DISTDET: A Cost-Effective Distributed Cyber Threat Detection System

Feng Dong[1,2] Liu Wang[3] Xu Nie[3] Fei Shao[4] Haoyu Wang[1]‡ Ding Li[5]
Xiapu Luo[6] Xusheng Xiao[7]

[1]*School of Cyber Science and Engineering, Huazhong University of Science and Technology* [2]*Sangfor Technologies Inc*
[3]*Beijing University of Posts and Telecommunications* [4]*Case Western Reserve University*
[5]*Key Laboratory of High-Confidence Software Technologies (MOE), School of Computer Science, Peking University*
[6]*The Hong Kong Polytechnic University* [7]*Arizona State University*
‡*Corresponding author*

Abstract

Building provenance graph that considers causal relationships among software behaviors can better provide contextual information of cyber attacks, especially for advanced attacks such as Advanced Persistent Threat (APT) attacks. Despite its promises in assisting attack investigation, existing approaches that use provenance graphs to perform attack detection suffer from two fundamental limitations. First, existing approaches adopt a centralized detection architecture that sends all system auditing logs to the server for processing, incurring intolerable costs of data transmission, data storage, and computation. Second, they adopt either rule-based techniques that cannot detect unknown threats, or anomaly-detection techniques that produce numerous false alarms, failing to achieve a balance of precision and recall in APT detection. To address these fundamental challenges, we propose DISTDET, a distributed detection system that detects APT attacks by (1) performing light weight detection based on the host model built in the client side, (2) filtering false alarms based on the semantics of the alarm proprieties, and (3) deriving global models to complement the local bias of the host models. Our experiments on a large-scale industrial environment (1,130 hosts, 14 days, ∼1.6 billion events) and the DARPA TC dataset show that DISTDET is as effective as sate-of-the-art techniques in detecting attacks, while dramatically reducing network bandwidth from 11.28Mb/s to 17.08Kb/S (676.5× reduction), memory usages from 364MB to 5.523MB (66× reduction), and storage from 1.47GB to 130.34MB (11.6× reduction). By the time of this writing, DISTDET has been deployed to 50+ industry customers with 22,000+ hosts for more than 6 months, and identified over 900 real-world attacks.

1 Introduction

Advanced cyber attacks, such as advanced persistent threat (APT) attacks, have penetrated into many well-defended targets, causing significant financial losses [1, 34]. These attacks usually combine various advanced attack techniques (e.g., 0-day, fileless and living off the land) into different attack steps [37, 39], making the entire attack more stealthy. Traditional intrusion detection and prevention systems (IDPS) can only detect a few known exploits, and fail to capture the entire APT attack due to the lack of causality analysis capability. Recently, provenance graph based detection [16, 18, 29, 49] is considered as a promising way to combat APT attacks, where system monitoring is applied to collect auditing logs of system calls. Provenance graph represents the system execution as a directed acyclic graph (DAG), where nodes represent system entities (e.g., processes, files, and network connections) and edges represent system event (e.g., a process creating a file). Using provenance graphs, detection tools can obtain the contextual information of an APT attack by constructing a chain of events that lead to the alarm event reported by anomaly detection tools. Such contextual information is effective in revealing advanced attack tactics such as distinguishing benign uses of ZIP from ransomware [15, 18].

Despite of the promising early results, provenance graphs are impractical for real-time APT detection in the industrial setting due to two fundamental challenges: *intolerable computational overheads* and *poor balance in precision and recall for detection* [17, 18]. On the one hand, as constructing provenance graphs consumes significant computing resources, existing work mainly adopts the centralized architecture, where monitored hosts (referred to as clients) upload the collected logs for centralized processing. However, the number of clients in an enterprise cluster is usually more than 1,000 in practice, and thus the costs of data transmissions, data storage, and computations incurred by the centralized architecture well exceed the security budgets of most enterprises [40]. For example, based on the analysis of the Cadets dataset of DARPA Engagement 3 [38], one client generates 3.7G of log data per day on average, and 687MB of memory is required if we adopt the state-of-the-art approach Unicorn [16] for APT detection. Thus, if a cluster of 1,000 hosts is monitored, 3.61TB of data needs to be transmitted and stored per day, which requires 351Mb/s network bandwidth and 671GB memory. *These daunting numbers make it infeasible for most*

enterprises to adopt the provenance graphs for APT detection. On the other hand, it is difficult to achieve a balance of precision and recall in detection. Rule-based approaches usually rely on generic patterns to improve the recall, but they can also misidentify benign behaviors as threats [18]. Moreover, rule-based methods are severely limited by expert knowledge and cannot detect unknown threats. While anomaly detection-based approaches are able to detect unknown threats, they are known to produce numerous false positives and suffer from the notorious alarm fatigue problem [18].

To address these fundamental challenges, in this paper, we propose DISTDET, the first cost-effective detection system that synergistically combines **distributed computing**, **anomaly detection**, and **false alarm filtering** techniques for detecting and investigating cyber attacks APT attacks. Specifically, DISTDET adopts a novel distributed detection architecture to minimize the overall computational cost by shifting part of the APT detection to the clients and transmitting only summary graphs that represent potential attacks to the server, which greatly reduces the costs of data transmission and storage. Furthermore, DISTDET synergistically combines anomaly detection and false alarm filtering to detect unknown threats (improving the recall) and combat alarm fatigue (improving the precision). In particular, with the novel synergy of these techniques, DISTDET (1) *achieves high precision in detecting APT attacks with low costs*, (2) *presents more contextual information (summary graphs) than the existing detection tools that report only the suspicious event*, and (3) *seamlessly supports provenance graph based analysis through client-side caches*. We next describe the technical challenges and our insights in addressing these challenges.

Technical Challenges. There are three major challenges on developing a distributed system for real-time APT detection:

① *How can we ensure the overheads incurred by the APT detection will not affect the performance of the clients' daily business?* For example, to minimize the client side impacts, the internationally renowned Endpoint Detection and Response (EDR) company, CrowdStrike [9], has a memory footprint of less than 25.36MB, a CPU footprint of 1%-3%, and about 5MB of data transferred per day. Similarly, if we consider the acceptable overhead is 3% based on the ratio for information security investment, for a client host with 4GB RAM and 80GB disk, DISTDET needs to limit the CPU usage to be within 3%, the memory to be within 120MB, and the storage space to be within 2.4GB.

② *How to deal with the notorious alarm fatigue of anomaly detection in* DISTDET*?* While anomaly detection improves the recall, many benign but unseen behaviors can also be flagged as anomalies, facing a more serious alarm fatigue problem than the rule-based approach [16, 49].

③ *How can the distributed system overcome the local bias and achieve similar detection effectiveness as the centralized architecture?* Since the distributed system does not transmit all the event logs to the server for centralized analysis, the models built locally by each host will lack global information, which may compromise the detection effectiveness.

Key Insights. To address these challenges, the design of DISTDET is powered by three key insights:

- **Lightweight Client-Side Detection**: While it is infeasible to perform computationally intensive APT detection, clients are able to conduct light-weight detection by building a compact and expressive index based on the observed normal behaviors and identify unseen behaviors based on the index as alarms, and the server can perform more expensive analysis on the alarms sent by the clients to further filter false alarms. In this way, the clients can not only minimize computational costs, but also eliminate the needs for uploading all the observed logs.
- **Unique Properties of False Alarms**: Most of the false alarms are caused by rare benign behaviors, which are not observed during the learning period and thus cannot be learned by the model. Through careful inspections of the large amount of false alarms, we found that false alarms typically possess some unique properties: (1) the alarms representing the same behaviors will be repetitively reported over a period of time; (2) many false alarms are related to the benign behaviors triggered by semantically similar commands; (3) the contexts for these alarms are generally known to represent benign behaviors.
- **Global View of Service Behaviors**: Most enterprises adopt load-balancing and thus hosts that provide the same type of services, such as web servers, typically execute the almost same set of processes to serve the outside world. As each host is running in different phases, the benign behaviors observed locally during the learning period are only limited to certain phases. While local models can easily lead to false alarms in detection, a global model built in the server can observe the behaviors in all the phases and can complement the missing observations in the local models.

Contributions. Based on these key insights, DISTDET develops three major components to address the challenges:

(1) Host-based Anomaly Detection. DISTDET includes the novel designs of a *hierarchical system event tree* (HST, see § 4.1.1) and *Alarm Summary Graphs (ASG, see § 4.1.3)* to minimize the impacts on the clients' performance. For each client, DISTDET first builds an HST as the host model based on the logs collected during the learning period. HST is a compact index that categorizes auditing events based on their properties using a multi-layer tree, where the first layer concerns about the entity type (i.e., process, file, network sockets) and the last layer concerns about the parameters of the executed commands. Compared to maintaining the entire provenance graph for recording the observed behaviors, an HST model is very efficient to compute and it reduces the average memory footprint from 687MB [16] to 5.52MB (§ 5.2). Based on the created HST model, DISTDET detects events that deviate from the model as alarms (§ 4.1.2), i.e., performing *light-weight anomaly detection*. Then for each alarm, DIST-

DET generates an ASG and transmits the ASG to the server. An ASG is a summary graph that includes the process p that initiates the suspicious behavior reported in an alarm, and the events initiated by p's ancestor processes and descendent processes. As an ASG's size is significantly reduced from the original provenance graph, it drastically reduces the costs.

(2) False Alarm Filtering. DISTDET uses three steps to overcome the notorious alarm fatigue problem: alarm deduplication, semantic alarm aggregation, and alarm ranking based on contextual events. Based on the unique properties for false alarms, in the client side, DISTDET eliminates alarms that share the same properties based on the host model (§ 4.1.3), so that duplicated alarms triggered by repetitive benign behaviors can be filtered out. Then in the server side, DISTDET first applies a more computationally-intensive algorithm (§ 4.2.2) to aggregate the alarms whose properties exhibit similar semantics. Next, DISTDET provides a novel ASG Ranking algorithm (§ 4.2.3) that prioritizes alarms by considering whether their contextual events (i.e., ancestor processes and descendent processes in the ASGs) are considered as anomalous and whether these contextual events are rare among the ASGs.

(3) Global Model Derivation. To complement the missing behaviors observed during the learning period, DISTDET derives a global model by merging the HSTs sent by the clients periodically. As a result, each host that provides the same type of services will have a global view of the service behaviors, addressing the local bias in the models and achieving the similar effectiveness as a centralized architecture. *To the best of our knowledge, we are the first to propose such an approach to generate global models to improve detection.*

Evaluation. We built a prototype of DISTDET in roughly ∼20K lines of code and deployed it in an anonymized industrial environment that includes 1,130 hosts. A professional red team from a leading security company performed 4 distinct APT attacks on 7 out of the 1,130 hosts. We then applied DISTDET to detect these attacks for evaluating its effectiveness. We strictly enforced the cost limit (i.e., 3% overhead) in all the deployed hosts. We further compare DISTDET with the state-of-the-art approaches on the DARPA TC dataset [38]. ***In total, we collected the system events for 14 days in our deployed environment (∼1.6 billion events) and the DARPA TC dataset contains ∼147 million events. We are the first to evaluate APT detection systems using the real industry data at this scale, and we are able to reveal the practical challenges that previous research cannot observe.*** For example, many existing works, such as Unicorn [16], fail to work at this scale as each host will need 300-600MB to hold a provenance graph and the central server simply cannot handle the data of 1000+ hosts.

Our evaluation results demonstrate that DISTDET is highly cost-effective in detecting APT attacks for large-scale clusters. On average, *the total cost of protecting the 1,130 hosts is merely $68.93 USD, which is 56× less than the centralized architecture (∼ $3,842 USD)*: (1) the network cost is ∼15.476b/s, achieving a 676.5× reduction from ∼10.22Kb/s; (2) the storage cost is ∼130.34MB on average for the 14-day data persistence, achieving a 11.6× reduction from ∼1.47GB; (3) the memory footprint is ∼5.523MB, achieving a 66× reduction from ∼364MB. In terms of detection effectiveness, our results show that DISTDET achieves an almost perfect recall for the 4 real APT attacks, and is as good as Unicorn for the attack cases in the DARPA dataset. In particular, on average, DISTDET*'s false alarm filtering component effectively reduces the false alarms from 230 alarms per host per day to 0.71 alarms per host per day, saving 99.69% of the required inspection efforts from security analysts*. By the time of this writing, DISTDET has been deployed to 50+ companies (with over 22,000 hosts) for over 6 months, and over 900 real-world attacks were identified by DISTDET.

Table 1: Representative attributes of system entities

Entity	Attributes
File	Name, Directory, Owner/Group, FileID, etc.
Process	MD5, Name, User, Command, etc.
Network	IP, Port, Protocol

Table 2: Representative attributes of system events

Operation	Start/Exit, Create/Modify/Delete/Rename, Connect/Listen
Time/Sequence	Record Time, Event Sequence
Misc.	Subject ID, Object ID

2 Background and Motivating Example

2.1 System Auditing Events

System auditing events describe the interactions between two system entities, which can be represented as a 3-tuple, $evt = \langle subject, option, object \rangle$. As shown in the existing studies [10, 20, 28, 51], in most cases, subjects represent process entities (e.g., Chrome), and objects represent files, processes, or network entities. Thus, the collected system audit events are mapped to three types, i.e., *file events*, *process events*, and *network events*. Following the established trend, in this work, we focus on these three types of events and the main security-related attributes of the entities and events are shown in Table 1 and 2. Representative attributes of entities include file name, process command line, IP, etc. Representative attributes of events include event operations (e.g., create a file), event time (e.g., start time/end time), etc.

2.2 Motivating Example

Figure 1 shows part of the ASGs generated by DISTDET during the F-Lateral attack campaign. It includes 20 attack steps represented as F1 to F20 (see § B.2 in Appendix for details). During a two-week monitoring in our evaluation, the anonymized industrial environment generated a total of ∼1.6 billion system events (1.6TB) from 1,132 hosts (Industry and

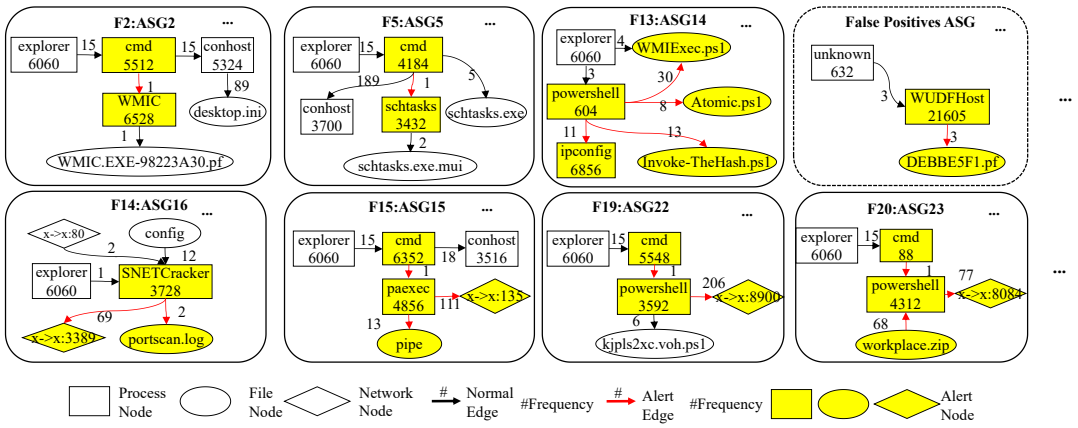


Figure 1: Part of Alarm Summary Graphs (ASGs) for the F-Lateral attack campaign

Public Arena datasets together), and distinguishing truly significant attacks from these ∼1.6 billion events is definitely "finding a needle in a haystack". The centralized architecture adopted by the existing works will require ∼11.28Mb/s bandwidth to transmit the raw logs to the server in real time, ∼401GB (Unicorn [16]) memory for graph computation, and ∼1.63TB disk space to store the logs, which well exceeds the acceptable costs of most enterprises' security budgets.

How DISTDET detects the F-Lateral attack. DISTDET first builds the HST as the host model and transmits it to the server to derive the global model, which helps address the local bias. As shown in Figure 1, DISTDET takes advantage of the derived global model to detect unobserved behaviors as alarms, and generates ASGs for each alarm. The alarm events are represented as the red edges inside the ASGs. Before generating the ASG, a sliding time window is first used to remove duplicate alarms, and the numbers of duplicates are recorded as the properties of the edge. For example, the alarm event `<paexec connect x->x:135>` in `F15:ASG15` is repeated 111 times, indicating that `paexec` establishes a lateral channel from host A to host B on port 135. Similarly, the frequency of a benign event denotes how many times the event occurs in the model, such as the benign event `<cmd, start, conhost>` in `F15:ASG15` that has occurred 18 times.

After the ASGs are sent to the server side, DISTDET parses the commands associated with the alarm events into a syntactic tree, and aggregates ASGs with similar commands. After aggregation, 29 ASGs are remained for detecting the F-Lateral attack, and DISTDET further computes their anomaly scores based on our ASG Ranking algorithm. 23 ASGs (including 2 false positive ASGs) out of these 29 ASGs have anomaly scores higher than the threshold τ_d and are reported by DISTDET as attack ASGs. As only 2 ground truth ASGs are missed, DISTDET is highly effective to detect the attack steps of the F-Lateral attack (precision of 91% and recall of 91%).

In total, the distributed architecture reduces the total cost by 56 times, i.e., reducing the network cost by 676 times ($11.28Mb/s \rightarrow 17.08Kb/s$), the memorial cost by 66 times ($401GB \rightarrow 6GB$), and the storage cost by 11.6 times ($1.63TB \rightarrow 143.83GB$). This makes DISTDET a cost effective approach for APT detection in the industrial settings.

3 Overview

Figure 2 shows the architecture of DISTDET, which consists of three components: (1) Host-based Anomaly Detection, (2) False Alarm Filtering, and (3) Global Model Derivation. The Host-based Anomaly Detection component flags anomaly events based on the host models as alarms in the client side, generates an ASG for each alarm, and sends the ASGs to the server. The False Alarm Filtering reduces false alarms by eliminating duplicated alarms in the client side, aggregating ASGs with similar semantics, and prioritizing ASGs with high anomaly scores. The Global Model Derivation receives the host models sent by the clients and merges the HSTs of the hosts that provide the same types of service to obtain global models. These global models are then distributed to the clients for complementing their host models.

Threat Model. Similar to most previous works [16–20, 26, 28, 29, 49] that detect APT attacks using system auditing logs, we assume the integrity of the data collection in the clients, i.e., the collected event streams are complete. We also assume that the attacker cannot manipulate or delete the auditing logs, i.e., log integrity is maintained all the time. In real-world attacks, the attacker may uninstall the auditing tools or tamper with the event stream. To mitigate this risk, tamper-evident logging solutions [35] can be adopted to ensure the integrity of the collected event stream and it is not the focus of this work. Same as other anomaly detection methods [16, 49], we also face the problem of model poisoning, i.e., the model is contaminated with attack behaviors. We assume no attacks are performed during the training period of our model. We also do not consider the attacks performed using side channels or inter-procedural communications (IPC) that cannot be captured by system auditing logs. It is beyond the scope of this work since our focus is to develop a cost-effective system for detecting APT attacks using system auditing logs.

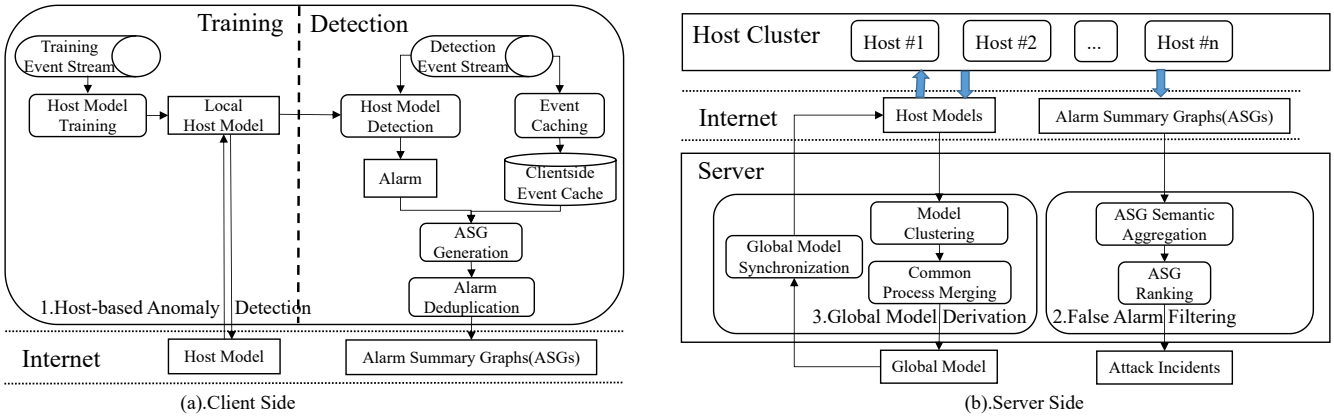


Figure 2: The overall architecture of DISTDET

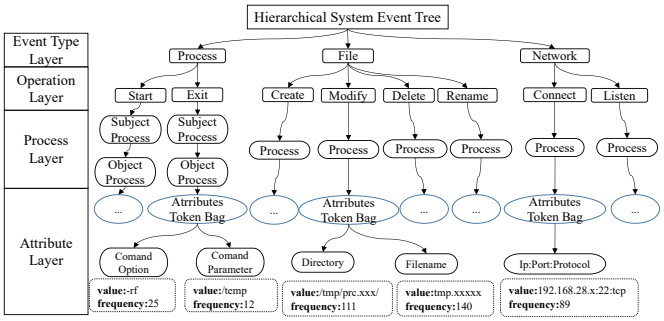


Figure 3: Host model - Hierarchical System Event Tree (HST)

4 Design Details of DISTDET

4.1 Host-based Anomaly Detection

This component has two modes: the training mode and the detection mode. In the training mode, it trains a host model from the system event stream for each client. In the detection mode, it detects events deviating from the model as alarms and generates ASGs based on the alarms.

4.1.1 Host Model Training

Figure 3 illustrates the trained Hierarchical System Event Tree (HST) model, which categorizes auditing events based on their properties using a multi-layer tree. Specifically, an HST model consists of four layers of nodes: *Event type*, *Operation*, *Process* and *Attribute*, where each layer concerns about a specific set of properties of the events. Given a system event *evt*, an HST finds a matched node in each layer to represent the event's corresponding attribute values, or create a new node if the values are not found.

The event type layer of an HST has three submodels: *process*, *file* and *network*. Based on *evt*'s event type *t*, DISTDET uses a submodel whose event type matches *t* to find the nodes in the operation layer. Then DISTDET finds or creates a node in the operation layer based on *evt*'s operation type, such as `start` and `exit` operations for a process event. After that, DISTDET continues to find or create a node in the process layer based on the process that initiates the event *evt*. Here the *md5* of the process is used as the unique identifier to represent the process. Finally, DISTDET builds nodes in the attribute layer to represent *evt*'s other attributes based on *evt*'s event type. For example, a process event is mainly represented using its command line attributes, including command options and arguments. Meanwhile, DISTDET performs on-demand tokenization for each attribute node to improve the generalization of the node representation. For example, some file paths such as `/tmp/prc.280002378/` and `/tmp/prc.280002379/` share the same prefix `/tmp/prc`. Thus, by tokenizing the numeric strings, DISTDET can use one node `/tmp/prc.xxx/` to represent these two file paths and records its frequency, as shown in Figure 3. Table 10 in Appendix § A.1 shows other representative scenarios for our tokenization.

In the learning mode, DISTDET categorizes the observed system events based on their proprieties and builds corresponding nodes in the HST. The frequency of each event is also recorded in the corresponding attribute node, which will be used by the subsequent ASG Ranking (§ 4.2.3). When the proportion of newly created nodes is less than 0.1% for two consecutive days, the model is considered as converged. Our experiment result (§ C.3 in Appendix) shows that most servers reach convergence within 3 days. Note that the HST model can be updated based on the global model that contains more observed benign events. More importantly, the complexity of finding nodes for a system event is limited to hash checks of the attribute values in the four layers, and thus it requires very low computational resources and memory consumption, meeting the demand of low overhead in the client.

4.1.2 Host Model Detection

After an HST is converged in a client, DISTDET switches to the detection mode. Given a system event *evt′*, DISTDET searches the HST to find whether there are nodes that match *evt′*' attributes. If not found, *evt′* is reported as an alarm. That is, any event no observed during the learning period is considered as anomaly, which will detect as many anomalous

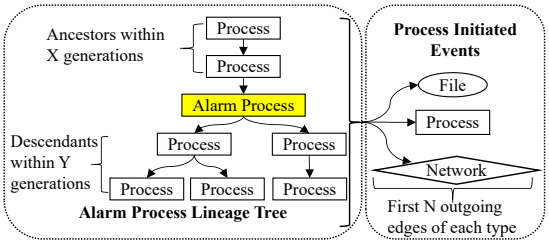


Figure 4: ASG Generation

events as possible. Besides, the HST allows DISTDET to use different sub-models to complement each other for anomaly detection. Consider a user who one day uses Chrome to access the Internet on a public library host. Due to his access to a malicious website, the host is unfortunately subject to a drive-by download attack and Chrome starts to execute Powershell. Since the library host is used by different people for surfing the Internet every day, there is a lot of randomness in the network addresses accessed and Chrome's network sub-model fails to converge. Fortunately, Chrome's process sub-model is well learned and does not involve calling too many processes. Thus, DISTDET can detect the anomalous behavior of Chrome by using the process sub-model.

4.1.3 ASG Generation

For an alarm, DISTDET generates an ASG that contains the anomalous event and its contextual events. As shown in the provenance graph-based approaches [4, 10, 13, 14, 16, 18, 29, 49, 51], processes spawn child processes to work together for certain system tasks, such as file compression and web downloads. To capture such critical information, for the subject process p in an alarm event evt (referred to as an alarm process), DISTDET first builds a process lineage tree of p and identify its ancestors within X generations and the descendants within Y generations. Then DISTDET includes the first N outgoing edges of each type (process, file and network) of these identified nodes (i.e., their initiated events) to form an ASG. The selection of optimal values of X, Y and N is presented in Appendix § C.1. Figure 4 shows an example process lineage tree for an alarm process. To improve the efficiency in building process lineage trees, DISTDET caches the observed events from the event stream (see § A.2 in Appendix).

4.2 False Alarm Filtering

While host-based anomaly detection incurs low overhead in detecting potential attacks, it suffers from the alarm fatigue problem. Thus, DISTDET includes the false alarm filtering component that takes three steps to overcome the problem.

4.2.1 Alarm Deduplication

Alarm deduplication is based on the insight that a considerable number of false alarms represent the same behaviors. For

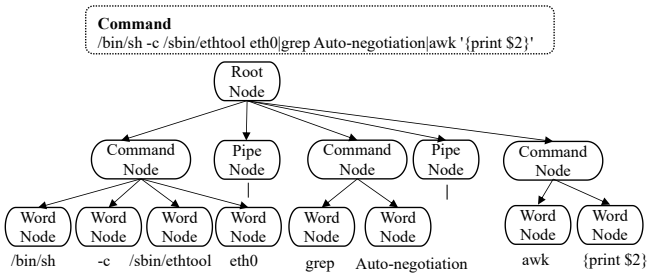


Figure 5: Syntactic tree of a command in an alarm event

example, an unscheduled upgrade operation will repetitively add and delete numerous files in a short period of time, resulting in a series of false alarms. To eliminate such duplicate alarms, DISTDET maintains a time window for each alarm a and all the subsequent alarms whose alarm events are the same as a's will be discarded. At the end of a time window, the frequency of the duplicate alarm events will be also recorded in the ASG generated for a. In this way, a sudden surge of recurring alarms will only have a limited impact. Based on our pilot studies, we set the time window to 24 hours.

4.2.2 ASG Semantic Aggregation

Besides the duplicate alarms for the same behavior, we also observed that a large number of false alarms are related to similar executed commands, with only differences in some arguments or operation objects. Thus, DISTDET aggregates semantically similar alarms to further reduce the reported alarms. Specifically, this process consists of three steps: (1) DISTDET constructs a syntactic tree of a command by tokenizing the words in the command. Figure 5 shows an example syntactic tree for the command `</bin/sh -c /sbin/ethtool eth0|grep Auto-negotiation|awk 'print $2'>`. (2) DISTDET computes the similarity between two command trees. Following the previous study [31], the similarity of two trees is measured by how many common sub-fragments they have. Specifically, we consider two commands to be similar if their similarity exceeds a threshold τ_s. (3) DISTDET aggregates the ASGs with similar alarm events, and records the frequency of the aggregated ASG. In this way, the redundancy of generated alarms/ASGs can be further reduced. Note that ASG aggregation is performed on the server side,

4.2.3 ASG Ranking

The final step in false alarm filtering is to prioritize the alarms by considering their rareness (frequency) as well as whether their contextual events (i.e., ancestor processes and descendent processes in the ASG) are also anomalous. We observed that many false alarms had contextual events that were generally known to represent benign behaviors. Thus, we design a ranking algorithm that considers the anomaly scores of alarm events and their ancestors/descendants.

Given an ASG, there are two types of events, i.e., *normal* and *alarm*. For normal events, the event frequency is recorded when DISTDET trains the host model. For alarm events, the event frequency refers to the number of repetitions recorded in the alarm deduplication process (§ 4.2.1). Suppose there is an ASG containing x alarm events and y normal events. For alarm events, since they contribute more to the anomaly of the ASG, the anomaly score is calculated as follows:

$$AS(evt^a) = \frac{\sum_{i=1}^{x} \frac{1}{freq(evt_i^a)}}{x} \quad (1)$$

where $freq$ indicates the frequency of each alarm event. For normal events, the anomaly score is calculated as follows:

$$AS(evt) = \frac{\sum_{j=1}^{y} evt_j}{y}, evt_j = \begin{cases} 1 & freq(evt_j) < freq_{ave} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

where $freq(evt_j)$ indicates the frequency of the normal event evt_j in the host model, and $freq_{ave}$ indicates the average frequency of the normal events of the same type. For example, consider an event <*java, /bin/bash, process_start*>. We first get $freq(evt_j)$ by the frequency of this event recorded in the model. Then we focus on the events that have happened in the model where *Process Layer* $\in$ *java* and *Operation Layer* $\in$ *process_start* and calculate their average frequency to get $freq_{ave}$. If $freq(evt_j) < freq_{ave}$ we consider this event contributing somewhat to the anomaly score (added to the numerator), otherwise we ignore it. This is to prevent the high-frequent normal events from having a large impact on raising the anomaly score. The anomaly score $AS(ASG)$ is calculated as follows:

$$AS(ASG) = \alpha AS(evt^a) + (1-\alpha) AS(evt), \alpha = \begin{cases} 1-\alpha_f & x = 1 \\ \alpha_f & x > 1 \end{cases} \quad (3)$$

where α_f is the weight that has different values depending on the number of alarm events. The underlying intuition is that when x is 1, the anomaly score is obtained mainly using the normal events; when x is greater than 1, the alarm events are considered more important for the overall anomaly. α_f is set to 0.9 through experiments in Appendix § C.1 Note that for the attacks that occur quite a lot in a short period of time (e.g., ransomware and brutal force), we handle them by checking the frequency of suspicious behaviors (e.g., 100 times in 1 second), without relying on the anomaly scores.

We rank all ASGs and calculate a cut-off threshold τ_d to differentiate false alarms and real attack incidents. If the anomaly score of an ASG is larger than τ_d, it is categorized as a true alarm otherwise a false alarm. The threshold value is related to the current enterprise configuration, such as the number of hosts and system monitoring events.

4.3 Global Model Derivation

A global model built in the server can complement the host model built in each client, especially for clients that provide the same type of services such as web servers or database servers. For example, suppose a Mangodb cluster contains 2 hosts h_a and h_b that provide the storage services, and they both have the same set of weekly scheduled tasks. During their training periods, h_a captures these tasks in its host model, but h_b misses these tasks as they happen to be scheduled outside the training period. DISTDET addresses this local bias by deriving a global model from both h_a and h_b and distributes the global model to update the host models in h_a and h_b.

More specifically, this process consists of four steps:

1) DISTDET extracts the list of service-specific processes from each host model. The process nodes in the host models are classified into two categories, i.e., system process and service-specific process. The system processes come with the OS installation, which are the same in all the hosts with the same type of OS, such as `/init` and `/usr/bin` for Linux. In contrast, service-specific processes are the processes of the software applications, and the combination of these processes can indicate the type of services the host provides. We obtained the list of system processes by summarizing the system processes of different operating systems. Thus, each model can derive a list of service-specific processes by excluding the system processes.
2) For the extracted service-specific processes in a host, DISTDET computes the word embeddings of the extracted processes' names, such as $< p_1, p_2, ..., p_{10} >$ of the model-1 in Figure 6. DISTDET uses word2vec [8], a popular pre-trained word embedding model, to compute the embeddings of the process names for each host.
3) DISTDET clusters the models based on their embedding vectors using the k-means algorithm. In this way, the hosts that provide the same type of services are grouped in the same clusters. The value of k is set based on the current enterprise setting, which is 146 in our evaluation (see § 5.4).
4) DISTDET then merges the behaviors of the common processes in the same cluster. As shown in Figure 6, model-1, model-2, and model-7 are in the same cluster and they share the common processes p_1 and p_2. DISTDET merges the behaviors of p_1 and p_2 in these three models to obtain p_1' and p_2', and then replaces p_1 and p_2 in the local model with p_1' and p_2' to obtain the global models, see § A.3 in Appendix for details. The global models are then distributed to each host to update the host local models.

5 Evaluation

We built DISTDET (~20K lines of code in C++) upon an industrial EDR tool of a leading security company and deployed it in the company's industry arena for adversarial engagement (1,130 hosts). We then performed real attack campaigns in the

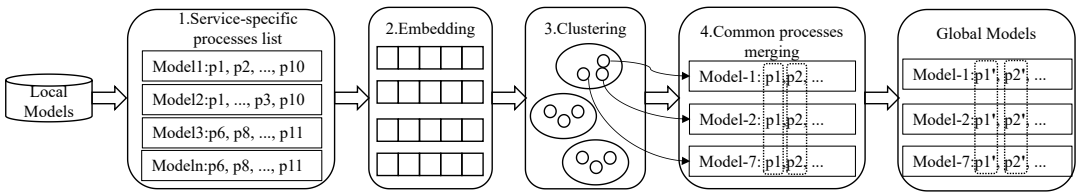


Figure 6: Global Model Derivation

deployed environment to collect the ground truth dataset for evaluation. Due to the intolerable overhead incurred by the centralized architecture for 1,000+ hosts, we cannot compare the detection effectiveness with the state-of-the-art techniques on the industry dataset. Thus, we resort to the DARPA TC dataset [38] for comparing DISTDET with the state-of-the-art techniques. In particular, our evaluations aim to address the following research questions:

RQ1. How much can DISTDET reduce the costs, compared to the existing centralized detection approaches?

RQ2. What is the overall effectiveness of DISTDET?

RQ3. How effective is DISTDET in filtering false alarms?

RQ4. How well does DISTDET perform when conducting investigation on ASGs?

RQ5. How well does DISTDET perform when it is deployed in real-world industrial environments?

5.1 Evaluation Setup

5.1.1 Industry Arena Dataset

DISTDET is deployed in a leading security company's industry arena, which includes 1,130 hosts (168 windows hosts and 962 Linux hosts) and a virtual machine server (16 CPUs, 32GB RAM, and 1TB hard disk) running 64bit CentOS 7.4. We deployed the client side components of DISTDET to these hosts and the server side component to a server that manages the cluster of hosts. We conducted a 14-day long adversarial engagement to evaluate the costs and the effectiveness of DISTDET. In total, the collected system auditing logs contain ∼1.6 billion system events (1.63TB), as shown in Table 3 (row-1). Column "EventRate" shows the number of captured system events per second (eps) and Column "Event Size" indicates the average size of captured events. We can see that the event rate of our collected dataset is much lower than DARPA's, which owes to the data deduplication [52] performed by our auditing tool. We further present the distribution of the types of events in Appendix § B.1. A professional red team from the security company performed four distinct attacks, including the well-known APT29 [43] and the APT32 [44] attacks, and two attacks that exploit the recent vulnerabilities (e.g., Log4j2 and Weblogic) on the hosts running the vulnerable (CVE-2021-44228 [33]) version of Apache Log4j2 [11] and the vulnerable (CVE-2020-14645 [32]) version of Oracle WebLogic [3]. We show the mappings between the attack steps (e.g., A1 indicates a step of the APT29 attack) and the stages (e.g., "initial access") in the MITRE ATT&CK matrix [45] in Table 4. The 4 attacks (116 attack steps) cover all the 12 attack stages. Note that all hosts have completed the host model training before the attacks.

Table 3: Overview of the evaluation datasets

Dataset	Host Num	Days	Data Size	Event Num	Event Rate	Event Size
Industry Arena	1,130	14	1.63 TB	1.6B	1.189 eps	1100 Byte
DARPA-CADETS	1	11	39G B	41 M	43 eps	1013 Byte
DARPA-THEIA	1	11	80 GB	106 M	111 eps	810 Byte
Public Arena	2	6	7GB	16M	15 eps	483 Byte

5.1.2 Darpa TC Dataset

The Darpa TC dataset was released by the DARPA TC program, generated during a red-team vs. blue-team adversarial engagement in April 2018 [38]. We used the attack datasets from the THEIA and CADETS teams in the DARPA TC program. THEIA is a system that tags and tracks multi-level host events and data by instrumenting the Ubuntu Linux machines during the engagement. The CADETS data is derived from the FreeBSD DTrace data. Table 3 and 5 shows the details of these datasets, including six attack campaigns, where CADETS 1-4 are four repeated attacks, and THEIA 5-6 are two two consecutive attacks. In total, the DARPA TC datasets contain ∼147 million system events (119GB) from 2 hosts in 11 days. We convert the DARPA datasets to the compatible data formats for DISTDET to analyze, and use the pre-attack data for training the host models.

5.1.3 Public Arena Dataset

One recent work Log2vec [12] converts users' behavior logs (i.e., user logs) into a heterogeneous graph and detects APT attacks by discovering anomaly nodes in the graph. The system audit logs in the Industry Arena and Darpa TC datasets cannot be applied to Log2vec due to lack of user and login information. Therefore, to compare the performances of Log2vec and DISTDET, we further craft the Public Arena dataset, which contains logs in both system audit logs and user logs generated during attacks. Table 3 and 5 show the details of the Public Arena dataset, including two attacks, where `Insider` [5] is a common threat that steals sensitive information, and `Lateral` [5] is a typical lateral movement attack where APT actors move from host A to host B. The attacks were conducted by the company's red team on 2 hosts (windows) located in the public cloud workspace. In total, the Public Arena dataset contains ∼16 million system audit logs (7GB) and 682 user logs (91KB) from 2 hosts in 6 days. We

Table 4: Mappings between the attack steps and the stages in the MITRE ATT&CK matrix

Attack Campaign	Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Command & Control	Exfiltration	Impact	Total
A-APT29	A1	A2-A6	A7-A11	A12-A15	A16-A23	A24	A25-A31	A32	A33	A34-A38	-	-	38
B-APT32	B1	B2-B8	B9-B11	B12,B13	B14-B31	B32-B34	B35-B44	B45,B46	B47	B48-B51	B52	B53	53
C-Log4j2	C1	C2	C3	-	-	-	C4-C7	C8-C10	-	-	-	-	10
D-Weblogic	D1	D2	D3	-	-	-	D4-D7	D8-D10	-	-	-	-	10
Total	4	14	10	6	26	4	25	9	2	9	1	1	111
E-Insider	E1	-	-	-	E8	-	E2-E4	-	E5	-	E6-E7	-	8
F-Lateral	F1	F2	F5	-	F3-4,F18	F12	F6-11,F13-14	F15-17	-	F19	F20	-	20

Table 5: Statistics of ground truth for attack campaigns

Dataset	Attack Campaign	Host Num	Host Type	Duration	# of Events	# of Attack Events	Attack ASG
Industry Arena	A-APT29	2	Windows	1d0h7m	358K	489	45
	B-APT32	2	Windows	1d2h13m	355K	423	68
	C-Log4j2	2	Linux	1d14h43m	459K	220	12
	D-weblogic	1	Linux	0d0h12m	1.7M	268	14
DARPA TC	Cadets 1-4	1	FreeBSD	10d23h28m	41M	2313	24
	Theia 5-6	1	Linux	9d20h6m	106M	1266	15
Public Arena	E-Insider	1	Windows	0d11h44m	1.9M	1508	8
	F-Lateral	2	Windows	0d2h45m	783K	4415	23

make the Public Arena dataset publicly available [5] to enable reproducible study and facilitate further research.

5.1.4 Labeling Ground Truth

We identify the attack events as the ground truths in the Industry Arena, Darpa TC and Public Arena datasets. To do so, we developed scripts to flag attack events in the raw logs. Our scripts take as input the config files that contain the key attributes used in each attack, such as timestamps, host IDs, host IPs, process commands, process file paths, and match the corresponding attack events in the raw logs accordingly. The configuration files for the APT attacks were generated by the red team during the attacks, and the configuration files for the Darpa TC dataset were generated from their ground truth files. We consider a detection result of DISTDET as a true positive if the ASG contains a ground truth attack event.

Table 5 shows the details of the ground truth. The number of attack events is much larger than the generated ASGs. This is because our ASGs aggregates duplicate events generated by some attack steps. For example, the C9 step in the C-Log4j2 attack (network scan) generates 180+ attack events. For the Public Arena dataset, we marked 21 malicious user logs, including 11 for the E-Insider and 10 for the F-Lateral.

5.1.5 Evaluation Metrics

Cost Metrics. Since few existing works have focused on evaluating the cost of attack detection, we propose a new metric for evaluating the cost of APT detection in a large-scale cluster of hosts, which is based on the actual cost spent by the customers when using the security products. In general, the total cost has a positive correlation with the number of clusters that need to be protected and can be divided into three components: *network*, *computation* and *storage*. Among them, the network cost is mainly the bandwidth required for data transmission from the clients to the server. EDR products such as CrowdStrike are delivered in the form of SaaS model, where all clients transfer telemetry data to a server deployed in the public cloud, and the network accounts for a major portion in the total cost. The computational cost is mainly the memory usage in the detection process. The storage cost is mainly for the persistence of the raw data (the size of the detection results is almost negligible compared to raw data). Although real-time detection can be done without persisting the data, the subsequent attack investigation and responses require the raw data. Thus, the cost of data persistence should take into consideration. We measure the host cost as follows:

$$HostCost = Cost_N * \alpha_n + Cost_C * \alpha_c + Cost_S * \alpha_s \quad (4)$$

where $Cost_N$ denotes the network bandwidth per host, $Cost_C$ denotes the memory footprint per host for data processing, $Cost_S$ denotes the average storage size per host for data persistence, and α_n, α_c, α_s denote the price indexes of network bandwidth (yearly), memory, and storage respectively. More specifically, $Cost_N$ is computed using the average number of transmissions per host per second. The amount of transmissions varies between the centralized (total size of events) architecture and the distributed architecture (total size of ASGs and host models). $Cost_C$ is computed using the memory footprint in the server for the centralized architecture, while it includes the memory usage of both clients and the server for the distributed architecture. $Cost_S$ is computed using the average daily raw data size generated by a single host for the centralized architecture, while it uses the size of the event cache data in the clients for the distributed architecture. *Note that these cost metrics are summarized based on industrial security products, which directly affects the enterprises' security budgets and can represent the real-world usages.*

Effectiveness Metrics. We use precision and recall to evaluate the overall effectiveness of DISTDET.

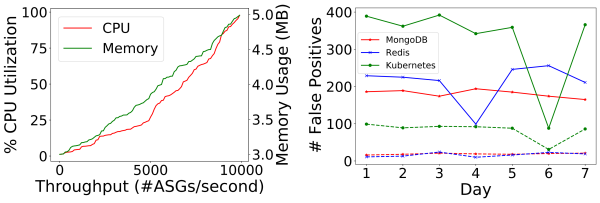
5.2 RQ1: Cost Reduction by DISTDET

We compare the costs of DISTDET and Unicorn [16] (i.e., one of the most representative centralized architecture) on the Industry Arena dataset. We implemented Unicorn to consume the event logs from real-world hosts and used the optimal configuration from the original paper.

Storage Cost ($Cost_S$). For DISTDET, $Cost_S$ includes event caching and models generated by all hosts within 14 days, with a total of 143.83GB, of which the models occupy 220.7MB. The average model size is ∼200KB, which is related to the type of application running on the host. $Cost_S$ of

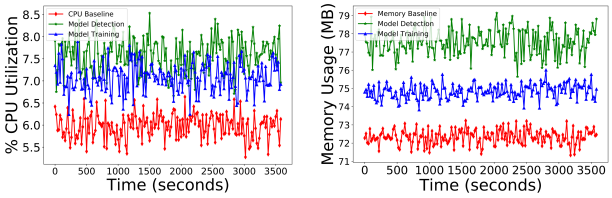
Table 6: Cost comparison between the distributed architecture (DISTDET) and the centralized architecture (Unicorn)

Architecture	$Cost_N$	$Cost_S$	$Cost_C$	Host Cost	Cluster Totalcost
Distributed (DISTDET)	15.47b/s (cluster:17.08Kb/s)	130.34MB (cluster:143.83GB)	client side: 5.52MB server side: 3KB Total: 5.523MB (cluster:6GB)	Net: 0.00016USD Storage: 0.013USD Computing: 0.048USD Total: 0.061 USD	68.93 USD
Centralized (Unicorn)	10.22kb/s (cluster:11.28Mb/s)	1.47GB (cluster:1.63TB)	364MB (cluster:401GB)	Net: 0.107USD Storage: 0.146USD Computing: 3.15USD Total: 3.4 USD	3842 USD



(a) Throughput (server side). (b) False alarms reported by the global (dotted) /local models (solid).

Figure 7: Throughput of DISTDET (server side) and effectiveness of global model.



(a) CPU utilization (client side) (b) Memory usage (client side)

Figure 8: Client side Computational overhead of DISTDET

the centralized architecture is the raw event data size generated by all hosts within 14 days, which is 1.63TB. DISTDET reduced the storage cost from 1.47GB/host (1.63TB/1130) to 130.34MB/host (143.83GB/1130), a reduction of 11.6×.

Network Cost ($Cost_N$). For DISTDET, all hosts transferred 2.462GB of data (i.e., ASGs and models) in 14 days, with an average bandwidth of 17.08Kb/s (2.462GB/14days). For the centralized architecture, all hosts transmitted 1.63TB of data to the server in 14 days, with an average bandwidth of 11.28Mb/s (1.63TB/14days). As shown in Table 6, compared with UNICORN, DISTDET reduces the bandwidth from the average 11.28Mb/s to 17.08Kb/s (676.5× reduction).

Computational Cost ($Cost_C$). We measure the CPU and memory usages of DISTDET on both the clients and the server. Figure 8 shows the CPU utilization as well as the memory footprints of DISTDET on the clients, where the red line represents the baseline consumption of the EDR process without DISTDET deployed, and the blue and green lines represent the consumption of the EDR process with DISTDET deployed during model training and model detection, respectively. During the model training period, DISTDET only builds HST, the average CPU utilization and memory footprint of the EDR process are 6.63% and 74.89MB respectively. Compared with the baseline (CPU: 5.63%, memory: 71.42MB), the average overhead of CPU and memory are 1% and 3.47MB, respectively. During the detection period, the costs include HST detection, event caching, ASG generation and alarm deduplication. DISTDET introduces an average overhead of 1.225% and 5.52MB for CPU and memory, respectively. *This incurs negligible impacts on the daily business of the clients.*

Figure 7(a) shows the CPU and memory utilization on the server side. When the number of ASGs fed to the server reaches 10,000 per second, the CPU utilization reaches ∼98%, while the memory usage is only around 5MB. This means that DISTDET can process at most ∼10,000 ASGs per second in this setting. In total, the 1,130 hosts have a throughput of 0.063 ASGs/s with the original memory usage of 3.2MB and each host has a memory footprint of only 3KB (3.2MB/1130) on the server side (Table 6). In addition, DISTDET takes 3.8 minutes and an average of 12.05MB of memory to derive the global models for all 1130 hosts. Since the global models only need to be derived once and it takes less time, we ignore its computational cost. For the centralized architecture, the average memory usage of Unicorn running on the server side is 364MB/host (the server memory is not enough to run all host data). DISTDET reduced the average computational cost from 364MB/host to 5.523MB/host (5.52MB for the client side and 3KB for the server side, i.e., a 66× reduction).

Overall Cost Reduction. We acquired the typical prices of enterprise network bandwidth, server storage, and server memory from a well-known cellular network provider [42] and Dell [22, 23], through which we derived the values of the price indexes α_n, α_s and α_c. On average, DISTDET reduces the host cost (the expense of securing a single host) from 3.4 USD to 0.061 USD (56× reduction). Thus, to protect the entire cluster of 1,130 hosts, DISTDET costs only 68.93 USD compared to 3,842 USD for the centralized system. Besides, the money spent is only the cost that we can quantify, and there are other hidden costs such as business impairment and technical barriers. For example, when employing the centralized system, using 11.28Mb/s network bandwidth for security detection can severely impact other daily businesses. On the other hand, since the server side requires 401GB of memory to detect APT attacks in real time, an additional cluster with such scale needs to be set up. These obstacles greatly limit the real-world usage of existing centralized approaches.

Table 7: Detection results on the evaluation datasets. The results of UNICORN are from [16].

Dataset	Precision	Recall	Accuracy	F1
Industry Arena (DISTDET)	0.99	1.0	0.99	0.99
DARPA CADETS&THEIA (DISTDET)	0.98	1.0	0.98	0.98
DARPA CADETS&THEIA (DeepLog)	0.14	0.02	0.45	0.04
DARPA CADETS (UNICORN)	0.98	1.0	0.99	0.99
DARPA THEIA (UNICORN)	1.0	1.0	1.0	1.0
Public Arena (DISTDET)	0.88	0.9	0.83	0.89
Public Arena (Log2vec)	0.53	0.43	0.97	0.47

5.3 RQ2: Effectiveness of DISTDET

5.3.1 Effectiveness on the Industry Arena Dataset

Table 7 (row1) shows the overall results. In total, DISTDET reported 141 attack ASGs out of 11K ASGs, of which 2 were false positives (FPs). Since there are 139 labelled attack ASGs, DISTDET managed to miss no true attacks, achieving a recall of 100% and a precision of 99%. In particular, we found that FPs were caused by the following two reasons: (1) *Rare benign behaviors*. The `Windows Explorer.EXE` started an unexpected theme update, resulting in the file creation of `\\Local\\ Microsoft\\ Windows\\ ActionCenterCache\\xx.png`. The related events were recorded with low frequencies in the model, causing a false alarm. (2) *Missing logs in the collection*. Due to the missing logs in the collection, a scheduled task `apt-daily.service` was flagged as an attack, which is a service that regularly refreshes the list of available packages in the Ubuntu server. This indicates that the completeness of the data collection and the frequencies of the normal edges in the model play important roles in the precision of DISTDET. In practice, we can collect log data as completely as possible and reinforce the models to ensure the detection effectiveness. Further, we show that DISTDET is able to restore the attack chain in § C.2.

5.3.2 Comparison with State-of-the-Art

We compare DISTDET with the state-of-the-art anomaly detection systems including Unicorn, DeepLog [30] and Log2vec [12]. We contacted the authors of Log2vec to get the prototype code, and obtained the source code of Unicorn [2] and DeepLog [46] from GitHub. Due to the memory limitation, the state-of-the-art systems (Unicorn, Deeplog and Log2vec require 364MB, 7.5GB and 10.63GB memory per host respectively) cannot handle all the 1,130 hosts in the Industry Arena dataset. Therefore, we compare DISTDET with Unicorn and DeepLog on the Darpa TC dataset. Since Log2vec cannot handle system audit logs, we compare its performance with DISTDET on the Public Arena dataset.

DISTDET vs. Unicorn As shown in Table 7, DISTDET achieves almost the same precision and recall with Unicorn on the DARPA TC dataset. Yet, DISTDET outperforms Unicorn in terms of interpretability and granularity of the results. DISTDET uses an ASG, which records the attack nodes and the contextual events, as the detection result. This provides better interpretability and can effectively facilitate the subsequent investigation by the security analysts. Unicorn uses the entire provenance graph that tracks the system state transitions as the detection result. Security analysts have to find the attack nodes from the whole graph for follow-up investigation and reaction, which requires a lot more manual effort.

DISTDET vs. DeepLog As shown in Table 7, the performance of DeepLog on DARPA TC dataset is quite poor (with F1 of 0.04). The reason is that the natural language sequence model used by DeepLog is difficult to distinguish between attack logs and benign behavior logs. First, many attack logs have no casual and sequential relationships with normal behavior logs. For example, the first attempt to exploit Nginx server in the DARPA CADETS 1 campaign generated a network activity log <Nginx, connect, source IP, destination IP>. This attack step could be initiated by the attacker at any time and had no sequential relationship with other benign behaviors. Secondly, unlike the system logs used for system diagnosis in DeepLog experiment, which have many types of logs, system audit logs only have three types of logs: process, network, and file. The sequence models constructed with only these three types of logs cannot achieve satisfactory performances.

DISTDET vs. Log2vec Table 7 (row 6-7) shows that although Log2vec can achieve a high accuracy, its recall is quite low. Thus, DISTDET outperforms Log2vec greatly, with recall of 89% and 47%, respectively. There are mainly two reasons. First, Log2vec only leverages login, file and web network logs to detect attacks, which cannot detect attacks captured in other important logs (e.g., process logs). Second, Log2vec requires long-term user logs to construct logical relationships among days (516 days in [12]), which cannot be easily adopted in the industry settings. Note that, DISTDET reported three false negatives in the Public Arena dataset, all of which are caused by insider attacks (e.g., the attacker logged in host B from host A using `mstsc`, and the user of host A also did this during the training period). These missed behaviors can be restored during the investigation phase using cached data on the client side, which will be described in § 5.5.

5.4 RQ3: False Alarm Filtering

As aforementioned, DISTDET performs false alarm filtering at multiple stages. On the one hand, the detection by the comprehensive global model can reduce the false positives caused by local bias. On the other hand, DISTDET deals with alarm fatigue through client side alarm deduplication and server side semantic alarm aggregation. We next measure the effectiveness of false alarm filtering for each stage.

Effectiveness of Global Model. As DISTDET builds a global model for hosts with the same type of service, we first group the hosts into different service clusters. Among 1,130 hosts, 25 hosts have < 5 processes listed in their host models, which were new or template hosts that do not actually conduct any business. Thus, we clustered the remaining 1,105 hosts us-

ing the method described in Section 4.3. In total, we get a total of 146 clusters with an average of 7.57 hosts per cluster, see Appendix § C.1 for details. Overall, for different clusters, the global model can reduce false positives by an average of around 70%. Particularly, we select the `MongoDB, Redis and Kubernetes` clusters, which provide the most representative services for the enterprises, as case studies to show the effectiveness of the global model. DISTDET is applied to train a local model for each host in the `MongoDB, Redis and Kubernetes` clusters, and derived global models based on the local models. We then deployed the global model to each host along with its local model to perform a week-long detection. We measured the alarms reported by both the models in each cluster. As we did not perform any attack, any alarm reported is a false positive. Figure 7(b) shows the results. The number of false positives generated by the global models (dotted lines) is much less than that of the local models (solid lines). When relying on only the local models for detection, the average number of false positives generated per host is 240, while the average number reported by the global models is 39. *This shows that the global model can effectively complement the host models to eliminate local bias.*

Effectiveness of Alarm Deduplication. In our two-week experiment, 1,130 hosts generated a total of ∼1 million alarms, with an average of 69 alarms per host per day. This is a labor-intensive task for security analysts to handle these alarms. Using a sliding time window of 24 hours for deduplication, the number of alarms is reduced to ∼76,000 (4.8 alarms/host/-day). This indicates that DISTDET reduced 93.03% of the duplicate alarms in the clients with a small overhead, saving a lot of network bandwidth.

Effectiveness of Alarm Semantic Aggregation. After semantic aggregation, 11,242 ASGs (0.71 alarms/host/day) were remained out of the ∼76,000 ASGs with the similarity threhold τ_s of 0.85 (see Appendix § C.1 for details). This shows that semantic aggregation can further eliminate 85.21% of the alarms. In addition, two experienced security analysts manually investigated 50 samples each day from the filtered alarms (700 alarms in total), and no attacks were found.

In summary, after the above steps, DISTDET *reduces the total number of alarms from 3.6 million (230 alarms/host/day) to 11k (0.71 alarms/host/day), saving 99.69% of the security analysts' labor cost without filtering true positives.*

5.5 RQ4: Investigation with DISTDET

DISTDET persists data on the client through event caching (see Appendix A.2), so that analysts can obtain the missing provenance graph data from the cache on demand. We measure the effectiveness of log reduction in the event caching and the performance of investigation using cached data.

Effectiveness of Log Reduction. We compare DISTDET with the state-of-the-art system audit log reduction approaches, i.e., NodeMerge [41] and Full Dependence (FD) [21], on the Darpa TC dataset. We reproduced NodeMerge to measure its effectiveness, and the result of FD was from [21]. Table 8 compares the log size reduction ratio (i.e., ratio of the data size before and after the reduction) of these approaches, and Table 12 in Appendix § C.4 presents the detailed results. The average reduction ratio of DISTDET (1.33) is somewhat less than NodeMerge (3.37) and FD (4.8). This is expected, because DISTDET mainly compresses repeated long strings in node attributes, most (96.1%) of which are *null* in the Darpa TC dataset. NodeMerge merges read-only file nodes, while FD mainly merges similar edges between node pairs. Since these nodes and edges have a higher proportion in the Darpa TC dataset, they achieve better reduction results. More importantly, these approaches compress different parts of the log data, they are complementary to each other and can be applied together. Furthermore, DISTDET only consumes 2MB of memory to caches all data in real time, strictly conforming to the overhead constraint on the client. As a comparison, due to the complex compression algorithms, NodeMerge (928.61MB) and FD (290MB) are only suitable for offline analysis and may be applied once a day or a week.

Table 8: Log reduction on the DARPA TC dataset.

Tools	NodeMerge (Raw Data)	FD (without CSR)	DISTDET (Raw Data)
Reduction	3.37X	4.8X	1.33X
Memory	928.61MB	290MB	2MB

Table 9: The distribution of attacks DISTDET identified.

Attack Type	Percentage(%)	Attack Type	Percentage(%)
Cryptomining	27	Ransomware	20
Infectious Malware	6	Macro virus	2
Worm	7	Rootkit	7
Brute Force	2	Other	29

Effectiveness of Investigation. We perform causality analysis and connect the same nodes (processes, files, or networks) in different ASGs in chronological order to restore the attack scenario graph for each campaign. When ASGs fail to connect, we request the missing association nodes from the cached data on the client. We use F-Lateral as the case study to show the investigation process in Appendix § C.5. In total, through on-demand investigation, we find all the missed attack ASGs, indicating that DISTDET preserves all the information that is necessary to perform an attack investigation. Table 6 shows that it only needs 130MB storage to cache the industry data generated in 14 days (9MB/day/host) on average, and 1.6GB for 0.5 years.

5.6 RQ5: Application of DISTDET in the Wild

By the time of this writing, DISTDET has been beta deployed to 50+ industry customers for more than 6 months, covering 22,000+ hosts in total. The customers cover different industries such as finance, retail, and healthcare providers, and the customer clusters cover various common business types

such as databases, apps, and electronic transactions. In these customer clusters, DISTDET generated an average of 4,400 alarms per day (0.2 alarm/host/day) after ASG semantic aggregation and about 90 alarms per day were flagged as attack ASGs by ASG ranking. The daily false positive rate fluctuates in the range of 1-10%. In practice, we had 5-6 full time security operators handling the alarm data with a processing efficiency of 800 alarms per person per day. Overall, 900+ real attacks were found during roughly 6 months (5 attack campaigns per day on average). Table 9 shows the distribution of these 900+ attacks, with ransomware and cryptomining account for roughly half of the attacks.

The industry customers have also deployed other EDR products on their clients, while we observed that only 25% of the attacks can be flagged by them. DISTDET has exhibited superior performance over existing EDR products mainly in two aspects: (1) *the ability to detect unknown attacks*. DISTDET uses a fine-grained anomaly detection model such that it is capable of detecting attacks that cannot be flagged by existing rule-based EDR products. (2) *The ability to overcome alarm fatigue*. Compared with ordinary EDR products that generate 700 alarms per host per day, the alarm rate of DISTDET in customer clusters is only 0.2 alarms per host per day. This allows security operators to respond more quickly. For example, in an anonymized company, 4 hosts were infected with the mining virus `mkatz.ini` [47]. Our security operators quickly noticed the mining behavior from the 16 alarms generated in 3 days by these 4 hosts with DISTDET deployed. Actually, the attack is also detected by other rule-based detection tools (i.e., we further checked all the alarms generated by other rule-based detection tools after we identifying these attacks). However, due to the overwhelming number of false alarms (10,000+ alarms) generated, the attack is missed by security operators. This demonstrates DISTDET's ability to address alarm fatigue in the real-world settings.

6 Discussion

Evasion Attacks. Adversaries may launch mimicry attacks [36, 39] that camouflage attacks as benign behaviors to evade detection. However, conducting mimicry attacks on provenance graph based models is more challenging than on small sequence of events, as it is difficult to launch an entire attack campaign using all normal behaviors. Moreover, unlike Unicorn [16] and Provdetector [49] which use graph level and path level data for modeling, DISTDET uses event-based HST to build a fine-grained model, making the evasion more costly, i.e., attackers have to keep the most fine-grained attributes of the attack events consistent with those in the model.

Model Building and Updating. Some hosts may exhibit behaviors that make it difficult for DISTDET's host models to converge, e.g., public Internet hosts have random network access. While this will compromise the capabilities of DISTDET in detecting anomalies in network accesses, DISTDET still can protect the host by using other sub-models (i.e., process models and file models). Besides, unlike existing modeling methods that require a lot of effort to retrain the models after deployment, DISTDET's host models can be easily updated based on the manually identified false alarms.

Limitations. DISTDET shares assumptions and limitations with other anomaly-based detection systems: (1) *Attack noise in the training period*: Our host models can be contaminated when attack behaviors occur during the training period. In a live industrial setup, we use our accumulated attack knowledge such as heuristic rules to minimize the impact of attacks in the training period. The model will be uploaded to the cloud to find out whether it is polluted, and mitigate the polluted part. (2) *Poor generalization*: Our HST models in the clients are efficient to compute but have limited generalization to other types of benign behaviors. In future work, we plan to leverage the cloud to train behavior models based on business needs by using more computational expensive techniques, such as graph embeddings and CNN, and distribute this model as the baseline model to complement the HST models.

7 Related Work

Causality Analysis via System Audit Logs. Causality analysis based on system auditing logs aims to automatically reconstruct a series of events that represent attack steps [24]. As causality analysis suffers from the dependency explosion problem, recent efforts have been made to mitigate the dependency explosion problem by performing fine-grained causality analysis [25, 27], prioritizing dependencies [18, 26], customized kernel [6], and optimizing storage [21, 41, 52]. Causality analysis can work with DISTDET to build a more detailed provenance graph upon our ASGs to reveal more contextual information of the attacks.

Anomaly Detection. Recent log anomaly detection systems [30, 50] usually convert logs into sequences and then employ different machine learning methods to detect abnormal logs in sequences. However, these systems do not consider the logical relationships among the nodes of provenance graphs, which is the key to represent the contextual information of APT attacks. Thus, they cannot achieve satisfactory performances and are more often used for system diagnosis. Furthermore, as described in § 5.3.2, machine learning methods require a large computational cost to train models (DeepLog requires 7.5GB per host), which significantly limits their adoption in the industrial settings. DISTDET addresses this fundamental limitation by employing a novel distributed detection architecture to minimize the overall cost by shifting part of the APT detection to the clients and collects summarized graphs (i.e., ASGs) instead of all logs.

False Alarm Filtering. Existing alert correlation and deduplication systems [17, 29, 48, 53] perform alert deduplication or correlation based on alert properties, whose performances are not robust as they are specific to their domains and heavily

rely on their alert detection systems. Different from these works, DISTDET adopts a novel four-step approach (global model, alert deduplication, alert semantic aggregation and alert ranking) that integrates monitoring and detection with alarm filtering and provides novel techniques in every step to achieve better filtering performance. Also, none of the existing works leverage global models of process behaviors to filter alarms. For example, NoDoze [18] is an alert ranking technique that assigns an anomaly score based on the frequency to combat threat alert fatigue produced by the rule-based host IDPS. However, NoDoze is actually a centralized system that requires all logs being sent to the server for training, which is impractical in the industry settings, especially when considering that the filtering effectiveness of NoDoze is only around 84%. The 1,130 hosts in the Industry Arena generate 260K alarms every day, and security analysts need to handle 42K alarms every day after using NoDoze (VS. 800 alarms reported by DISTDET).

8 Conclusion

We propose DISTDET, a cost-effective detection system that detects APT attacks through distributed computing, anomaly detection, and false alarm filtering. In particular, DISTDET adopts a novel distributed detection architecture to minimize the overall cost by shifting part of the APT detection to the clients and transmitting only summary graphs that represent potential attacks to the server, which greatly reduces the costs of data transmission and storage. Our evaluations on real industry data show that DISTDET effectively reduces the detection cost by 56 times, while maintaining a comparable detection accuracy as the state-of-the-art techniques.

Acknowledgement

This work was supported in part by National Key R&D Program of China (2021YFB2701000), the National Natural Science Foundation of China (grants No.62072046 and 62172009), ZDSY20200811143600002, the Fundamental Research Funds for the Central Universities (HUST 3004129109), and HKPolyU Grant (ZVG0).

References

[1] Yahoo discloses hack of 1 billion accounts, 2016. https://techcrunch.com/2016/12/14/yahoo-discloses-hack-of-1-billion-accounts/.

[2] Open-source implementation of Unicorn, 2020. https://github.com/crimson-unicorn.

[3] Oracle weblogic, 2022. https://www.oracle.com/java/weblogic.

[4] ALSAHEEL, A., NAN, Y., MA, S., YU, L., WALKUP, G., CELIK, Z. B., ZHANG, X., AND XU, D. ATLAS: A sequence-based learning approach for attack investigation. In *Proceedings of the USENIX Security Symposium* (Aug. 2021).

[5] ANONYMOUS. The Public Arena dataset, 2022. https://github.com/security0528/PublicArena.

[6] BATES, A. M., TIAN, D., BUTLER, K. R. B., AND MOYER, T. Trustworthy whole-system provenance for the linux kernel. In *Proceedings of the USENIX Security Symposium* (2015), pp. 319–334.

[7] CALIŃSKI, T., AND JA, H. A dendrite method for cluster analysis. *Communications in Statistics - Theory and Methods 3* (01 1974), 1–27.

[8] CHURCH, K. W. Word2vec. *Natural Language Engineering 23*, 1 (2017), 155–162.

[9] CLEARNETWORK, I. Crowdstrike lightweight agent, 2021. https://www.clearnetwork.com/managed-crowdstrike/.

[10] FANG, P., GAO, P., LIU, C., AYDAY, E., JEE, K., WANG, T., YE, Y., LIU, Z., AND XIAO, X. Back-propagating system dependency impact for attack investigation. In *Proceedings of the USENIX Security Symposium* (2022).

[11] FOUNDATION, T. A. S. Apache log4j2, 2022. https://logging.apache.org/log4j/2.x/.

[12] FUCHENG, L., YU, W., DONGXUE, Z., XIHE, J., XINYU, X., AND DAN, M. Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise. In *CCS* (2019), ACM, pp. 1777–1794.

[13] GAO, P., XIAO, X. S., LI, D., LI, Z. C., JEE, K. K., WU, Z. Y., KIM, C. H., KULKARNI, S. R., AND MITTAL, P. SAQL: A stream-based query system for real-time abnormal system behavior detection. In *Proceedings of the USENIX Security Symposium* (2018), pp. 639–656.

[14] GAO, P., XIAO, X. S., LI, Z. C., XU, F. Y., KULKARNI, S. R., AND MITTAL, P. AIQL: Enabling efficient attack investigation from system monitoring data. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC)* (2018), pp. 113–125.

[15] GOOGLE. Understand tasks and back stack, 2020. https://developer.android.com/guide/components/activities/tasks-and-back-stack.

[16] HAN, X., PASQUIER, T., BATES, A., MICKENS, J., AND SELTZER, M. Unicorn: Runtime provenance-based detector for advanced persistent threats. *NDSS* (2020).

[17] HASSAN, W. U., BATES, A., AND MARINO, D. Tactical provenance analysis for endpoint detection and response systems. In *2020 IEEE Symposium on Security and Privacy (SP)* (2020), IEEE, pp. 1172–1189.

[18] HASSAN, W. U., GUO, S., LI, D., CHEN, Z., JEE, K., LI, Z., AND BATES, A. Nodoze: Combatting threat alert fatigue with automated provenance triage. In *NDSS* (2019).

[19] HOSSAIN, M. N., MILAJERDI, S. M., WANG, J., ESHETE, B., GJOMEMO, R., SEKAR, R., STOLLER, S., AND VENKATAKRISHNAN, V. {SLEUTH}: Real-time attack scenario reconstruction from {COTS} audit data. In *USENIX Security* (2017), pp. 487–504.

[20] HOSSAIN, M. N., SHEIKHI, S., AND SEKAR, R. Combating dependence explosion in forensic analysis using alternative tag propagation semantics. In *2020 IEEE Symposium on Security and Privacy (SP)* (2020), IEEE, pp. 1139–1155.

[21] HOSSAIN, M. N., WANG, J. A., SEKAR, R., AND STOLLER, S. D. Dependence-preserving data compaction for scalable forensic analysis. In *Proceedings of the USENIX Security Symposium* (2018), pp. 1723–1740.

[22] JD.COM. Dell (dell) server hard disk enterprise-class mechanical hard disk 4t sas 7200 rpm 3.5-inch merchant warehouse, 2021. https://item.jd.com/25662718500.html#none.

[23] JD.COM. Dell (dell) server/workstation memory 32g ddr4 recc 2933/3200 memory, 2021. https://item.jd.com/13490989528.html.

[24] KING, S. T., MAO, Z. M., LUCCHETTI, D. G., AND CHEN, P. M. Enriching intrusion alerts through multi-host causality. In *NDSS* (2005).

[25] LEE, K. H., ZHANG, X., AND XU, D. High accuracy attack provenance via binary-based execution partition. In *NDSS* (2013).

[26] LIU, Y., ZHANG, M., LI, D., JEE, K., LI, Z., WU, Z., RHEE, J., AND MITTAL, P. Towards a timely causality analysis for enterprise security. In *NDSS* (2018).

[27] MA, S., ZHANG, X., AND XU, D. Protracer: towards practical provenance tracing by alternating between logging and tainting.

[28] MILAJERDI, S. M., ESHETE, B., GJOMEMO, R., AND VENKATAKRISHNAN, V. Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting. In *CCS* (2019), pp. 1795–1812.

[29] MILAJERDI, S. M., GJOMEMO, R., ESHETE, B., SEKAR, R., AND VENKATAKRISHNAN, V. HOLMES: real-time APT detection through correlation of suspicious information flows. In *IEEE S&P* (2019), IEEE, pp. 1137–1152.

[30] MIN, D., FEIFEI, L., GUINENG, Z., AND VIVEK, S. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)* (2017), ACM, pp. 1285–1298.

[31] MOSCHITTI, A. Efficient convolution kernels for dependency and constituent syntactic trees. *Springer, Berlin, Heidelberg* (2006).

[32] NATIONAL VULNERABILITY DATABASE. Cve-2020-14645 detail, 2022. https://nvd.nist.gov/vuln/detail/CVE-2020-14645.

[33] NATIONAL VULNERABILITY DATABASE. Cve-2021-44228 detail, 2022. https://nvd.nist.gov/vuln/detail/CVE-2021-44228.

[34] NPR. Home Depot Confirms Data Breach At U.S., Canadian Stores, 2014. http://www.npr.org/2014/09/09/347007380/home-depot-confirms-data-breach-at-u-s-canadian-stores.

[35] PACCAGNELLA, R., DATTA, P., HASSAN, W. U., BATES, A., FLETCHER, C., MILLER, A., AND TIAN, D. Custos: Practical tamper-evident auditing of operating systems using trusted execution. In *NDSS* (2020).

[36] PARAMPALLI, C., SEKAR, R., AND JOHNSON, R. A practical mimicry attack against powerful system-call monitors. In *Acm Symposium on Information* (2008), p. 156.

[37] PINNA, E., AND CARDACI, A. Gtfobins is a curated list of unix binaries that can be used to bypass local security restrictions in misconfigured systems, 2022. https://gtfobins.github.io/.

[38] PROGRAM, D. T. C. The DARPA Transparent Computing (TC) program Data Release, 2022. https://github.com/darpa-i2o/Transparent-Computing.

[39] PROJECT, L. Living off the land binaries and scripts (and now also libraries), 2022. https://github.com/LOLBAS-Project/LOLBAS.

[40] SCHWARTZ, S. Security accounts for just 5.7 https://www.cybersecuritydive.com/news/security-budget-gartner/587911/.

[41] TANG, Y., LI, D., LI, Z., ZHANG, M., JEE, K., XIAO, X., WU, Z., RHEE, J., XU, F., AND LI, Q. Nodemerge: Template based efficient data reduction for big-data causality analysis. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)* (2018).

[42] TELECOM, C. China telecom - business line, 2021. https://www.ct10000.sh.cn/product.php?item=020&_ref=nav.

[43] THE MITRE CORPORATION. Apt29 detail, 2022. https://attack.mitre.org/groups/G0016/.

[44] THE MITRE CORPORATION. Apt32 detail, 2022. https://attack.mitre.org/groups/G0050/.

[45] THE MITRE CORPORATION. Mitre att&ck matrix for enterprise, 2022. https://attack.mitre.org/matrices/enterprise/.

[46] THIJSVANEDE. PyTorch implementation of Deeplog, 2021. https://github.com/Thijsvanede/DeepLog.

[47] TRENDMICRO. Trojan.Win32.MIMIKATZ.ADU, 2019. https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/Trojan.Win32.MIMIKATZ.ADU.

[48] VALEUR, F., VIGNA, G., KRUEGEL, C., AND KEMMERER, R. A. Comprehensive approach to intrusion detection alert correlation. *IEEE TDSC 1*, 3 (2004), 146–169.

[49] 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).

[50] XIA, B., YIN, J., XU, J., AND LI, Y. Loggan: a sequence-based generative adversarial network for anomaly detection based on system logs. In *International Conference on Science of Cyber Security* (2019), Springer, pp. 61–76.

[51] XU, Z., FANG, P., LIU, C., XIAO, X., WEN, Y., AND MENG, D. Graph summarization on system audit logs for attack investigation. In *IEEE S&P* (2022).

[52] XU, Z., WU, Z., LI, Z., JEE, K., RHEE, J., XIAO, X., XU, F., WANG, H., AND JIANG, G. High fidelity data reduction for big data security dependency analyses. In *CCS* (2016), pp. 504–516.

[53] ZHAI, Y., NING, P., AND XU, J. Integrating ids alert correlation and os-level dependency tracking. In *International Conference on Intelligence and Security Informatics* (2006), Springer, pp. 272–284.

Appendices

Appendix A Design of DISTDET

A.1 Tokenization of the HST

Table 10 shows representative types of tokenization.

Table 10: Representative types of tokenization

Type	Original command/filepath/ip	Tokenized command/filepath/ip
hash	docker inspect 5facb37bd928	docker inspect xxxxxxxxxxxx
pid	taskset -cp 0-2 30110	taskset -cp 0-2 xxxxx
number	date -d 1073876 second ago +%s	date -d xxxxxxx second ago +%s
ip	192.168.28.37	192.168.28.x
filename	/net/cali2e2859502e8	/net/calixxxxxxxxxx

A.2 Event Caching

The goal of Event Caching is to cache the event stream for generating ASGs, which requires forward and backward tracing of the alarms. However, the biggest challenge is how to achieve caching of event streams on the limited storage resources of client. To this end, we use a lightweight disk-based database *sqlite* to cache data. The central idea is to minimize the storage space for duplicate strings. By analyzing a large amount of event stream data, we found that the most repeated strings in events are *command* and *file paths*. For this, we use a multi-level hashing mechanism to reduce the storage space for these long strings. Figure 9 illustrates the Event caching method. We first perform the primary hash, which extracts the duplicate long commands and file paths from the events and stores them in the form of hash index, generating a command hash table and a file hash table respectively. After that, there are still a lot of duplicate shared long paths in both tables. Thus we dump the hash of the duplicate long paths

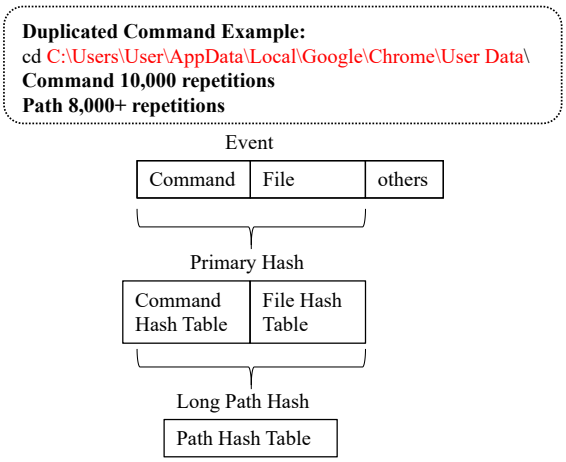


Figure 9: Event Caching

from the two tables into a path hash table. Here, a duplicate string is deemed to be a long path if it has more bits than the hash value, which is 19 bits in this work. As shown in Figure 9, the example command is repeated 1,000+ times in the host, and the path in the command is repeated 8,000+ times. Through two-level hash, a total of 18,000+ repeated storage elements are saved. The results in Section 5.2 show that our event caching method losslessly caches one day's data in 9.31MB (130.34MB/14day) on average, with an average compression rate of 10 times. The memory footprint in the agent is extremely small.

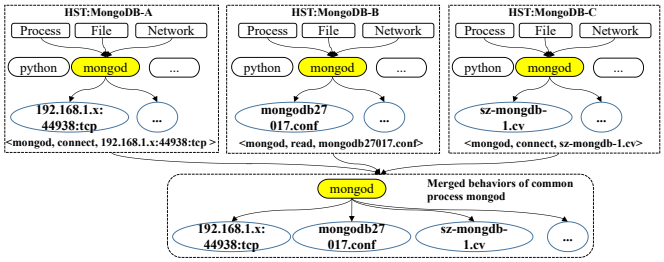


Figure 10: Case study of Global Model Derivation

A.3 Case study of Global Model Derivation

Figure 10 shows how the behaviors of the common process, `mongod`, are merged in the MongoDB cluster. MongoDB-A, MongoDB-B and MongoDB-C have unique events `<mongod, connect, 192.168.1.x: 44938: tcp>`, `<mongod, read, mongodb 27017.conf>` and `<mongod, connect, sz-mongdb-1.cv>` respectively. DistDet merges these unique events into the global model and updates the host HSTs correspondingly. DistDet then repeats to merge all common processes in the cluster. In the experiment, we limit the maximum number of leaf nodes per process node to 500 (configurable) to mitigate the leaf node explosion in the model. Since the tokenization heuristics cover most of the attribute changes, no over-limited nodes are found in the 1,132 host models in the Industry Arena dataset. If the number of leaf nodes exceeds the maximum value, the excess leaf nodes will generate false alarms because they are not learned into the model. Operators will discover these special cases when analyzing the models and add corresponding tokenization heuristics to mitigate the problem.

Appendix B Dataset Details

B.1 Dataset Event Type

Table 11 shows the proportion of event types in the datasets.

Table 11: Dataset Event Type

Dataset	Process	File	Network	Other
Industry	19.45%	8.06%	36.22%	36.27%
Darpa	5.10%	43.70%	7.40%	43.70%
Public	0.21%	86.86%	8.53%	4.40%

B.2 Attack Steps of the F-Lateral Campaign

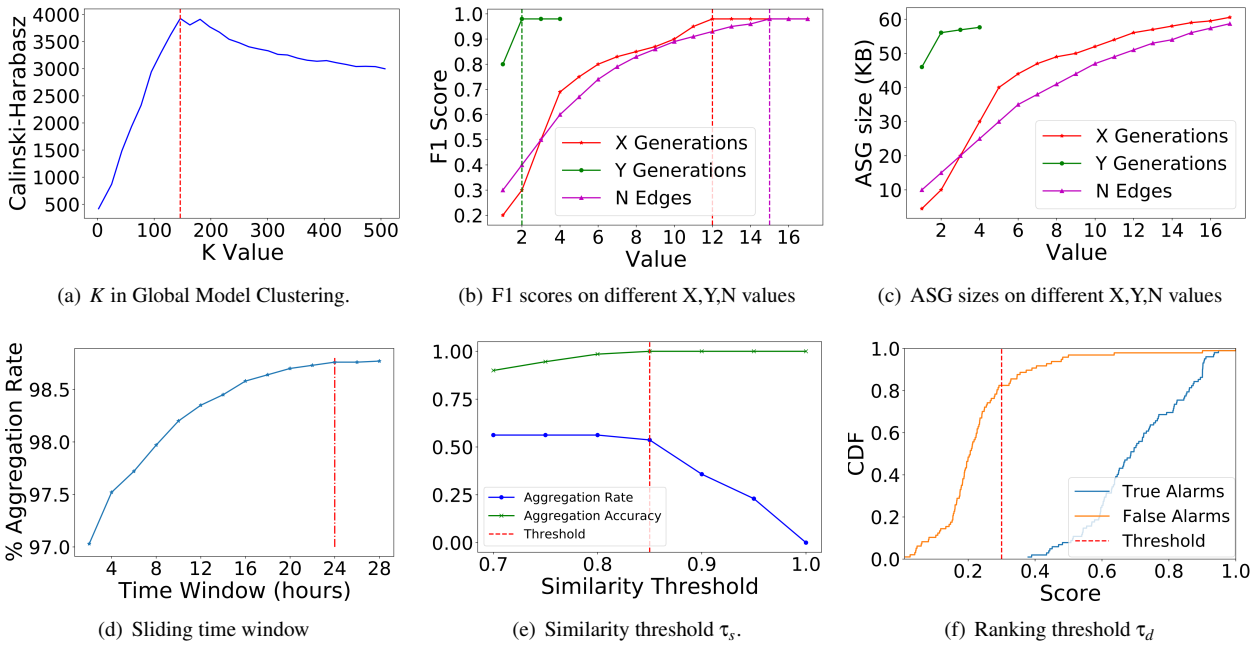
The APT actor compromises the target host A, moves laterally to host B and steals important data on host B. **F1:** Brute force cracking into the target host A; **F2:** Use the `WMI` tool to query host information; **F3:** Disable security tools; **F4:** Disable `windows defender`; **F5:** Add scheduled tasks; **F6:** Get a list of accounts in host A; **F7:** Gather information about domain trust relationships; **F8:** Enumerate files and directories; **F9:** Get process information; **F10:** Obtain a list of other systems by IP address, host name, or other logical identifier on the network that can be used to move laterally from host A; **F11:** Get detailed information about the operating system and the hardware, including versions, patches, service packs, and architectures; **F12:** Execute `mimikatz` to get a ticket for lateral movement; **F13:** Execute `WMIexec` to obtain the password of target host B; **F14:** Execute `SNETCracker` to brute force other hosts on the intranet; **F15:** Execute `paexec` to move laterally to host B, and execute the `ipconfig` command; **F16:** Download the remote control Trojan twice through `paexec`, but the Trojan cannot be started; **F17:** Execute `mstsc` to move laterally to host B; **F18:** Disable `windows defender` on host B; **F19:** Execute `CMD` on host B, download the remote control Trojan, and run successfully; **F20:** Steal files through the remote control Trojan;

Appendix C Experiment Results

C.1 Finding the Optimal Parameter Values

Figure 11 shows how the key parameters influence the performance on detecting attacks in the Industry Arena dataset.

***K* in Global Model Clustering.** We compare the results of clustering using different K values, ranging from 2 to 500.



(a) K in Global Model Clustering. (b) F1 scores on different X,Y,N values (c) ASG sizes on different X,Y,N values

(d) Sliding time window (e) Similarity threshold τ_s. (f) Ranking threshold τ_d

Figure 11: Performance of DISTDET on different parameters.

In Figure 11(a), when the value of K is 146, the Calinski-Harabasz index [7] that measures the K-Means clustering effect reaches the maximum value, and the clustering achieves the best performance.

Parameters in ASG Generation (X, Y, N). We employ F1 score and ASG size to measure the effectiveness and cost of DISTDET under different X, Y, N values. As shown in Figure 11(b) and Figure 11(c), as the parameters of X generations ancestors, Y generations descendants and first N outgoing edges increase, the number of context nodes in ASGs increases, which improves the ranking results of ASGs. But it consumes more memory and transmission bandwidth. When the value of X, Y and N exceeds 12, 2 and 15, respectively, we find that the newly added context nodes have no significant impact on the F1 score, but instead increase the ASG sizes. Thus, we adopt the ancestors within 12 generations and the descendants within 2 generations. For each identified process node, we use the first 15 outgoing nodes of each type (if available).

Sliding Time Window. For the optimal value of the sliding time window, we measured the aggregation rate by varying the value of the time window, as shown in Figure 11(d). As the time window is increased to more than 24 hours, the increases of the aggregation rate become marginal. We thus set the sliding time window to 24 hours.

Similarity Threshold τ_s. We selected 313 alarms as the test set, including 176 alarms with similar semantics (22 groups of different types, 8 in each group) and 137 alarms with dissimilar semantics. We calculated the aggregation rate (similar alarms/total alarms) and aggregation accuracy by varying τ_s. Figure 11(e) shows that the larger the threshold τ_s, the higher the aggregation accuracy and the lower the aggregation rate. When the threshold τ_s is lower than 0.85, the aggregation accuracy rate is lower than 100%, which will introduce false positives, and when it is higher than 0.85, the aggregation rate is significantly lower than 56.23% (maximum aggregation rate). Therefore, the semantic aggregation was performed on the server with the similarity threshold τ_s of 0.85.

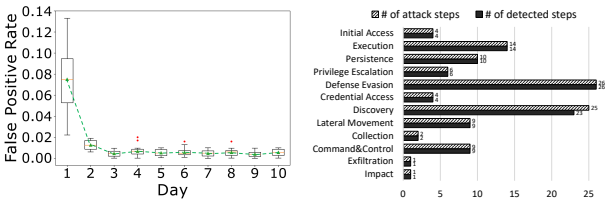
Weighting Factor α_f and Ranking Threshold τ_d. To determine the optimal threshold value, we first created a tested dataset leveraging some of the most commonly used attack techniques including redis 0day vulnerability, anti-virus webshell, reverse shell. Overall, our tested dataset covered 28 hosts, which generated 52 attack ASGs (true alarms) and 87 normal ASGs (false alarms) in total (labeled as described in §5.1.4). We then calculated the anomaly score for each of these ASGs using different τ_d values. The results show that, when the weighting factor α_f is less than 0.9, the anomaly scores of attack ASGs and normal ASGs are crossed and indistinguishable. Therefore, $\alpha_f = 0.9$ is used in our evaluations. Figure 11(f) shows the distribution of true alarms and false alarms. We can see a significant difference in the distribution of anomaly scores for the false alarms and the true alarms. Thus, we set the ranking threshold τ_d to 0.3 (shown as the red line), where the majority portion (more than 80%) of the false alarms can be filtered.

C.2 Detection Coverage

We further use the *detection coverage* that is often used in security companies and the MITRE ATT&CK [45] to measure the ability of restoring the entire attack campaign. We

Table 12: Log reduction details on the Darpa TC dataset

Darpa dataset	NodeMerge		DISTDET			
	Read-only files count(%)	Reduction Factor	Total Path count	Long Path count(%)	Null Path count(%)	Reduction Factor
Cadets	23.4M(69.1%)	4.58	82.7M	4.5M(5%)	66M(79.8%)	1.21
Theia	13M(66.8%)	2.89	226.6M	0(0%)	226.6M(100%)	1.55
Trace	2.7M(34.5%)	3.45	63.3M	0(0%)	63.3M(100%)	1.33
FiveDirections	34.5M(23.5%)	2.81	47.8M	0(0%)	47.8M(100%)	1.26
Total	73.6M(35.4%)	3.37	420.4M	4.5M(1.1%)	403.8M(96.1%)	1.33



(a) False positives reported by the host models over time (box plot).

(b) Detection results of each attack step.

Figure 12: Convergence of host models and detection coverage of attacks.

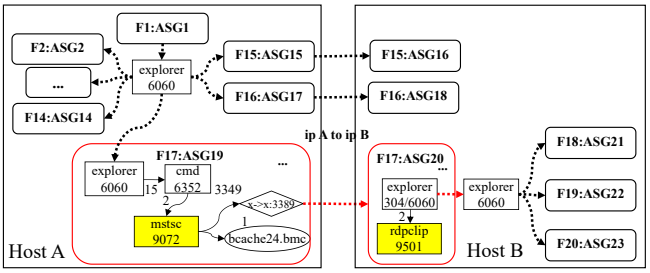


Figure 13: Investigation process of the F-Lateral campaign

compute the *detection coverage* of the attack campaign, i.e., the ratio of detected attack steps to the total number of steps in a given attack campaign. Figure 12(b) shows the results for each attack. We can see that the detection results cover all the attack steps in the MITRE ATT&CK matrix, indicating that the anomaly detection capability of DISTDET is independent of the exploit types and attack techniques adopted by the attackers. Overall, 109 (98%) of the 111 attack steps in the four attacks were detected. The two missed steps are the C5 step of C-Log4j2 and the D6 step of D-Weblogic. Manual analysis revealed that both steps were missed due to the data collection problem.

C.3 Convergence of Host Models

To understand whether and how soon our local models can reach convergence, we measure the number of false alarms reported by the host models over time. Specifically, we randomly selected 50 hosts that provide representative business services, including test servers, mail servers, DNS servers, SOAPA servers, yum repositories, Desktop, etc. For each host, we started the model training, saved a model every day, and obtained 10 models after 10 days. Then each of the ten models was deployed on the host for detection. Figure 12(a) shows the results. We use the evolution of the false positive rate (FPR) over time to measure whether the model achieves convergence. We can see that the FPRs decrease quickly in the first three days, and the FPRs of most hosts drop below 1% after three days of training. We can also notice that there are some fluctuations of the FPRs in some hosts (i.e., isolated nodes on the box plot), which are mainly caused by scheduled tasks and unexpected maintenance tasks performed by the users. To mitigate these issues, domain knowledge such as whitelists can be further adopted to reduce the false alarms.

C.4 Log reduction on the Darpa TC dataset

Table 12 shows the number and proportion of reduction objects for NodeMerge and DISTDET in the Darpa TC dataset.

C.5 Case Study of Investigation

Figure 13 shows the investigation process of the F-Lateral campaign, where the nodes fetched from the client are marked with red boxes. Since the attack steps are all conducted through the user interface (UI), the ancestor of most of the alarm processes in ASGs is `explorer`, which is the Windows UI management process. ASG1-18 and ASG21-23 are connected into two subgraphs by the `explorer` on host A and host B, respectively, lacking associated nodes in the middle. Therefore, we request the `explorer` descendant nodes that occurred between ASG18 and ASG21 from the host A client side, and find the `mstsc` (the APT actor moved from host A to host B) process and related events, thus obtaining ASG19. Similarly, we find `rdplcip` (host B accepts host A's login) process related events from the nodes requested by host B client side, and get ASG20. The ASGs (ASG15-20) across hosts are all connected using the network events of Host A and Host B. As a result, the scenario graph of the F-Lateral is completely restored. We share the rest attack graphs along with our Public Arena Dataset [5] to the community.