Paradise: Real-time, Generalized, and Distributed Provenance-Based Intrusion Detection

Yafeng Wu, Yulai Xie, Member, IEEE, Xuelong Liao, Pan Zhou, Senior Member, IEEE
Dan Feng, Member, IEEE, Lin Wu, Xuan Li, Avani Wildani, Darrell Long, Fellow, IEEE

Abstract—Identifying intrusion from massive and multi-source logs accurately and in real-time presents challenges for today's users. This paper presents Paradise, a real-time, generalized, and distributed provenance-based intrusion detection method. Paradise introduces a novel extract strategy to prune and extract process feature vectors from provenance dependencies at the system log level, and it stores them in high-efficiency memory databases. Using this strategy, Paradise does not depend on the specific operating system type or provenance collection framework. Provenance-based dependencies are calculated independently during the detection phase, thus, Paradise can negotiate all detection results from multiple detectors without extra communication overhead between detectors. Paradise also employs an efficient load-balanced distribution scheme that enhances the Kafka architecture to efficiently distribute provenance graph feature vectors to the detectors. The experimental results demonstrate that our method has a high detection accuracy with a low time overhead.

Index Terms—APT, Intrusion detection, Provenance, Real-time

1 Introduction

The problem of enterprises and network users being exposed to sophisticated attacks is now becoming more serious. For example, APT (Advanced Persistent Threat) attacks have become mainstream [1], causing huge economic losses to enterprises. However, intrusions are often buried in a wide range of average daily behaviors, thus making them very difficult to detect (*needle in a haystack*).

Mainstream methods analyze causal dependencies through a syscall audit log in order to explore the path from the intrusion source to the infected file. Those methods also improve detection accuracy through log clustering [2] and partitioning [3] techniques as much as possible. However, these methods are offline and cannot satisfy real-time requirements. Typical APT attacks may remain latent inside the enterprise for half a year or more. By the time they are located, confidential documents in the enterprise may have been stolen or modified.

Researchers have proposed provenance-based IDSs (Intrusion Detection Systems) [4–7] which use in-memory databases (e.g., Redis [8]) or multi-threading technology [9] to perform online analysis of anomalies within the provenance path or graph, greatly improving detection accuracy and reducing detection time. These IDSs are based on provenance frameworks (e.g., SPADE [10], PASS [11], Hi-Fi [12], and CamFlow [13]), which provide powerful security and integrity for information flow capture [14].

However, with the *high concurrency of events* and *complex internal dependencies*, provenance-based IDSs may fail when facing a series of challenges in big data environments:

- *Interoperability*: The designers of these provenance collection systems [10–13] have made different choices regarding what activity to record and how to represent it. Although some of these systems use standards such as W3C PROV [15] (e.g., CamFlow) or Open Provenance Model [16, 17] (e.g., SPADE) to establish a public shared interface for provenance data, the provenance graphs collected under different standards for the same event may be different. For example, there is almost no consensus on how a specific activity should be represented in the provenance graphs of the different models in Figure 1. In addition, different methods work at different system layers (kernel space and user space), so some information may not be available for a given IDS [18].
- *Real-time performance*: Although some studies have proposed methods [7, 19–21] for obtaining better detection accuracy, their algorithms require an entire provenance

- *Yafeng Wu, Yulai Xie (Corresponding author), and Lin Wu are with Hubei Engineering Research Center on Big Data Security, School of Cyber Science and Engineering, Wuhan National Laboratory for Optoelectronics, Key Laboratory of Information Storage, Huazhong University of Science and Technology, Wuhan 430074, P.R. China. E-mail: {yafengwu, ylxie, linwu}@hust.edu.cn*
- *Pan Zhou is with Hubei Engineering Research Center on Big Data Security, School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan 430074, P.R. China. E-mail: panzhou@hust.edu.cn*
- *Xuelong Liao and Dan Feng are with the School of Science and Technology, Wuhan National Laboratory for Optoelectronics, Key Laboratory of Information Storage, Huazhong University of Science and Technology, Wuhan 430074, P.R. China. E-mail: cserlxl@gmail.com, dfeng@hust.edu.cn*
- *Xuan Li is with the NSFOCUS Inc. A9-3/F Optical Valley Software Park, Wuhan, China. Email:lixuan2@nsfocus.com*
- *Avani Wildani is with Dept. of Computer Science Emory University 400 Dowman Dr., W401 Atlanta, GA 30322 USA. Email: avani@mathcs.emory.edu*
- *Darrell Long is with Jack Baskin School of Engineering, University of California, Santa Cruz, CA 95064 USA. Email: darrell@ucsc.edu*

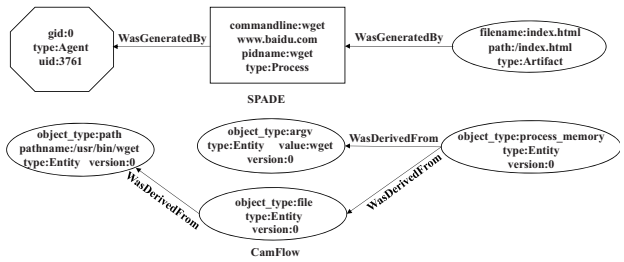

Fig. 1: A *wget* system call, as recorded by SPADE and CamFlow. This figure clearly illustrates nontrivial structural differences in how *wget* is represented as dependencies between processes (rectangles), artifacts or resources (ovals), and an agent (octagon).

graph over a period of time as input in order to extract the features of the entire graph. The investigation of APT attacks usually requires querying and comparing massive amounts of data in the real world, leading to poor real-time performance.

We propose Paradise, a real-time, generalized, and distributed provenance-based intrusion detection method, to solve these challenges. Paradise consists of three steps: a collecting and preprocessing step, a data distribution step, and an intrusion detection step. First, the collecting and preprocessing step collects and stores the provenance of objects (e.g., files, processes, and agents) in a memory database (e.g., Redis) in order to speed up the query and detection. It also filters out some unnecessary provenance information (e.g., temporary files or system configuration files) defined as unrelated to intrusion, in order to improve detection accuracy [7]. Then, it extracts process dependencies from provenance to build process feature vectors, which are the calling frequencies of the process in different types. In addition, it includes an adaptive monitoring and cluster load balancing mechanism in the data distribution step that achieves efficient distribution and parallel detection. Paradise exploits the different kinds of process dependencies in the provenance graphs employed in the intrusion detection step to enable efficient intrusion detection. At the same time, Paradise adds a flag to the vector to both indicate whether the vector belongs to a subgraph or a whole provenance graph and label the timestamp. Finally, to discover the attack story, Paradise negotiates all detection results of subgraphs from multiple detectors.

Paradise can be applied for intrusion detection in different provenance models (e.g., W3C PROV [15] or OPM [16, 17]) without extra data conversion based on the proposed feature vector extraction strategy. It is independent of the host operating system and provenance collection system, and it does not require prior program instrumentation.

Our contributions are as follows

- We propose Paradise, a real-time, generalized, and distributed provenance-based intrusion detection system that considers the feature vector of vertex *neighbor graph (see 2.4)* to accurately identify anomalies in massive provenance data in big data environments.
- We present a novel and simple strategy to adaptively convert collected provenance into feature vectors independent of the underlying provenance-collecting platform (e.g., Linux, Windows) and different provenance models.
- We design an adaptive monitoring and cluster load balancing mechanism to achieve effective data distribution and parallel detection. This scheme automatically achieves load balancing for different detectors by monitoring cluster performance metrics, such as CPU usage and memory usage rate, and automatically scheduling the data.
- We evaluate Paradise on a series of real-world normal and vulnerable applications. Paradise significantly outperforms FRAP [5], Unicorn [19], and Pagoda [7] in terms of detection rate, false alarm rate, and detection time.

The rest of this paper is organized as follows. We introduce the background and motivation of our work in section 2. Methodology and specific implementation are discussed in Section 3. Section 4 describes the experiments and results of Paradise. Section 5 surveys the related work from three aspects: host-based intrusion detection, forensic analysis, and distributed intrusion detection. Finally, we discuss the limitations and improvements of our work in Section 6 and conclude our work in Section 7.

2 Background and Motivation

We first describe the provenance and provenance models with a motivating example, then introduce the Kafka architecture [22] and state the threat model and assumption.

2.1 Motivating Example

Consider the following attack scenario in Figure 2: The user uses *wget* to download and *tar* to decompress a malicious *phishing-emial.tar* which contains a piece of malicious code. The attacker also uses malicious code to obtain a reverse shell on the target system. After obtaining the shell, the attacker first tries to mimic the user's typical behavior (e.g., using *gcc* to compile the program and upload to Github), and get some sensitive files (e.g., *file A* and *file B*). Then he imitates normal behavior in order to avoid detection. He first merges these sensitive files into *file C*, copies the *file C* as the *file A.c*, and finally uses *gcc* to compile *file A.c* and upload it to Github. Note that the compilation of the *file A.c* was unsuccessful because *file A.c* is not a legal source file.

2.2 Provenance

Provenance describes the "reasons" (causal relationships) that lead to the occurrence of a "thing" (entity), that is, how an entity depends on others and how interactions between entities lead to specific states. Provenance can be understood as a record of the generation and history of system data [23].

Since provenance has been widely used in various fields (e.g., search [24] and security [25]), many provenance collection systems have been built. Typical systems include SPADE [10], TREC [26], PASS [11], LinFS [27], Hi-Fi [12], ZOOM [28], and CamFlow [13]. These systems collect provenance in different layers. For example, ZOOM collects

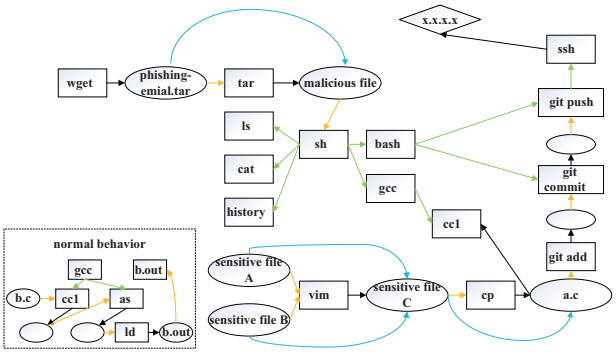

Fig. 2: The provenance graph for the motivation example. The dotted box is the normal behavior of the system, and the other parts are intrusion behaviors. For readability, we use the specifications defined by OPM (W3C PROV is also a good choice) to display part of the provenance graph. Nodes in the graph are system entities (rectangles are processes, ovals are files, agents are octagons and diamonds are sockets), and edges among nodes represent system calls. Different colored edges represent different types of dependencies defined by OPM.

provenance in the application layer. PASS mainly collects provenance in the OS (Operating System) layer. SPADE can collect provenance in both application and OS layers. In addition, SPADE can collect provenance in distributed environments, and PASS can be extended to network-attached [29] and cloud environments [30]. SPADE is a cross-platform provenance collection system that enables low latency and high availability in distributed environments. SPADE collects provenance information that conforms to the OPM provenance model [10]. It not only automatically generates and collects provenance, but also supports the manual import of provenance information. The provenance represents the dependency relationships between different objects. For example, the provenance record: *the process P creates the file A* indicates that the generation of the *file A* depends on the *process P*. The provenance data can be stored in different types of databases, so a graph database can be used to view the complete provenance graph and path visually.

Current provenance-based intrusion detection systems are designed based on different collection systems [10, 11, 13]. For instance, Pagoda [7] is built on the PASS [11] system, and FRAP [5] is built on the CamFlow [13] system. Because they may be built on different systems, enabling efficient provenance interoperability between different provenance systems is still a great challenge.

OPM [16] and W3C PROV [15] are designed for better provenance interoperability between different provenance systems. They capture the causal relationships between entities such as artifacts, processes, and agents. Later, a provenance graph is defined as a directed acyclic graph (DAG). We use the OPM model to describe our provenance methods because of its simplicity and clarity; nothing in our work explicitly depends on an OPM framing. Edges of the OPM provenance graph belong to one of the following five causal relationships:

1) **Used**: This relationship represents the use of a file for the execution of a process.
2) **WasGeneratedBy**: This relationship represents that the generation of a file depends on the process.
3) **WasTriggeredBy**: This relationship represents that a parent process generates a child process.
4) **WasControlledBy**: This relationship represents that the execution of a process P is controlled by an agent.
5) **WasDerivedFrom**: This relationship represents that a new file is derived from an old file.

The different colored edges in Figure 2 represent the above-mentioned different types of OPM relationships.

2.3 Challenges

When identifying the source of an attack or intrusion, the analyst needs to identify malicious behavior that has been detected so far, as well as malicious behavior that has not yet been exposed. Malicious events are usually buried in daily benign activities, and analysts often spend too much time investigating and identifying evidence-related benign events, leading to a high false alarm rate. Attack behavior can be represented as the abstraction of audit data. Abstracting high-level behavior from low-level audit events conforming to different provenance models, reducing the workload of analyzing the event behavior, and ensuring the accuracy of intrusion detection face the following challenges:

Generalized: Due to non-interoperability, different IDSs cannot adapt well to different collection frameworks. For example, Pagoda [7] is designed based on the PASS [11] and SPADE [10] systems, while Unicorn [19] is designed based on the CamFlow [13] system. The node of the W3C PROV model collected by CamFlow contains the *cf:secctx* attribute, used for the data histogram constructed by Unicorn. However, the node of the OPM model collected by SPADE, whose reporter is Linux Audit, does not contain this attribute. When the Unicorn detection method is applied to detect the provenance collected by other systems, many formatting conversions are required, which can easily lead to errors such as losing critical information, resulting in low detection efficiency and accuracy.

Distributed: Existing host-based provenance intrusion detection systems [6, 7, 31] typically use similarity-based detection methods and are easily affected when the length of a provenance sequence changes. Moreover, distributed provenance-based intrusion detection systems [19–22, 32] usually map the provenance graph into high-dimensional vectors, and then judge intrusion by the distance between those vectors. The IDSs mentioned above either take the complete provenance graphs as input, or divide them into subgraphs, and then compare the similarities between those subgraphs. Those IDSs usually regard obtaining subgraphs of provenance graphs as a community detection problem. However, as the provenance graph is a DAG, most of the data used in classic community detection algorithms lack the notions of *closeness* [33] or *importance* [34] between provenance entities and dependencies.

Real-time: The most state-of-art host-based provenance intrusion detection systems [6, 7, 19, 20, 32] usually consider the contextual knowledge of each vertex's multi-level ancestors and descendants. Although rich contextual semantic knowledge can ensure high detection accuracy, it inevitably

increases detection time due to traversing the same vertex or dependency several times. For example, Pagoda [7] needs to use a BFS or DFS algorithm to obtain each path in the provenance graph during detection. Unicorn [19] needs to obtain the label of each vertex to construct *graph histograms* before detection. This label describes the R-hop neighborhood of the vertex; building all vertex labels will also repeatedly traverse the node multiple times. More importantly, this problem is particularly prominent in those IDSs that handle many vertices and explosive dependency data, such as low and slow APT attack mode data.

2.4 Exploring process characteristics for intrusion detection

We have found that most edges connected with the process node represent the system's internal operations in Figure 2. Therefore, we are inspired to use process feature vectors to detect harmful behaviors of processes. As shown in Figure 2, we can easily see that the dependencies of the *process cc1* node in the intrusion provenance graph are significantly different from those in the normal graph. Thus, we can regard the process node, not the provenance subgraph or the whole provenance graph and path, as the smallest unit for intrusion detection. There are challenges in precisely extracting the process feature vector in real-time; the constructed feature vector, however, can be dynamically updated at a small cost. Inspired by reference [34], Paradise solves these problems by introducing the concept of the *neighbor graph*.

neighbor graph: If two nodes u and v are connected by an edge, they are called neighbors. The *neighbor graph* of node u is a subgraph G composed of u and its neighbor nodes directly connected to it.

For example, in Figure 2, a process *vim* first needs to read *file A* and *file B*, and then generates *file C*. The event provenance graph generated according to OPM will produce two paths, *file A* $\rightarrow$ *vim* $\rightarrow$ *file C*, and *file B* $\rightarrow$ *vim* $\rightarrow$ *file C*. Any single path cannot easily reveal how this event occurred. However, Paradise describes this event very clearly by exploring dependencies of the process *vim* and its *neighbor graph*.

Generalized: According to this process *neighbor graph*, Paradise constructs the feature vectors of the process nodes. Specifically, the dimension of the feature vector is the number of process relationship types in a specific provenance model, and the value is the number of each type of relationship (e.g., the feature vector of process *cc1* is [0,0,1,1] in Figure 2). Paradise does not require additional node attributes (e.g., *cf:secctx*) or edge attributes (e.g., *cf:jiffies*) for feature vector construction. Thus, provenance data with different models can be detected using the same strategy without format matching and conversions.

Distributed: These process feature vectors are independent of each other during detection, so the feature vectors are suited for distributed intrusion environments. Different detection machines in a distributed cluster do not communicate with one another when using those vectors for detection, thereby reducing network transmission overhead.

Real-time: Paradise constructs feature vectors based on neighbor graphs, avoiding traversing the same vertex multiple times, thereby reducing the detection time. Although Paradise sacrifices contextual semantic information to a certain extent, the experimental results on real-world applications demonstrate its detection accuracy (see Section 4.2).

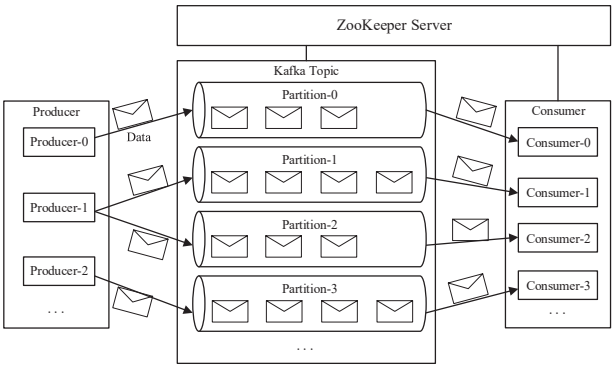

Fig. 3: Kafka framework.

2.5 Kafka for intrusion detection

Kafka is a distributed messaging system based on ZooKeeper [35] that supports partitioning and multiple replica mechanisms [36]. It can process massive amounts of data in real-time and is often used for log distribution. Thus, Kafka can serve as an intermediary and distribute messages from multiple hosts to different detectors quickly to satisfy real-time intrusion detection [22].

Figure 3 shows the basic architecture of Kafka. The producer posts a message to the specified *topic* of the Kafka server. A *topic* can be divided into multiple partitions. The consumer group consumes the message from a *topic*, while the different consumers consume different partitions. Kafka uses ZooKeeper distributed coordination services to ensure high availability, and ZooKeeper can judge which consumers are working at any time. Adding or deleting a consumer in a consumer group will trigger the partition reallocation mechanism. Kafka can realize the balanced distribution of messages to enable parallel detection, therefore improving detection efficiency through a rebalance mechanism. Thus it enables real-time intrusion detection on multiple hosts and enhances scalability.

Although Kafka can guarantee the high-speed distribution of information, it needs to set a static distribution strategy in advance. However, detection is a dynamic process, and detectors with the same hardware conditions may have different CPU and memory usage within a certain period. For example, the consumption rate of a heavily loaded machine may lag far behind the production speed. In this case, the detector may have an anomaly, causing the cluster detection time to be too long or even creating a denial of service. Therefore, extending the Kafka architecture to ensure the cluster's load balance during the dynamic detection process is still a significant challenge.

2.6 Threat Model and Assumptions

Our approach is designed with distributed clusters running a distributed application that has been replicated on many worker nodes. Workers can run as containers, virtual

machines, or bare metal hosts for provenance collecting. First, we assume that a modeling period exists during which system administrators can safely run their systems to capture regular system activity. Second, we assume a trusted computing platform is utilized for safeguarding the kernel against attacks, so the underlying operating system, audit engine, and monitoring data are all part of the TCB (trusted computing foundation). Because Paradise detects intrusions based on the provenance process entities and their dependencies, some attacks that are unrelated to the process can be challenging to detect. For example, the OpenSSL heartbleed vulnerability (CVE-2014-0160) cannot be detected because data leakage in memory will not generate system calls. Note that it being difficult to detect attacks does not mean attacks are undetectable. For example, CamFlow collects *wget* event provenance. Although *wget* is a file entity in the W3C PROV model, the file can still interact with specific processes, so Paradise can still detect the abnormal dependencies and make forensic analysis.

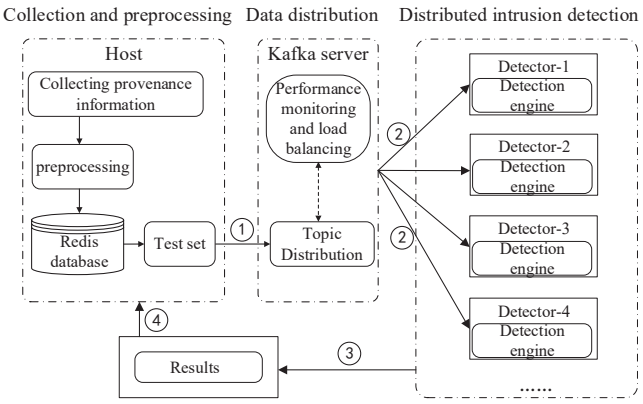

Fig. 4: Paradise system framework. (① is putting the preprocessed data into kafka *topic*. ② is dynamically distributing the data to the detection cluster. ③ is summarizing the results of different machines to obtain the final detection result. ④ is employed for forensic analysis.)

3 System Design and Implementation

We first describe the overall architecture of Paradise, then introduce specific design and implementation details. For simplicity and clarity, we only present provenance that conforms to the OPM model as an example. Of course, different provenance models can also be handled using the same strategy of Paradise without losing detection accuracy.

3.1 Overall System Design

The overall system is shown in Figure 4. The system is mainly divided into three modules: provenance collection and preprocessing, data distribution, and distributed intrusion detection.

The provenance collection and preprocessing module collects provenance and then converts the provenance graph into process feature vectors. Later, those preprocessed feature vectors are stored in the Redis memory database to speed up intrusion detection and forensic analysis by avoiding unnecessary disk I/O.

The data distribution module uses Kafka architecture to distribute process feature vectors to different detectors reasonably, paving the way for distributed real-time detection. Although Kafka can evenly distribute the workload to the partitions under the specified *topic*, it does not acquire the resources of the detectors and may distribute the workload to a detector with a heavy load. Therefore, we design a performance monitoring and load balancing scheme to dynamically adjust the load allocated to each detector according to the detector's resources, and thus achieve load balancing of the detection cluster.

The distributed intrusion detection module uses multiple detectors to perform real-time detection and aggregates the results of all detectors. At the same time, Paradise can support the distributed IDSs by adding a flag to the vector to determine whether the vector is a subgraph or a whole provenance graph. Finally, after summarizing the results of all detectors, we perform forensic analysis by looking up the intrusion process to help administrators quickly locate the system vulnerability. For more specific details, see 3.4.1.

3.2 Collecting And Preprocessing

3.2.1 Provenance Collection

Paradise utilizes SPADE [10] to collect the provenance of both normal and anomalous applications. It can also use other provenance tracking systems (e.g., CamFlow [13] and PASS [11]), which intercept system calls and generate file-level provenance.

SPADE supports either structured or unstructured databases to store provenance. For example, Graphviz is a commonly used visual graphical analysis tool that uses Graphviz DOT [37] to store and analyze provenance data in order to provide a structured view. Provenance nodes and edges collected by SPADE are output to a file in DOT syntax, which can fully present the provenance graph for intrusion analysis.

However, Graphviz is not conducive to querying, and the provenance logs need to be further parsed. The provenance node contains the object identifier and attributes such as *type*, *name*, and *time*. The provenance edge represents the type of system call. It contains the object identifiers of the child node and the parent node and attributes such as *type*, *time*, and *operation*.

3.2.2 Provenance Preprocessing

Paradise does not preserve the provenance of objects that only reside on disks for a short period in order to save storage space. Typically, these objects include temporary files or pipes that appear during program execution [7]. They bridge the transfer of information between different entities but are unlikely to store intrusive information.

Paradise mainly detects process feature vectors that record event behavior. The process nodes have attributes such as *name*, *type*, and *commandline* and attributes such as *eid*, *uid*, *gid*, and *ppid*, which will vary between repeated executions and thus should be removed. The characteristics of processes are primarily reflected in the type of dependencies between them. For simplicity and clarity, we only present the provenance that conforms to the OPM model

as an example. Of course, different provenance models also can be handled using the same strategy.

TABLE 1: Provenance database

Database	Key	Value
NameDB	node hash value	node name
ParentDB	child node hash value	parent node hash value
ChildDB	parent node hash value	child node hash value
VectorDB	process node hash value	process feature vector

Paradise focuses on four kinds of typical process dependencies defined in OPM: *Used*, *WasControlledBy*, *WasGeneratedBy*, and *WasTriggeredBy*. Paradise computes the frequency of the dependency type to obtain a process feature vector represented as [No. of *Used*, No. of *WasControlledBy*, No. of *WasGeneratedBy* No. of *WasTriggeredBy*] in algorithm 1. Specifically, each process feature vector describes the substructure of a provenance graph. Note that *isSub* represents whether the vector is a subgraph or a complete graph, and *proName* is the corresponding vector field indicating which event the subgraph belongs to.

The provenance is stored in a Redis memory database, as shown in Table 1. Among other reasons, we build around Redis because the provenance in Redis will not disappear in the case of a power failure. NameDB stores the mapping relationship between the provenance node hash value and the node name, using the hash value as the key, and the node name as the value. ParentDB and ChildDB store the dependency between the child node and the parent node. VectorDB stores the process node and its provenance feature vector. The feature vector of a process is extracted and constructed according to Algorithm 1.

Note that there exist many edges that start from the same node in a provenance graph. It takes time to identify all these edges in the graph query process when storing these edges as separate key-value items. Paradise employs Redis to store these edges into a single item where the key is the start node, and the value is a set of the end nodes of edges from the same start node. This improves query efficiency by reducing the number of items in the database.

3.3 Adaptive Data Distribution

The data distribution module is used to transfer the feature vectors to the detection cluster based on an improved version of Kafka. The host that collects the provenance information is the producer, a detection cluster is a consumer group, and the Kafka server is an intermediary for message distribution.

Though Kafka could distribute data evenly to the different detectors, detectors with the same hardware may have different resource consumption in a given period. Once the consumption rate lags far behind the production speed, the detector may have an anomaly such as a denial of service. In order to better avoid data loss and achieve load balancing of the detection cluster, we design a performance monitoring and data distribution module to enhance the Kafka architecture to address these issues. The performance monitoring and load balancing module monitors the detector performance by periodically acquiring detector performance metrics (such as CPU usage and memory usage) in order to control the amount of data distributed to detectors.

Algorithm 1 Extract Process Vector

Input: $G(V, E)$ //Provenance Graph
Output: Y //Process Vector

for each $dep_t : (v_i \rightarrow v_j)$ **in** E, $dep_t \in \{f_1 : Used, \cdots, f_4 : wasTriggeredBy\}$ **do**
  if type of v_i **is** $Process$ **then**
    $v_i.f_t++$
  end if
  if type of v_j **is** $Process$ **then**
    $v_j.f_t++$
  end if
end for
for each v_i **in** V **do**
  if type of v_i **is** $Process$ **then**
    create new instance $Y_i(isSub, proName, y_{i1}, y_{i2}, y_{i3}, y_{i4})$
    for all y_{ij} **in** Y_i **do**
      $y_{ij} = v_i.f_j$
    end for
  end if
end for
return Y

This method can automatically and reasonably allocate data, as well as ensuring load balancing and making full use of the performance of the detector cluster.

The steps for the data distribution module are as follows:

Step 1: Each detector collects its performance indicators, such as CPU usage and memory usage.

Step 2: If the value of a detector's (named detectorA) performance metrics is greater than the performance threshold T_C, the load of the machine is considered to be too heavy. DetectorA will not subscribe to the *topic* but "sleep" for a unit of time, thus the other detectors will consume the data originally intended to be sent to detectorA.

Step 3: When the performance indicator value is less than the performance indicator threshold T_C or after "sleeping" for a certain period of time, detectorA resubscribes to the *topic* for consumption.

This data distribution module can realize a self-controlled consumption rate according to the consumption capacity of the detector, avoid denial of service and network congestion issues with detectors, and achieve load balancing of entire clusters.

3.4 Distributed Intrusion Detection

Real-time intrusion detection is achieved by using multiple detectors to detect provenance information distributed by improved Kafka servers. This module is mainly divided into two parts: intrusion detection and forensic analysis.

3.4.1 Intrusion Detection

Typically, when the workload of a detector is very heavy, and the detector needs to "sleep" for a while, it is much easier to estimate the time interval based on a vector-based detection method, rather than graph [7] or path-based detection [6] methods. Since the vector format is fixed, the detection time of each vector with Paradise is more stable

Algorithm 2 Distributed intrusion detection

Input: X_i, $Y[n]$, $isSub$, $proName$, T, k
$//X_i$: Test set feature vector
$//Y[n]$: Training set feature vectors
$//isSub$: A flag to determine whether the vector is a subgraph or a complete graph
$//proName$: A corresponding field to indicate which event the provenance subgraph belongs to
$//T$: Predefined similarity threshold
$//k$: Predefined threshold of the nearest vector's number
Output: $temp-result$, $results$ //Detection results

for each feature $x_j \in X_i$ **do**
  $x_j = (1 - e^{-ax_j})/(1 + e^{-ax_j})$
end for
for each process feature vector Y_j **in** $Y[n]$ **do**
  $d(X_i, Y_j) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_4 - y_4)^2}$
  if $d(X_i, Y_j) == 0$ **then**
    $isNormal = true$ // X_i **is normal**
    exit
  end if
end for
sort $d(X_i, Y[n])$ **in increasing order**
get top k **records and calculate avg** $d(X_i, Y[n])$
if avg $d(X_i, Y[n]) > T$ **then**
  $isNormal = false$ // X_i **is abnormal**
else
  $isNormal = true$ // X_i **is normal**
end if
if $isSub$ **then**
  upload $(X_i, isNormal, proName)$ **to *temp-result***
else
  upload $(X_i, isNormal, proName)$ **to *results***
end if

and has less fluctuation. The graph or path-based detection methods, on the other hand, judge anomaly degrees of a single provenance path or the whole provenance graph. Thus the detection time may vary significantly for different events, which is not conducive to load balance in distributed environments.

The intrusion detection algorithm is shown in Algorithm 2. Paradise first calculates the similarities between a process feature vector X_i in the test set and all normal process feature vectors $Y[n]$ in a training set to determine whether X_i is abnormal. Paradise mainly detects units of processes, and the feature vectors of each process in the provenance graph are independent. Unlike sliding window detection (FRAP) [5], the sequence of detection does not affect Paradise detection results. The different feature vectors of the same provenance graph can be determined by any detectors, so it is very suitable for distributed detection environments.

We use Euclidean distance to calculate the similarity between the feature vectors. Then we arrange the results in ascending order and calculate the average value of the first k shortest distances. If the average distance is higher than a predefined T, the system is considered to be attacked and an alert is triggered. After getting the detection result, Paradise judges whether the process vector belongs to a subgraph for each vector in a detector. If this vector belongs to a provenance subgraph, we will output the result to a *temp-result* in the detector. Otherwise, we directly upload this result to the final *results*. We merge all the *temp-results* for different detectors into the final *results*. Generally speaking, intrusion detection in Paradise includes two steps; the first step is to calculate and compare the similarities of the process vectors, and the second step is to record and summarize the subgraph detection results.

In Algorithm 2, Paradise first obtains the feature vector X=[x_1, x_2, x_3, x_4] of the process node according to Algorithm 1. x_1 indicates the frequency of *Used*, x_2 indicates the frequency of *WasGeneratedBy*, x_3 indicates the frequency of *WasTriggeredBy*, and x_4 indicates the frequency of *WasControlledBy*. Then, Paradise uses Equation 2 to normalize the feature vector in training and testing to obtain the feature vectors X=(x_{f1}, x_{f2}, x_{f3}, x_{f4}) and Y=(y_{f1}, y_{f2}, y_{f3}, y_{f4}) respectively.

$$x_{f_i} = (1 - e^{-af_i})/(1 + e^{-af_i}) \quad (2)$$

In Equation 2, e is a natural constant, f_i is a feature in the feature vector, and a is a constant, the value of which depends on the frequency of the process relationship type. Note that for different process characteristics, a is generally different. Since some system daemons always read and write files in the system, we have adopted a normalized method to handle feature vectors in order to reduce the impact of these normal processes. In our experiments, we tested the case where a is 1, 10, and 100. When the a value for *WasControlledBy* is 100 and the a value for other types of relationships is 1, the detection rate is 100% and the false alarm rate is 0%. Through this normalization formula, the values can be well distributed between 0 – 1. We perform vector deduplication on the training set to enhance the detection efficiency and avoid repeatedly calculating the similarity distance.

The detector in Paradise only extracts the process node feature vectors and does not traverse the whole provenance graph. More importantly, Paradise could be applied to different provenance models with the same strategy in order to avoid data conversion and reduced detection accuracy (see 2.4 Exploring process characteristics for intrusion detection).

3.4.2 Forensic Analysis

After Paradise identifies an intrusion process, we use BFS (Breadth-first search) for intrusion forensic analysis, which uses the anomaly process node as the tracking root node. A BFS is used in forensic investigation because Paradise constructs the feature vector based on the neighbor graph, which a BFS can represent.

After the root of intrusion is identified, all nodes connected to the intrusion source root are regarded as abnormal nodes. Then, they are added to the anomalous provenance events and marked as an intrusion. The edge is also added to the anomalous provenance graph, which can be further used to analyze the files affected by the intrusion. For example, Paradise identifies the process *cc1* and process *sh* feature vectors as abnormal so that the intrusion file *malicious file* and the leaked *sensitive file C* can be easily located in Figure 2. After knowing that a *malicious file* caused the intrusion, administrators can quickly locate the vulnerability in the *wget* process.

4 Experimental Evaluation

Our assessment addresses the following research questions:

Q1: Can Paradise detect intrusions on the different provenance model datasets? (§4.2)

Q2: How accurate at detection is Paradise compared to an existing state-of-the-art IDS on open-source datasets? (§4.2)

Q3: Can a dynamic distribution strategy accelerate Paradise's detection speed? (§4.3)

Q4: What are Paradise's memory utilization and provenance growth overhead during system execution? (§4.4)

We compare the detection accuracy between Paradise and Pagoda, FRAP, and Unicorn on different datasets (Q2). We show that Paradise can detect intrusion in actual APT activities (Q1). Finally, we used CamFlow in a controlled laboratory environment to create our own simulated APT attack dataset to test whether Paradise can be accelerated in a distributed environment (Q3) and analyze Paradise's overhead performance (Q4).

4.1 Experimental Setup

The experimental host runs a Windows 10 Pro 64-bit operating system. There are thirteen virtual machines in the host as shown in Table 2. Three machines are used as the invaded machine to collect provenance, one machine is used as the attack machine, one machine is used as the Kafka server, and the other eight machines are used as the detection cluster. To evaluate the performance of Paradise, different kinds of applications are tested in Table 4. These provenance data are obtained by different provenance collection systems and have different numbers of training files and process feature vectors.

PASS traces: Vsftp trace is already used in PIDAS [6]. The vulnerable application proftp (1.3.3c) has a backdoor command execution vulnerability that allows remote attackers to access the system. The phishing email is a typical APT attack on users who click a web link in a malicious email to open a downloadable malicious backdoor program.

SPADE traces: Attackers can use malicious code to exploit application traces in order to execute remote attacks [6, 8].

Simulated CamFlow trace: We simulate an attacker initiating an exploit via the Metasploit framework from a remote machine to attain the root shell, thereby obtaining simulated traces. Note that although we simulate those attacks, we still collect the simulation data with the CamFlow system.

Open-Sourced CamFlow SC-1 trace: The Attacker discovers that an enterprise continuously downloads Debian packages through *wget* (version 1.17 CVE-2016-4971), which is vulnerable to arbitrary remote file upload. The attacker inserts a popular remote access trojan (RAT) into a Debian package and hacks one of the repositories, redirecting all legitimate package download requests to the attacker's FTP server, which hosts the RAT-embedded package. The CI server mistakenly installs this malicious package when it downloads and installs the legitimate package. The attack uses this RAT to create a reverse TCP shell session on the CI server by establishing a C&C channel. Then the attack changes the CI server settings to obtain control of the CI deployment output.

TABLE 2: Descriptions of Machines

Machine type	Operating system	Number of processors	RAM	Hard disk
Invaded machine (1)	Ubuntu 16.04 LTS	1	8G	500G
Invaded machine (2)	fedora29	4	16G	200G
Invaded machine (3)	Ubuntu 16.04 LTS	2	16G	100G
Attack machine	Kali Linux 18.4	2	4G	100G
Kafka server	Ubuntu 18.04 LTS	8	8G	100G
Detector	Ubuntu 18.04 LTS	1	2G	200G

TABLE 3: Confusion Matrix

		Predictive Value	
		Abnormal	Normal
Actual Value	Abnormal	TP	FN
	Normal	FP	TN

DARPA OpTC trace: The DARPA TC program achieved a better knowledge of APT attacks by collecting several datasets, including many harmful behaviors that mimic APT situations in an enterprise network environment. The Operationally Transparent Cyber (OpTC) dataset [38] is the most recent version of these datasets. Almost 17 billion records in this dataset detail benign and abnormal behaviors through a network and host-level log. The dataset's sheer size and abundance of information make it ideal for detecting APTs. The malicious events of this host include instances of data filtering through *Netcat* and *RDP*, and malware upgrades through *notepad* (When the susceptible program *notepad* is updated, it connects to a malicious server and downloads a binary program called *update.exe*). All these malicious activities are consistent with the behavior of APT. Therefore, this OpTC dataset is an excellent sample for APT detection or APT classification research. The OpTC organizers first record benign behavior, followed by malware injection behavior. The OpTC dataset contains malicious events executed within three days of the evaluation period, and benign behaviors ran continuously during the injection activity. Paradise is a host-based intrusion detection system. We extract and label provenance data from the cluster hosts that contain both abnormal and normal behaviors based on the ground truth [39] in the period.

Compared with the SPADE and CamFlow (not including percona-root trace) traces, the PASS trace has less dependencies. The first two applications of CamFlow traces are collected in OPM format, and the last four applications of CamFlow traces are collected in W3C PROV format. The OpTC dataset is based on the eCAR [41] data format.

4.2 Paradise vs State-of-the-Art

4.2.1 Evaluation Indices

As shown in Table 3, the cross-list of the confusion matrix can be used to evaluate the detection rate and false alarm rate of the intrusion detection methods.

TP (True Positive) represents the number of abnormal provenance graphs correctly classified as abnormal;

TN (True Negative) represents the number of normal provenance graphs correctly classified as normal;

FP (False Positive) represents the number of normal provenance graphs misclassified as abnormal;

FN (False Negative) represents the number of abnormal provenance graphs misclassified as normal.

The detection rate T_{TPR} (True Positive Rate) shows the percentage of intrusion provenance graphs that have been

TABLE 4: Descriptions of Intrusion Traces (① represented using CamFlow to create simulated attack dataset).

Provenance system	Dataset name	CVE Number	Train set	Test set		Source
			Number of files	Normal behavior Number of files	Abnormal behavior Number of files	
SPADE	samba	CVE-2017-7494	100 (Raw data 101 MB)	51 (Raw data 10.4 MB)	51 (Raw data 40.7 MB)	[8]
	tomcat	CVE-2017-1217	100 (Raw data 36.4 MB)	51 (Raw data 21.7 MB)	51 (Raw data 14.9 MB)	
	wordpress	CVE-2019-8943	100 (Raw data 36.3 MB)	51 (Raw data 19.2 MB)	51 (Raw data 20.7 MB)	
	webmin	CVE-2019-9624	100 (Raw data 112 MB)	51 (Raw data 46.9 MB)	51 (Raw data 75.5 MB)	
CamFlow	samba	CVE-2017-7494	100 (Raw data 9.08 MB)	100 (Raw data 2.82 MB)	100 (Raw data 2.06 MB)	① [40]
	webmin	CVE-2019-9624	100 (Raw data 4.31 MB)	94 (Raw data 4.41 MB)	64 (Raw data 8.49 MB)	
	tomcat-remote	CVE-2016-5425	99 (Raw data 1.75 GB)	10 (Raw data 178 MB)	10 (Raw data 642 MB)	①
	percona-root	CVE-2016-6664	99 (Raw data 4.64 MB)	10 (Raw data 480 KB)	10 (Raw data 884 KB)	
	mysql-race	CVE-2016-6663	100 (Raw data 5.07 MB)	10 (Raw data 520 KB)	10 (Raw data 520 KB)	
	SC-1	CVE-2016-4971	100 (Raw data 46.0 GB)	25 (Raw data 16.6 GB)	25 (Raw data 13.9 GB)	[19]
PASS	vsftp	CVE-2011-2523	300 (Raw data 1.07 MB)	48 (Raw data 228 KB)	110 (Raw data 0.99 MB)	[7]
	proftp	N/A	30 (Raw data 1.84 MB)	12 (Raw data 873 KB)	102 (Raw data 520 KB)	
	phising email	—	20 (Raw data 1.55 MB)	57 (Raw data 4.09 MB)	200 (Raw data 11.0 MB)	
ETW eCAR	OpTC DARPA	—	10 (Raw data 10.4 GB)	2 (Raw data 2.45 GB)	6 (Raw data 3.23 GB)	[38]

TABLE 5: Average detection rate and average false alarm rate for Paradise for all the datasets (except SC-1 and Optc Darpa datasets) in different thresholds k and T (k represents the number of the nearest vectors and T represents the similarity threshold).

Evaluation Index	threshold T	SPADE				PASS				CamFlow			
		Value k											
		1	3	5	7	1	3	5	7	1	3	5	7
Detection Rate	0.1	99.5%	99.5%	99.5%	99.5%	99.8%	99.8%	99.8%	99.8%	100.0%	100.0%	100.0%	100.0%
	0.3	99.5%	99.5%	99.5%	99.5%	42.3%	76.2%	76.2%	76.2%	99.3%	100.0%	100.0%	100.0%
	0.5	99.5%	99.5%	99.5%	99.5%	7.0%	40.2%	66.8%	66.8%	42.8%	99.3%	100.0%	100.0%
	0.7	99.5%	99.5%	99.5%	99.5%	0.0%	6.0%	33.0%	33.0%	15.4%	15.4%	15.9%	17.0%
	0.9	99.5%	99.5%	99.5%	99.5%	0.0%	0.0%	0.0%	0.0%	15.4%	15.4%	15.4%	15.4%
False Alarm Rate	0.1	4.5%	6.0%	7.5%	9.5%	3.9%	3.9%	17.8%	26.1%	0.0%	0.0%	0.0%	0.0%
	0.3	0.0%	0.5%	2.0%	2.0%	2.7%	3.9%	3.9%	3.9%	0.0%	0.0%	0.0%	0.0%
	0.5	0.0%	0.0%	0.0%	0.0%	2.7%	2.7%	2.7%	2.7%	0.0%	0.0%	0.0%	0.0%
	0.7	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	2.7%	2.7%	0.0%	0.0%	0.0%	0.0%
	0.9	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	2.7%	0.0%	0.0%	0.0%	0.0%

correctly classified. The calculation method is shown in Equation 3:

$$T_{TPR} = TP/(TP + FN) \tag{3}$$

The false alarm rate T_{FAR} indicates the fraction of normal provenance graphs that have been judged as anomalies. The calculation method is shown in Equation 4:

$$T_{FPR} = FP/(TN + FP) \tag{4}$$

Winnower [21] is the most relevant provenance-based distributed detection system to Paradise. However, there is a lack of unified datasets and data formats for provenance graph-based detection. Neither winnower code nor datasets are publicly available. We also do not find the result of its related detection metrics, such as detection rate and false alarm rate. Therefore, it is not very convenient for us to do comparative experiments with Winnower. So, we measure these evaluation indices by comparing four intrusion detection methods:

- detection method based on sliding window (FRAP) [5]
- detection method based on provenance graph and path (Pagoda) [7]
- detection method based on data histogram (Unicorn) [19]
- detection method based on process feature vectors (Paradise)

Pagoda [7] identifies hidden and complex attacks (such as APT) in a big data environment by considering the anomaly degree of a single provenance path and the entire graph. Unicorn [19] models system behavior using structured provenance graphs that show causality relationships between system objects and summarize the graph as a data histogram, which can evolve over time. Since UNICORN is a successor of FRAP, we choose the FRAP detection method as the baseline for comparison of detection accuracy. We first perform a sensitivity analysis of Paradise, then we show the results of each application trace for all intrusion detection methods in Figure 5.

4.2.2 Threshold Chosen

Table 5 shows how different thresholds k and T impact the detection accuracy of Paradise. Threshold T represents the similarity threshold. The k nearest vectors are selected to calculate the similarity distance.

Note that the theoretical upper limit of the T between different provenance models is generally different. For example, the theoretical upper limit of T for OPM is 2 (Euclidean distance between the vector [0, 0, 0, 0] and the vector [1, 1, 1, 1]). However, the similarity distance between different provenance model vectors in our experimental results is mostly less than 1. So the similar distance thresholds T are set to 0.1, 0.3, 0.5, 0.7, and 0.9. The smaller the distance between vectors, the more similar they are.

In theory, the larger the value of k, the larger the average of the k shortest similarity distance, which will increase both detection rate and false alarm rate. The false alarm rate can be very high when k exceeds 3. In that case, the provenance graph can be easily judged as abnormal even if a few process vectors are similar to the rule database vectors.

In SPADE and CamFlow traces, most traces can be detected correctly. However, there is an abnormal file in the webmin trace whose feature vectors are all in the normal training set, which causes the file to be incorrectly detected, resulting in an average detection rate of 99.5%. In the best case, the threshold ($T \geq 0.3$) for testing the data collected by SPADE is slightly larger than the threshold used in PASS

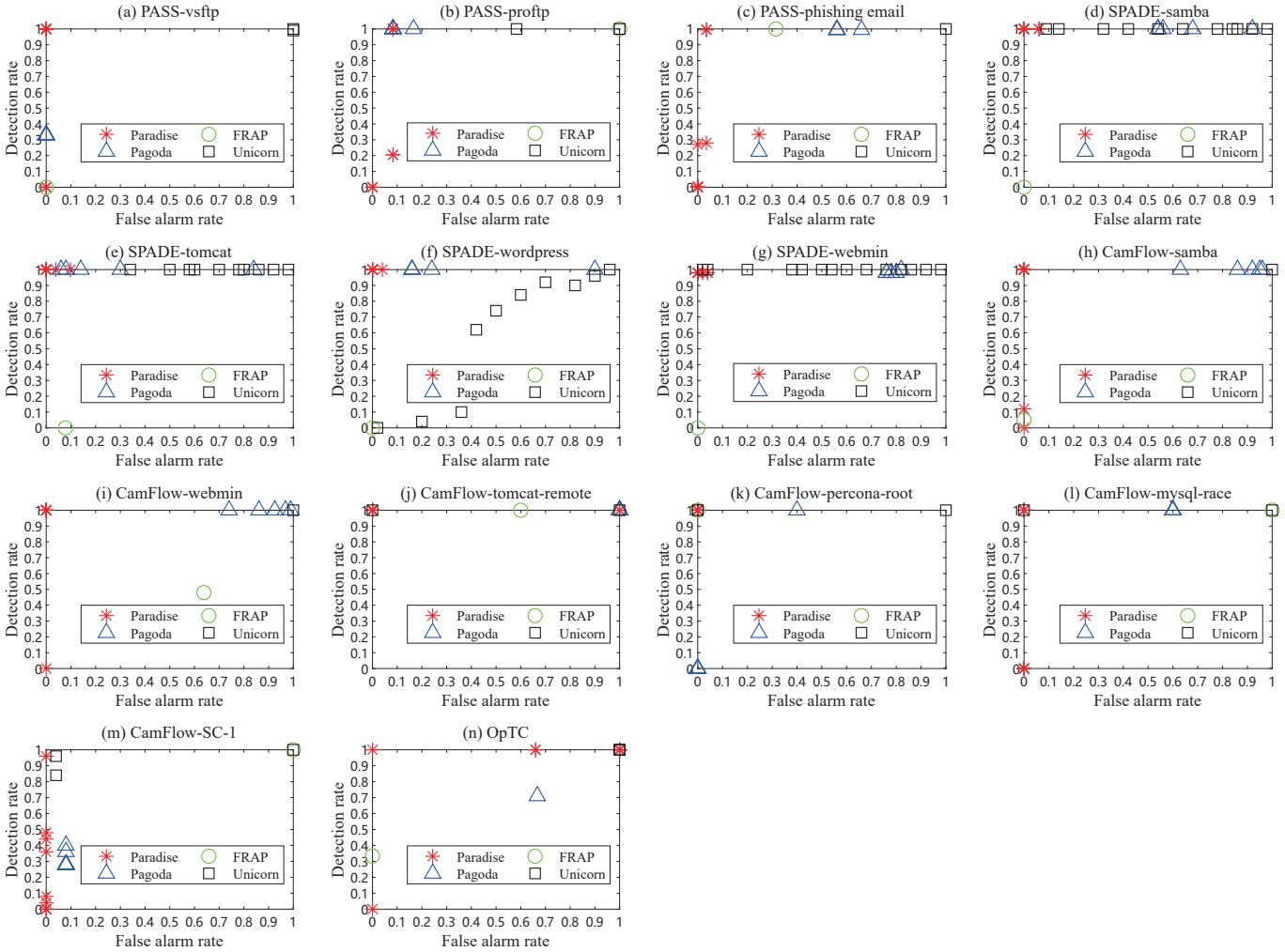

Fig. 5: Detection rate versus false alarm rate for different applications using different intrusion detection methods. For each application, we show the value for Paradise for different thresholds T. We also show the value for Pagoda for different $graph$ thresholds and set the $path$ threshold to 0.9 to minimize the false alarm rate. Each point in the figure shows the performance of a method at a certain threshold. FRAP has only one point because it only judges whether the intrusion instance is located in the existing cluster in the rule database and does not define multiple thresholds like the other two methods. Pagoda and Paradise methods have only one point in some specific applications because their values overlap under different thresholds. The upper-left point achieves the best performance in both detection rate and false alarm rate.

($T = 0.1$) because the differences between process feature vectors in PASS are not as obvious as the others. Hence, a smaller threshold ($T = 0.1$) needs to be taken. When using Paradise to detect PASS traces, T is set around 0.1, and the k is set around 3, when using Paradise to detect SPADE traces, T exceeds 0.3, and the k is set around 3, and when using Paradise to detect CamFlow traces, T is lower than 0.3.

Note that in Figure 1, *wget* is a file entity collected by the CamFlow system in the provenance graph. This file entity also interacts with specific process entities in the SC-1 dataset. Therefore, Paradise can also detect this intrusion. However, the threshold k needs to be expanded (e.g., one percent of the quantity in the rule base but no more than one-tenth of the quantity in the rule base) for accurate detection. The threshold T remains unchanged at 0.1, and the experimental results are shown in Figure 5.

4.2.3 Detection Results

1) PASS trace: Paradise is significantly better than the other methods, with a lower false alarm rate and a higher detection rate in the best case. However, compared with Pagoda, Paradise has a vast distribution in the detection rate. The main reason is that Pagoda detects more rigorously in fewer dependencies instead of simply considering the dependencies related to the process node. Paradise only takes into account the average similarity distance of process feature vectors. Since there are fewer process nodes in the PASS trace training data, the changes in the average similarity distance of process vectors are more sensitive in Paradise, leading to a wide distribution of the detection rate. FRAP is a streaming computing method that uses a sliding window to extract the feature; thus FRAP cannot fully detect the original provenance graph, resulting in poor performance. Unicorn has poor detection accuracy on these datasets partly because the dataset is converted to conform

to the Unicorn format, which will lose accuracy. Additionally, these datasets have fewer dependencies, making Unicorn unable to establish a complete detection model.

2) SPADE trace: Paradise performs best on the SPADE trace. The false alarm rate of Pagoda is very high, especially in the case of the dependency explosion in the four server applications of the SPADE trace. False alarms are easy to cause in Pagoda when the $graph$ threshold is small. Paradise achieves a lower detection rate when the T is large. Moreover, A large threshold T indicates that Paradise can tolerate significant differences between the process feature vectors. Therefore, Paradise can easily misjudge abnormal vectors as the rule database vectors (see section 4.2.2). This phenomenon shows that we can choose an appropriate threshold to obtain the best results. The detection accuracy of FRAP is very low because it only regards a part of the event graph as the detection object at a time. Due to the loss of precision of the converted data format, Unicorn cannot achieve ideal detection results.

3) Simulated CamFlow trace: In most applications, the performance of Paradise is significantly better than the other methods. The false alarm rate for Paradise is nearly 0% in Figure 5. Unlike the SPADE trace, all CamFlow trace files used to test the false alarm rate contain some normal events different from the normal training dataset. Paradise only considers the process feature vectors for classification and does not care about the path length and size of the provenance graph. Since Paradise calculates the average similarity distance, it is less sensitive to whether the path is in the rule database. Thus, the false alarm rate of Paradise is much better than that of Pagoda. Note that Paradise and Unicorn could achieve ideal detection results on traces that conform to the W3C PROV model. In the traces conforming to the OPM model, Paradise's detection accuracy is significantly better than Unicorn owing to loss of data accuracy in the conversion of Unicorn's data format.

There are two types of intrusion traces (samba and webmin) that use different provenance models (OPM and W3C PROV) and different provenance collection systems (SPADE and CamFlow). However, Paradise can still get ideal detection results, which shows that Paradise can be applied to different provenance models and provenance collection systems without data conversion, avoiding information loss during conversion.

We also evaluated the detection accuracy of Unicorn's open-source SC-1 dataset for fairness, as shown in Table 5.

4) Open-Sourced CamFlow SC-1 trace: Paradise has a false alarm rate of 0% under different thresholds k. The main reason may be that the original Unicorn SC-1 dataset is not continuous in time, so Paradise employs an active learning method called *Expected Model Change* [42] for training set construction. Among the 125 normal files, files with significant differences in process feature vectors were selected as the ruleset to ensure that Paradise can collect normal user behaviors as much as possible. In order to further verify the problem of false alarm rate, we drew on Xanthus's [43] related work. Based on the original normal behavior of Unicorn, we repeatedly simulated 100 groups of normal behavior on our virtual machine, tested the false alarm rate, and found that the false alarm rate was still 0% with the same best threshold in the SC-1 dataset.

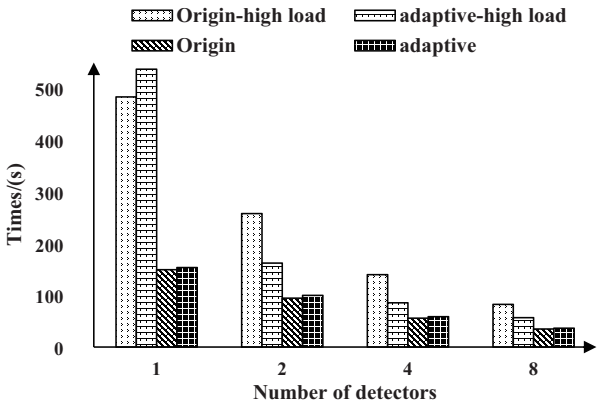

Fig. 6: Average detection time for different numbers of detectors on CamFlow trace. High-load means there is one detector machine in the cluster that is under a high load.

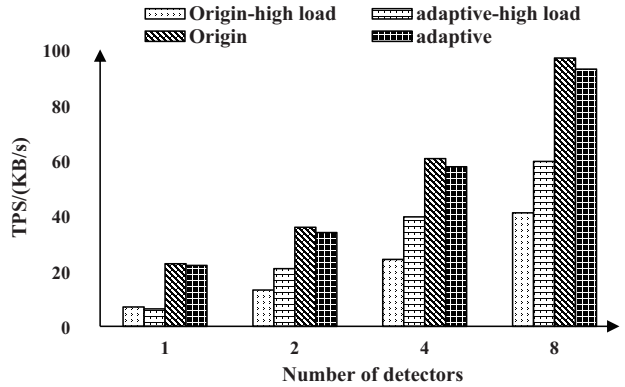

Fig. 7: Cluster throughput for different numbers of detectors on CamFlow traces. High-load means there is one detector machine in the cluster that is under a high load.

5) DARPA OpTC trace: The open-sourced DARPA Operationally Transparent Cyber (OpTC) dataset is comprised of six days of normal behavior data and abnormal behavior data for the next three days, which are continuous in time. We take the normal behavior of the later four days as the rule database, the first two days' behavior as the false alarm detection set, and the abnormal behavior as the detection set. The reason for this selection is to ensure detection accuracy by considering *concept drift* as defined in Unicorn [19] (The underlying statistical features of the streaming provenance graph alter over time as a result of changes in system behavior). Because of the *concept drift* problem, if the threshold is low ($T < 0.5$), Paradise's false alarm rate will be abysmal. The result shows that Paradise could identify abnormalities in long-running systems on a variety of applications with accuracy ($T > 0.5$). OpTC is in the eCAR format data, which lacks the critical node attribute information defined in Unicorn [19], leading to a poor false alarm rate. The attack initiated APT assaults utilizing several attack channels during the engagement, accounting for less than 0.0016 percent of the audit data volume. Even though the assaults are hidden within a plethora of normal behavior, Paradise can still identify the intrusion without prior knowledge of the attack.

Overall, in most applications, the performance of Par-

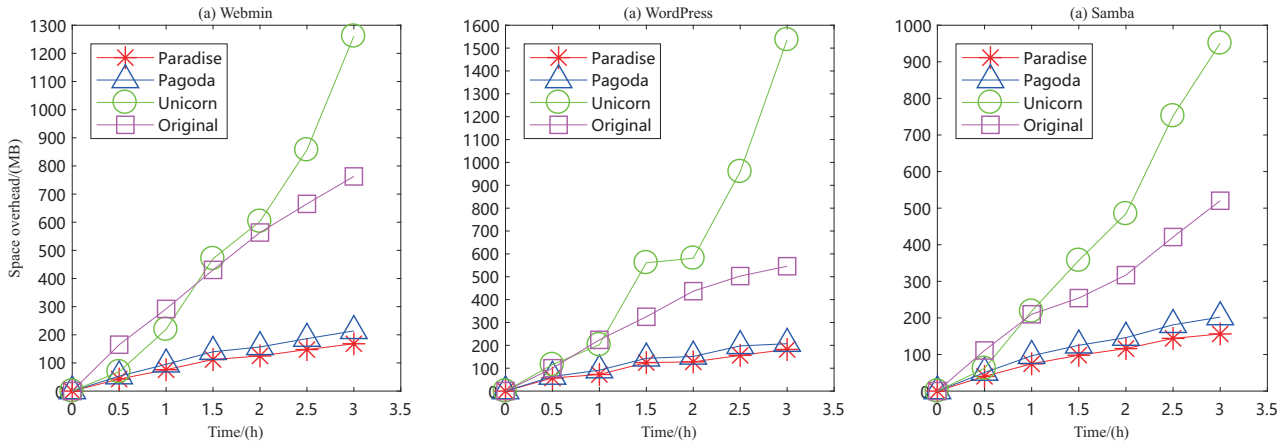

Fig. 8: Provenance growth overhead of different intrusion detection methods for webmin, wordpress and samba applications.

adise is significantly better than the other methods. Compared with Pagoda, Paradise mainly uses process features as detection objects instead of provenance graphs or paths, thereby reducing to some extent the attacker's means of evading detection (e.g., adding many normal behaviors in anomaly graphs). In addition, Paradise also needs to summarize all the detection results of subgraphs to obtain the final result in order to obtain an excellent detection accuracy. The poor accuracy of FRAP is because it uses a sliding window to detect only a part of the provenance graph each time. Compared with Unicorn, Paradise has better generalization capabilities and can be applied to different provenance models without additional data conversion.

4.3 Scalability Analysis

In this experiment, we use the CamFlow tomcat-remote dataset [19], of which 1.75 GB data is used as the training set and 820 MB data is used as the test set. The detector machines in the cluster are tested under low load (CPU and memory resources are sufficient for Paradise intrusion detection) and high load (only 30% CPU resource is available for Paradise intrusion detection by using *Stress-ng* [44] tools) respectively. Cluster detection time overhead and cluster throughput are the average results of running the Paradise six times. It is worth noting that after an average of 86.86s of preprocessing steps, we finally get 3.3 MB process feature vector data that needs to be distributed for detection.

The test results are shown in Figure 6 and Figure 7. Generally speaking, cluster detection time decreases with the increase of cluster detectors, and cluster throughput increases with the increase of cluster detectors.

For high load cases when the number of detectors in the cluster is 2, 4, and 8, the adaptive data distribution scheme can effectively reduce the detection time of the cluster by 25%–30% and improve the throughput of the cluster by 45%–60% simultaneously. This is because multiple detectors share the tasks of the detector with the higher load to alleviate the performance bottleneck of the cluster, ensuring the detection efficiency of the cluster. However, if the detection cluster has only one detector with a high load, there are no additional machines to share the load, but the improved scheme still needs to monitor the performance indicators (CPU and memory resource) of the detector (which may sleep for a certain time under high load conditions), leading to a worse result than the original case.

For the low load case, similarly, the improved scheme needs to monitor the performance indicators of the detector. However, cluster throughput will be only reduced by 3%–5%, and the cluster detection time will be only reduced by 3%–6%.

4.4 Overhead Analysis

We mainly analyze the three aspects of provenance growth overhead, memory overhead, and time overhead.

1) Provenance growth overhead. We tested the provenance growth overhead of the three intrusion detection methods, for webmin, wordpress, and samba datasets [8] as shown in Figure 8. Paradise reduces the space overhead by about 70% compared to the original case by aggregating similar provenance items and only storing the process nodes and their dependencies. It has less space overhead than Pagoda, because Pagoda extracts all the nodes and their dependencies, while Paradise only extracts the process nodes and their feature vectors. Unicorn computes the characteristics of the provenance graph and summarizes the graph as a data histogram based on the degrees of the node. It retains all the features of the provenance graph on the disk, and the number of node labels will increase over time. Therefore, it has a larger space overhead than the original provenance data. Paradise generates an average of 73 MB of provenance information per hour and 1.71 GB for one day which is acceptable using today's cheap disk.

2) Memory overhead. The runtime memory overhead for the three intrusion detection algorithms is sampled per second, and the average value is computed to obtain the memory overhead generated by running the Tomcat application as shown in Figure 9. The memory overhead for both Pagoda and Paradise mainly consists of two parts: the memory overhead of the detection process and the Redis server process used to speed up the query and detection. Redis server processes for both Pagoda and Paradise have comparable memory overhead (18 MB). The Paradise detection process detects intrusion by computing the similar distance between process feature vectors of each intrusion

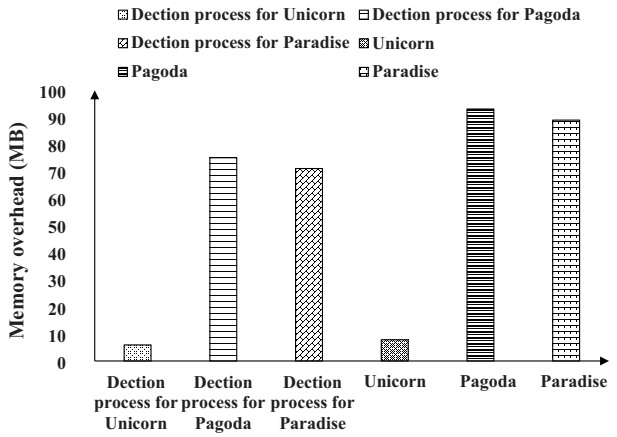

Fig. 9: Memory overhead of different intrusion detection methods on Tomcat application.

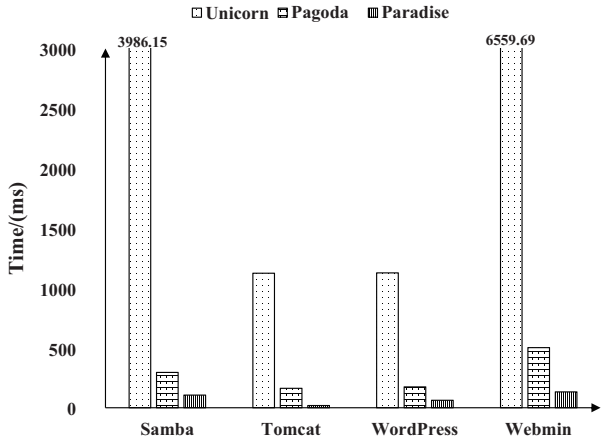

Fig. 10: Average detection time of different intrusion detection methods for a wide variety of intrusion applications in one detector machine.

event, thus it needs to traverse the nodes and edges in the whole provenance graph only once. However, the Pagoda detection process detects intrusion by taking into account the anomaly degrees of both the path and graph of each intrusion event, thus it needs to traverse the nodes and edges in the whole provenance graph many times. Therefore, Pagoda consumes more memory than Paradise. As Unicorn stores provenance on disk, the memory overhead for Unicorn lies in the detection process. As it only extracts part of the provenance graph for detection each time, it has the least memory overhead.

3) Time overhead. The test time overhead is the average of running the intrusion detection method six times. The test result is presented in Figure 10. Both Pagoda and Paradise get the rules from the Redis database, so the detection time is relatively small. Unicorn needs to get full context-knowledge of multiple levels of ancestors and descendants of each node, greatly increasing the detection time. Compared with Pagoda, Paradise only extracts the process features in the provenance graph for detection, without matching the provenance paths one by one, so it has the smallest time overhead.

5 Related Work

In this section, we review related work on host-based intrusion detection systems, forensic analysis, and distributed intrusion detection.

5.1 Host-based intrusion detection

There are some existing works on host-based intrusion detection. Forrest et al. [45] first proposed to extract short system call sequences from normal processes and used these short sequences to build a normal database to detect anomalies. Hofmeyr et al. [46] calculated the Hamming distance between the different sequences of system calls. By comparing the minimum Hamming distance with a user-defined threshold, anomalies can be identified. Jiang et al. [47] proposed a varied-length n-gram model to characterize the provenance and constructed automata to detect anomalies. Liao et al. [48] proposed O-kNN, a kNN-based intrusion detection method that uses TF-IDF [31] for vector preprocessing and classification according to the cosine similarity between vectors. However, the cosine similarity is prone to misjudgment, resulting in a decrease of detection accuracy. Yao et al. [49] proposed a unified framework that depicts the anomaly detection models and ranks them by their detection capabilities and proved the theoretical accuracy limit of the anomaly detection.

Recently, with the research hotspots of neural networks, some researchers have applied them to intrusion detection. Creech et al. [50] obtained semantic features by analyzing discontiguous system call patterns, and used these semantic features as the input of a neural network to determine whether traces are abnormal. Sun et al. [51] proposed ZePro, a method to use a probabilistic Bayesian network to identify zero-day attack paths. Shu et al. [52] proposed a content detection technique by comparing the similarity of two sample sequences to detect the leakage of sensitive information in file content or network traffic. To achieve high analytical throughput, the method can be parallelized on multiple CPU and general purpose GPU platforms. Shone et al. [53] proposed NDAEs, a deep learning classification model constructed by stacked Non-symmetric Deep Auto-Encoder (NDAE), which combines NDAE with RF classification methods to reduce analysis overhead and improve detection accuracy. Although machine learning algorithms begin to show advantages in intrusion detection, the training of neural networks can be complex with a large time overhead.

There are also a variety of works on provenance-based intrusion detection. Han et al. [5] proposed FRAP to use normal behavior to construct a model, and used a dynamic sliding window method to compare a new instance with the model to judge whether an attack has occurred. Nwafor et al. [54] proposed a method for intrusion detection using the provenance of the sensor data of IoT devices. This method compares the observed provenance graph with the known provenance graph of an application to determine whether an intrusion has occurred. However, in a massive data environment, the provenance graph size will become large and the provenance path will become long, resulting in a poor performance. Milajerdi et al. [20] proposed HOLMES to detect advanced persistent threats. HOLMES forms a

provenance graph by using APT-related events. Then it builds a high-level scenario graph (HSG) as an abstraction over the provenance graph to bridge the semantic gap between a low-level system-call view and a high-level APT kill-chain view, and uses HSG to judge whether an attack has occurred. Han et al. [19] proposed Unicorn which uses an efficient graph analysis method (called graph sketching) that contains rich contextual semantics and historical information of provenance to judge APT attacks. In our previous work, we proposed PIDAS [6] to use the inherent dependence in provenance information to analyze an attack path. Later, we designed Pagoda [7] to identify stealthy and sophisticated attacks (e.g., APT) in big data environments by taking into account the anomaly degrees of both a single provenance path and the whole provenance graph. However, all these methods are designed for the detection in a single host, and thus they have limited detection capabilities and cannot tolerate the failure of a single node.

5.2 Forensic Analysis

Provenance has also been extensively used in forensic analysis [55–60]. Chris et al. [60] developed a tool called EviHunter that summarizes the AED (Application Evidence Database) by static analysis of a large number of applications to automatically identify evidence data in the permanent storage of Android devices. Zhou et al. [55] proposed secure network provenance (SNP), which provides network forensics by tracking down faulty or abnormal nodes and assessing the damage such nodes may cause to the rest of the system. This method collects the provenance of the entire system, and is not suitable for online detection requirements in massive data environments because of the dependency explosion. Many techniques such as partitioning and logging [3, 61] have been suggested to reduce the size of the provenance. Tang et al. [62] also proposed a data reduction system named NodeMerge for online storage that directly processes system-related data and achieves data reduction according to their access pattern.

5.3 Distributed Intrusion Detection

MIDeA [63] is a parallel intrusion detection architecture, which uses multi-queue NICs, multiple CPUs, and multiple GPUs to avoid locking and thus optimizes data transfers between the different processing units. Kargus [64] uses multiple CPUs and multiple GPUs to increase the throughput of pattern matching, and thus enables efficient parallel detection of massive network data. Rathore et al. [65] proposed a real-time intrusion detection system using Hadoop [66] for efficient parallel processing in big data environments where data generation is fast. Tan et al. [67] designed a framework based on MapReduce [68] for cloud computing systems. It combines HIDS (Host-based Intrusion Detection System) with NIDS (Network-based Intrusion Detection System) for intrusion detection through collaborative agent detection sensors. However, Vasilomanolakis et al. [69] pointed out that the framework is not scalable enough to identify new advanced persistent threats. Solaimani et al. [70] developed a generic statistical real-time framework for multi-source stream data using Apache Spark [71] and Kafka. This detection architecture can improve the detection performance from the aspect of architecture deployment, but it cannot reasonably divide large-scale datasets to adapt to distributed algorithms. Winnower [21] is the most relevant provenance-based distributed detection system to Paradise. Winnower provides scalable compression using the DFA learning method across different executions to remove redundancy and generate DFA models. Furthermore, Winnower applies graph grammar principles to enable efficient behavioral modeling and comparison in the provenance domain. However, the high overheads associated with DFA learning are still a big obstacle for real-time requirements.

6 Discussion

Overall, Paradise is essentially an anomaly-based detection (AD) system, which generates a statistical model of normal behavior by monitoring the system during an attack-free period. It considers any activity out of the normal range as an anomaly. Therefore, comparison with detection methods [20, 72] based on abnormal training data is beyond the scope of our discussion on Paradise. These methods require specific knowledge about the attack strategies that a potential attacker could follow to discover a vulnerability. The major limitation of this technique is that it cannot correlate unknown attacks since their prerequisites and consequences are not defined. Moreover, attack profiles to be used for training are hard to find.

We assume that the number of system behavior patterns is finite and exhaustive. If Paradise observes a new normal behavior, it will send a false alert, although the false alarm problem exists in many IDSs [7, 19, 73], not only Paradise.

While producing virtually faultless results on the OPTC DARPA dataset, Paradise pulls just the characteristics of the neighbor graph for real-time consideration, rather than using the contextual knowledge of the multi-level ancestors and descendants of each process. In the future, we will perform experimental tests on more datasets to verify whether it is necessary to extract multiple levels of ancestors and descendants of each process of Paradise.

We set a time threshold M_{time} (one day in our experiments) instead of directly raising alerts for the monitored system. Paradise detects abnormal nodes in a streaming mode to detect threats in real time. Therefore, a benign node may be detected as abnormal before reaching its final stage. We set a waiting threshold M_{time} for a node to reach its final stage. If it has not changed to benign within M_{time}, we consider it an abnormal node. An incremental update process may detect a node several times under the streaming mode. Therefore, an abnormal node may be detected as benign at first. When the node has a new system call relationship within the waiting time M_{time}, it will be detected again. There is usually a time interval between the execution of two consecutive stages of a multi-step attack, and it is difficult to specify a quantitative time limit because the time interval between each stage can vary widely, depending on the attacker. A larger M_{time} means that the evolution time of the node is longer, which reduces the real-time performance of the system to ensure detection accuracy. We recommend that administrators maintain a whitelist of system entities associated with reliable software to maintain high detection

accuracy while ensuring real-time performance. Additionally, we will identify the abnormal node and issue an alarm when the node that labels abnormal occurrences reaches a certain number. Dynamically updating process node vectors in Paradise can be difficult. Paradise does not use the *concept drift* [19] technology for modeling to learn the system's evolution but instead increases the number of call relationships on the original node vector. We will continue to work on this topic in the future, such as taking the temporal *forgetting factor* [19] into account while normalizing the feature vector.

7 Conclusions

Efficient intrusion detection and analysis in big data environments has become a significant challenge for users today. This paper proposes a real-time, generalized, and distributed intrusion detection system named Paradise. It exploits the characteristics of provenance graphs by analyzing different types and amounts of dependency relationships, pruning and storing them in a highly efficient memory database, and then constructing vectors for different events in provenance graphs without extra data conversion. Thus it can be built on a wide variety of provenance collection systems. Since the process vector is an independent unit during the detection process, Paradise can integrate all the detection results from multiple detectors. Paradise also includes an effective performance monitoring and load balancing scheme that enhances the Kafka architecture to enable the efficient distribution of provenance graph feature vectors to the detection cluster. The experimental evaluation demonstrated the efficiency of our system and method.

Acknowledgment

This work was supported in part by the National Science Foundation of China under Grant No. 61972449, 61972448, U1705261, and 61821003, CCF-NSFOCUS Kun Peng research fund, and the Fundamental Research Funds for the Central Universities under Grant No.2019kfyXKJC021, and the Fundamental Research Funds for the Central Universities HUST under Grant No.2021JYCXJJ049. Corresponding author: Yulai Xie, ylxie@hust.edu.cn

References

[1] "APT trends report," https://www.w3.org/Submission/2013/SUBM-prov-json-20130424/, 2021.
[2] M. Landauer, F. Skopik, M. Wurzenberger, and A. Rauber, "System log clustering approaches for cyber security applications: A survey," *Computers & Security*, vol. 92, p. 101739, 2020.
[3] K. H. Lee, X. Zhang, and D. Xu, "High accuracy attack provenance via binary-based execution partition." in *Network and Distributed System Security Symposium*, 2013.
[4] S. Vakili, J. P. Langlois, B. Boughzala, and Y. Savaria, "Memory-efficient string matching for intrusion detection systems using a high-precision pattern grouping algorithm," in *Proceedings of the symposium on architectures for networking and communications systems*, 2016, pp. 37–42.
[5] X. Han, T. Pasquier, T. Ranjan, M. Goldstein, and M. Seltzer, "FRAPpuccino: fault-detection through runtime analysis of provenance," in *9th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 17)*, 2017.
[6] Y. Xie, D. Feng, Z. Tan, and J. Zhou, "Unifying intrusion detection and forensic analysis via provenance awareness," *Future Generation Computer Systems*, vol. 61, pp. 26–36, 2016.
[7] Y. Xie, D. Feng, Y. Hu, Y. Li, S. Sample, and D. Long, "Pagoda: a hybrid approach to enable efficient real-time provenance based intrusion detection in big data environments," *IEEE Transactions on Dependable and Secure Computing*, 2018.
[8] Y. Xie, Y. Wu, D. Feng, and D. Long, "P-Gaussian: provenance-based gaussian distribution for detecting intrusion behavior variants using high efficient and real time memory databases," *IEEE Transactions on Dependable and Secure Computing*, 2019.
[9] W. Jiang, H. Song, and Y. Dai, "Real-time intrusion detection for high-speed networks," *Computers & security*, vol. 24, no. 4, pp. 287–294, 2005.
[10] A. Gehani and D. Tariq, "SPADE: support for provenance auditing in distributed environments," in *Proceedings of the 13th International Middleware Conference*, 2012, pp. 101–120.
[11] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. I. Seltzer, "Provenance-aware storage systems." in *USENIX Annual Technical Conference, General Track*, 2006, pp. 43–56.
[12] D. J. Pohly, S. McLaughlin, P. McDaniel, and K. Butler, "Hi-Fi: collecting high-fidelity whole-system provenance," in *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012, pp. 259–268.
[13] T. Pasquier, X. Han, M. Goldstein, T. Moyer, D. Eyers, M. Seltzer, and J. Bacon, "Practical whole-system provenance capture," in *Proceedings of the Symposium on Cloud Computing*, 2017, pp. 405–418.
[14] X. Han, T. Pasquier, and M. Seltzer, "Provenance-based intrusion detection: opportunities and challenges," in *10th USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, 2018.
[15] "A json representation for the PROV data model," https://securelist.com/apt-trends-report-q1-2021/101967/, 2021.
[16] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, and J. Myers, "The open provenance model core specification (v1.1)," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 743–756, 2011.
[17] P. Groth and L. Moreau, "Representing distributed systems using the open provenance model," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 757–765, 2011.
[18] T. Zhu, J. Wang, L. Ruan, C. Xiong, J. Yu, Y. Li, Y. Chen, M. Lv, and T. Chen, "General, efficient, and real-time data compaction strategy for APT forensic analysis," *IEEE Transactions on Information Forensics and Security*, 2021.
[19] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "UNICORN: runtime provenance-based detector for advanced persistent threats," in *Network and Distributed System Security Symposium*, 2020.
[20] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "HOLMES: real-time APT detection through correlation of suspicious information flows," in *IEEE Symposium on Security and Privacy*, 2019.
[21] W. U. Hassan, L. Aguse, N. Aguse, A. Bates, and T. Moyer, "Towards scalable cluster auditing through grammatical inference over provenance graphs," in *Network and Distributed Systems Security Symposium*, 2018.
[22] M. Liu, Z. Xue, X. Xu, C. Zhong, and J. Chen, "Host-based intrusion detection system with system calls: Review and future trends," *ACM Computing Surveys*, vol. 51, no. 5, p. 98, 2018.
[23] T. Gibson, K. Schuchardt, and E. G. Stephan, "Application of named graphs towards custom provenance views." in *Workshop on the Theory and Practice of Provenance*, 2009.
[24] S. Shah, C. A. Soules, G. R. Ganger, and B. D. Noble, "Using provenance to aid in personal file search." in *USENIX Annual Technical Conference*, 2007, pp. 171–184.
[25] J. Cheney, "A formal framework for provenance security," in *IEEE 24th Computer Security Foundations Symposium*, 2011, pp. 281–293.
[26] A. Vahdat and T. E. Anderson, "Transparent result caching," in *USENIX Annual Technical Conference*, 1998.
[27] D. Devecsery, M. Chow, X. Dou, J. Flinn, and P. M. Chen, "Eidetic systems," in *11th USENIX Symposium on Operating Systems Design and Implementation*, 2014, pp. 525–540.
[28] O. Biton, S. Cohen-Boulakia, and S. B. Davidson, "Zoom* userviews: querying relevant provenance in workflow systems," in *Proceedings of the 33rd international conference on Very large data bases*, 2007, pp. 1366–1369.
[29] K. K. Muniswamy-Reddy, U. Braun, D. A. Holland, P. Macko, D. Maclean, D. Margo, M. Seltzer, and R. Smogor, "Layering in provenance systems," in *USENIX Annual*, 2009.
[30] K.-K. Muniswamy-Reddy, P. Macko, and M. I. Seltzer, "Provenance for the cloud." in *Conference on File and Storage Technologies*, vol. 10, 2010, pp. 15–14.
[31] J. H. Paik, "A novel TF-IDF weighting scheme for effective ranking," in *Proceedings of the 36th international conference on Research and development in information retrieval*, 2013, pp. 343–352.
[32] A. Alsaheel, Y. Nan, S. Ma, L. Yu, G. Walkup, Z. B. Celik, X. Zhang, and D. Xu, "ATLAS: A sequence-based learning approach for attack investigation," in *30th USENIX Security Symposium*, 2021.
[33] R. Badie, A. Aleahmad, M. Asadpour, and M. Rahgozar, "An efficient agent-based algorithm for overlapping community detection using nodes¡¯ closeness," *Physica A: Statistical Mechanics and its Applications*, vol. 392, no. 20, pp. 5231–5247, 2013.
[34] M. Barre, A. Gehani, and V. Yegneswaran, "Mining data provenance to detect advanced persistent threats," in *11th International Workshop on Theory and Practice of Provenance*, 2019.
[35] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "ZooKeeper: wait-free coordination for internet-scale systems." in *USENIX annual technical conference*. Boston, MA, USA, 2010.
[36] J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: a distributed messaging system for log processing," in *Proceedings of the NetDB*, 2011, pp. 1–7.
[37] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull, "Graphviz and dynagraph¡ªstatic and dynamic graph drawing tools," in *Graph drawing software*. Springer, 2004, pp. 127–148.
[38] M. Anjum, S. Iqbal, B. Hamelin *et al.*, "Analyzing the usefulness of the DARPA OpTC dataset in cyber threat detection research," *arXiv preprint*

arXiv:2103.03080, 2021.
[39] "The description of the optc dataset groundtruth." https://drive.google.com/drive/folders/1n3kkS3KR31KUegn42yk3-e6JkZvf0Caa, 2021.
[40] "CamFlow trace," https://github.com/yulaixie/yulaixie.github.io/tree/master/datasets/camflow, 2020.
[41] "Cyber analytics repository model." https://car.mitre.org/data_model/, 2019.
[42] G. M. Alqaralleh, M. A. Alshraideh, and A. Alrodan, "A comparison study between different sampling strategies for intrusion detection system of active learning model." *J. Comput. Sci.*, vol. 14, no. 8, pp. 1155–1173, 2018.
[43] X. Han, J. Mickens, A. Gehani, M. Seltzer, and T. Pasquier, "Xanthus: Push-button orchestration of host provenance data collection," in *Proceedings of the 3rd International Workshop on Practical Reproducible Evaluation of Computer Systems*, 2020, pp. 27–32.
[44] "Stress-ng," https://github.com/ColinIanKing/stress-ng, 2014.
[45] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for unix processes," in *IEEE Symposium on Security and Privacy*, 1996, pp. 120–128.
[46] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," *Journal of computer security*, vol. 6, no. 3, pp. 151–180, 1998.
[47] G. Jiang, H. Chen, C. Ungureanu, and K. Yoshihira, "Multiresolution abnormal trace detection using varied-length n-grams and automata," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 37, no. 1, pp. 86–97, 2006.
[48] Y. Liao and V. R. Vemuri, "Use of k-nearest neighbor classifier for intrusion detection," *Computers & security*, vol. 21, no. 5, pp. 439–448, 2002.
[49] X. Shu, D. D. Yao, and B. G. Ryder, "A formal framework for program anomaly detection," in *International Symposium on Recent Advances in Intrusion Detection*, 2015, pp. 270–292.
[50] G. Creech and J. Hu, "A semantic approach to host-based intrusion detection systems using contiguousand discontiguous system call patterns," *IEEE Transactions on Computers*, vol. 63, no. 4, pp. 807–819, 2013.
[51] X. Sun, J. Dai, P. Liu, A. Singhal, and J. Yen, "Using bayesian networks for probabilistic identification of zero-day attack paths," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 10, pp. 2506–2521, 2018.
[52] X. Shu, J. Zhang, D. D. Yao, and W.-C. Feng, "Fast detection of transformed data leaks," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 3, pp. 528–542, 2015.
[53] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, pp. 41–50, 2018.
[54] E. Nwafor, A. Campbell, and G. Bloom, "Anomaly-based intrusion detection of IoT device sensor data using provenance graphs," in *1st International Workshop on Security and Privacy for the Internet-of-Things*, 2018.
[55] W. Zhou, Q. Fei, A. Narayan, A. Haeberlen, B. T. Loo, and M. Sherr, "Secure network provenance," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, 2011, pp. 295–310.
[56] A. Bates, D. J. Tian, K. R. Butler, and T. Moyer, "Trustworthy whole-system provenance for the Linux kernel," in *Proceedings of the USENIX Security Symposium*, 2015.
[57] A. Bates, W. U. Hassan, K. Butler, A. Dobra, B. Reaves, P. Cable, T. Moyer, and N. Schear, "Transparent web service auditing via network provenance functions," in *International Conference on World Wide Web*, 2017.
[58] G. Jenkinson, L. Carata, T. Bytheway, R. Sohan, R. N. Watson, J. Anderson, B. Kidney, A. Strnad, A. Thomas, and G. Neville-Neil, "Applying provenance in APT monitoring and analysis: Practical challenges for scalable, efficient and trustworthy distributed provenance," in *9th USENIX Workshop on the Theory and Practice of Provenance*, 2017.
[59] T. Pasquier, X. Han, T. Moyer, A. Bates, O. Hermant, D. Eyers, J. Bacon, and M. Seltzer, "Runtime analysis of whole-system provenance," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2018, pp. 1601–1616.
[60] C. C.-C. Cheng, C. Shi, N. Z. Gong, and Y. Guan, "EviHunter: identifying digital evidence in the permanent storage of android devices via static analysis," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2018, pp. 1338–1350.
[61] Y. Kwon, F. Wang, W. Wang, K. H. Lee, W.-C. Lee, S. Ma, X. Zhang, D. Xu, S. Jha, and G. F. Ciocarlie, "MCI: modeling-based causality inference in audit logging for attack investigation." in *Network and Distributed System Security Symposium*, 2018.
[62] Y. Tang, D. Li, Z. Li, M. Zhang, K. Jee, X. Xiao, Z. Wu, J. Rhee, F. Xu, and Q. Li, "Nodemerge: template based efficient data reduction for big-data causality analysis," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2018, pp. 1324–1337.
[63] G. Vasiliadis, M. Polychronakis, and S. Ioannidis, "MIDeA: a multi-parallel intrusion detection architecture," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, pp. 297–308.
[64] M. A. Jamshed, J. Lee, S. Moon, I. Yun, D. Kim, S. Lee, Y. Yi, and K. Park, "Kargus: a highly-scalable software-based intrusion detection system," in *Proceedings of the ACM conference on Computer and communications security*, 2012, pp. 317–328.
[65] M. M. Rathore, A. Ahmad, and A. Paul, "Real time intrusion detection system for ultra-high-speed big data environments," *The Journal of Supercomputing*, vol. 72, no. 9, pp. 3489–3510, 2016.
[66] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *IEEE 26th symposium on mass storage systems and technologies (MSST)*, 2010, pp. 1–10.
[67] Z. Tan, U. T. Nagar, X. He, P. Nanda, R. P. Liu, S. Wang, and J. Hu, "Enhancing big data security with collaborative intrusion detection," *IEEE cloud computing*, vol. 1, no. 3, pp. 27–33, 2014.
[68] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox, "Twister: a runtime for iterative mapreduce," in *Proceedings of the 19th ACM international symposium on high performance distributed computing*, 2010, pp. 810–818.
[69] E. Vasilomanolakis, S. Karuppayah, M. Mühlhäuser, and M. Fischer, "Taxonomy and survey of collaborative intrusion detection," *ACM Computing Surveys (CSUR)*, vol. 47, no. 4, pp. 1–33, 2015.
[70] M. Solaimani, M. Iftekhar, L. Khan, and B. Thuraisingham, "Statistical technique for online anomaly detection using spark over heterogeneous data from multi-source vmware performance data," in *IEEE International Conference on Big Data*, 2014, pp. 1086–1094.
[71] S. Salloum, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang, "Big data analytics on Apache Spark," *International Journal of Data Science and Analytics*, vol. 1, no. 3-4, pp. 145–164, 2016.
[72] M. Ficco, "Security event correlation approach for cloud computing," *International Journal of High Performance Computing and Networking 1*, vol. 7, no. 3, pp. 173–185, 2013.
[73] J. Zeng, L. C. Zheng, Y. Chen, K. Ji, and J. Mao, "WATSON: Abstracting behaviors from audit logs via aggregation of contextual semantics," in *Network and Distributed System Security Symposium*, 2021.

Yafeng Wu received master's degree in computer science from Huazhong University of Science and Technology in 2019. He is currently pursuing his Ph.D. degree in Huazhong University of Science and Technology (HUST). His research interests mainly include digital provenance and intrusion detection.

Yulai Xie received the B.E. and Ph.D. degrees in computer science from Huazhong University of Science and Technology (HUST), China, in 2007 and 2013, respectively. He was a visiting scholar at the University of California, Santa Cruz in 2010 and a visiting scholar at the Chinese University of Hong Kong in 2015. He is now an associate professor in School of Cyber Science and Engineering, in HUST, China. His research interests mainly include cloud storage and virtualization, digital provenance, intrusion detection, machine learning and computer architecture.

Xuelong Liao was born in 1996. He received master's degree in computer science from Huazhong University of Science and Technology in 2019. His research interests include intrusion detection and computer architecture.

Pan Zhou is currently an associate professor with Hubei Engineering Research Center on Big Data Security, School of Cyber Science and Engineering, Huazhong University of Science and Technology (HUST), Wuhan, P.R. China. He received his Ph.D. in the School of Electrical and Computer Engineering at the Georgia Institute of Technology (Georgia Tech) in 2011, Atlanta, USA. He received his B.S. degree in the Advanced Class of HUST, and a M.S. degree in the Department of Electronics and Information Engineering from HUST, Wuhan, China, in 2006 and 2008, respectively. He held honorary degree in his bachelor and merit research award of HUST in his master study. He was a senior technical member at Oracle Inc., America, during 2011 to 2013, and worked on Hadoop and distributed storage system for big data analytics at Oracle Cloud Platform. He received the 'Rising Star in Science and Technology of HUST' in 2017. His current research interest includes: security and privacy, big data analytics and machine learning, and information networks.

Dan Feng received her B.E, M.E. and Ph.D. degrees in Computer Science and Technology from Huazhong University of Science and Technology (HUST), China, in 1991, 1994 and 1997 respectively. She is a professor and director of Data Storage System Division, Wuhan National Lab for Optoelectronics. She is also dean of the School of Computer Science and Technology, HUST. Her research interests include computer architecture, massive storage systems, parallel file systems, disk array and solid state disk. She has over 100 publications in journals and international conferences, including FAST, USENIX ATC, ICDCS, HPDC, SC, ICS and IPDPS. Dr. Feng is a member of IEEE and a member of ACM.

Lin Wu received her M.E. degree from Guangxi Normal University, in 2020. She is currently pursuing her Ph.D. degree in Huazhong University of Science and Technology (HUST). Her research interests are digital provenance and intrusion detection.

Xuan Li received the Ph.D. degree in control science and control engineering at the School of Automation, Huazhong University of Science and Technology, Wuhan, China. After completion of his degree, Dr. Li served as a researcher at NSFOCUS Inc. A9-3/F Optical Valley Software Park, Wuhan, China. His main research interests include edr, intrusion detection and response, threat modelling, and DevSecOps.

Avani Wildani received her B.S. degree in Computer Science and Mathematics at Harvey Mudd College and her Ph.D. in Computer Science at UC Santa Cruz. She is now an assistant professor in the Department of Computer Science, in Emory University, US. Her interests mainly include information storage and retrieval across different storage models, with application domains including access prediction, data deduplication, archival economics, power management, wireless mesh networks, auditory receptive field characterization, and pollution monitoring.

Darrell Long received his B.S. degree in Computer Science from San Diego State University, and his M.S. and Ph.D. from the University of California, San Diego. Dr. Darrell D.E. Long is Distinguished Professor of Computer Engineering at the University of California, Santa Cruz. He holds the Kumar Malavalli Endowed Chair of Storage Systems Research and is Director emeritus of the Storage Systems Research Center. His current research interests in the storage systems area include high performance storage systems, archival storage systems and energy-efficient storage systems. His research also includes computer system reliability, video-on-demand, applied machine learning, mobile computing and cyber security. Dr. Long is Fellow of IEEE and Fellow of the American Association for the Advancement of Science (AAAS).