstm: From applications to modeling techniques and beyond—systematic review. *Journal of King Saud University - Computer and Information Sciences*, 36, 2024.

[2] Abdulellah Alsaheel, Yuhong Nan, Shiqing Ma, Le Yu, Gregory Walkup, Z. Berkay Celik, Xiangyu Zhang, and Dongyan Xu. ATLAS: A sequence-based learning approach for attack investigation. In *USENIX Security Symposium (USENIX Security)*, Anaheim, CA, 2021. USENIX Association.

[3] Enes Altinisik, Fatih Deniz, and Husrev Taha Sencar. Provg-searcher: A graph representation learning approach for efficient provenance graph search. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, New York, NY, USA, 2023. Association for Computing Machinery.

[4] Rody Arantes, Carl Weir, Henry Hannon, and Marisha Kulseng. Operationally transparent cyber (optc), 2021.

[5] Asad Arfeen, Saad Ahmed, Muhammad Asim Khan, and Syed Faraz Ali Jafri. Endpoint detection & response: A malware identification solution. In *International Conference on Cyber Warfare and Security (ICCWS)*, pages 1–8, USA, 2021. IEEE.

[6] Daniel P. Bovet and Marco Cesati. *Understanding the Linux Kernel: from I/O ports to process management*. " O'Reilly Media, Inc.", 2005.

[7] Robin Buchta, Carsten Kleiner, Felix Heine, Daniel Mahrenholz, Uwe Mönks, and Henning Trsek. Graphwatch: A novel threat hunting approach for apt activities based on anomaly detection. In *IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 01–04, 2024.

[8] Zijun Cheng, Qiujian Lv, Jinyuan Liang, Yan Wang, Degang Sun, Thomas Pasquier, and Xueyuan Han. Kairos: Practical intrusion detection and investigation using whole-system provenance. In *IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, 2024. IEEE.

[9] DARPA/I2O. Tc ground truth report e3 update. https://drive.google.com/open?id=1QlbUFWAGq3Hpl8wVdzOdIoZLFxkII4EK, 2020.

[10] Feng Dong, Shaofei Li, Peng Jiang, Ding Li, Haoyu Wang, Liangyi Huang, Xusheng Xiao, Jiedong Chen, Xiapu Luo, Yao Guo, and Xiangqun Chen. Are we there yet? an industrial viewpoint on provenance-based endpoint detection and response tools. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, New York, NY, USA, 2023. Association for Computing Machinery.

[11] Feng Dong, Liu Wang, Xu Nie, Fei Shao, Haoyu Wang, Ding Li, Xiapu Luo, and Xusheng Xiao. DISTDET: A Cost-Effective distributed cyber threat detection system. In *USENIX Security Symposium (USENIX Security 23)*, Anaheim, CA, 2023. USENIX Association.

[12] Peng Fei, Zhou Li, Zhiying Wang, Xiao Yu, Ding Li, and Kangkook Jee. Seal: Storage-efficient causality analysis on enterprise logs with query-friendly compression. In *USENIX Conference on Security Symposium (USENIX Security)*, Anaheim, CA, 2021. USENIX Association.

[13] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric, 2019.

[14] Center for Internet Security. The solarwinds cyber-attack: What you need to know. https://www.cisecurity.org/solarwinds, 03 2021.

[15] Center for Internet Security. The solarwinds cyber-attack: What you need to know. https://www.cisecurity.org/solarwinds, 03 2021.

[16] Akul Goyal, Xueyuan Han, G. Wang, and Adam Bates. Sometimes, you aren't what you do: Mimicry attacks against provenance graph host intrusion detection systems. In *Network and Distributed System Security Symposium (NDSS)*, 2023.

[17] Mohit Goyal, Kedar Tatwawadi, Shubham Chandak, and Idoia Ochoa. Deepzip: Lossless data compression using recurrent neural networks. *arXiv preprint arXiv:1811.08162*, 2018.

[18] Luca Gregori, Paolo Missier, Matthew Stidolph, Riccardo Torlone, and Alessandro Wood. Design and development of a provenance capture platform for data science. In *IEEE International Conference on Data Engineering Workshops (ICDEW)*, pages 285–290, 2024.

[19] Google Threat Intelligence Group. Hello 0-days, my old friend: A 2024 zero-day exploitation analysis. https://cloud.google.com/blog/topics/threat-intelligence/2024-zero-day-trends, 04 2025.

[20] Xueyuan Han, Thomas Pasquier, Adam Bates, James Mickens, and Margo Seltzer. Unicorn: Runtime provenance-based detector for advanced persistent threats. In *Network and Distributed System Security Symposium (NDSS)*, San Diego, California, USA, 2020. The Internet Society.

[21] Wajih Ul Hassan, Adam Bates, and Daniel Marino. Tactical provenance analysis for endpoint detection and response systems. In *IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, 2020. IEEE.

[22] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. Nodoze: Combatting threat alert fatigue with automated provenance triage. In *Network and Distributed System Security Symposium (NDSS)*, San Diego, California, USA, 2019. The Internet Society.

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

[24] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R Lyu. A survey on automated log analysis for reliability engineering. *ACM Computing Surveys (CSUR)*, 54(6):1–37, 2021.

[25] Md Nahid Hossain, Junao Wang, R Sekar, and Scott D Stoller. Dependence-preserving data compaction for scalable forensic analysis. In *USENIX Conference on Security Symposium (USENIX Security)*, pages 1723–1740, Anaheim, CA, 2018. USENIX Association.

[26] Sagar Imambi, Kolla Bhanu Prakash, and GR Kanagachidambaresan. Pytorch. *Programming with TensorFlow: solution for edge computing applications*, pages 87–104, 2021.

[27] Muhammad Adil Inam, Yinfang Chen, Akul Goyal, Jason Liu, Jaron Mink, Noor Michael, Sneha Gaur, Adam Bates, and Wajih Ul Hassan. Sok: History is a vast early warning system: Auditing the provenance of system intrusions. In *IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, 2023. IEEE.

[28] Zian Jia, Yun Xiong, Yuhong Nan, Yao Zhang, Jinjing Zhao, and Mi Wen. MAGIC: Detecting advanced persistent threats via masked graph representation learning. In *USENIX Security Symposium (USENIX Security)*, Philadelphia, PA, 2024. USENIX Association.

[29] Kaspersky. Apt q1 2023 playbook: advanced techniques, broader horizons, and new targets. https://www.kaspersky.com/about/press-releases/apt-q1-2023-playbook-advanced-techniques-broader-horizons-and-new-targets, 04 2023.

[30] Yonghwi Kwon, Fei Wang, Weihang Wang, Kyu Hyung Lee, Wen-Chuan Lee, Shiqing Ma, X. Zhang, Dongyan Xu, Somesh Jha, Gabriela F. Cretu-Ciocarlie, Ashish Gehani, and Vinod Yegneswaran. Mci : Modeling-based causality inference in audit logging for attack investigation. In *Network and Distributed System Security Symposium (NDSS)*, San Diego, California, USA, 2018. The Internet Society.

[31] FortiGuard Labs. Global threat landscape report. https://www.fortinet.com/content/dam/fortinet/assets/threat-reports/threat-report-1h-2023.pdf, 08 2023.

[32] Ravie Lakshmanan. Researchers uncover years-long cyber espionage on foreign embassies in belarus. https://thehackernews.com/2023/08/researchers-uncover-decade-long-cyber.html, 08 2023.

[33] Shaofei Li, Feng Dong, Xusheng Xiao, Haoyu Wang, Fei Shao, Jiedong Chen, Yao Guo, Xiangqun Chen, and Ding Li. Nodlink: An online system for fine-grained apt attack detection and investigation. In *Network and Distributed System Security Symposium (NDSS)*, San Diego, California, USA, 2024. The Internet Society.

[34] Theodore A Linden. Operating system structures to support security and reliable software. *ACM Computing Surveys (CSUR)*, 8(4):409–445, 1976.

[35] Robert Love. *Linux kernel development*. Pearson Education, 2010.

[36] Shiqing Ma, X. Zhang, and Dongyan Xu. Protracer: Towards practical provenance tracing by alternating between logging and tainting. *Network and Distributed System Security Symposium (NDSS)*, 2016.

[37] Matin Maleki. Process management in linux. *Developers ultimate guide: Linux Bash scripting*, 2022.

[38] Neethu Elizabeth Michael, Ramesh C Bansal, Ali Ahmed Adam Ismail, A Elnady, and Shazia Hasan. A cohesive structure of bi-directional long-short-term memory (bilstm)-gru for predicting hourly solar radiation. *Renewable Energy*, 222:119943, 2024.

[39] Sadegh M. Milajerdi, Birhanu Eshete, Rigel Gjomemo, and V.N. Venkatakrishnan. Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, New York, NY, USA, 2019. Association for Computing Machinery.

[40] Taima Rahman Mim, Maliha Amatullah, Sadia Afreen, Mohammad Abu Yousuf, Shahadat Uddin, Salem A. Alyami, Khondokar Fida Hasan, and Mohammad Ali Moni. Gru-inc: An inception-attention based approach using gru for human activity recognition. *Expert Syst. Appl.*, 216(C), April 2023.

[41] The Hacker News. Researchers uncover years-long cyber espionage on foreign embassies in belarus. https://thehackernews.com/2023/08/researchers-uncover-decade-long-cyber.html, 08 2023.

[42] John O'Gorman. *The Linux Process Manager: The internals of scheduling, interrupts and signals*. John Wiley & Sons, Inc., 2003.

[43] Bofeng Pan, Natalia Stakhanova, and Suprio Ray. Data provenance in security and privacy. *ACM Computing Surveys*, 55(14s), jul 2023.

[44] Mian Pan, Ailin Liu, Yanzhen Yu, Penghui Wang, Jianjun Li, Yan Liu, Shuaishuai Lv, and He Zhu. Radar hrrp target recognition model based on a stacked cnn–bi-rnn with attention mechanism. *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–14, 2022.

[45] Thomas Pasquier, Xueyuan Han, Mark Goldstein, Thomas Moyer, David Eyers, Margo Seltzer, and Jean Bacon. Practical whole-system provenance capture. In *Symposium on Cloud Computing*, pages 405–418, 2017.

[46] Kexin Pei, Zhongshu Gu, Brendan Saltaformaggio, Shiqing Ma, Fei Wang, Zhiwei Zhang, Luo Si, Xiangyu Zhang, and Dongyan Xu. Hercule: attack story reconstruction via community discovery on correlated log graph. In *Annual Conference on Computer Security*

Applications (ACSAC '16), page 583–595, New York, NY, USA, 2016. Association for Computing Machinery.

[47] João Felipe Pimentel, Juliana Freire, Leonardo Murta, and Vanessa Braganholo. A survey on collecting, managing, and analyzing provenance from scripts. *ACM Computing Surveys*, 52(3), June 2019.

[48] ranok. Darpa transparent computing program engagement 3 data release. https://github.com/darpa-i2o/Transparent-Computing, 2020.

[49] Khalid Sayood. *Introduction to data compression(Fifth Edition)*. The Morgan Kaufmann Series in Multimedia Information and Systems. Morgan Kaufmann, 2018.

[50] R. Sekar, Hanke Kimm, and Rohit Aich. eaudit: A fast, scalable and deployable audit data collection system. In *IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, 2024. IEEE.

[51] stevegrubb. Linux auditd. https://github.com/linux-audit/audit-userspace, 08 2024.

[52] Yutao Tang, Ding Li, Zhichun Li, Mu Zhang, Kangkook Jee, Xusheng Xiao, Zhenyu Wu, Junghwan Rhee, Fengyuan Xu, and Qun Li. Nodemerge: Template based efficient data reduction for big-data causality analysis. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, New York, NY, USA, 2018. Association for Computing Machinery.

[53] threaTrace detector. groundtruth. https://github.com/threaTrace-detector/threaTrace/, 2023.

[54] Benjamin E Ujcich, Samuel Jero, Richard Skowyra, Adam Bates, William H Sanders, and Hamed Okhravi. Causal analysis for software-defined networking attacks. In *USENIX Conference on Security Symposium (USENIX Security)*, Anaheim, CA, 2021. USENIX Association.

[55] Mati Ur Rehman, Hadi Ahmadi, and Wajih Ul Hassan. Flash: A comprehensive approach to intrusion detection via provenance graph representation learning. In *IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, 2024. IEEE.

[56] waa. Optc red team ground truth. https://drive.google.com/drive/folders/1n3kkS3KR31KUegn42yk3e6JkZvf0Caa, 10 2020.

[57] Qi Wang, Wajih Ul Hassan, Ding Li, Kangkook Jee, Xiao Yu, Kexuan Zou, Junghwan John Rhee, Zhengzhang Chen, Wei Cheng, Carl A. Gunter, and Haifeng Chen. You are what you do: Hunting stealthy malware via data provenance analysis. In *Network and Distributed System Security Symposium (NDSS)*, San Diego, California, USA, 2020. The Internet Society.

[58] Su Wang, Zhiliang Wang, Tao Zhou, Hongbin Sun, Xia Yin, Dongqi Han, Han Zhang, Xingang Shi, and Jiahai Yang. Threatrace: Detecting and tracing host-based threats in node level through provenance graph learning. *IEEE Transactions on Information Forensics and Security*, 17:3972–3987, 2022.

[59] Min Xia, Haidong Shao, Xiandong Ma, and Clarence W. de Silva. A stacked gru-rnn-based approach for predicting renewable energy and electricity load for smart grid operation. *IEEE Transactions on Industrial Informatics*, 17(10):7050–7059, 2021.

[60] Sheng Xiang, Penghua Li, Yi Huang, Jun Luo, and Yi Qin. Single gated rnn with differential weighted information storage mechanism and its application to machine rul prediction. *Reliability Engineering & System Safety*, 242, 2024.

[61] Yulai Xie, Dan Feng, Yuchong Hu, Yan Li, Staunton Sample, and Darrell Long. Pagoda: A hybrid approach to enable efficient real-time provenance based intrusion detection in big data environments. *IEEE Transactions on Dependable and Secure Computing*, 17(6):1283–1296, 2020.

[62] Zhang Xu, Zhenyu Wu, Zhichun Li, Kangkook Jee, Junghwan Rhee, Xusheng Xiao, Fengyuan Xu, Haining Wang, and Guofei Jiang. High fidelity data reduction for big data security dependency analyses. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, New York, NY, USA, 2016. Association for Computing Machinery.

[63] Zhiqiang Xu, Pengcheng Fang, Changlin Liu, Xusheng Xiao, Yu Wen, and Dan Meng. Depcomm: Graph summarization on system audit logs for attack investigation. In *IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, 2022. IEEE.

[64] Fan Yang, Jiacen Xu, Chunlin Xiong, Zhou Li, and Kehuan Zhang. PROGRAPHER: An anomaly detection system based on provenance graph embedding. In *USENIX Security Symposium (USENIX Security 23)*, Anaheim, CA, 2023. USENIX Association.

[65] Na Yi, Jianjun Xu, Limei Yan, and Lin Huang. Task optimization and scheduling of distributed cyber–physical system based on improved ant colony algorithm. *Future Generation Computer Systems*, 109:134–148, 2020.

[66] Le Yu, Shiqing Ma, Zhuo Zhang, Guanhong Tao, X. Zhang, Dongyan Xu, Vincent E. Urias, Han Wei Lin, Gabriela Felicia Ciocarlie, Vinod Yegneswaran, and

Ashish Gehani. Alchemist: Fusing application and audit logs for precise attack provenance without instrumentation. In *Network and Distributed System Security Symposium (NDSS)*, San Diego, California, USA, 2021. The Internet Society.

[67] Hao Yue, Tong Li, Di Wu, Runzi Zhang, and Zhen Yang. Detecting apt attacks using an attack intent-driven and sequence-based learning approach. *Computers and Security*, 140(C), July 2024.

[68] Jun Zeng, Zheng Leong Chua, Yinfang Chen, Kaihang Ji, Zhenkai Liang, and Jian Mao. Watson: Abstracting behaviors from audit logs via aggregation of contextual semantics. In *Network and Distributed System Security Symposium (NDSS)*, San Diego, California, USA, 2021. The Internet Society.

[69] Jun Zeng, Chuqi Zhang, and Zhenkai Liang. Palantír: Optimizing attack provenance with hardware-enhanced system observability. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2022.

[70] Lei Zeng, Yang Xiao, and Hui Chen. Linux auditing: Overhead and adaptation. In *IEEE International Conference on Communications (ICC)*, pages 7168–7173, USA, 2015. IEEE.

[71] Jun Zengy, Xiang Wang, Jiahao Liu, Yinfang Chen, Zhenkai Liang, Tat-Seng Chua, and Zheng Leong Chua. Shadewatcher: Recommendation-guided cyber threat analysis using system audit records. In *IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, 2022. IEEE.

[72] Bo Zhang, Yansong Gao, Boyu Kuang, Changlong Yu, Anmin Fu, and Willy Susilo. A survey on advanced persistent threat detection: A unified framework, challenges, and countermeasures. *ACM Computing Surveys*, 57(3), November.

[73] Tiantian Zhu, Jiayu Wang, Linqi Ruan, Chunlin Xiong, Jinkai Yu, Yaosheng Li, Yan Chen, Mingqi Lv, and Tieming Chen. General, efficient, and real-time data compaction strategy for apt forensic analysis. *IEEE Transactions on Information Forensics and Security*, 16:3312–3325, 04 2021.

[74] Tiantian Zhu, Jie Ying, Tieming Chen, Chunlin Xiong, Wenrui Cheng, Qixuan Yuan, Aohan Zheng, Mingqi Lv, and Yan Chen. Nip in the bud: Forecasting and interpreting post-exploitation attacks in real-time through cyber threat intelligence reports. *IEEE Transactions on Dependable and Secure Computing*, pages 1–18, 2024.

[75] Tiantian Zhu, Jinkai Yu, Chunlin Xiong, Wenrui Cheng, Qixuan Yuan, Jie Ying, Tieming Chen, Jiabo Zhang, Mingqi Lv, Yan Chen, Ting Wang, and Yuan Fan. Aptshield: A stable, efficient and real-time apt detection system for linux hosts. *IEEE Transactions on Dependable and Secure Computing*, 20(6):5247–5264, 2023.

[76] Michael Zipperle, Florian Gottwalt, Elizabeth Chang, and Tharam Dillon. Provenance-based intrusion detection systems: A survey. *ACM Computing Surveys*, 55(7):1–36, July 2023.