APTSHIELD: A Stable, Efficient and Real-Time APT Detection System for Linux Hosts

Tiantian Zhu †, Jinkai Yu, Chunlin Xiong ‡, Wenrui Cheng †, Qixuan Yuan †, Jie Ying †, Tieming Chen †, Member, IEEE, Jiabo Zhang, Mingqi Lv ‡, Yan Chen †, Fellow, IEEE, Ting Wang †, and Yuan Fan

Abstract—Advanced Persistent Threat (APT) attacks have caused massive financial loss worldwide. Researchers thereby have proposed a series of solutions to detect APT attacks, such as dynamic/static code analysis, traffic detection, sandbox technology, endpoint detection and response (EDR), etc. However, existing defenses are failed to accurately and effectively defend against the current APT attacks that exhibit strong persistent, stealthy, diverse and dynamic characteristics due to the weak data source integrity, large data processing overhead and poor real-time performance in the process of real-world scenarios. To overcome these difficulties, in this paper we propose APTSHIELD, a stable, efficient and real-time APT detection system for Linux hosts. In the aspect of data collection, audit is selected to stably collect kernel data of the operating system so as to carry out a complete portrait of the attack based on comprehensive analysis and comparison of existing logging tools; In the aspect of data processing, redundant semantics skipping and non-viable node pruning are adopted to reduce the amount of data, so as to reduce the overhead of the detection system; In the aspect of attack detection, an APT attack detection framework based on ATT&CK model is designed to carry out real-time attack response and alarm through the transfer and aggregation of labels. Experimental results on both laboratory and Darpa Engagement show that our system can effectively detect web vulnerability attacks, file-less attacks and remote attack trojan attacks, and has a low false positive rate, which adds far more value than the existing frontier work.

Index Terms—Advanced persistent threat, data collection, data compaction, real-time detection.

I. INTRODUCTION

ADVANCED Persistent Threat Attacks are escalating to the harm of the current society, and it is often organized by groups of hackers with certain national, governmental or other organizational backgrounds—the hackers are usually well-organized, well-targeted, highly skilled and aggressive, often against the government, core infrastructure (e.g., energy, transportation, communication) and key industries (e.g., military, finance, health care). APT attacks have occurred frequently in recent years, showing a high incidence of high risk in the world, such as: Stuxnet worm attack in Iranian nuclear power plant [1], BlackEnergy virus attack in Ukraine’s power system [2], and the user information leakage attack in Target [3]. The COVID-19-themed attacks have also been frequently reported [4], e.g., APT groups have tried to attack firms working on COVID-19 vaccines and they have used spear-phishing emails to entice users to download and execute malicious attachments so as to steal target-related data and destroy medical infrastructure [5].

APT detection has become an important research topic widely concerned with academia and industry field. A series of traditional attack detection schemes such as static code analysis [6], [7], dynamic sandbox detection [8], [9], malicious traffic analysis [10], [11], and hooking technology [12], [13] can be applied to fight against APT attacks. However, the detection effect of the above methods is not ideal due to the problems such as weak de-obfuscate ability, high computational overhead, and low system stability in the practical use. Furthermore, with the extension of the network boundaries and the increment of various 0-day vulnerabilities, it is almost impossible to continue to use these traditional methods to detect APT attacks. Therefore, the idea of Detection and Response (DR) came into being, which is divided into Endpoint Detection and Response (EDR) and Network Detection and Response (NDR). In EDR, the relevant security agent will collect and analyze activity data of the application program when it is running in the user’s host/endpoint [14]. While NDR will monitor how threats enter the network and how they move laterally in the network [15].

Nowadays, the market for EDR solutions is expanding rapidly. To this end, the DARPA Transparent Computing (TC) program has tried to organize multiple scientific research units to conduct engagements to enable the prompt detection of APTs and other cyber threats [16]. EDR tools usually record a large number of system events to a central database. By adopting techniques

1545-5971 © 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
such as indicators of compromise (IOC), behavior analysis, and machine learning to analyze data, it is hoped that threats will be detected and responded to at an early stage.

However, in the life cycle of APT attacks, the current EDR system (especially the EDR system of the Linux host) does not form a comprehensive and effective solution to the problems of selecting data sources for detection, massive data analysis and storage, association of suspicious behaviors in context, detection of 0-day threats, as well as energy consumption and real-time performance in detecting APT attacks. The main challenges of the current EDR system based on the Linux host are as follows:

1. **How to select reliable, stable and semantically rich data sources.** To carry out complete and accurate APT detection and forensics analysis in the enterprise environment, it is necessary to retain the complete log data of the entire enterprise for subsequent analysis. Nowadays, there are a few studies focusing on the collection and analysis of system events. However, the existing endpoint system log collection tools are varied, and few work are able to analyze and evaluate these tools under the current Linux system by combing with collecting performance, data integrity, and availability.

2. **How to reduce the amount of data required for real-time detection to improve the detection efficiency.** The duration of APT attacks is usually much longer than other attacks. According to the report from Trustwave [17], the average latency time of APT attacks is about 83 days, and some of them are as long as several years. Massive data of APT attacks have made new requirements for analysts: It is critical to improve real-time detection efficiency and reduce storage overhead by effective data compaction strategy. Some existing data compaction work [18], [19], [20], [21], [22], [23] tries to use fine-grained taint tracking technology to delete redundant events. However, these methods always rely on known software models which are lack of generalization. Although some studies [24], [25], [26] have proposed general data compaction methods based on audit log, their algorithms read the dependency graph composed of long-term log data into memory at one time, resulting in huge computing and memory overhead.

3. **How to construct a real-time APT detection framework with high adaptability, high coverage, high precision, low false positives and constant memory overhead.** The contextual data analysis studies [27], [28] usually preserve context information via the provenance graph. However, the size of the provenance graph will explode over time due to the long duration of APT attacks, rendering these approaches inevitably suffer from efficiency and memory problems [29]. The ideal detection scheme should have good expansibility, be able to cover different types of sophisticated APT attacks, effectively define APT related suspicious behaviors, and comprehensively utilize the full contextual attack chain to realize real-time and accurate detection of attacks.

To cope with the above challenges, we propose a stable, efficient and real-time APT detection system for Linux hosts, called APTSHIELD. First, in order to select reliable, stable and semantically rich data sources, we make a comprehensive and detailed evaluation on Linux system log collection tools. Second, for the purpose of reducing the storage overhead and improve the detection efficiency, we reduce the amount of log data by means of redundant semantic skipping and non-viable node pruning. Finally, we construct an APT attack detection framework based on ATT&CK model [30], and the framework can carry out real-time attack response and alarm through the transfer and aggregation of labels. In general, the contributions of this paper are as follows:

- Different from previous studies, we make a detailed comparison and analysis of the advantages and disadvantages of the existing data source. We deploy multiple indicators and then obtain the optimal data sources for stable APT detection on Linux through performance analysis and comprehensive judgment.
- We use redundant semantics skipping and non-viable entity pruning to improve the efficiency of real-time APT detection and reduce data storage overhead for forensic analysis. Our data compaction methods can be carried out in real-time streaming data to ensure the performance of the detection system. Also, the compaction effect is better than that of existing studies without affecting the accuracy of final results.
- Based on the ATT&CK model, we constructed an information aggregation framework through system data flow and control flow. With the help of tactics, techniques and procedures (TTP), the atomic suspicious characteristics of system entities and their transmission rules are defined in the framework to realize the aggregation of the contextual information of APT attacks. Different from traditional single-point detection methods, our framework can aggregate the information of the attack chain into specific entities, and realize the whole chain detection and real-time alarm of APT attacks with constant memory overhead.
- We implement a stable, efficient and real-time APT detection prototype system on Linux and conduct experiments on the dataset of Darpa Engagement as well as the dataset that simulates APT attacks in real-world scenarios. The experimental results show that our system can effectively detect web vulnerability attacks, file-less attacks and remote access trojan attacks in a real-time manner, and has a low false positive rate, which adds far more value than the existing frontier studies.

II. RELATED WORK

In this section, we review notable studies in APT detection and compare with them in data collection, data reduction and detection performance to highlight the novelty of our approaches.

**Data Collection.** In order to carry out practical and effective APT detection, it’s crucial to select the suitable data source which meets the characteristics of non-tampering, low resource consumption, and stability. Network data flow [31], [32] are widely used by previous studies for intrusion analysis and detection, but its universality was insufficient to record all attack operations. The application data [33] shows the inherent characteristics of the program when it is running, but the analyst cannot
associate the information of application level with that of system level. Also, some studies [34], [35] use instrumentation to collect taint records, which may cause huge memory overhead. Some logging tools are also used by researchers, such as NanoLog [36], Log4j2 [37], Spdlog [38], Glog [39], and BoostLog [40]. Nanolog is a nanosecond logging system, but it can only operate on some static strings. Although the remaining systems (i.e., Log4j2, Spdlog, Glog, and BoostLog) are able to meet some requirements, such as low resource consumption and fidelity, they are highly coupled with the application itself, and cannot be used as a general log infrastructure. In view of the deficiencies of the above data sources, kernel-based data collection tools are used by researchers, such as Auditd [41], Sysdig [42], Lttng [43], and Auditbeat [44]. However, there are few studies that conduct a complete and detailed analysis of these popular data collection tools based on the needs of the real-world environment.

Different from the previous work, in this paper we compare the performance of the above four kernel-based data collection tools when the system is no/full load through experiments.

Data Reduction. LogGC [18] pioneered the idea of garbage collection for audit log. The authors of LogGC combined with BEEP [19] to delete events that had no lasting impact on the system. Subsequently, NodeMerge [45] presented a template-based data compaction system with the assistance of an improved FP-Growth algorithm. Some specific patterns such as image loading and system configuration were extracted for data compaction. Similarly, Conan [29] prefILTERed the duplicated read events through semantic recognition to reduce the detection efficiency. Zhu et al. [46] maintained a long list to record redundant events. However, the above three methods are only applicable to specific event types, and have limitations in complex scenarios (e.g., when files are accessed by multiple processes, or there are a large number of file write operations, or the real-time performance of the system will deteriorate as the collection time increases).

ProTracer [20] tried to improve the compaction rate by dividing the target program into multiple units to perform fine-grained taint tracking. Later, MPI [21] improved ProTracer by utilizing a semantics aware program annotation and instrumentation technique to partition execution, which was able to generate execution partitions with rich semantic information. To avoid application instrumentation or kernel modification, MCI [22] utilized LDX [23] to acquire precise causal models for a set of primitive operations. The compaction effect of above methods depends on a large number of software models. However, there are a lot of unpredictable software running in the real-world production environment, and it is very difficult to ensure the coverage of these methods. In addition, software updates may invalidate the above methods.

Xu et al. [47] presented the concept of trackability. By aggregating events under the same trackability (i.e., deleting multiple equivalent data streams and retaining only one of them), they could reduce a large portion of data while preserving events relevant to a forensic analysis. However, this method only considers the characteristics of a single node instead of the global semantics, which makes the compaction effect very limited. To solve this problem, both Hossain et al. [24] and Hassan et al. [26] presented the data compaction algorithm based on the global semantics of the provenance graph. But during the calculation, the whole data of the provenance graph needs to be read into memory at once, which brings additional I/O and memory overhead. Furthermore, the above methods cannot process real-time streaming data and guarantee the real-time performance of APT detection.

In contrast, APTShield is able to compact (i.e., redundant semantics skipping and non-viable node pruning) the real-time streaming data to improve the efficiency of real-time APT detection, reduce data storage overhead for forensic analysis, and achieve a better compaction effect than existing work without affecting the accuracy of final result.

APT Detection. In order to effectively monitor APT attacks, provenance graphs generally record the coarse-grained data from the kernel level [48], including subjects (e.g., processes and sockets), objects (e.g., files), and edges (e.g., system events). BackTracker [49] and PriorTracker [50] tried to use provenance graph to identify the entry point and the effect of the attack through backward tracking and forwarding tracking, respectively. Sun et al. [51] utilized Bayesian Networks to identify zero-day attack paths on the provenance graph. Similarly, NoDoze [52] performed attack triage within the provenance graph to find anomalous paths. During the provenance-based attack investigation, analysts usually came across the dependency explosion problem. To mitigate this problem, Ma et al. [20] presented a lightweight provenance tracing system to reduce the memory overhead via unit-based execution partitioning [19], as well as their improved studies [21],[53]. As discussed in data reduction, the above three methods mainly rely on software models and are not universal.

Due to the persistence of APT attacks, it is difficult for a security analyst to pick out “needle-in-a-haystack” attacks. Learn-based APT detection on long-term provenance graphs has been proposed by some researchers. Barre et al. [54] tried to extract statistic features of key processes, by adopting a random forest model, their system was able to deliver a detection rate of about 50%. The results show that simple feature engineering without considering the context of the provenance graph cannot effectively characterize complex APT attacks. Berrada et al. [55] designed a comprehensive experiment on provenance graphs by combining existing techniques for APT anomaly detection and they found that simple score or rank aggregation techniques were effective at improving detection performance. Han et al. [56] presented UNICORN which could model system behaviors via structured provenance graphs with a graph sketching technique. Although UNICORN requires no prior expert knowledge of APT attack patterns or behaviors, the anomaly-based system will undoubtedly introduce a large number of false alarms, which is difficult to be practical in real-world scenarios.

In fact, rule-based APT detection is more in line with commercial needs, it has been shown that rule-based EDR systems are suitable for addressing the noise problem [57] (i.e., false alarms and duplicate alerts). Expert policies have been proposed by Sleuth [27] and Holmes [28] for attack reconstruction on provenance graphs. To match possible exploits of localized components in the provenance graph, empirical labels and prior specifications were used by Sleuth and Holmes, respectively.
However, the size of the provenance graph would explode over time due to the long duration of APT attacks, rendering these two approaches inevitably suffer from efficiency and memory problems. Poirot [58] tried to detect APT attacks by measuring the offline similarity between a provenance graph and a query graph. The query graph was obtained based on the expert knowledge from cyber threat intelligence (CTI). The disadvantage of Poirot is that only the occurred attacks are able to be further detected. It is a critical limitation given that composing the elaborate description of a new category of APT requires significant forensics efforts. Conan [29] detected APT attacks by a three-phase model and pre-defined rules. Although Conan is capable of capturing the suspicious behaviour in a real-time manner, it has the following two problems. First, in the case of a large number of concurrent operations in the system (e.g., a large number of file read and write operations at the same time), the real-time performance of Conan cannot be guaranteed because the event generation rate of the target host may be much larger than the event throughput of the detection system. Second, the three-phase model proposed by Conan only reflects the path of suspicious data/control flow, and it has no performance on the tactics used by attackers. That is to say, it lacks interpretable guidance for the analysis of security analysts. Unlike the previous methods, APTSHIELD can detect long-term APT attacks with a constant level of memory overhead. By adopting the data compaction strategies, APTSHIELD is capable of guaranteeing the real-time performance. Also, APTSHIELD is able to cover different types of sophisticated APT attacks (i.e., web vulnerability attacks, file-less attacks and remote access trojan attacks), effectively define APT related suspicious behaviors based on ADIT&CK model, and comprehensively utilize the full contextual attack chain to realize accurate detection of attacks.

III. BACKGROUND

In this section, we first introduce system entities and system events, followed by the dependency graph composed of them. Then, we describe the dependency relationship in the dependency graph. Finally, we introduce how to use provenance graph for forensic analysis.

A. System Entity and System Event

Generally, the system entity is able to be divided into two categories: subject and object. The subject is the initiator of the event. In most cases, the subject refers to the process. Specifically, users are treated as subjects when there is a user-related event. The object is the target of the event, such as files, network IPs, and channels. It is worth mentioning that processes (e.g., child processes) can also be treated as objects. System events usually record the actions initiated by the subject to the object, such as file reading, file writing, process creation, etc.

B. Dependency Graph

Dependency graph $G$ can be represented as a combination of $(V, E)$, where $V$ represents all nodes (system entities) and $E$ stands for all edges (system events). For any edge $e \in E$, there is $e = (u, v, t)$, where $u, v \in V$, $u$ represents the subject, $V$ represents the object, and $t$ represents the time stamp of the event. For the two edges in the dependency graph, $e_1 = (u_1, v_1, t_1)$, $e_2 = (u_2, v_2, t_2)$, we consider that there is a dependence between $e_1$ and $e_2$ if $v_1 = u_2$ and $t_1 < t_2$.

C. Forensic Analysis

Forensic analysis is also known as trace analysis, and its main purpose is to help analysts understand when, how and what impact the attack has been made. In the process of APT analysis, analysts have two tasks. The first one is to determine the entrance of the attack (e.g., the initial process and file entering the terminal and the source IP/port of the attack). The analysis process to achieve this goal is called backward analysis, which is tracing suspicious information flow from the process marked as suspicious through the opposite direction of information flow. Backward analysis was first proposed by BackTracker [49]. During backward analysis, the timestamp is used to judge the causa relationship between different events. For the second task, forward analysis is used to analyze the impact of attacks [24], [46], such as accessing to sensitive information, tampering with system configuration, etc. The origin of forward analysis is usually the attack entry point obtained by backward analysis or any point in the attack chain.

IV. SYSTEM DESIGN

In this section, we first introduce the threat model and the overall architecture of our proposed APT detection system. We then discuss several important topics in its design, including data collection, data compaction, and APT detection framework.

A. Threat Model

Based on famous APT attack, RSA SecurID tokens leakage [59], we considers the following attack scenarios: an employee received an email with the words "recruitment plan". The employee downloaded and opened the attachment, then was hit by the latest 0-day vulnerability. At the same time, the attacker established a command&control connection with the employee's host, continuously downloaded and executed the remote access Trojan that the attacker had prepared, which was not able to be detected via the anti-virus software. As a result, the attacker now had access to the target host in the corporate environment. Further, the attacker used this machine as a springboard to perform lateral movement. And the other employee’s or senior manager’s host that associated with the employee’s were successively controlled by the attacker. Then the attacker could stay for months or even years until he completed the ultimate goal, such as the theft of confidential documents or the destruction of the integrity of the system.

In the threat model of this article, the logs generated by the operating system are considered credible. We assume that the system is not attacked before the start of data collection. Attackers can compromise the target host in a variety of ways, including but not limited to 0-day vulnerabilities, Webshell, and RAT.
primary task of the attacker is to run the malicious code on the victim’s machine, and then collect target information through remote command&control. Attackers will hide themselves in normal system activities through disguise, with the ultimate goal of stealing high-value data or destroying the integrity and availability of the system.

B. System Overview

The architecture of APTSHIELD is illustrated in Fig. 1. It consists of three parts: data collection module, data compaction module, and APT detection framework. First, kernel-based tools are adopted for data collection. Through comprehensive analysis, we choose auditd as the collector on the client side since it provides sufficient data with low overhead. The collected system logs (including entities and events) will construct a dependency graph. Second, redundant semantics skipping and nonviable node pruning will be adopted to compact system logs. The main advantage of data compaction is to improve the efficiency of real-time APT detection and reduce memory costs for forensic analysis. Third, APT detection framework is proposed based on the ATT&CK model. The atomic suspicious characteristics of system entities (i.e., process labels and file labels) and their transfer rules (through events with different semantics) are defined in the framework to realize the aggregation of the contextual information of APT attacks. The APT will be alerted from a specific entity (through judgement rules), and finally the related attack chain will be found through forensic analysis.

C. Data Collection

In order to detect APT in a real-time manner with high efficiency, it is crucial to select optimal data sources and data collectors. To this end, in this section we first sort out the data sources for APT detection under the Linux system, and then analyze and select the best data collector that meets the requirements.

1) Data Requirements: According to the ATT&CK model, the host-based APT detection system needs to collect relevant data from different stages (e.g., initial access, untrusted execution, data exfiltration, etc.). We summarize requirements of both data sources and data collectors into the following three aspects:

1) Tamper-proof. APT attacks are camouflaged. In order to avoid detection, APT attackers often hide their attack traces (e.g., bash history and log message can be easily modified/deleted by an attacker, thereby increasing the difficulty of attack detection). We need to select credible data sources to ensure that intruders cannot avoid detection by tampering with data.

2) Stability and Low Overhead. APT attacks are persistent. That means data collection is a long-term process. First, the data collector needs to be directly deployed on the user’s host. If the data collection takes up too much resources, it will affect the user’s normal usage. Second, the detection system needs to perform stable and real-time detection and alarm, this requires data to be collected and transmitted in real-time without any data loss. Third, the operation of the data collector cannot affect the stability of the system (e.g., it should not be requested to modify the kernel).

3) Rich semantics. APT attacks are diverse. It is necessary to analyze a series of data such as file IO data, network IO data, inter-process communication data, process attribute modification data, sensitive system operation data, and file attribute modification data. Based on three requirements above, the kernel-level logs are used as the main data source. We analyze and select the best data collector currently available on Linux, which will be described in details in the next section.

2) Data Source Analysis and Selection: As introduced in Section II, current data collectors that are active in the community are sysdig, auditd, lttng, and auditbeat. With the Intel Xeon E5 CPU (8 cores) and 64G memory running on Ubuntu 16.04, we analyze and compare the performance (CPU and memory usage) of these data collectors when the host is no-load and full-load (i.e., use commands to increase the CPU usage to more than 95%). The executed commands include but are not limited to file reading and writing, file downloading, inter-process communication, executing shell scripts, accessing pages, etc.). The results are shown in Table I.

It can be seen from Table I that auditd and auditbeat have better overall performance when the server is fully loaded. The overhead (memory) of auditd is the same under both no-load and full-load because its memory usage is related to the buffer size. In the actual test, the buffer size can be controlled within 40MB to meet the data demand under full load. If sysdig is used as a collector, there will be service crashes and data loss.
when the system is fully loaded. The output data generated by lttng is a binary byte stream, and parsing requires a lot of costs. Considering that auditd is a module that comes with the kernel and has the lowest cost, we finally select auditd as the data collector in this article. In addition, we use the stress testing tool UnixBench [60] to monitor the state of auditd when it is turned on/off. The results show that when auditd is turned on, the total score of UnixBench drops about 0.3% compared to that is turned off, which is basically negligible.

D. Data Compaction

If the collected raw data is sent directly to the server for processing, it will consume a lot of storage and network resources due to the long duration of the APT attack. Therefore, in the real-world enterprise environment, these data should be compacted first, and then sent to the storage/detection server. The data compaction method should meet the following conditions: (1) Real-time. The data consumption rate should be greater than the data generation rate to avoid the impact of cached data on the real-time performance of the detection system; (2) High efficiency. The data compaction method has low CPU and memory usage, and will not affect the operating system itself. Also, it will greatly reduce the data transmission bandwidth and data storage cost; (3) High accuracy. The compacted data should maintain the dependencies in the original data, and should not have a negative impact on the APT detection and forensic analysis of APT attacks. In this section, we will first give the insight for data compaction. Then we introduce two algorithms in our data compaction method: redundant semantics skipping and non-viable node pruning.

1) Insight for Data Compaction: In order to perform data compaction without affecting the detection results, it is important to analyze the contextual semantics of dependency graph. From the perspective of semantic transfer, we assume that entity \( u \) has an event \( e_1 \) directed to entity \( v \) at time \( t_1 \). It is considered that entity \( u \) has an impact on entity \( v \) at time \( t_1 \). Then at time \( t_2 \) \((t_2 > t_1)\), there is an event \( e_2 \) pointing to entity \( w \) from entity \( v \), so it is considered that entity \( v \) has an impact on entity \( w \) at time \( t_2 \). Since the event \( e_2 \) occurs after the event \( e_1 \), it can be considered that the entity \( u \) has an impact on the entity \( w \), which is transmitted through the entity \( v \).

Take Fig. 2 as an example. The circle represents the process, the rectangle represents the file, the diamond represents the IP/DNS, the arrow represents the event, and the number on the arrow represents the time point of the events (the smaller the number, the earlier the event occurs). Process \( P \) is affected by the event generated by \( x.com \) at time \( t_1 \), and \( P \) obtains semantic information from the network. After that, the system event with process \( P \) as the initial entity will have a direct impact on other entities (i.e., file \( A \), file \( B \), and process \( Q \)). However, considering that process \( P \) is only affected by the event generated by \( x.com \) before time \( t_5 \). It can be judged that from time \( t_1 \) to time \( t_5 \), other entities pointed by process \( P \) are only affected by the network information from \( x.com \) to \( P \). After the event at time \( t_5 \) occurs, other entities pointed by process \( P \) are simultaneously affected by the network information from both \( x.com \) to \( P \) and \( y.com \) to \( P \). According to observations above, we can see that the impact of the event on the entity is lasting. From the perspective of contextual semantics, when an entity receives an event containing external information, the entity will contain certain characteristics transmitted by the event. And these characteristics will affect the flow of information from this entity. Our main idea/insight of data compaction is derived from these observations: If the semantics of source entity has not changed and the destination entity has received multiple same information streams (events) from the source entity, these events will have the same semantics, which can be combined to delete redundant semantics.

The result of redundant semantics skipping is shown in Fig. 2. The data compaction process is as follows:

1) After the time \( t_1 \), process \( P \) already has the semantics from \( x.com \), and the event at \( t_2 \) and \( t_11 \) do not introduce any new semantics and can be deleted.

2) Although during \( t_5 \) and \( t_7 \) an event (at \( t_6 \)) from \( P \) to \( Q \) occurred, it only affects the semantics of \( Q \) and the semantics of \( P \) remains unchanged. The events at time \( t_5 \) and \( t_7 \) have the same semantics, and we can delete the event at time \( t_7 \).

3) We can find that events at \( t_4, t_8 \), and \( t_9 \) have the same subject and object. The event at time \( t_5 \) has changed the semantics of \( P \), so the event at time \( t_8 \) should be retained. Since the events at time \( t_8 \) and time \( t_9 \) have the same semantics, we can delete the event at time \( t_9 \).

Note that in this example, Process \( P \) is communicating with \( x.com \) at \((t_1, t_2)\), then \( y.com \) at \((t_5, t_7)\), and again with \( x.com \) at \((t_11, t_51)\). But the events at \( t_11 \) is deleted and that at \( t_51 \) is retained. The main reason is that in the real-world scenario, the IP/DNS

| Collector     | No-load CPU (%) | No-load Memory (MB) | Full-load CPU (%) | Full-load Memory (MB) |
|---------------|----------------|---------------------|-------------------|-----------------------|
| Auditd       | 0.2            | 31.2                | 2.0               | 31.2                  |
| lttng         | 2.0            | 403.6               | 23.0              | 467.6                 |
| Sydig         | 19.0           | 108.4               | 22.0              | 463.4                 |
| Auditbeat     | 1.2            | 136.4               | 7.1               | 170.4                 |

All the results are the average based on 10 tests independently.
(e.g., external network) may be compromised by an attacker, thereby causing semantic changes. To this end, we design a time window $T$ to retain the network receiving events at a low frequency. For example, if $T = 50$, then after every 50 time intervals, the semantics of IP/DNS will be refreshed and retained again. We may consider two status: First, the semantics of $x.com$ has not been changed during process $P$ switching context back and forth (i.e., $x.com$ has not been influenced by other processes or attackers), the edges ($t_2, t_{11}$) will be deleted. Second, $x.com$ is exploited by attackers during interaction and the semantics of $x.com$ changes, the edge ($t_{51}$) will be retained. In order to select the most appropriate parameters, we have made the statistical analysis in a real enterprise environment (an Internet company of hundreds of employees, the audit log volume collected from Linux hosts reaches the PB level per day) for one month. Observations show that the optimal value of $T$ is between 40 to 60 seconds. When $T$ is small, a mass of redundant network events are retained (low usability). While when $T$ is large, the investigation results may be badly affected if the network session had changed or the remote site was compromised (low accuracy). To balance the usability and accuracy, we choose $T = 50$ (i.e., 50 seconds) in this paper. In actual use, the administrator can dynamically adjust $T$ according to the network status.

2) Redundant Semantics Skipping: APTSHIELD maintains a table to store the latest semantic information of all processes. When the semantics of the incoming event is the same as that in the semantic table, the event can be considered as redundant and deleted. If there are a large number of repeated read and write operations, deleting redundant semantic events can greatly reduce execution time and memory overhead.

The algorithm of redundant semantics skipping is shown in Algorithm 1. The input is the real-time streaming data (events) and the output is whether to delete the event or not (all events considered by APTSHIELD is shown in Appendix C, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TDSC.2023.3243667). We use a table called $LST$ to record the newest sets of (entity, event) that have been occurred. When processing the real-time event, our algorithm will determine whether the subject $S$ exists in $LST$. If it exists, then the corresponding event $e$ that is the same as stored event will be skipped. However, the efficiency of this algorithm will become lower as size of $LST$ increases (i.e., maintaining the latest semantics of $LST$ requires frequent queries and deletions, especially when the table is large). Therefore, we propose to empty the $LST$ when it reaches a certain size. Fortunately, by observing data samples in real enterprise scenarios, we find that most (about 95%) redundant semantics occurred continuously in a short period of time, so by setting a small threshold, a very considerable compaction effect can be achieved. The choice of the size of $LST$ is based on sensitivity analysis. In the test, we find that when the size is smaller than 5, some redundant events will still be retained. While when the size is larger than 5, the efficiency of our redundant semantics skipping algorithm gradually decreases. To balance the compaction performance and algorithm efficiency, we finally choose 5 as the size of $LST$. It should be noted that our method does not have any loss of context semantics regardless of size of $LST$ changes. In addition, when the event $e$ is a WRITE/RECV event, the event $e'$ stored in $LST$ related to $O$ need to be deleted. The reason is that the semantics of the object $O$ will be changed due to the WRITE/RECV event. For example, if process $P$ continuously reads file $A$, since the semantics of subject and object have not changed, the first READ event will eventually be retained. However, if a process $Q$ writes file $A$ during this period, which changes the semantics of the file $A$, the semantics of the process $P$ will also change after the subsequent process $P$ reads the file $A$. Therefore, the next three events will be retained in the end, namely READ (process $P$ reads file $A$), WRITE (process $Q$ writes file $A$), and READ (process $P$ reads file $A$). The same practice applies to RECV.

3) Non-Viable Entity Pruning: After filtering out redundant semantics, we maintain a node tree to contain the relationship of

| Algorithm 1: Data Compaction of Redundant Semantics Skipping. |
|---------------------------------------------------------------|
| **Input:** | **Output:** |
| (1) Streaming data (events) in chronological order, each event $e_i$ contains a subject $S_i$ and an object $O_i$ | Whether the event is skipped |
| (2) The set of (entity, event) that have been occurred recently are denoted as Latest Semantic Table $LST$ | |
| **Initialize:** | |
| The set of (entity, event) $LST = \emptyset$ | |
| **for** $e_i \in$ streaming events **do**: | |
| if $S_i$ not exists in $LST.Keys$ then | |
| $LST.add((S_i, e_i))$ | |
| else | |
| if $e_i$ equals to $e'_i$ then | |
| Skip the event | |
| Continue | |
| else | |
| Delete $(S_i, e_i)$ in $LST$ | |
| Add $(S_i, e_i)$ to $LST$ | |
| if $O_i$ exists in $LST.Keys$ then | |
| Delete $(O_i, e'_i)$ in $LST$ | |
| **end if** | |
| **end if** | |
| **end if** | |
| **if** $LST.size \geq 5$ then | |
| $LST = \emptyset$ | |
| Continue | |
| **end if** | |
| if $e_i \in \{\text{WRITE, RECV}\}$ then | |
| **for** $(S'_i, e'_i) \in LST$ **do**: | |
| if $O_i$ exists in $LST.Keys$ then | |
| Delete $(S'_i, e'_i)$ in $LST$ | |
| **end if** | |
| **end for** | |
| **end if** | |
all processes for efficient forensic analysis. For all process entities that do not have a parent node (i.e., independent processes), we connect them by creating a virtual root process. In order to keep a stable memory overhead when the system runs for a long time, non-viable entity pruning is used to reduce the number of processes. As shown in Algorithm 2, if a process satisfies: (1) it executes the exit event, (2) it does not have potential harmful functionalities (PHF, it will be introduced in Section IV-E2) and child nodes (When the child node has PHFs, the deletion of the parent node will affect the forensic analysis), this process will be pruned from the node tree.

Moreover, we move inactive file nodes (in this article, inactivity is defined as no change for more than 5 minutes, and the time interval can be changed according to actual needs) to the disk to further reduce the memory overhead. They will be took out from the disk in real-time when needed.

**Algorithm 2. Data Compaction of Non-Viable Entity Pruning**

**Input:**
Streaming data (only EXIT events) in chronological order.
Each event \( e_i \) contains a subject \( P_i \).

**Output:**
Whether the process is pruned

**Initialize:**
For all processes are stored in a Process Tree \( PT \)
1: for \( e_i \in \text{streaming events do} \):
2: if \( e_i \in \{ \text{EXIT} \} \) then
3: if \( \text{isleaf}(P_i) \& \& \neg \text{PHF}(P_i) \) then
4: prune \( P_i \)
5: end if
6: end if
7: end for

\( \text{E. APT Detection Framework} \)

In this section, we first introduce the insight for label-based APT detection. Then we explain the definition of labels and label delivery rules based on ATT&CK model. Finally we give the attack judgment rules.

1) **Insight for Label-Based APT Detection:** Due to the persistence of APT attacks, the detection methods based on the dependency graph suffer from efficiency and memory problems. To detect long-term APT attacks with a constant level of persistence of APT attacks, the detection methods based on the attack judgment rules. The semantics of these two file reading events are not the same. In the ATT&CK model, R1 is mostly the behavioral characteristic in the Initial Access stage of the APT attack, which may lead to malicious code execution, while R2 may be the Data Exfiltration stage where the attacker is stealing user information, which may eventually lead to user information leakage. In order to solve the problems above, we define a series of entity labels, as well as transfer rules for labels between entities to visually describe the APT attack. In summary, APTSHIELD gathers the semantics of the complete attack chain into target entity to realize the abstraction of large-scale attack features with constant-level memory overhead.

2) **Label Definition:** (1) **Process Labels.** We divides process labels into two categories: status labels and behavior labels, a selective list of process labels is shown in Table II. The status labels represent the label that semantics in the process will be passed to the child process with process/thread related events. The behavior labels indicate what the process has done, and are used to accurately locate the threat. For example, if a process has network connection, we label it with \( P_{S1} \), for we can not trust the data from network. If a process executes a command without being allowed, we label this process with \( P_{B4} \). A detailed description of the process label is in Appendix A, available in the online supplemental material.

2) **File Labels.** We divide file labels into two categories: untrusted labels and high-value labels, a selective list of file labels is shown in Table III. The untrusted files refer to the file containing untrusted data from the network, and the high-value files refer to the file containing sensitive data. For example, if a file contains data from the network, we label it with \( FU/2 \), for the untrusted data may cause an attack later (code execution). A detailed description of the file label is in Appendix B, available in the online supplemental material.
Some processes with specific labels (functionalities) may directly or indirectly cause potential harm to system security. We define PHF as a function that has the potential to cause harm (in other words, the attacker’s goal would not be achieved without performing these functions). Admittedly, legitimate programs may also perform these functions. In principle, all labels in Table II can be regarded as PHFs. However, the selection of PHFs needs to comprehensively consider both the performance of data compaction and the efficiency of forensic analysis in practical scenarios. For example, if sensitive commands (such as chmod, tcpdump, ifconfig, etc.) are frequently executed by normal users, setting “process executes a sensitive command” to PHF will cause a large number of nodes to be retained during the non-viable entity pruning, thereby increasing the memory usage of APTSHIELD. Similarly, for sensitive files (i.e., system configuration files, user sensitive information, and application critical files), marking uncommon behaviors such as “process executes a sensitive file” as PHF on the personal host can not only maintain low cost of memory, but also facilitate efficient forensic analysis (analysts can directly restore attack chain from memory without reading the raw data from disk). On the enterprise host, since processes may interact frequently with sensitive files, a finer-grained PHF selection strategy needs to be formulated in combination with specific tasks. In this paper, we select PS2, 3, 5 and PB1, 2, 5 as PHF. When used in a real-world environment, analysts are able to select labels with suitable attack characteristics as PHF in combination with specific enterprise scenarios to meet the performance requirements of data compaction and forensic analysis at the same time.

3) Event Selection: After defining the labels of system entities (i.e., processes and files), it is necessary to clarify effective events between entities in order to facilitate the transfer and aggregation of contextual information. Inspired by [27], [29], the events involved are a part of the events that exist in the Linux kernel data collected by auditd, including process events such as fork, execute, LoadELF, file operation, entity attribute modification, network operation, etc. A selective list of events we used is in Appendix C, available in the online supplemental material.

4) Transfer Rules: To gather the semantics of the complete APT attack chain into target entity, we design a label transfer rule based on the ATT&CK model, a selective list of transfer rules is shown in Table IV. We divide APT attacks into five major stages, which are initial access, untrusted execution, lateral movement, suspicious behavior, and data exfiltration. Furthermore, the stage of suspicious behavior contains persistent stronghold, privilege escalation, credential access, and information collection. In Table IV, label 1 and label 2 represent semantic labels carried by system entities, and the direction represents the flow direction of system events, which is also the label transmission direction. For example, if a process with label PS1 writes a file, we label the file with label FU2, when a file with label FU2 is accessed by a process, we label the process with PS3. A detailed description of the transfer rules is in Appendix D, available in the online supplemental material. If a certain process contains some labels in above five stages, the process has the possibility to represent an ongoing APT attack, the APT judge rules will be described in Section IV-E5.

Note that HOLMES [28] extracts TTPs from the audit logs (i.e., TTP rule specifications). In fact, the idea of transfer rules in APTSHIELD is fundamentally different from that in HOLMES. HOLMES is about searching for different isolated TTPs (i.e., single-hop for one event) and then trying to correlate them one by one to understand the full attack picture. While APTSHIELD uses untrusted data flow, untrusted control flow, and high-value data flow to correlate the potential threat in low-level audit logs through the transfer and aggregation of labels, and then discover the underlying TTPs (i.e., multi-hop for several events). For example, the following suspicious pattern “(1) Process A read file from the network, (2) Process A wrote a file F, (3) Another process B read F” can be captured by APTSHIELD, but not by HOLMES. Compared with HOLMES, APTSHIELD is able to track and contextualize suspicious behaviors on a host and identify malicious acts in real-time, with low overhead and strong interpretation. Analysts can expand Table IV according to different techniques under specific tactics.

5) Judgement Rules: The attack judgment rules used in this paper are listed in Table V. The last column specifies the prerequisites for the transfer rule to match. The prerequisites can specify conditions on the parameters of the alarm being matched. The administrator is able to judge whether an attack has occurred based on the different labels contained in the process. In other words, the labels of different entities on the attack chain will eventually come to a process through transfer and aggregation, when labels contained in the process match target judgment rules, the corresponding alerts (threats or APTs) will be reported to the administrator. For example, a process that contains the Webshell/RAT/Living-off-the-land attack labels, as well as the label PB5 may be the alert point of APT attack. A detailed description of the judgement rules is in Appendix E, available in the online supplemental material.

V. EVALUATION

In our evaluation, we first describe the experimental setup. Then we introduce the effectiveness of data compaction, the accuracy and effectiveness of APT detection, and the overhead of the system in turn.

A. Experimental Setup

Our datasets consist of two parts, one is collected from our laboratory, and the other is from Darpa Engagement. Table VI summarizes the property of our dataset.
TABLE IV
A SELECTIVE LIST OF TRANSFER RULES

| Stage                  | Label | Event | Label2 | Direction | Description                                                                 |
|------------------------|-------|-------|--------|-----------|-----------------------------------------------------------------------------|
| Initial Access         | P1    | Write | P2     | D         | A network-connected process wrote a file                                   |
|                        | P2    | Read  | P2     | R         | The process reads a file containing network data                           |
|                        | P4    | Write | P4     | D         | The process, which has accessed network data, writes data to files          |
| Unwanted Execution     | P2    | Read/Map | P2     | R         | The file uploaded by the user is loaded or read by processes               |
|                        | P1    | LoadIf | P2     | R         | The file, which contains the codes from the network, is executed           |
|                        | P1    | Write | P2     | D         | The process, which has executed the codes from the network, writes files   |
| Lateral Movement       | P1    | Write | P1     | D         | The process, which has caused the Webshell attack, writes files             |
|                        | P1    | Read/Map | P1     | R         | The process reads files written by Webshell                                |
| Suspicous Behavior     | P2    | Read  | P2     | R         | The process reads files written by Living-off-the-land attack, writes files |
|                        | P3    | Read  | P3     | R         | The process reads files that hold sensitive information such as temp/passw  |
| Data Exfiltration      | P3    | Read  | P3     | R         | The process reads files that have historical command such as hash/history   |
|                        | P5    | Read  | P5     | R         | The process reads files that contain high-value data                       |

Label 1 and label 2 represent semantic labels carried by system entities, and the direction represents the flow direction of system events. D stands for forward, back, R stands for back to front.

TABLE V
A SELECTIVE LIST OF JUDGEMENT RULES FOR DIFFERENT ATTACKS

| Alert                  | Condition          | Prerequisites                                           |
|------------------------|--------------------|---------------------------------------------------------|
| Download&Execution     | PBI                | os(fd,fp) = match(fp, name, "[mystep*/]") \rightarrow p.labels.add("PST") |
|                        | PBI                | propReg(fd, f) = if (f.labels.contains("PST") \&\& \&\& e = Write) then f.labels.add("PST") |
|                        | Webshell           | os(fd,fp) = match(fp, name, "[mystep*/]") \rightarrow p.labels.add("PST") |
|                        | RAT                | os(fd,fp) = match(fp, name, "[mystep*/]") \rightarrow p.labels.add("PST") |
| Living-off-the-land    | P3 & P3            | os(fd,fp) = match(fp, name, "[mystep*/]") \rightarrow p.labels.add("PST") |
|                        | RAT                | os(fd,fp) = match(fp, name, "[mystep*/]") \rightarrow p.labels.add("PST") |
| Suspicious Behavior    | P3 (P3 & P3)       | os(fd,fp) = match(fp, name, "[mystep*/]") \rightarrow p.labels.add("PST") |
|                        | Data Exfiltration  | os(fd,fp) = match(fp, name, "[mystep*/]") \rightarrow p.labels.add("PST") |
|                        | APT                | os(fd,fp) = match(fp, name, "[mystep*/]") \rightarrow p.labels.add("PST") |

The prerequisites can specify conditions on the parameters of the alarm being matched.

TABLE VI
DETAILS OF OUR DATASETS

| Dataset | Duration (mins) | Platform | Source | Attack Description |
|---------|----------------|----------|--------|--------------------|
| L-1     | 21             | Ubuntu 16.04 (64 bit) | Laboratory | Webshell attack from the backdoored Apache |
| L-2     | 14             | Ubuntu 16.04 (64 bit) | Remote Access Trojan from the phishing website |
| L-3     | 36             | Ubuntu 16.04 (64 bit) | Living-off-the-land attack from vulnerable service |
| E-1     | 485            | Ubuntu 14.04 (64 bit) | Darpa Engagement | Information gather and exfiltration (cleaner attack) Malicious file download and execute (he attack) |
| E-2     | 482            | Ubuntu 14.04 (64 bit) | Darpa Engagement | In-memory attack with firefox |

1) The Dataset From Laboratory: For the data from our laboratory, two participants (red team) in our laboratory were responsible for instrumenting OS and carrying out attack campaigns (on three hosts with Ubuntu 16.04), while the other two participants (blue team) performed data collection, data compaction and attack detection in a real-time manner. The benign background activities were also being carried out on the machines used for experimentation, such as web browsing, chatting, and document editing. In order to get an adversarial engagement, the blue team had no prior knowledge of the attacks planned by the red team. As shown in Table VI, L-1, L-2, and L-3 were the APT datasets generated by simulating real scenes in our laboratory. Each dataset contained normal data and attack data. The attacks contained in above three datasets...
Table VII
RESULTS OF APTSHIELD (REDUNDANT SEMANTICS SKIPPING) COMPARED WITH THAT OF FULL DEPENDENCE PRESERVATION (FD, PRESENTED BY HOSSAIN ET AL. [24]). SUM STANDS FOR THE TOTAL NUMBER OF EVENTS

| Dataset | Sum | Skip for Event | Skipped Ratio (%) | Saved Time (ms)/1000K for APTSHIELD |
|---------|-----|----------------|-------------------|-------------------------------------|
| L-1     | 1142480 | APTSHIELD | 33.43 | 1716 |
| L-2     | 1070143 | FD | 33.42 | |
| L-3     | 1006603 | LogGC | 33.39 | |
| E-1     | 27978043 | APTSHIELD | 21.44 | |
| E-2     | 33743425 | FD | 17.76 | |

Remain stands for the number of events remaining after being skipped.

(L-1, L-2, and L-3) are: Webshell attack from the backdoored Apache, Remote Access Trojan from the phishing website, and Living-off-the-land attack from vulnerable service. A detailed description of attacks contained in L-1, L-2, and L-3 are in Appendix F, available in the online supplemental material.

2) The Dataset From Darpa Engagement: For the data from Darpa Engagement, two hosts with Ubuntu 14.04 were deployed in advance for the data collection (blue team) and the APT attack (red team). Note that the blue team had no prior knowledge about the attack prepared by the red team. Similarity, when the red team was attacking the host, other activities of normal programs on the host were also proceeding simultaneously. Benign activities contained browsing websites, downloading and executing binary files, reading/writing emails and documents, etc.. Overall, more than 99% of the events in the dataset had nothing to do with attacks. As shown in Table VI, E-1 and E-2 are the APT datasets generated by Darpa Engagement. Each dataset contained normal data and attack data. The attacks contained in above two datasets (E-1 and E-2) are: Information gather and exfiltration, Malicious file download and execute (ccleaner attack and hc attack), and In-memory attack with firefox. A detailed description of attacks contained in E-1 and E-2 are in Appendix G, available in the online supplemental material.

B. System Performance

Here, we will introduce the effectiveness of data compaction, the accuracy and effectiveness of APT detection, and the system overhead.

1) Performance of Data Compaction: (1) Effectiveness of Redundant Semantics Skipping

Table VII gives results of redundant semantics skipping. For the datasets (L-1, L-2, and L-3) from our laboratory, each dataset contains about 1000K events, and about 350K pieces of events can be skipped on average through the redundant semantic skipping strategy. In the real-time APT detection, APTSHIELD can save 1000ms-2500ms for every 1000K samples. For the datasets (E-1 and E-2) from Darpa Engagement, the datasets contain about 28000K-34000K events, and about 6000K pieces of events are able to be skipped on average through the redundant semantic skipping strategy. Although the proportion of skipped events has decreased compared with the data from laboratory (down from 35% to 20%), the time saved per 1000K events has increased significantly, reaching 7800ms-11000ms.

We find that APTSHIELD can save more time on the Darpa Engagement dataset, the reason is that the process of file reading and writing events takes more time, and there are a large number of such events on the Darpa Engagement dataset.

2) Effectiveness of Non-Viable Entity Pruning

Some non-viable entities will neither pose a threat nor affect the forensic analysis. Therefore, we can delete these non-viable entities based on EXIT events and PHF labels to reduce memory consumption. We note that LogGC [18] proposes a dead-end event garbage-collected mechanism to delete entities. Results of non-viable entities pruning compared with that of dead-end event garbage-collected in LogGC are shown in Table VIII. It is shown that the compaction effect of non-viable entity pruning is better than dead-end event garbage-collected on all datasets. From the perspective of the process, LogGC treats process kill events as destructive and retains all of them (LogGC only deletes the process that write to stdout). However, most process kill events are irrelevant to the attack and can be deleted. To this end, we propose the conditions of PHF to maximize process pruning while ensuring attack semantics. The results on different datasets show that the proportion of pruned processes is about 3% to 8% for APTSHIELD (for LogGC, the ratio for pruned processes is less than 3%). From the perspective of the files, LogGC only filters the temporary file whose entire life time belongs exclusively to a single process when a file deletion event occurs. We consider moving inactive file nodes into disk to further reduce the memory overhead (temporary files deleted by LogGC can obviously be covered by our method). The reason file pruning ratio of L-3 and E-1 is relatively high is because these two datasets contain a large number of inactive temporary files. These temporary files are all moved to disk.

3) Comparison With Related Work

As shown in Table IX, we compared APTSHIELD to other existing data compaction methods from the following three aspects: the real-time processing capability of the generated data stream (real-time), compaction effect/ratio (efficiency), and system overhead when running data compaction methods (overhead).

LCD [47] considers the event compaction of the single node in a real-time manner, but it does not link the contextual semantics, which could cause difficulty in forensics analysis. Moreover, LCD cannot handle network events.

The input of FD is the whole dependency graph, which can only start after data collection is completed (i.e., FD uses offline cached data). Although redundant semantics skipping and FD are compaction methods that preserve global dependencies, we consider process/thread related events more deeply in redundant semantics skipping. There are three system call functions related
to processes and threads on Linux: fork, vfork and clone. The child process/thread will copy/share the resources of the parent process when fork and vfork are invoked, the semantics of the child process/thread are equivalent to that of the parent process. Clone related events are strongly extensible, in that they can specify to create a new namespace or turn the created process into a sibling process of the parent process, which may introduce new semantics. So, we retain clone related events and conditionally skip fork/vfork related events (according to Algorithm 1) on Linux. On the contrary, FD does not take the above into account and retains all events that related to processes and threads. As shown in Table VII, the reason why APTSHIELD is slightly better is because the fork/vfork related events (i.e., if a child process is forked by a process or parent process whose semantics have not changed, the related event will be deleted) is also compacted. Note that source dependence preservation (SD) was also proposed by Hossain et al. [24] to further improve compaction effect. However, SD has the following disadvantages that make redundant semantics skipping more applicable in our scenario: First, SD only retains some dependencies and destroys the global semantics. Second, the analyst needs to perform a forward analysis from a suspicious entity to get the complete graph of the attack in SD. However, the entry point is usually complex with multiple information flows. The forward analysis from a hub process will cause the dependency explosion problem. By deploying the SD algorithm, the overhead of system will grow rapidly. Third, the SD algorithm requires the entire provenance graph as an input, rendering it hard to process real-time streaming data and apply in the real-world scenario. Our redundant semantics skipping method does not have the above problems.

GS [46] is able to establish global dependencies as FD does, but the compaction system of GS does not have a discarding mechanism (e.g., the processing of lengthy list fields, killed processes, and unused files). As time increases, the real-time performance and efficiency of the system will decrease. Our compaction method is able to improve the efficiency of real-time APT detection and reduce data storage overhead for forensic analysis by adopting redundant semantics skipping and non-viable entity pruning. All in all, the requirements of real-time performance, efficiency, and overhead can be jointly addressed by APTSHIELD. It outperforms all the above related methods.

(4) Correctness of Data Compaction
Since we can easily obtain the ground truth that includes the attack execution steps and the related entities in our datasets, we compare and verify the consistency of APT-related attributes in the attack detection results before and after compaction. Since our redundant semantics skipping does not break the contextual semantics of the audit log and non-viable entity pruning does not delete attack-related entities, the results show that the APT-related attributes before and after compaction are exactly the same on all datasets. The attack chain corresponding to each attack is described in detail in Section V-B2.

2) Accuracy and Effectiveness of APT Detection: In this section, we will explain how the APT detection framework can effectively detect attacks in Table VI.

L-1: Webshell Attack From the Backdoored Apache. The APT attack chain in dataset L-1 is shown in Fig. 3. The process of the APT detection framework is as follows:

1) The folder directory /var/www/html/uploads/ in the web service is to save uploaded files. When the Webshell file shell.php has been uploaded to this directory by

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
the Apache vulnerability, we label this file with FU1 according to the file label definition (time point 1-3).

2) If the file shell.php is accessed or loaded by a process, the process will be labeled with PS4 by the transfer rule (PS4, READ/MMAP, FU1, R) (time point 4). Note that the Apache in Fig. 3 is a subprocess, which will not cause label taint (i.e., other behaviors of apache will not be labeled).

3) When a process with PS4 executes any commands, the process will be labeled with PB4 (time point 5-6). In order to track the spread of attacks after the invasion, APTSHIELD will label files accessed by the process (with PB4) as FU4 according to the transfer rule (PB4, WRITE, FU4, D), which indicates that these files may be threatened by the subsequent proliferation of Webshell. When the file is accessed by other processes, the semantics will be passed back to the process (labeled with PB4) by the transfer rule (PB4, READ/MMAP, FU4, R), which makes subsequent infiltrations of Webshell fully recorded and detected.

4) The process of Webshell (PB4) writes a controllable script (/tmp/cleanup.sh), (/tmp/cleanup.sh) will be labeled with FU4 by the transfer rule (PB4, WRITE, FU4, D) (time point 9 and 16).

5) The script (FU4) is loaded and executed by a privileged process (/bin/sh), (/bin/sh) will be labeled with PB4 by the transfer rule (PB4, READ/MMAP, FU4, R), (/bin/sh) executes nc to create command\&control between the user and the attacker (time point 17-23).

6) The process sh (PB4) continues to access sensitive files, such as /etc/crontab(FH1), /etc/sudoers(FH2), /etc/passwd(FH3), and .bash_history(FH4), it will be labeled with the corresponding label (i.e., PB6, PB7, PS6, and PST) indicating that it has read or written these sensitive files (time point 26-35).

7) When a process sh has read or written sensitive files that pass high-value information out, the file secret written by it will be labeled with FH5 (time point 36-38), indicating that the file secret may contain high-value information (PS6 - 7/PB6 - 7, WRITE, FH5, D).

8) If another process cat subsequently accesses the file with the FH5 label, it may be labeled with PB8 due to the threat of data exfiltration (PB8, READ, FH5, R) (time point 39-40). Finally, the process that satisfies this series of attack chains may generate the threat of APT attacks (including Initial Access, Lateral Movement, Suspicious Behavior, and Data Exfiltration). Therefore, it is reported that there may be APT which uses Webshell as the entrance to the process that contains labels of Webshell and PB8 at the same time.

L-2: Remote Access Trojan From the Phishing Website. For the APT in L-2, the detection flow of APTSHIELD is as follows:

1) Through a phishing link, the target host downloads the Trojan file (vmp.elf). Since the file is transmitted over the Internet, it will be labeled with FU2.

2) After a period of time, the file (vmp.elf) is executed, and the target process will be labeled with PS1 by the transfer rule (PS1, EXECUTE, FU2, R), indicating that the process executes files originating from the Internet may cause a threat.

3) Subsequent attacks and proliferation are the same as they are in the Webshell experiment. Finally, it is reported that there may be APT which uses RAT as the entrance for the process that contains labels of RAT and PB8 at the same time.

L-3: Living-off-the-Land Attack From Vulnerable Service. For the APT in L-3, the detection flow of APTSHIELD is as follows:

1) Through an exploit script, the threatening process will interact with a file named "(null)". Therefore, the nonexistent file will be labeled with FU3.

2) When a process interacts with the file, the process will be labeled with PS5 by the transfer rule (PS5, READ/MMAP/LOADELF, FU3, R). When the process containing the label PS5 continues to execute the sh or bash command, it will be labeled with PB5, which means that the process has got the shell and carried out an in-memory attack.

3) Subsequent attacks and proliferations are the same as they are in the Webshell experiment. Finally, it is reported that there may be APT that uses Living-off-the-land as the entrance for the process that contains labels of Living-off-the-land and PB8 at the same time.

E-1: Information Gather and Exfiltration. The detection flow of APTSHIELD is as follows:

1) The file (/etc/passwd) containing sensitive information is labeled with FH3.

2) When a process reads this file, the process will be labeled with PS6 by the transfer rule (PS6, READ, FH3, R).

3) The process writes the high-value data on /dev/pts/1, /dev/pts/1 will be labeled with FH5 by the transfer rule (PS6, WRITE, FH5, D). By this way, the user may have exfiltrated this information.

E-1: Malicious File Download and Execute (Ccleaner Attack). The attack chain is shown in Fig. 4. The detection flow of APTSHIELD is as follows:

1) A file named ccleaner is downloaded to the local through scp, this file will be labeled with FU2. Because the process that uses the scp command is likely to include an internet connection PS1, the file will be labeled with FU2 by the transfer rule (PS1, WRITE, FU2, D).
2) After the file is executed by the process, the process will be labeled with PB1 by the transfer rule (PB1, EXECUTE, FU2, R). Therefore, the process containing the PB1 label may pose a threat (download and execution).

**E-1: Malicious File Download and Execute (hc Attack)**. The attack chain is shown in Fig. 5. The detection flow of APT-SHIELD is as follows:

1) A file named hc is downloaded to the local through scp, this file will be labeled with FU2.
2) After the hc file is executed by the process, the process will be labeled with PB1 by the transfer rule (PB1, EXECUTE, FU2, R). Therefore, the process containing the PB1 label may pose a threat (download and execution).
3) The process hc reads the sensitive file (/etc/passwd) will be labeled with PS6 by the transfer rule (PS6, READ, FH3, R).

**E-2: In-Memory Attack With Firefox**. The attack chain is shown in Fig. 6. The detection flow of APT-SHIELD is as follows:

1) Use Firefox’s write-executable memory space to cause in-memory attacks through sshd process. The sshd process interacts with the "/(null)" file, and the process is labeled with PS5 by the transfer rule (PS5, READ/MMAP/LOADELF, FU3, R).
2) Later, the process with PS5 loads the file /tmp/libnet.so, which is created by firefox visiting external website, to get the shell, the process will be labeled with PB5. Processes that contain labels PB5 and PS5 may cause the threat of in-memory attacks.
3) Finally, a file named audiobackup is downloaded and executed through the sshd in-memory attack. The detection flow is the same as the hc and ccleaner attack.

**C. Overhead of APT-SHIELD**

1) **CPU and Memory Usage**: The CPU and memory usage of APT-SHIELD for five different datasets are shown in Table X. We use performance monitor to record the average overhead of the system while it is running. Here, the CPU usage means the usage rate on a single-core CPU (thread). In the absence of pruning, the average single-core CPU usage of APT-SHIELD is 5.3%, 5.8%, 4.7%, 9.7%, and 10.1% on L-1, L-2, L-3, E-1, and E-2, respectively, while the average memory usage is 90.1 MB, 99.7 MB, 85.6 MB, 117.7 MB, and 130.3 MB on L-1, L-2, L-3, E-1, and E-2, respectively. By deploying non-viable entity pruning (to further reduce the system overhead), the average single-core CPU usage is reduced by 1.1%, 0.7%, 0.4%, 2.0%, and 1.6% on L-1, L-2, L-3, E-1, and E-2, respectively, while the average memory usage is reduced by 14.8 MB, 22.6 MB, 14.9 MB, 22.2 MB, and 19.7 MB on L-1, L-2, L-3, E-1, and E-2, respectively. All in all, in the case of real-time detection, APT-SHIELD can maintain low CPU usage and satisfactory memory overhead.

2) **Runtime Performance**: We measure the runtime performance of APT-SHIELD for CPU-intensive and I/O-intensive operations, respectively. Measurements are performed on two cases, one without APT-SHIELD (baseline) and one with APT-SHIELD opened. We conduct ten independent experiments and the average results are shown in the Fig. 7. The vertical axis of Fig. 7 represents the ratio of the time spent in different operations after running APT-SHIELD compared to the baseline. For CPU-intensive commands (i.e., using ar to archive 1500 pictures, using bzip2 to compress a 2G photo album, using dump
TABLE XI
REAL-TIME PERFORMANCE OF APTSHIELD

| Dataset | Generation Rate (event num/pers) | Consumption Rate (event num/pers) | Speed-up Ratio |
|---------|-----------------------------------|----------------------------------|----------------|
| L-1     | 907                               | 102155                           | 112.7          |
| L-2     | 1274                              | 100836                           | 79.2           |
| L-3     | 289                               | 69451                            | 309.2          |
| E-1     | 961                               | 149710                           | 155.7          |
| E-2     | 1167                              | 169668                           | 145.4          |

Comparison of data consumption rate and data generation rate.

to back up the contents in /home/user, using gcc to compile a .c file with 1000 lines, using wc to count the words in a 2G compressed package), the increment of average runtime overhead is 4.06%. For IO-intensive commands (i.e., using cp to copy a 2G compressed package, using dd to convert a 2G file, using scp to upload a 2G photo album, using wget to download a 3G image file), the increment of average runtime overhead is 7.50%. It can be concluded that enabling APTSHIELD has little effect on the overall system runtime performance.

3) Real-Time Performance: We test the real-time performance of the APTSHIELD by comparing data consumption rate and data generation rate. Results are shown in Table XI. The speed-up ratio means the event consumption rate divided by event generation rate of the target host. It can be seen from Table XI that APTSHIELD can consume events 79 to 309 times faster than the average event generation rate of the host, which shows the real-time performance of APTSHIELD is feasible.

4) Comparison With Related Work: We compare APTSHIELD with state-of-the-art studies to show the advantage of our work.

(1) Conan [29]. Conan only considers the data compaction in the storage stage, while APTSHIELD can perform compaction and detection at the same time. In addition, APTSHIELD also prunes invalid/unused entities to reduce overhead. In our experiment, the average memory usage of Conan on the dataset from laboratory (L-1, L-2, and L-3) is about 90 MB. For the dataset from Darpa Engagement, the average memory usage of Conan is about 140 MB. Compared with APTSHIELD, the memory overhead of Conan is increased by about 30% to 50%, and the growth rate of CPU usage reaches to 10% to 20%. Furthermore, APTSHIELD has a better semantic interpretation and good detection efficiency for in-memory attacks.

(2) Sleuth [27] & Holmes [28]. According to the description in Section II, Sleuth will read all the data into the memory at one time during detection, which cannot ensure real-time performance. The memory Sleuth used is related to the size of the dataset. In our experiment, the memory usage of Sleuth on L-1, L-2 and L-3 ranges from 80 MB to 160 MB, while on E-1 and E-2, it ranges from 300 MB to 400 MB. For Holmes, authors in [28] showed that a nearly linear growth in memory consumption. In our experiment, the memory usage of Holmes on L-1, L-2 and L-3 ranges from 90 MB to 150 MB, while on E-1 and E-2, it ranges from 500 MB to 1 GB. Compared with APTSHIELD, the memory overhead of Sleuth and Holmes is increased by at least 60% and 20%, respectively. APTSHIELD has a good real-time performance and can quickly respond to threats that occur in the system.

In the experiments of this paper, APTSHIELD does not show any false positives and false negatives, the main reasons are as follows: First, we optimize the detection rules of known attacks according to ATT&CK, which can ensure the coverage of known attacks and is not easy to generate false negatives. Second, APTSHIELD is able to correlate long-term context behavior semantics and abstract large-scale attack features through label definition and transfer rules to track unknown attacks. However, due to the small sample of real APT attacks, the current test can only cover some but not all attack scenarios. There may still be some false positives and false negatives in real environments, which we will discuss in detail in Section VI. It is worth noting that although the above-mentioned attacks can also be detected (without false positives) using existing studies [27], [28], [29], the detection performance (real-time ability, memory and CPU overhead) of APTSHIELD is better than the above-mentioned existing work.

VI. DISCUSSION

A. False Positive

We discuss the following scenarios that may generate false positives:

Normal Meta Behavior. Since an APT attack is composed of multiple different behaviors, each individual behavior (we call it meta behavior) does not represent the occurrence of an attack. For example, READ events to high-value files such as /etc/passwd may occur during the APT attack stage (when the user is authenticated, /etc/passwd will be accessed), but related processes should not be identified as threats. In order to prevent single-point false positives, APTSHIELD adopts the method of contextual semantic transfer to detect the APT attack chain.

Dependency Explosion. Dependency explosion is a fundamental challenge in host-level audit logging approaches. There are two main ways to alleviate dependency explosion and realize the whole chain detection and real-time alarm of APT attacks in APTSHIELD:

(1) Graph compaction. Through redundant semantics skipping and non-viable entity pruning, APTSHIELD is able to delete redundant semantics and process massive data in real-time with low costs. Also, comparison of data consumption rate and data generation rate in Section V-C3 shows that the event consumption rate is far greater than the generation rate, which reveals that our system can guarantee real-time performance.

(2) Label mechanism. Through the atomic suspicious characteristics of system entities and their transmission rules, APTSHIELD is able to aggregate the information of the attack chain into specific entities in real-time instead of storing the whole provenance graph. Since labels on an entity are a finite set (e.g., there will only be one FU1 per file), a finite number of labels can maintain a constant level of memory overhead. As shown in Tables VII and X, the data from Darpa Engagement is 30 times larger than that from laboratory, but the memory overhead when processed with APTSHIELD is almost the same. Furthermore, the transmission of the label enables the contextual semantics of
the attack to finally be aggregated to a specific entity, ensuring the whole chain detection. However, defining and formulating strict transfer rules requires huge practical experiences and manual efforts. Here, we discuss two potential ways to further assist APTSHIELD to mitigate dependency explosions in host-level audit data: 1) Label fading. Fade labels of a benign subject after a certain threshold of time unless it exhibits suspicious behavior, or before transferring it to the object that it writes into [61]. 2) Path scoring. Path scoring is performed according to whether there is anomaly [52] or whether it can be mapped with TTPs [26], and label transfer is implemented when the path score meets a specific threshold. These thresholds/hyperparameters can be dynamically adjusted according to actual scenario needs (e.g., low FPs in daily operation and maintenance, or high TPs in APT detection and other meritorious campaigns).

B. False Negative

Here, we discuss scenarios that may lead to false positives and provide potential solutions.

The Completeness of Entity Labels. Since Tables II and III lists only a selective set of process and file labels, one may concern about some malicious instances that are not covered by these labels. Recall that APTSHIELD uses different tactical stages in the ATT&CK model to describe a complete APT attack. For the necessary tactics of APT attacks such as Initial Access and Untrusted Execution, our entity labels and transfer rules can totally track untrusted data flow (e.g., the data from network or device), and untrusted control flow (e.g., suspicious process creation). For tactics that reveal attacker’s intent such as Information Collection and Data Exfiltration, our entity labels and transfer rules show the source and destination of confidential (high-value) data inside the system. By adopting entity labels and transfer rules, APTSHIELD can reflect the nature of the attack by tracking untrusted data flow, untrusted control flow, and high-value data flow. While other tactics which may be performed by attackers such as Lateral Movement, Persistent Stronghold, Privilege Escalation and Credential Access, based on the freely configurable features of labels, security analysts can easily expand corresponding labels with knowledge bases such as Atomic Red Team [62]. The context association of different tactics through the aggregation of labels can greatly reduce false negatives, and at the same time provide the ability to explain the entire attack chain.

VII. CONCLUSION

In this paper, we design a stable, efficient, and real-time APT detection system, called APTSHIELD. Unlike previous studies with the problems of the weak data source integrity, large data processing overhead and poor real-time performance, APTSHIELD can collect reliable, stable and semantically rich data sources via audit, reduce the overhead of the detection system in real-time via redundant semantics skipping and non-viable node pruning, and carry out real-time APT attack detection through the transfer and aggregation of labels based on ATT&CK model. Experimental results on both laboratory and Darpa Engagement show that APTSHIELD can effectively detect web vulnerability attacks, file-less attacks and remote access trojan attacks, at the same time has a low false positive rate, which adds far more value than the existing frontier work.

REFERENCES

[1] What is Stuxnet, Sep. 1, 2021. [Online]. Available: https://www.mcafee.com/enterprise/en-hk/security-awareness/ransomware/what-is-stuxnet.html
[2] BlackEnergy APT attacks in Ukraine, Sep. 1, 2021. [Online]. Available: https://www.kaspersky.com/resource-center/threats/blackenergy
[3] Missed alarms and 40 million stolen credit card numbers: How target blew it, Sep. 1, 2021. [Online]. Available: https://www.bloomberg.com/news/articles/2014--03-13/target-missed-warnings-in-epic-hack-of-credit-card-data
[4] APT trends report Q3 2020, Sep. 1, 2021. [Online]. Available: https://securelist.com/apt-trends-report-q3--2020/99204/
[5] APT groups target firms working on COVID-19 vaccines, Sep. 1, 2021. [Online]. Available: https://www.bankinfosecurity.com/microsoft-warning--a15363
[6] A. D. Bolton and C. M. Anderson-Cook, “APT malware static trace analysis through bigrams and graph edit distance,” Statist. Anal. Data Mining: ASA Data Sci. J., vol. 10, no. 3, pp. 182–193, 2017.
[7] G. Laurenza, L. Aniello, R. Lazzaretto, and R. Baldoni, “Malware triage based on static features and public APT reports,” in Proc. Int. Conf. Cyber Secur Cryptogr. Mach. Learn., Springer, 2017, pp. 288–305.
[8] D. Liu, H. Zhang, H. Yu, X. Liu, Y. Zhao, and G. Lv, “Research and application of APT attack defense and detection technology based on big data technology,” in Proc. IEEE 9th Int. Conf. Electron. Inf. Emerg. Comput., IEEE, 2019, pp. 1–4.
[9] I. Rosenberg, G. Sicard, and E. O. David, “DeepAPT: Nation-state APT attribution using end-to-end deep neural networks,” in Proc. Int. Conf. Artif. Neural Netw., Springer, 2017, pp. 91–99.
[10] G. Zhao, K. Xu, L. Xu, and B. Wu, “Detecting APT malware infections based on malicious DNS and traffic analysis,” IEEE Access, vol. 3, pp. 1132–1142, 2015.
[11] L. Huang, J. Xue, W. Han, Z. Kong, and Z. Niu, “Detection of malicious domains in APT via mining massive DNS logs,” in Proc. Int. Conf. Mach. Learn. Cyber Secur., Springer, 2020, pp. 140–152.
[12] N. A. S. Mirza, H. Abbas, F. A. Khan, and J. Al Muhtadi, “Anticipating advanced persistent threat (APT) countermeasures using collaborative security mechanisms,” in Proc. Int. Symp. Biometrics Technol. Secur., IEEE, 2014, pp. 129–132.
[13] A. Khizar, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda, “UNVEIL: A large-scale, automated approach to detecting ransomware,” in Proc. 25th USENIX Secur. Symp., 2016, pp. 757–772.
[14] What is endpoint detection and response, Sep. 1, 2021. [Online]. Available: https://www.gigamon.com/resources/endpoint/what-is-endpoint-detection-and-response.html
[15] 2020 Gartner market guide for network detection and response, Sep. 1, 2021. [Online]. Available: https://www.gartner.com/resources/security/analysis/endpoint/what-is-endpoint-detection-and-response.html
[16] Transparent computing program, Sep. 1, 2021. [Online]. Available: https://github.com/darpa-ict/Transparent-Computing
[17] Trustwave global security report, Sep. 1, 2021. [Online]. Available: https://www.trustwave.com/en-us/resources/library/documents/2020-trustwave-global-security-report/
[18] K. H. Lee, X. Zhang, and D. Xu, “LogGC: Garbage collecting audit log,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., 2013, pp. 1005–1016.
[19] K. H. Lee, X. Zhang, and D. Xu, “High accuracy attack provenance via binary-based execution partition,” in Proc. Netw. Distrib. Syst. Secur. Symp., 2013, pp. 1–16.
[20] S. Ma, X. Zhang, and D. Xu, “ProTracer: Towards practical provenance tracing by alternating between logging and tainting,” in Proc. Netw. Distrib. Syst. Secur. Symp., 2016, pp. 1–15.
[21] S. Ma, J. Zhai, F. Wang, K. H. Lee, X. Zhang, and D. Xu, “MPI: Multiple perspective attack investigation with semantic aware execution partitioning,” in Proc. 26th USENIX Secur. Symp., 2017, pp. 1111–1128.
[22] Y. Kwon et al., “MCI: Modeling-based causality inference in audit logging for attack investigation,” in Proc. Netw. Distrib. Syst. Secur. Symp., 2018, pp. 1–15.
Tiantian Zhu received the PhD degree in computer science from Zhejiang University, Hangzhou, China, in 2019. He is currently an associate professor with the College of Computer Science and Technology, Zhejiang University, and a visiting scholar at the Chinese Academy of Sciences, China. His research interests include mobile security, system security, and artificial intelligence.

Jinkai Yu received the MS degree in software engineering from the Zhejiang University of Technology, Hangzhou, China, in 2022. His research interests include cybersecurity, system security, and artificial intelligence.

Chunlin Xiong received the PhD degree in cybersecurity from Zhejiang University, Hangzhou, China, in 2021. He is currently a postdoctoral student jointly trained by Shenzhen Institute of advanced technology, Chinese Academy of Sciences, and Sangfor Inc. His research interests include system and information security.
Wenrui Cheng received the BE degree in computer science and technology from Huzhou University, Huzhou, China, in 2022. She is currently working toward the MS degree in software engineering with the Zhejiang University of Technology. Her current research interests include knowledge driven APT attack sample generation and detection.

Qixuan Yuan received the BE degree in computer science and technology from ShanDong Normal University, Jinan, China, in 2022. She is currently working toward the MS degree in computer science and technology with the Zhejiang University of Technology. Her current research interests include APT and cyber threat intelligence.

Jie Ying received the BE degree in digital media from the Zhejiang University of Technology, Hangzhou, China, in 2020. He is currently working toward the PhD degree in computer science with the Zhejiang University of Technology. His current research interests include APT and cyber threat intelligence.

Tieming Chen (Member, IEEE) received the PhD degree in computer software and theory from BeiHang University, Beijing, China, in 2011. He is now a professor with the College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou, China. His research interests include cyberspace security and intelligence security. He is a member of the ACM.

Jiabo Zhang received the BE degree in network engineering from Wenzhou University, Wenzhou, China, in 2021. His current research interest include cyberspace security.

Mingqi Lv received the PhD degree in computer science from Zhejiang University, Hangzhou, China, in 2012. His research interests include spatiotemporal data mining and ubiquitous computing.

Yan Chen (Fellow, IEEE) received the PhD degree in computer science from the University of California, Berkeley, CA, USA, in 2003. He is a professor with the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL, USA. His research interests include network security, measurement, and diagnosis for large-scale networks and distributed systems.

Ting Wang received the PhD degree in computer science from Zhejiang University, Hangzhou, China, in 2015. Her research interests include system security, formal methods, and artificial intelligence.

Yuan Fan received the MS degree from San Jose State University. He is the founder and CEO of DAS-Security, Hangzhou. He has a deep study in intrusion detection, cloud security, IoT system security, and industrial Internet security.