Host-Based Intrusion Detection System with System Calls: Review and Future Trends

MING LIU, Shanghai Jiao Tong University, China, and University of Technology Sydney, Australia
ZHI XUE, Shanghai Jiao Tong University, China
XIANGHUA XU, Hangzhou Dianzi University, China
CHANGMIN ZHONG, Joowing Australia, Australia
JINJUN CHEN, Swinburne University of Technology, Australia

In a contemporary data center, Linux applications often generate a large quantity of real-time system call traces, which are not suitable for traditional host-based intrusion detection systems deployed on every single host. Training data mining models with system calls on a single host that has static computing and storage capacity is time-consuming, and intermediate datasets are not capable of being efficiently handled. It is cumbersome for the maintenance and updating of host-based intrusion detection systems (HIDS) installed on every physical or virtual host, and comprehensive system call analysis can hardly be performed to detect complex and distributed attacks among multiple hosts. Considering these limitations of current system-call-based HIDS, in this article, we provide a review of the development of system-call-based HIDS and future research trends. Algorithms and techniques relevant to system-call-based HIDS are investigated, including feature extraction methods and various data mining algorithms. The HIDS dataset issues are discussed, including currently available datasets with system calls and approaches for researchers to generate new datasets. The application of system-call-based HIDS on current embedded systems is studied, and related works are investigated. Finally, future research trends are forecast regarding three aspects, namely, the reduction of the false-positive rate, the improvement of detection efficiency, and the enhancement of collaborative security.

CCS Concepts: • **Security and privacy** → **Intrusion detection systems**; **Artificial immune systems**;

Additional Key Words and Phrases: Cybersecurity, system call, intrusion detection, cloud computing, big data

ACM Reference format:
Ming Liu, Zhi Xue, Xianghua Xu, Changmin Zhong, and Jinjun Chen. 2018. Host-Based Intrusion Detection System with System Calls: Review and Future Trends. *ACM Comput. Surv.* 51, 5, Article 98 (November 2018), 36 pages.
https://doi.org/10.1145/3214304

This work is supported by the China Scholarship Council.
Authors' addresses: M. Liu, Shanghai Jiao Tong University, School of Electronics, Information and Electrical Engineering, Minhang, Shanghai, 200240, China, and University of Technology Sydney, Faculty of Engineering and IT, Ultimo, NSW, 2007, Australia; email: Ming.Liu-2@student.uts.edu.au; Z. Xue, Shanghai Jiao Tong University, School of Electronics, Information and Electrical Engineering, Minhang, Shanghai, 200240, China; email: zxue@sjtu.edu.cn; X. Xu, Hangzhou Dianzi University, School of Computer Science, Hangzhou, Zhejiang, 310018, China; email: xhxu@hdu.edu.cn; C. Zhong, Joowing Australia, Glen Waverley, VIC, 3150, Australia; email: Changmin.zhong@joowing.com; J. Chen, Swinburne University of Technology, Faculty of Science, Engineering and Technology, Hawthorn, VIC, 3122, Australia; email: jinjun.chen@gmail.com.

0360-0300/2018/11-ART98 $15.00
https://doi.org/10.1145/3214304

1 INTRODUCTION

The host-based intrusion detection system (HIDS) has been gaining attention in the community of cybersecurity for over two decades. Compared with the network-based intrusion detection system (NIDS), HIDS has the superiorities of fine granularity and the ability to detect internal attacks. HIDS analyzes auditing data from operating systems, whereas NIDS analyzes data from network traffic. System-call-based HIDS is about analyzing collected Linux system call traces. This approach is effective on common hosts. However, due to the rapid development of data center facilities and techniques, recently HIDS has suffered from the well-known big data challenge. It is challenging for traditional HIDS to handle the growing amount of system call traces or other kinds of auditing data generated from various Linux applications installed in a contemporary data center. Linux system call traces produced from a large data center are a kind of big data, which are massive and complicated. Therefore, they bring new challenges to conventional data processing methods. Haider et al. claimed these kinds of challenges according to their practical experience, and they listed some currently deployed systems and their performance in specific numbers for comparison [47]. Traditional database management systems and data mining methods on a single host may not be able to handle massive system call traces efficiently. Processing large amounts of system call traces has become an essential requirement for modern data centers. Traditional host-based intrusion detection systems are mostly about performing intrusion analysis on an independent host with standalone detection software installed. System call traces for training are loaded into the memory of a single host that has static computing and storage capacity. There are no interactions among HIDS installed on different hosts. This form of deployment has two main shortcomings:

System security → With standalone HIDS software installed on every single host, it is difficult to perform comprehensive system call analysis to detect complex and distributed attacks among multiple hosts. Nowadays, new varieties of zero-day attacks keep being generated. Intruders may even choose the intrusion detection software itself as the attack target based on the software vulnerabilities.
Detection efficiency → Training decision engines with raw Linux system call traces on a single host that has static computing and storage capacity is time-consuming. The size of RAM on a single host may not be capable of handling a large quantity of real-time system call traces or intermediate datasets such as data mining models. If intermediate datasets and permanent detection results are stored in hard disks, it may delay the real-time detection or queries of the records.

Therefore, innovations have to be made to address these shortcomings. To the best of our knowledge, there are limited surveys discussing host-based intrusion detection systems with system calls and the corresponding big data challenge. Forrest et al. discussed applications and results of system call monitoring as well as data modeling in anomaly intrusion detection [33]. This work was composed years ago, yet many new techniques and applications have been generated since that time. Hu [54] composed an introductory article about host-based anomaly intrusion detection. This article emphasized the application of the Hidden Markov Model, and other data mining models were briefly introduced. Ahmed et al. composed a survey about network anomaly detection techniques and the dataset issues with the introduction of the ADFA-LD dataset [2]. This survey was mostly about network-based anomaly detection systems. Chandola et al. proposed a comprehensive taxonomy to classify studies of anomaly detection for discrete sequences into three distinct categories [19]. Their work mainly focused on the theoretical analysis of algorithms, whereas issues such as system call datasets or new applications with system-call-based HIDS were rarely discussed. Our article endeavors to present a clear overview of the development for HIDS with system calls and provide inspirational future research trends in the current big data and cloud computing environment. The main contributions of this article are described below:

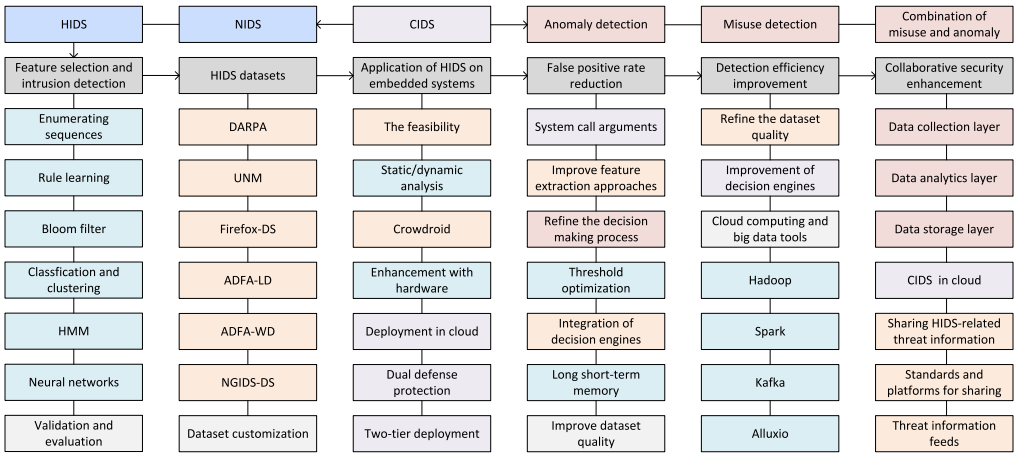

Fig. 1. Roadmap of this article.

(1) An overview of the development of system-call-based HIDS is presented.
(2) Algorithms and techniques relevant to system-call-based HIDS are investigated, including various data mining methods.
(3) The application of system-call-based HIDS on embedded systems is studied, and related works are investigated.
(4) The HIDS dataset issues are analyzed; currently available datasets with system calls and approaches for researchers to generate new datasets are discussed.
(5) Limitations relating to current system-call-based HIDS are discussed, and future research trends are forecast regarding three aspects, namely, the reduction of the false-positive rate, the improvement of detection efficiency, and the enhancement of collaborative security.

The rest of this article is composed as follows. In Section 2, we present an overview of intrusion detection systems. In Section 3, algorithms and techniques of system-call-based HIDS are outlined. In Section 4, HIDS dataset issues are analyzed, and dataset generation methods are discussed. In Section 5, the application of HIDS on embedded systems is studied. In Section 6, future trends of HIDS research are forecast. In Section 7, we give concluding remarks of this article. A roadmap of this article is demonstrated in Figure 1.

2 AN OVERVIEW OF INTRUSION DETECTION SYSTEMS

An intrusion detection system, or IDS, monitors the real-time data of a network or hosts to detect malicious behaviors [29]. It generates an alarm when it detects an intrusive activity. There are two ways to categorize intrusion detection systems:

Types of analyzed data → Network-based intrusion detection system (NIDS), host-based intrusion detection system (HIDS), and collaborative intrusion detection system (CIDS) [81]
Types of attacks → Misuse detection system and anomaly detection system [81]

2.1 Categorization Based on the Types of Analyzed Data

2.1.1 Network-Based Intrusion Detection System. NIDS checks communications in a network to observe intrusions, for instance, applying data mining methods to network traffic data for the detection of anomalies. Currently, issues related to NIDS have been intensively researched and notable outcomes have been achieved. For instance, Tan et al. proposed a system to detect

Table 1. A Typical System Call Sequence [34]

"...*open*, *read*, *mmap*, *mmap*, *open*, *getrlimit*, *mmap*, *close*..."

denial-of-service (DoS) attacks for interconnected server systems by applying multivariate correlation analysis [103]. Geometrical correlations are generated from selected network features. They apply anomaly detection on benign network data to detect zero-day attacks. The process of multivariate correlation analysis is augmented and accelerated by using the triangle-area-based method. The KDD99 dataset is adopted to assess the capability of their IDS on raw and normalized datasets, demonstrating the enhanced detection rate. They further used computer vision methods to solve the DoS attack problem [104]. Network traffic entries are considered as corresponding images by taking multivariate correlation analysis. Those images that were used as observed objects are composed with Earth Mover's Distance. The experiment with 10-fold cross-validations shows high detection accuracy.

Limitations of NIDS. NIDS is often deployed as hardware devices between the Internet and the Intranet of an enterprise. The advantage of this form of deployment is that the NIDS can detect network intrusions immediately. However, it is difficult for traditional NIDS to process encrypted packets transmitting on the network, and the speed of network flow may be downgraded due to the deployment of NIDS devices. Moreover, internal attacks are difficult to detect by NIDS in this case.

2.1.2 Host-Based Intrusion Detection System. HIDS monitors activities such as system or shell logs within hosts to discover unauthorized behaviors. HIDS can perform various kinds of data mining methods such as artificial neural networks on host auditing data to discover attacks. System-call-based HIDS is mainly discussed in this article. In Unix-like operating systems, system calls are referred when a kernel service from the operating system is requested by a running process. System calls are significant interactions between programs and the system kernel. A system-call-based HIDS monitors real-time system call traces to detect abnormal system call sequences. Table 1 shows a typical system call sequence. Data processed by system-call-based HIDS is fine-grained. System-call-based HIDS can trigger alarms when abnormal system call traces are detected from normal traces. By using system calls as the input data, a HIDS can manipulate the most original information of an operating system. System-call-based HIDS has gained attention in the past 20 years because of the increasing number of attacks focusing on Linux servers. System-call-based HIDS has been developed for intrusion detection in virtual hosts and embedded platforms such as smartphones.

Challenges for HIDS. The execution speed of a HIDS is commonly measured by summing up all times of training and testing. Considering the HIDS needs to handle a large quantity of fine-grained system call traces, the speed may be limited. Meanwhile, it is tedious to maintain and update a large number of traditional HIDS software installed on every host or virtual host in a network, compared with the number of NIDS devices. Therefore, novel HIDS approaches that can reduce the execution time while keeping the acceptable detection accuracy are demanded [85]. Moreover, traditional HIDS does not show enough robustness against advanced persistent threats. Therefore, it is necessary for future HIDS to work collaboratively with other security mechanisms.

2.1.3 Collaborative Intrusion Detection System. CIDS is about collaboratively combining NIDS, HIDS, and other security mechanisms in a network for more efficient and effective detection of cyber attacks. Vasilomanolakis et al. proposed a detailed survey that classified CIDS into three types, centralized CIDS, decentralized CIDS, and distributed CIDS [109].

Limitations of Traditional CIDS. Traditional CIDS methods often ignore centralized summarization and comprehensive analysis of network traffic data and host logs, and traditional CIDS may not be efficient enough to handle the summarization of large-volume real-time data streams.

2.2 Categorization Based on the Types of Attacks

2.2.1 Misuse Detection System. The misuse detection system (signature-based intrusion detection system) defines libraries of acknowledged attack signatures and raises alarms when the network traffic or system operations match any attack signatures in the library. Predefined by the system administrators, the library tries to precisely list and persist every possible abnormal network and system behavior. Other known and unknown behaviors are treated as normal. Therefore, the misuse detection system can effectively discover attack methods that are already identified.

Limitation of Misuse Detection System. The misuse detection system is often criticized for having a high missed alarm rate. The increasing amount of zero-day attacks make this approach gradually obsolete. Intruders can simply obfuscate their attack methods to bypass the predefined intrusion libraries.

2.2.2 Anomaly Detection System. The anomaly detection system (ADS) requires no knowledge of known intrusions. Chandola et al. provided a comprehensive survey of the anomaly detection system [18]. In the area of IDS, ADS is classified into network-based anomaly detection system (NADS) and host-based anomaly detection system (HADS). NADS identifies anomalies from normal network traffic. HADS monitors hosts to discover anomalies from normal system behaviors. HADS is often deployed in systems whose normal behaviors do not change frequently. System-call-based HADS is about utilizing system call traces to build normal databases or data mining models of normal system behaviors. Then these databases or models can be used as criteria for anomaly detection. Therefore, anomaly detection system can detect zero-day attacks.

Challenge for Host-Based Anomaly Detection System. A high false-alarm rate (FAR) is the challenge. Novel system call traces are being generated along with an increasing number of new Linux applications. As HADS only holds normal databases or data mining models of known behaviors, new normal system call traces that do not conform to the databases or models may be falsely regarded as intrusions.

2.3 Combination of Misuse Detection with Anomaly Detection

Although IDS can be categorized into these two groups, the combination of a misuse detection system with an anomaly detection system is a current trend for the development of a complete real-time IDS infrastructure to detect both known and unknown intrusions. Concerning system-call-based HIDS, when a new system call trace is approaching, after preprocessing, first it can be compared with predefined rules of known attacks. If no attack is detected, then it can be compared with the normal databases or checked by other anomaly detection algorithms. The trace can also be passed to the security personnel for further investigations. The combination of misuse detection with anomaly detection has been adopted by many current research works such as [9].

3 ALGORITHMS AND TECHNIQUES OF HIDS

The intrusion detection system was briefly introduced from different perspectives in the previous section. In this section, algorithms and techniques mainly used in the area of system-call-based HIDS will be discussed.

3.1 Preprocessing and Feature Selection

As system call traces collected by tracing tools are the original data, preprocessing and feature selection methods have to be applied to get clean and typical features for training. Similar to natural language processing, methods such as the *n*-gram, sliding window algorithm, bag-of-words model, and term frequency-inverse document frequency (TF-IDF) method can be implemented in system-call-based HIDS. An *n*-gram in HIDS is often referred to as a contiguous sequence of *n* system calls extracted from a system call trace within a particular time interval [121]. The sliding window algorithm with window size *n* is often utilized to scan the complete system call trace to generate *n*-grams of system calls, which are used for training normal databases or data mining models. The *n*-gram method has been proven to be effective by researchers [113, 60, 129]. Creech et al. [25] used multiple-length sliding windows to scan the complete system call trace and multiple-length *n*-grams were generated. This method combined with ELM is proven to be effective to increase the detection rate and lower the false-alarm rate.

Choosing the Optimal n-Gram. Tan et al. provided a theoretical and experimental investigation of choosing various sequence lengths [102]. 6-Gram and 7-gram are claimed to have better performance in UNM and ADFA-LD datasets, respectively [102, 67]. In [5], system call traces are collected from virtual machine user programs; 6-gram shows the best time efficiency and 10-gram can achieve the optimal detection result.

3.2 Enumerating Sequences

The enumerating sequences-based method, known as "sequence time-delay embedding (STIDE)" [88], is the first kind of system-call-based HIDS method introduced by Forrest et al. [34, 114, 53, 33]. STIDE is inspired by the natural immune systems of organisms. This approach is claimed to be simple and efficient to deploy for possible real-time implementation. Parameters of system calls are removed, endeavoring to reduce system load and obtain the best result with system call identifiers only. In this method, short sequences are extracted from normal execution of processes. Normal databases are constructed with these short sequences to detect anomalies. Databases do not have to contain every possible permutation of system calls; otherwise, no novel sequence can be detected as an anomaly. The two main stages of this method are described below:

- The first stage is for database construction. During this stage, the sliding window algorithm is applied to scan normal system call traces and generate short sequences, and then normal databases representing signatures of normal behaviors are constructed with these short sequences. Sequences are stored as a particular data structure in each database to reduce storage consumption and improve comparison efficiency.
- The second stage is for intrusion monitoring. During this stage, similar to the first stage, short sequences from testing system call traces are extracted and compared against the normal databases to get the number of mismatches. The number of mismatches is then used to identify anomalies from normal system behaviors.

Hamming Distance. Hofmeyr et al. utilized the Hamming distance, which is acquired by counting different positions of two system call sequences, to get the number of mismatches [53]. Every short sequence extracted from testing system call traces is compared with its corresponding normal database to find a known sequence with the minimal Hamming distance. If this minimal distance is larger than a user-defined threshold value, it means all sequences within the normal database may deviate from the testing sequence. Thus, the testing sequence is then regarded as an anomaly.

The Problem of Hamming Distance-Based Method. To guarantee a testing sequence is an anomaly, the sequence needs to be compared with all entries of the database to ensure every obtained

Hamming distance is larger than the threshold. Therefore, the process of finding the minimal Hamming distance may be time-consuming, especially if there are many anomalies or normal databases are large.

The Challenge of Enumerating Sequences-Based Method. In general, the enumerating sequences-based method requires building one normal database for each program. Consequently, different systems may generate different sets of normal databases. In the practical environment, normal databases need to be regularly updated to add new normal system call sequences generated by system and software upgrades. Therefore, efficient methods for building, updating, and maintaining those normal databases need to be designed.

3.3 Rule Learning

Lee et al. implemented a rule-learning-based method with continuous system call sequences to describe normal and abnormal kernel behaviors [70, 69]. During the data preprocessing stage, the sliding window algorithm is applied to traverse normal and intrusive traces to generate short sequences. Each short sequence is represented as a class-labeled vector, either normal or abnormal. Thus, a labeled dataset is formed and separated as training and testing partitions for the experiment. The rule learning algorithm RIPPER is applied to the training data. Testing sequences deviating from the set of predefined if-then rules are recognized as anomalies. The rules created are claimed to be concise for real-time implementation. Jiang et al. [60] generated n-grams of multiple lengths with normal system call traces and created an automaton of normal behaviors, and the automaton is used to detect anomalous behaviors. Tandon [106] designed some variants of the LEARD rule algorithm [76] to generate rules with system call sequences and their arguments. Qing et al. proposed an anomaly detection approach based on rough set theory [127]. A minimized set of rules is extracted from normal system call sequences to define a normal behavior model [127].

The Limitation of These Rule Learning-Based Methods. The rules described in the methods above are derived from traditional small-scale datasets. Therefore, these rules may be outdated for mining deep patterns from a large amount of system call traces generated by a data center.

3.4 Bloom Filter

The Bloom filter is often used for searching whether a new entry is in a set or not. Compared with STIDE, Bloom-filter-based methods have the advantages of light memory occupation, high searching speed, and effective privacy preservation. When a new short sequence is passed to the system, a set of hash functions can be applied to map the sequence to different positions in the Bloom filter. The Murmurhash [8], which is proven to have the advantages of simple implementation and high resistance of hash collisions, is often utilized with the Bloom filter. The two main stages for Bloom-filter-based HIDS methods are described below:

- During the construction stage, first, all bits in the Bloom filter are set to zero, k hash functions are selected manually, and all training normal short sequences will be processed by them. The hashing results are corresponding positions in the Bloom filter, which will be set to one.
- During the detection stage, for a new short system call sequence, k hashing results are generated and compared with corresponding bits in the Bloom filter. If at least one bit is zero, it means that the system call sequence is an anomaly. However, if all bits are one, the new short sequence may still be an anomaly since there is a small possibility of hash collision.

Anagram. Wang et al. proposed an anomaly detector called Anagram that applies a mixture of high-order n-grams and semisupervised training method on system call sequences [113]. Bloom filters are implemented in this method to save space and enhance privacy-preserving ability. Anagram can detect anomalies and model malicious behaviors with a "high detection rate and low false positive rate" [113] and shows robustness to some mimicry attacks [111].

Limitation of Bloom-Filter-Based Methods. Bloom filter has shown simplicity and effectiveness when computational space is inadequate. However, the limitation of the Bloom filter is that it allows false positives [14], yet it is expected that the FPR of a HIDS can be minimized. Moreover, considering the improvements in hardware and cloud computing capabilities recently, space is not the most significant factor. Therefore, improvements are expected to make Bloom filters further applicable for HIDS.

3.5 Classification and Clustering

The classification- and clustering-based methods here are referred to as a set of commonly used machine-learning algorithms that have been applied to system-call-based HIDS. Well-known models such as the k-nearest neighbor algorithm (kNN) [72, 129], k-means clustering algorithm (KMC) [117, 119], decision trees [129, 9], support vector machine (SVM) [129, 25, 118, 115, 7, 64], and Bayesian networks [87] have been implemented on system-call-based HIDS. For instance, Yuxin et al. proposed a behavior-based detection method with a semantic analysis approach [129]. An "executable" is presented as assembly code and then transferred to a control flow graph with flow-based analysis. Based on the control flow graph, a running tree is built from which the system call execution paths are extracted. Combining these paths can generate a system call stream from an "executable." System calls are represented as different integers. Features are extracted by using the *n*-gram approach with the sliding window algorithm and information gain method, and normalized feature vectors are formed. KNN, decision tree, and SVM classifiers are adopted for training and testing.

The Challenge. In the era of big data, how to fit traditional machine-learning algorithms in the current distributed computing environment is a challenge. It is necessary to improve the efficiency of current parallelization methods, which can be both on the algorithm aspect and on the data aspect; machine-learning algorithms can be redesigned to be executable in parallel, and large-scale datasets can be segmented properly to fit distributed machine-learning algorithms.

3.6 Hidden Markov Model

The Hidden Markov Model (HMM) is a doubly stochastic model that contains a finite number of hidden states [94]. Yeung et al. verified that applying HMM training on system call sequences operates better than modeling frequency distributions of shell command logs [128]. Hu [54] provided a detailed tutorial about how to apply HMM on system calls. HMM is widely known for its three fundamental problems shown in Table 2, which are applicable in system-call-based HIDS.

Initiation of HMM-Based HIDS with System Calls. Warrender et al. initiated the HMM-based approach for HIDS with system calls [114]. Several HMMs are trained with dozens of states according to the number of system call types in each test program, as the number of states should be determined in advance before the creation of an HMM [80]. States are completely connected and reachable from each other. The probabilities of state movement and system call generation are stored, which demand sufficient storage space for these intermediate results. Compared with the high time complexity of training, testing requires each single incoming system call to be examined, which is different from other system call sequence-based approaches. An alert is raised if the probability of an incoming system call is under a particular threshold.

Table 2. Three Fundamental Problems of HMM in HIDS

The evaluation problem
Given an HMM $\lambda = (\Lambda, B, \pi)$, obtain $P(O\|\lambda)$, the probability of $O = \{O_1, O_2, \ldots, O_t, \ldots, O_T\}$, $1 \le t \le T$ [54]. Λ is the state transition probability matrix, B is the observation probability distribution, and π is the initial state distribution, O is the observation sequence with time t [42]. A high $P(O\|\lambda)$ indicates that O is normal, whereas a low $P(O\|\lambda)$ indicates that O is abnormal. → *In HIDS, this problem is related to testing if an incoming system call trace is an anomaly.*
The decoding problem
Given an observation sequence O and a model λ, find the most probable hidden state sequence $Q = \{q_1, q_2, \ldots, q_t, \ldots, q_T\}$, $1 \le t \le T$ to maximize the joint probability $P(O, Q\|\lambda)$ [108].
The learning problem
Given an observation sequence O and a model λ, obtain the parameters (Λ, B, π) to maximize $P(O\|\lambda)$ [54]. → *In HIDS, this problem is related to training a model with the input observation sequence, i.e., benign system call trace.*

Combination of HMM with STIDE. Inspired by Forrest and Warrender's methods, Qiao et al. combined HMM with normal databases to propose a hybrid HIDS, which aims to discover a distinct difference between normal and abnormal kernel behaviors [93]. Part of preprocessed raw system call traces is taken to train an HMM filter using the Baum-Welch algorithm. Then all traces are passed to that HMM filter to get optimal state transition sequences using the Viterbi algorithm. Based on the research of Lee et al., conditional entropy can be used to measure the quality of input data [71]. The lower conditional entropy of input data can result in a model of higher performance. The conditional entropy of state transition sequences is lower than that of system call sequences. Hence, state transition sequences can represent kernel activities better. Then a normal database is constructed using these state transition sequences. The state number is set to 15. Thus, the database constructed is more concise compared with previous methods. The robustness is improved as the normal database can be almost constructed with only a small portion of training data.

Challenges of HMM Modeling. In practice, HMM modeling has been criticized for its high time complexity, especially when system call traces are extraordinarily massive and complicated in modern large data centers. Moreover, deep patterns of system call sequences usually cannot be mined by a single-layer HMM [47].

3.7 Neural Networks

Many HIDS works are related to neural networks, including multilayer perceptron [1, 26], self-organizing maps neural network [73], radial basis function (RBF)-based neural networks [3], extreme learning machine [25], self-structuring confabulation network [21, 20], and so forth [84]. Ghosh et al. applied the Elman neural network on the DARPA dataset for anomaly detection, and the performance of the Elman nets is better than the back-propagation neural network in the experiments [40]. Recently, deep learning has shown its capability of discovering underlying patterns within big data and is widely used in various data mining applications. Therefore, deep learning may be applicable for HIDS in the big data environment [68].

The Challenge. When it comes to deep learning, although it has started to show advantages, due to the increasing amount of system calls being generated, the training of a deep neural network

Table 3. Confusion Matrix

Predicted/Actual Label	Actual Intrusive	Actual Benign
Predicted intrusive	TP	FP
Predicted benign	FN	TN

and its fine-tuning may be complicated and time-consuming. One solution is using multiple GPUs deployed on a physical host to accelerate the training process. However, this solution may be pricey and space-consuming.

3.8 Validation Method and Evaluation Metrics

Cross-Validation. K-fold cross-validation is widely used to validate data mining algorithms [112]. Although k can be any integer number, 10-fold cross-validation is often applied. In 10-fold cross-validation, the original dataset is divided into 10 partitions with the same size, nine partitions are used for training, and one partition is used for testing [58]. Then another nine partitions and the leftover one partition will be used for training and testing. This cross-validation approach will repeat for 10 times with different training and testing data, and 10 results will be generated and averaged to get the final result [63]. The K-fold cross-validation method can provide a more precise estimation of the whole dataset.

Precision/Recall/F-Measure. A set of metrics is available for the performance evaluation of intrusion detection techniques. As HADS usually has two detection results, i.e., normal or anomalous, the performance can be evaluated using the Precision/Recall criteria [27]:

$$Precision = \frac{TP}{TP + FP} \quad Recall = \frac{TP}{TP + FN} \quad FPR = \frac{FP}{FP + TN} \quad TPR = \frac{TP}{TP + FN}, \tag{1}$$

where TP, FP, TN, and FN denote True Positive, False Positive, True Negative, and False Negative, respectively [65]. The meaning of these terms for system-call-based HIDS is described below:

- True Positive (TP) → The label of a sequence is positive, and the prediction is also positive. The anomalous sequence is alarmed properly.
- False Positive (FP) → The label of a sequence is negative, but the prediction is positive. An FP or a false alarm in system-call-based HIDS indicates that a benign system call sequence is treated as an anomaly.
- True Negative (TN) → The label of a sequence is negative, and the prediction is also negative. The incoming system call sequence is treated as benign properly.
- False Negative (FN) → The label of a sequence is positive, but the prediction is negative. An FN or a missed alarm indicates that the system fails to raise the alarm for an anomalous system call sequence.

Table 3 shows the confusion matrix of these terms.

FPR denotes the False-Positive Rate and TPR denotes the True-Positive Rate. In the area of IDS, TPR is equal to the detection rate (DR) and is equal to $Recall$; the False-Positive Rate is equal to the False-Alarm Rate (FAR) [27]. F-measure is often adopted to combine Precision with Recall for evaluation:

$$F_\beta = \left(\beta^2 + 1\right) \cdot \left(\frac{Precision \cdot Recall}{\beta^2 \cdot Precision + Recall}\right). \tag{2}$$

When Recall and Precision are weighted equally, β is equal to one. In this case, the F_1-measure is formed:

$$F_1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}. \tag{3}$$

The ROC Curve. The detection accuracy can also be evaluated with the Receiver Operating Characteristic (ROC) curve, which provides an intuitive view of the relation between FPR and TPR. The area under the ROC curve (AUC) can be a simple metric to provide an overall evaluation of the detection accuracy and can be obtained by calculating the trapezoids between each point under the ROC curve [27].

Time/Memory Complexity. The time complexity of an algorithm indicates the total time required by the algorithm, and it is commonly represented by the big O notation [27]. Memory complexity of an algorithm indicates the total memory required by the algorithm.

The Challenge. Evaluation metrics presented in HIDS articles are not consistent. Sometimes only one pair of TPR and FPR values is given, and the AUC value is not shown. Therefore, standardized evaluation metrics are recommended so that comparing different research outcomes can be easier.

4 HIDS DATASETS

The Dilemma of HIDS Datasets. In the area of HIDS research, there are insufficient publicly available datasets for conducting experiments. Traditional HIDS datasets with system calls cannot represent contemporary Linux systems. There are many studies describing the limitations of current HIDS datasets available on the Internet [74, 137, 99, 10, 78, 75, 2]. Some widely used system-call-based HIDS datasets are described below and compared in Table 4.

4.1 Current Datasets for HIDS with System Calls

DARPA and UNM. DARPA and University of New Mexico datasets are the two most commonly used datasets for evaluation of HIDS [89]. DARPA and UNM datasets were collected many years ago and may be outdated. They may contain errors and lack volume, variety, and veracity from the perspective of real-world big data [124]. Those two datasets only have a few kinds of attacks with a small data scale. Therefore, they are not representative of current diversified attack methods. The DARPA dataset consists of both network and host auditing data. Maggi et al. provided a thorough analysis of the flaws and shortcomings of the DARPA dataset [74]. System call traces of the DARPA dataset are relatively simple, and the UNM dataset consists of system call identifiers only. Other information such as system call arguments is removed. Therefore, traditional HIDS datasets with system calls cannot represent contemporary Linux systems.

Firefox-DS. The Firefox dataset [84] is a newly developed HIDS dataset that is publicly available. Created using modern penetration testing techniques such as Metasploit, the dataset contains normal and anomalous system call traces from several programs. In Firefox-DS, normal traces are acquired by running standard executions of the Firefox Internet browser. Five up-to-date attacks such as memory corruption exploit, integer overflow attack, DOM exploit, and pointer exploit are launched against Firefox 3.5, and complete system call traces are collected. Traces are grouped according to distinct Firefox processes, and each trace contains massive system calls.

ADFA-LD. The ADFA Linux Dataset (ADFA-LD) [24, 25] was created on the Linux operating system with preset vulnerabilities. The system is attacked by penetration testing tools with several contemporary attack methods. The dataset has 833 benign traces for training, 4,372 traces for analyzing the false-alarm rate, and 746 traces with six kinds of attacks for testing. The development of ADFA-LD aims to provide a new benchmark for HIDS analysis on contemporary computer systems. The ADFA-LD dataset only contains system call numbers. Therefore, comprehensive analysis

Table 4. Comparison of HIDS Datasets

Dataset	Type	Generation	Advantages	Disadvantages
DARPA	• Network and host auditing data	• Solaris with attacks and collected by BSM	• Widely used as the benchmark	• Few kinds of attacks with small data scale • System call traces are simple
UNM	• System call process IDs and numbers	• Unix with different kinds of programs and intrusions	• Widely used as the benchmark	• System call arguments are removed • Not representative of current diversified attack methods
Firefox-DS	• Normal and malicious Firefox system call traces	• Firefox and five different attacks	• Large-scale dataset with the completeness of normal behavior	• Focuses on Firefox web browser only • Referred only in a few studies
ADFA-LD	• Normal and malicious system call traces	• Ubuntu Linux and Metasploit	• Representative of contemporary attacks	• Contains system call numbers only • Only has six types of attacks with small data scale
NGIDS-DS	• Labeled host logs and network packets	• PerfectStorm commercial hardware platform	• A new dataset that is synthetically realistic	• Referred only in a few studies

with system call arguments cannot be applied. Xie et al. [117, 119] provided an initial evaluation of ADFA-LD with experiments using the simple kNN classification and k-means clustering methods. They analyzed feature vectors, applied dimension reduction, and determined optimal distance functions. The experimental result shows that frequency-based algorithms save computational resources compared with short sequence-based methods. They further improved the performance and reduced the computational cost using a one-class support vector machine (SVM).

ADFA-WD. Microsoft Windows is a widely used personal computing system. Haider et al. analyzed two complex HIDS datasets collected by Creech et al. [23] for Windows, namely, the "Australian Defence Force Academy Windows Dataset (ADFA-WD)" and the "Australian Defence Force Academy Windows Dataset with a Stealth Attacks Addendum (ADFA-WD: SAA)" [44]. ADFA-WD involves acknowledging "Windows-based vulnerability-oriented zero-day attacks" [44] using automated penetration testing tools. As an extension of ADFA-WD, ADFA-WD: SAA is designed to test the effectiveness of HIDS against "Windows-based stealth attacks" by "crafting three

stealth attacks, namely, Doppelganger, Chimera, and Chameleon" [44]. These two datasets are not designed for Linux system call analysis.

NGIDS-DS. Haider et al. proposed a synthetical high-quality and realistic IDS dataset named the next-generation IDS dataset (NGIDS-DS) using the advantageous Ixia PerfectStorm commercial hardware platform [57] and infrastructure in ADFA [45]. The dataset contains 99 CSV files of labeled host logs for the design of HIDS as well as network packets for the design of NIDS. The host logs contain various information, such as the time of activities, process IDs, execution paths, and system call numbers for comprehensive system call analysis. They also assessed the quality of existing IDS datasets including DARPA and ADFA-LD with a fuzzy logic system based on the "Sugeno fuzzy inference model" [101].

4.2 Dataset Customization

With the rapid evolution of system and software, due to the drawbacks of existing datasets such as DARPA and UNM, datasets relevant to the current environment of cybersecurity need to be established. Acquiring a real-world intrusion dataset is difficult, as companies often do not want to share data with the public. Therefore, researchers tend to launch experiments and generate new datasets that can represent current intrusion methods. In their experiments, normal training data can be acquired by gathering benign system call traces during the normal execution of supervised processes. Intrusive testing data can be acquired by collecting attack system call traces caused by applying penetration tools such as Metasploit [95] on different kinds of vulnerabilities according to their Common Vulnerabilities Exposures (CVE) ID [22]. Table 5 compares typical system call tracing tools. Table 6 provides common tools for penetration testing.

4.3 The Challenge

In this section, we have analyzed the issues of traditional HIDS datasets, introduced several contemporary datasets for HIDS research, and offered techniques to generate new datasets. Researchers should make significant efforts to guarantee that all generated public datasets have high quality, which ensures that relevant studies will not be misled by the datasets.

5 THE APPLICATION OF SYSTEM-CALL-BASED HIDS ON EMBEDDED SYSTEMS

Embedded systems such as smartphones are gradually attracting users' attention due to their computational capability and portability. Meanwhile, the issue of smartphone security has arisen. New threats are continuously occurring due to new functionalities of the smartphone. Android is a popular mobile operating system with the Linux kernel, which is similar to Linux operating systems installed on desktops [116]. Android smartphones can be compromised by the same types of attacks against Linux. Conventional defensive techniques cannot be simply transferred to Android due to the embedded environment. The computational capability of smartphones is still limited for time-efficient malware detection. For Android devices, lightweight anomaly detection approaches should be developed so that the execution of detection software will not affect other applications. The detection software should conform to three design rules, i.e., accurate detection result, quick detection speed, and minimal resource usage.

5.1 The Feasibility of Applying System-Call-Based HIDS to Embedded Systems

Recently, system-call-based HIDS has been applied to embedded systems such as smartphones and is proven to be effective to detect intrusive behaviors. Feizollah et al. proposed a review of recent feature selection approaches for developing an effective malware detection system of mobile devices [30]. Features are categorized into four types, namely, "static features, dynamic features,

Table 5. Comparison of System Call Tracing Tools

Tracing Tools	Functional Summaries	Other Features	Platforms
strace[1]	The strace monitors communications including system calls between processes and the Linux kernel.	The operation of strace is promoted by ptrace.	Linux
KProbes and DProbes[2]	KProbes and DProbes can debug errors and monitor events in Linux kernel.	KProbes is the built-in mechanism of DProbes, which can insert probes dynamically into running code modules.	Linux
SystemTap[3]	SystemTap reduces the load of collecting information of the operating system, which helps to analyze functional errors.	The command line interface and scripting language are simple to use.	Linux
ltrace[4]	Similar to strace, ltrace can monitor and record system calls invoked by a running process.	ltrace can also intercept and print the system calls.	Linux
LTTng[5]	The "Linux Trace Toolkit next generation (LTTng)" is a modern open-source tracing platform, which consists of kernel modules to simultaneously trace the Linux kernel.	LTTng also has a kernel module to trace shell scripts.	Linux
BSM[6]	Solaris Basic Security Module (BSM) is an auditing tool provided by SUN Microsystems for system security. It provides comprehensive auditing of system processes.	BSM offers methods to check the history logs of actions and events.	Solaris
Truss[7]	Truss can execute a specified command, invoke system calls, and produce a trace, each line of which contains a system call name with arguments and values.	Truss is developed for other Unix-like operating systems except for Linux.	Solaris UnixWare AIX FreeBSD
DTrace[8]	DTrace is a real-time debugging tool for kernel and application errors. DTrace provides a full description of the executing system.	Some fine-grained information can be provided, e.g., an argument log or a process list.	FreeBSD NetBSD OpenBSD Mac OS
ktrace[9]	The ktrace is a utility that traces interactions including system calls between kernel and executing processes.	The traced information can be dumped to external hard drives and saved as files for debugging and analysis. The ktrace is similar to strace.	FreeBSD NetBSD OpenBSD Mac OS

hybrid features and applications metadata" [30]. System calls and network traffic are "two main types of dynamic features" [30]. Many researchers used system calls as dynamic features as applications use system calls to communicate with the mobile operating system [86]. They surveyed 100

[1] https://linux.die.net/man/1/strace/.
[2] https://lwn.net/Articles/132196/.
[3] https://sourceware.org/systemtap/wiki/.
[4] https://linux.die.net/man/1/ltrace/.
[5] http://lttng.org/docs/v2.9/.
[6] https://docs.oracle.com.
[7] https://docs.oracle.com/.
[8] http://dtrace.org/blogs/about/.
[9] https://www.freebsd.org/cgi/man.cgi?ktrace(1)/.

Table 6. Tools for HIDS Dataset Generation

Tools	Categories	Features
CVE[10]	• Dictionary of vulnerabilities	• Provides identifiers of publicly known cybersecurity vulnerabilities
Metasploit[11]	• Penetration testing framework	• Enables exploiting vulnerabilities
Kali Linux[12]	• Advanced penetration testing platform	• Provides built-in penetration testing tools
VirusShare[13]	• Malware repository	• Provides samples of live malicious code
VX Heaven[14]	• Virus repository	• Provides massive continuously updated virus samples and information
Contagio[15]	• Malware repository	• Provides contemporary malware samples • Provides malware analysis
AVCaesar[16]	• Website of malware analysis	• Provides a malware repository • Provides multiengine malware analysis
VirusTotal[17]	• Online multiengine malware detector	• Enables analysis of suspicious files and URLs

articles between 2010 and 2014, discovering 42% of all research works utilized dynamic features. System calls were found to be the most regular dynamic features used.

Static/Dynamic Analysis. Static analysis and dynamic analysis are two primary analysis methods for smartphone malware detection [31]. Static analysis is related to examining static patterns of applications such as source codes to address malicious behaviors without executing. Static analysis may not be valid after codes are obfuscated. Dynamic analysis is about executing applications in a safe and isolated sandbox to evaluate their running behaviors. Static and dynamic analysis can work together to enhance malware detection capabilities. Blasing et al. proposed "Android Application Sandbox (AASandbox)" that implements "static and dynamic analysis on Android programs" for malware detection [13]. Static prechecks are performed before the installation of Android applications. Dynamic analysis is about running the application in an isolated sandbox, and system calls are traced. The sandbox can be deployed in the cloud for distributed and resource-intensive execution.

Crowdroid. Burguera et al. provided a lightweight malware detection system called Crowdroid, which is available on Google Play [16]. They indicate that monitoring system calls can provide detailed low-level information, which is feasible to determine malware behaviors. The system

[10] https://cve.mitre.org/.
[11] https://www.metasploit.com/.
[12] https://www.kali.org/.
[13] https://virusshare.com/.
[14] http://vxheaven.org/.
[15] http://contagiodump.blogspot.com.au/.
[16] https://avcaesar.malware.lu/.
[17] https://www.virustotal.com/.

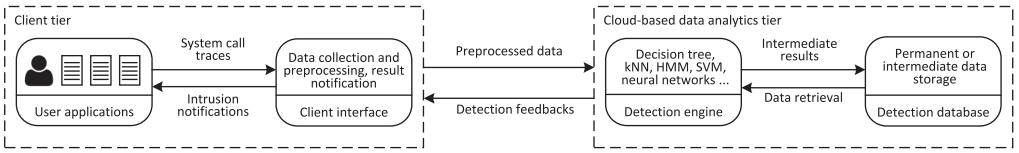

Fig. 2. A two-tier HIDS platform for mobile devices.

includes an overall framework for the collection of system call traces and the analysis of data; the two-means clustering algorithm is applied to classify benign and intrusive applications [16]. Users will be notified if an abnormal system call trace is detected by the system. The experimental results confirm the feasibility of collecting and analyzing system calls for malware detection. They discover that system calls *open*, *read*, *access*, *chmod*, and *chown* are mostly referred by malware [16]. They also provide a brief survey, demonstrating that many malware detection approaches are based on NIDS and HIDS techniques.

5.2 Enhancement with Hardware

Software-based HIDS may not show ideal performance against sophisticated intrusion techniques. Considering this issue, Das et al. proposed the first hardware-enhanced system-call-based HIDS framework named GuardOL with machine-learning approaches to detect malicious behaviors toward the embedded systems [26]. They claimed that hardware-based HIDS was resistant to malicious software. Attacks are launched using Ubuntu OS, and malware samples are downloaded from public malware repositories available on the Internet. Benign traces with arguments are collected from the normal operation of Linux applications with strace. For feature extraction, FCM is developed, which is a frequency-centralized model to comprehensively classify system calls with arguments to reduce the false-alarm rate. Offline analysis and online hardware implementation are conducted in FPGA. An artificial neural network model named multilayer perceptron is selected from a couple of machine-learning classifiers and trained in Weka with features extracted from benign and malicious traces. Ten-fold cross-validation is used to test the accuracy of prediction [26].

5.3 Cloud-Based HIDS with System Calls for Embedded Systems

Deploying system-call-based HIDS as a two-tier platform shown in Figure 2 is prevalent. The first tier is the client interface installed on users' devices for data collection. The second tier is a cloud-based and centralized data analytics unit. Detection feedback processed by the centralized unit can be returned to users' devices in a timely manner, and further defensive actions can be launched.

Dual Defense Protection. Su et al. proposed a "dual defense protection framework" for Android malware detection [100]. Applications can be uploaded to servers in the cloud for verification. The framework has two main components: a system call monitoring tool and network monitoring tool. The system call monitoring tool is utilized for evaluation of the "new application for potential malware," and the network monitoring tool is applied to solve the false-negatives problem [100]. The experiments show that the framework produces high detection accuracy with machine-learning classifiers. The Weka tool is used for experiments; the random forest classifier in the tool performs better than the J.48 classifier, achieving accuracies of 99.2% and 94.2%, respectively.

The Challenge of Deployment in Cloud. Although many research works have proven that it is feasible to transfer detection applications and algorithms to the cloud environment, it has to guarantee that the detection mechanisms in the cloud and the smartphone can be synchronized at all times. Therefore, the detection can be performed in a timely manner, and further responses to the malware will not be delayed.

6 FUTURE RESEARCH TRENDS OF HIDS

In previous sections, we concentrated on the review of HIDS development. In this section, future research trends of HIDS are discussed regarding three aspects, namely, the reduction of false-positive rate, the improvement of detection efficiency, and the enhancement of collaborative security. Possible challenges of these three aspects are discussed. This section aims to inspire researchers in the community to collaboratively push forward the frontier of HIDS.

6.1 Reduction of False-Alarm Rate

False alarms may consume and waste human resources of security personnel, as it is their responsibility to take appropriate actions on each intrusion alarm. Therefore, a high false-alarm rate can affect the performance of HIDS and has to be lowered down to a minimal level to save resources. Recently, researchers have tried multiple ways to reduce the false-alarm rate.

6.1.1 System Call Arguments. Although Forrest et al. initially eliminated all arguments of system calls to test the performance with that simple assumption, research works have shown that modeling system call arguments and return values together can enhance the detection accuracy and lower the false-alarm rate. Mutz et al. "applied multiple detection models to system call arguments," and anomaly scores from each model are combined into an overall score for judgment of anomalies [87]. Results have shown not only that taking system call arguments to train Bayesian networks can improve the detection accuracy but also computation and memory resources can be saved as well. Maggi et al. [74] also indicated that system call arguments could also contain anomalies, and they provided an unsupervised system that analyzes system call traces and arguments. Das et al. developed FCM, a frequency-centralized model, to comprehensively classify system calls with arguments to reduce the false-alarm rate [26].

The Problem. Although using system call arguments or other host logs can reduce the false-alarm rate, it may deviate from the simple hypothesis that Forrest initially made on system-call-based HIDS.

6.1.2 Improve Feature Extraction Approaches. Currently, novel feature extraction methods are being developed to achieve more effective detection.

Contiguous and Discontiguous System Call Patterns. Creech et al. designed a new HIDS approach with syntactic development and semantic hypothesis to lower the missed-alarm rate and false-alarm rate [25]. Word and phrase dictionaries are formed with various-length system call numbers. They trained an extreme learning machine (ELM) [56] for anomaly detection and achieved a notable detection rate and false-alarm rate. The result shows the effectiveness of handling mimicry attacks. Decision engines SVM, HMM, and ELM are applied for training and prediction. The approach is tested with DARPA, UNM, and ADFA-LD datasets.

Integer Data Zero-Watermarking. To select representative features from system call traces, Haider et al. proposed an "integer data zero-watermarking algorithm" to extract abstract hidden reliable or representative features from system call traces of the ADFA-LD and DARPA datasets [46, 48]. The proposed method in conjunction with multiple machine-learning algorithms shows acceptable real-time performance regarding accuracy and processing time.

Mutual Information. In the area of NIDS, considering that redundant features existing in big data may affect the efficiency and correctness of the classification result, Ambusaidi et al. [7] proposed a practical method with mutual information-based preprocessing and feature extraction for NIDS to deal with issues caused by duplicated and worthless features in the dataset. This method is potentially applicable in HIDS. The algorithm proposed helps to extract more critical features from raw

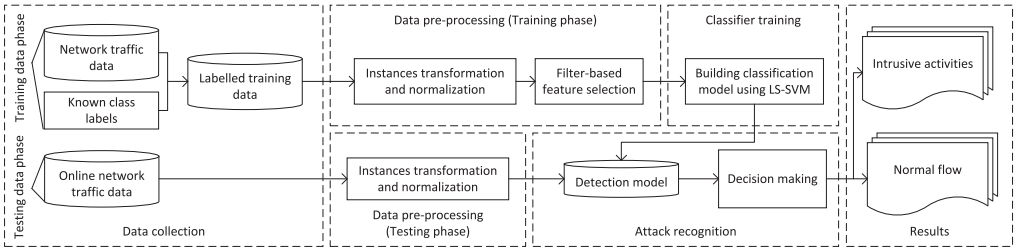

Fig. 3. LS-SVM-based intrusion detection system, modified from [7].

data and can deal with linearly as well as nonlinearly dependent features. They also implemented an IDS with Least Square SVM (LSSVM-IDS), which is combined with features extracted using the mutual information-based approach. The false-alarm rate is lowered, and the computational resources are saved. Figure 3 represents their proposed system.

6.1.3 Refine the Decision-Making Process. Nauman et al. proposed a "three-way decision-making approach" to reduce the false-positive rate [88]. The model is based on rough set theory with three options, namely, acceptance, rejection, or deferment. If the information of an intrusion is insufficient, then a decision will be deferred. "Game-theoretic rough sets (GTRS)" and "information-theoretic rough sets (ITRS)" [88] are exploited to construct the model, and the UNM dataset is taken for the experiment. A minimum 8.5% false-positive rate is achieved with the proposed approach.

6.1.4 Threshold Optimization. The threshold of HIDS controls the sensitivity. If the predicted value of an anomalous system call sequence is higher than a particular threshold, then an intrusion alarm will be triggered. Low sensitivity may cause a high missed-alarm rate, yet high sensitivity may cause a high false-alarm rate. A large number of false alarms can consume additional maintenance time of security personnel. Therefore, optimizing the threshold of HIDS is important and is a challenging work that requires knowledge and experiences. Laszka et al. [67] discussed related problems of generating the optimal threshold and built an algorithm for threshold selection. ADFA-LD is used for evaluation, showing that the proposed algorithm achieves a better result than the "optimal uniform strategy" and "locally optimal strategy" [67].

6.1.5 The Integration of Decision Engines.

Fuzzy Inference Engine. Hoang et al. built a "multi-layer model of program behaviors" [50] based on HMM and the enumerating sequences method using the temporal characteristics. They further proposed a hybrid HIDS scheme [51] that integrates HMM with the enumerating sequences method using a "fuzzy inference engine" to decrease the false-alarm rate. HMM is trained by adopting a modified HMMMOSA scheme [28] with optimal initialization of parameters, which reduces training time and the model volume. Input sequence parameters used by the "fuzzy inference engine" are "generated by HMM and normal databases" [51] as two detection engines. Fuzzy sets are created empirically, and fuzzy rules obtain assumptions from the two detection engines. For each testing input, a three-phase fuzzy reasoning process is applied to generate an output value.

Integration in the ROC Space. To reduce false alarms, Khreich et al. proposed a multiple-detector HIDS that integrates the decisions from various traditional detectors such as STIDE, HMM, and One-Class SVM with "Boolean combination in the Receiver Operating Characteristics (ROC) space" [64]. Experiments on the Linux dataset (ADFA-LD) and the Window dataset (CANALI-LD [17]) show that the proposed method can reduce the false-alarm rate and substantially increase the

true-positive rate on both of the two datasets [64]. To the best of our knowledge, when it comes to the detection accuracy of ADFA-LD, currently their work may achieve the state-of-the-art performance.

6.1.6 Long Short-Term Memory. Long short-term memory (LSTM) is an architecture of the recurrent neural network (RNN) first presented by Hochreiter et al. [52]. The LSTM network can lengthily persist temporal information and therefore is adaptive in the situation when there are notable time intervals between significant events within input sequences. Gates included within an LSTM block are helpful to determine when it is necessary to remember the input value and when it should retain, discard, or output the relative value.

Hierarchical LSTM. Although LSTM can handle relatively long sequences, different from other areas in which LSTM is employed such as natural language processing, a system call trace may contain thousands of system calls, which make corresponding long time dependencies difficult to capture by LSTM. This dilemma may be assisted by hierarchical LSTM, which has a hierarchical structure that can model the temporal transitions between single system calls as well as system call sequences with different granularities [91].

The Attention Model and Review Network. For implementing LSTM on system call traces for anomaly detection, the prediction will rely on the final hidden state. One assumption, in this case, is that the network has to encode all essential information of system call traces into equal-sized vectors, which may make it difficult to manage long traces, especially those exceeding the size of most training traces. Instead of compressing all information of system call traces into fixed-length vectors, the attention model [120] can encode input traces into a chain of vectors, and for prediction, a subset is flexible to be chosen from the chain accordingly, which may perform better on large traces generated by programs. Yang et al. indicated that the attention model operated sequentially and proposed a review network, which is an extension of the encoder-decoder framework, to augment global modeling capability [126].

The Challenge of LSTM. LSTM has gained notable advances in various sequence processing tasks such as speech processing. Thus, LSTM is potentially applicable in the area of system-call-based HIDS. However, the challenge is that modeling of LSTM is often complicated and computationally expensive.

6.1.7 Challenges Regarding This Trend. Controlling the false-alarm rates is still the most significant challenge and the critical point for both NIDS and HIDS, regardless of the kind of platform on which HIDS is deployed. Given the wide range of normal behaviors of the hosts in a network, it is possible that a detected anomaly is normal (i.e., legitimate). The false-positive rate can be affected by many factors, including feature selection, detection engine design, and threshold optimization. As new Linux applications and functions keep being developed, the generated system call traces can be unstable and miscellaneous. Therefore, how to preprocess the raw dataset and select the right features can influence significantly the performance of the detection system. Adjusting the detection threshold is also an arduous work, and it may require decisions from experienced security experts. Besides the quality of the designed detection engine, another essential factor that can affect the detection accuracy and false-positive rate is the quality of the training dataset.

6.2 Improvement of Detection Efficiency

Detection efficiency is another significant issue in real-time intrusion detection. Detection speed and accuracy are usually difficult to make well balanced. When there is a heavy load of the Linux kernel operation, it may take a significant amount of time to process the system call traces generated during a short period. Intermediate datasets may be too massive to be saved in the RAM of

a single machine. In this case, the real-time detection of intrusions may be delayed. Researchers have been seeking methods to improve the detection efficiency of HIDS.

6.2.1 Refine the Dataset Quality. Besides the detection accuracy, the quality of datasets can also influence the efficiency of detection engine training. Hu et al. provided an acceleration method of the HMM incremental training process, which reduces half of the training time for the UNM dataset [55]. During the preprocessing period, near-duplicate sequences of system calls are deleted. The detection rate of anomalies remains almost unchanged, whereas the false-alarm rate is downgraded as more than half of the original data are eliminated.

6.2.2 Improvement of Decision Engines.

Kernel State Modeling. Murtaza et al. developed a trace abstraction technique called Kernel State Modeling (KSM) to reduce the processing time and false-alarm rate of HADS by representing system call traces as traces of kernel modules [84, 85]. Abstract traces generated are used as the training data of previous approaches, such as STIDE and HMM. Experiments are launched with UNM, Firefox-DS, and ADFA-LD datasets. The result shows a lower false-alarm rate and less execution time. Their later work presented a creative and extensible Eclipse-based open-source HADS framework named the automated anomaly detection framework (Total ADS) [83], which can train a variety of decision engines with normal system call streams, raise anomaly alarms, and perform visualization automatically. The proposed framework allows training and testing of three remarkable HADS methods, namely, STIDE, HMM, and Kernel State Modeling (KSM) [84], with real-time system call streams.

Real-Time Self-Structuring Method. Recently, Chen et al. proposed a real-time self-structuring learning framework named anomaly recognition and detection (AnRAD) for unsupervised anomaly detection of streaming data [21, 20]. They aimed to exploit the computational capability of parallel computing platforms efficiently. The feature dependencies of the DARPA and ADFA-LD datasets are analyzed, respectively. Unlabeled system call data are used to train an efficient confabulation network with the self-structuring method. The result shows high-speed incremental learning of data streams as well as acceptable detection accuracy. The knowledge base can be updated regularly and rapidly. Meanwhile, the method implemented on parallel architectures obtains apparent speedups, with ideal performance and memory efficiency.

Nested-Arc-HSMM. Concerning the big data issue caused by a significant amount of system call traces, Haider et al. proposed a HIDS for hosts in the cloud environment by integrating "state summarization" and the novel "nested-arc hidden semi-Markov model (NAHSMM)" [47]. Designed according to the hierarchical HMM [32] structure, the NAHSMM has two layers of hidden Markov chains. Tested with NGIDS-DS, the proposed method shows acceptable detection accuracy, training efficiency, and scalability.

6.2.3 Capability of Cloud Computing. Currently, cloud computing provides a powerful infrastructure for scalable large-scale data processing by using its flexible computation and storage capability. With cloud computing, large-scale computation and storage resources such as RAMs, multiple GPUs, or solid-state drives can be allocated based on requirements. Therefore, the capability of cloud computing can be utilized to strengthen the performance of HIDS. Intrusion detection algorithms can run in the cloud to process real-time high-volume system call streams. Acceleration methods for HIDS used in the community can also be integrated into cloud computing platforms. Table 7 provides an overview and comparison of public/open-source cloud services.

6.2.4 Open-Source Big Data Tools. The development of scalable big data processing tools such as Hadoop shows high scalability for big data processing [136, 135, 133, 125, 123, 122]. These tools

Table 7. Overview of Public/Open-Source Cloud Services

Public cloud services → *Three forms*
Public cloud services usually have three forms, namely, "Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS)" [62]: • *SaaS* utilizes the Internet to provide services and applications. Most SaaS applications are accessible and controllable through a web browser. • *PaaS* provides an efficient cloud-based built-in middleware on which applications can be developed. • *IaaS* presents in-cloud servers, storage, and network hardware for users to rent instead of purchasing physical servers. IaaS users are responsible for managing allocated virtual machines. →*Google Compute Engine and Amazon EC2 are two commercial providers of public cloud services.*
Public cloud → *Advantages*
• *Scalability*. Deploying clusters in the public cloud is more flexible and scalable compared with deploying on physical machines. • *Efficiency*. Users pay for demanded computational capacity and time. The time required to launch instances and scale computational capacity can be significantly reduced. • *Flexibility*. Demanded computational resources can be flexibly allocated via a variety of console interfaces on the web page. Users can quickly boot, control, and manage various types of virtual machines (instances) with stable performance as required. • *Reliability*. Deploying HIDS in public cloud services provided by trusted large IT corporations such as Amazon and Google can be more reliable and less vulnerable compared with local networks, considering intruders may attack the intrusion detection software based on the vulnerabilities.
Public cloud → *Disadvantages*
• *Cost*. Uploading and downloading large amounts of data may cause additional data transfer cost charged by public cloud providers. →*Nevertheless, while software and hardware technologies for cloud computing are still being developed, the cost may gradually decrease, and the dependability may progressively increase with improving security measures.*
Open-source cloud → *Advantages*
• *Cost-effectiveness*. Open-source software usually is freely available on the Internet. • *Flexibility*. As source codes are available, researchers can build the cloud platforms themselves and customize the services to fulfill their functional requirements.
Open-source cloud → *Disadvantages*
Open-source cloud software is often developed by unpaid developers in loosely organized communities. Therefore, there are also some disadvantages. • *Usability*. Open-source cloud platforms are often cumbersome to deploy and do not have user-friendly APIs, which may influence the efficiency and effectiveness of working. Users usually cannot get instant customer support and therefore they need more effort of training and learning. • *Compatibility*. As the open-source cloud software is still under development, software with different versions are often not compatible with each other. Thus, the entire cloud platform may not be operating sometimes. • *Cost*. Although open-source cloud platforms are usually free for the services, the time-consuming installation and maintenance may cause other forms of expenses. • *Security*. Open-source software is often criticized regarding security, as the source codes are exposed to potential attackers. If a HIDS is deployed in an open-source cloud, then the system itself may be vulnerable.

Table 8. Comparison of Big Data Tools

Tools	Functional Summaries	Advantages	Limitations	For HIDS
Hadoop	• Distributed big data processing and storage • Computation →*MapReduce* • Storage →*HDFS*	• Large-scale static data processingy • Fault-tolerance capability	• Complete dataset must be loaded before processing →*High I/O cost*	• Long-term storage for HIDS results
Spark	• In-memory big data processing • Fault-tolerant distributed dataset abstraction →*RDD* • Lineages of RDDs →*DAG*	• Intermediate datasets are cached into distributed memory →*Facilitates iterative algorithms*	• Cost may be high due to the high RAM requirement	• Preprocessing and feature extraction • Training of decision engines
Spark Streaming	• Scalable fault-tolerant streaming data processing →*Discrete streams* • Divide streams into micro-RDD batches →*Predefined time intervals*	• Suitable for streaming system call traces	• Predefined time intervals →*Not actual real-time processing*	• Real-time intrusion detection of system call traces
Kafka	• Distributed messaging platform • Fault-tolerant data caching • Real-time processing of data streams	• Resilient to data loss →*Data streams can be duplicated for backup*	• The possibility of message lost	• Intermediary between multiple hosts and Spark cluster →*Propagating system call streams*
Alluxio	• In-memory distributed storage system • Designed for solving the problem of on-heap storage in Spark	• Off-heap storage →*Avoid the problem of full garbage collection (FGC) in JVM*	• Not suitable for long-term storage →*Data in RAM may be lost after services terminate*	• Off-heap storage solution in real-time HIDS →*Fault-tolerant and efficient execution*

are designed to perform underlying resource management such as task and fault-tolerant scheduling, providing simplified APIs to data engineers. Data mining tasks, especially for streaming data, can be distributed across clusters by these tools to achieve scalability. Fault tolerance is another important issue. With these big data tools, a failed job can be taken over by another worker rather than rolling back to the beginning and computing again, which saves computational time. Therefore, the integration of cloud computing and big data processing tools can provide a new vision for solving security problems. A set of open-source big data tools applicable for HIDS are described below and compared in Table 8.

Apache Hadoop. Hadoop is a popular open-source distributed big data processing and storage framework [96]. Next-generation MapReduce on YARN is the computing framework, and HDFS is the storage framework. HDFS can be used as long-term storage for HIDS results because of the fault-tolerant capability.

Apache Spark. Spark is an in-memory framework for distributed big data processing [79]. Different from MapReduce, intermediate datasets in Spark can be cached into distributed memory, which is reasonable for iterative statistical machine-learning algorithms. Spark has a master

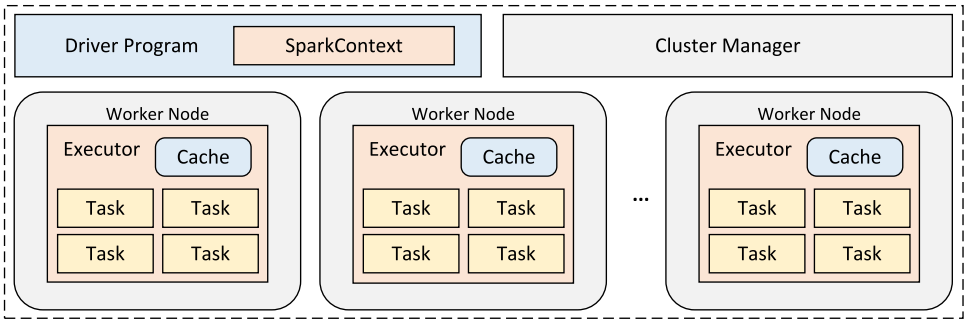

Fig. 4. A standard Spark cluster [39].

driver program, which controls its workers on a cluster. Consequently, this framework is a considerable solution for processing large-scale system call traces. Resilient Distributed Dataset, or RDD [131], represents Spark's fault-tolerant distributed dataset abstraction. Lineages of RDDs are represented by a Directed Acyclic Graph (DAG); if one partition of an RDD is lost, the DAG has the record of how that partition is acquired from other partitions [130]. Therefore, it only needs to recompute that partition according to its lineage, instead of recomputing the whole DAG again. Figure 4 shows the structure of a standard Spark cluster. Spark currently can run on three kinds of cluster managers, namely, Standalone, Mesos, and YARN. Recently, for deep learning acceleration, utilizing Spark in the cloud environment shows more flexibility compared with traditional methods. Philipp et al. introduced SparkNet [82], which implements a deep neural network training framework on Spark, including interfaces of loading data from other RDDs and the connection to Caffe [59].

Spark Streaming. Spark streaming is a scalable fault-tolerant streaming data processing framework [132]. System call traces are generated based on time intervals of hosts, which conform to the computing mechanism of Spark Streaming. To obtain efficient fault tolerance and low latency during real-time HIDS implementation, Spark Streaming can divide the incoming real-time streaming system call traces into small RDD batches according to micro time intervals.

Alluxio. Alluxio is an open-source in-memory distributed storage system that has a master-workers structure similar to HDFS [35]. As a component of the real-time scalable HIDS, Alluxio can be deployed in the same cluster with Spark. Alluxio was initially designed for solving the problem of on-heap storage in Spark. With off-heap storage, objects are stored out of the heap, which can avoid FGC in JVM. Alluxio can be an off-heap storage solution for real-time Spark-based HIDS implementation.

Apache Kafka. Kafka is a contemporary distributed messaging platform [37]. In the real-time HIDS data processing framework, Kafka can operate as an intermediary between multiple hosts and the Spark cluster [97], propagating system call streams from origin to destination.

Scalability of Algorithms in Spark. In the current big data environment, from the perspective of real-time implementation, the scalability of an algorithm should be ensured to deal with the increasing amount of data. Table 9 discusses the scalability of a set of data mining algorithms in big data tools, particularly for system call-based HIDS.

6.2.5 Challenge Regarding This Trend. Although the detection efficiency of HIDS can be improved with cloud computing and big data tools, how to effectively process massive HIDS data in a centralized and collaborative manner is still challenging.

Table 9. Scalability of Algorithms in Big Data Tools

Scalability of preprocessing and feature selection
• System call traces are capable of being distributed evenly to Spark workers so that data preprocessing can be operated in parallel.
Scalability of classification and clustering
• The scalability of the classification- or clustering-based HIDS method is capable of being accomplished by MLlib, which is a machine-learning library on Spark that helps to accelerate distributed machine-learning algorithms. • Classification models such as logistic regression, decision tree, and random forest and clustering models such as k-means, latent Dirichlet allocation, and Gaussian Mixture Model can be trained and predicted to distribute using MLlib [66]. • It is flexible to implement saving and loading RDDs of trained models with off-heap storage, which is appropriate for real-time HIDS prediction.
Scalability of normal databases
• The design of Spark is suitable for the method of STIDE. • In Spark, normal system call sequences can be either cached temporarily in memory or stored permanently in external storage such as HDFS. • As massive system call sequences may result in a large-scale traditional database, it may be inefficient to launch a database query when a new short system call sequence comes. Therefore, NoSQL databases can be utilized. NoSQL databases allow data accesses based on key-value pairs and values can be returned by the keys passed.

6.3 Enhancement of the Collaborative Security

Cloud computing and big data tools can be utilized to improve the detection efficiency of system-call-based HIDS. Currently, researchers are focusing on constructing comprehensive real-time HIDS for data center/cloud platforms. Moreover, to enhance the collaborative security, it is a significant trend that HIDS should combine with NIDS to form future CIDS.

Definition of Data Center. A data center, which contains computer systems, network systems, and database systems, can manage all data of a modern enterprise [97]. A data center is composed of a group of physical hosts. Virtual hosts are deployed on each of the physical hosts. A virtual host is an operating system administrated by the virtual machine monitor (VMM), which is a program installed on a host system that helps one physical host run on various computing environments. Applications such as databases, DNS servers, and web servers are usually distributed across multiple virtual hosts.

Threats of a Data Center. The comprehensive deploy mode of a data center can attract advanced intrusions. Virtualization technologies applied in a data center have led to the occurrence of new attack methods against vulnerabilities in virtual machines deployed on physical hosts. Hackers can exploit those vulnerabilities to set up Trojans and then obtain advanced system priorities or obtain the private information of data center users. The computational capacity of a data center may be taken by attackers to launch DDoS attacks toward the infrastructure of the data center. By exploiting a compromised virtual machine, an intruder can perform intrusions toward more VMs, the virtual machine monitor, or the operating systems of physical machines.

6.3.1 *Current System-Call-Based HIDS Approaches for Virtual Hosts.*

System-Call Capturing. From the perspective of HIDS, to deal with intrusions against a large data center, auditing data generated from hosts or virtual hosts of a data center need to be gathered for

intrusion analysis. Pfoh et al. designed a virtual machine introspection-based framework to capture and monitor system calls in real time [92]. This framework is based on a modified kernel-based virtual machine (KVM) and is segregated from guest OSs. Approaches for trapping desired events of system calls to the hypervisor are designed. Interrupt-based, syscall-based, and sysenter-based system calls are supported [92]. Users can control the granularity of tracing the system call data.

Bag of System Calls. As the public IaaS cloud environment is vulnerable to multiple novel intrusions, Alarifi et al. proposed a "bag of system calls" method to detect anomalies, particularly for mimicry attacks [5]. Experiments are conducted via Linux KVM, and system calls are collected from virtual machine user programs to guarantee the fine granularity. A normal profile is created for the experimental virtual machine. For the bag-of-system calls, sequence length 10 has the optimal detection result, whereas length 6 has the best time efficiency. Their next experiment [4] uses HMM with predefined capacity as the normal profile. Virtual machines are first running normally so that normal system calls can be gathered. Normal traces are the input for training, and malicious system calls generated by a DoS attack are for testing. The size of dataset required for classifier training and testing is small in their experiment as only simple services are installed on virtual machines and behaviors in public cloud are assumed static. Virtual machines are treated as black boxes by the premise that VMs are only reachable by IaaS users.

Intrusion Severity Analysis. Arshad et al. proposed an intrusion severity analysis method [9]. It assumes that system hardware is error-free. Decision trees are used in this method, as they are simpler to manipulate and require less training data than neural networks. A Virtual Machine System Call Handler collects and passes system calls to an anomaly or misuse intrusion Detection Engine, which may consult an Attack Database with existing intrusion signatures or a Virtual Machine Profile Engine for anomalies. A Virtual Machine Profile Engine can handle security profiles of VMs. Malicious system calls detected by the Detection Engine are passed to the "Severity Analysis Module" [9], which assesses intrusion severity, and the result is transferred to an "intrusion response system" [9] to make appropriate responses.

Structure-Based Approach. Gupta et al. developed a structure-based approach for anomalous processes detection in a private cloud environment [43]. The technique is claimed to be platform independent and portable to any cloud architecture. For model initialization, structures of programs are constructed and saved in a database. Structures are identified and created with logs generated by running programs. System call traces from virtual machines are monitored, and intermediate results are saved temporarily as key-value pairs for testing. Their method is still tested on the UNM dataset due to the complexity of collection and maintenance of real-world system call data generated from the cloud. The immediate system call database of key-value pairs is built based on programs of the UNM dataset. Anomalies identified will be recorded and notified to the cloud administrator for further actions. The cloud administrator is in charge of the whole system. The complexity of this method is $O(n^2)$ and the Perl hash method is adopted for further acceleration.

The First HIDS for Virtual Hosts with Spark and Kafka. To the best of our knowledge, Solaimani et al. first introduced an efficient and scalable real-time HIDS that performs comprehensive anomaly detection on various data streams such as CPU and memory performance data from virtual hosts [97]. The system has two major functional modules, i.e., Message Broker with Kafka cluster and Streaming Data Miner with Spark cluster. A virtual resource manager is designed and incorporated into the data center. Various data streams such as CPU and memory performance data are periodically gathered by the system from multiple virtual hosts and delivered through Kafka to the Spark cluster for analytics. vSphere Guest SDK [110] is used for continuously monitoring streaming data with the Kafka API integrated. CPU and memory usage percentages are

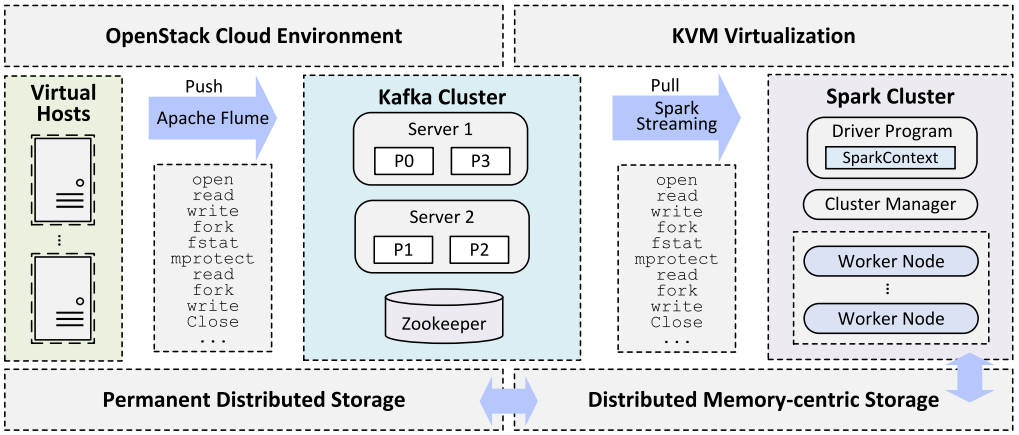

Fig. 5. A preliminary real-time scalable HIDS framework with big data tools in the cloud. The system call traces are from [34].

gathered by mpsat [41] and vmstat [49], respectively. Spark utilizes its built-in machine-learning approaches on discrete input streams for anomaly detection. A two-sample scalable Chi-square test is performed with Spark. Whenever abnormal behaviors are detected, the system will inform the resource manager, and further actions will be applied to abnormal virtual hosts. Detection results are saved to the resource pool for further resource allocation. Their next experiment was conducted in a VMware cluster, which involves five VMware hypervisor server systems, and each host has three virtual machines [98]. CentOS is set up on each VM. HDFS is utilized by Spark as the distributed storage system. Apache ZooKeeper is installed to manage message flow between the Kafka cluster and Spark cluster. The interactive esxtop [12] utility of the VMware ESXi host provides metrics of performance, such as CPU or memory usage data. Only CPU data is taken in their experiment. The data is periodically gathered and formed into feature vectors, which are continuously recorded into a CSV file cached in the Kafka cluster for further use. VMware resource usage is shown by the performance metrics. More resources should be provided when CPU-intensive programs are executing. Unexpected resource improvement, which is considered as the anomaly, would be rendered in the performance metrics. The nonupdatable model trained with benign data is used for prediction. Anomaly information analyzed by Spark is reported to the virtual resource management module, which manages and allocates computational resources.

6.3.2 Constructing a Real-Time Scalable HIDS with Big Data Tools in the Cloud. Inspired by current research works, we propose a preliminary real-time scalable HIDS framework with big data tools in the cloud for a data center. The framework is composed of three layers described in Table 10. Figure 5 demonstrates the framework deployed on an open-source private cloud computing environment. Computing facilities such as servers, routers, and switches are basic hardware components of the cloud. The KVM [15] open-source software included in the Linux operating system is used for infrastructure virtualization. A couple of virtual machines with their own virtualized hardware and Linux operating system can run on KVM; the OpenStack open-source cloud operating system is installed for resource management and interactions with applications of users [134]. Administrators can control all resources via the provided dashboard. Hadoop with HDFS and YARN [36] is installed to build and manage a cluster in the OpenStack-based cloud to realize scalable HIDS data processing and permanent storage. Apache Spark Streaming can run on the YARN cluster to process large HIDS data streams. Streaming system calls collected by Apache Flume [38]

Table 10. Three Layers of a Real-Time Scalable HIDS Framework with Big Data Tools in Cloud

Data collection layer
• This layer collects and caches real-time system call traces generated by multiple hosts. →*Apache Kafka can be adopted as the message broker.* • In practice, sensors are installed in hosts to gather system call traces.
Data analytics layer
• This layer comprehensively analyzes real-time system call traces from the data collection layer. • Spark MLlib can apply distributed machine-learning algorithms on feature vectors extracted from system call traces. Normal databases for anomaly detection can also be formed in this layer. • System call traces need to be pulled by Spark streaming from the Kafka server and compared with standard normal databases or predicted by machine-learning models.
Data storage layer
• This layer consists of distributed fault-tolerant data storage systems for saving and loading final detection results as well as intermediate HIDS datasets including trained machine-learning models and normal databases. • When it comes to host-based intrusion detection, given that normal databases or machine-learning models need to be regularly updated to deal with new attack methods, it requires high integrity and robustness for the storage system. Saving Spark RDDs in off-heap distributed memories can be a solution of intermediate dataset storage due to the DAG-based fault-tolerant mechanism of RDD. →*Long-term on-disk frameworks such as HDFS and temporary in-memory frameworks such as Alluxio are included in this layer.*

from hosts in the DMZ or intranet are pushed into the Kafka cluster for caching. Spark Streaming pulls system call traces accordingly from the Kafka server to analyze anomalies in a fine-grained manner. This framework can be easily extended to fulfill the requirement of new hosts set up in the data center. With a heavier data load as a result of an increasing number of hosts, Spark and Kafka clusters can be expanded, either in the way of enlarging internal distributed memory or adding more worker nodes. Furthermore, in a large data center, an overall scheduler with the highest priority controlled by security personnel is required to administer all computational resources including every single host. The scheduler should continuously monitor the complete data flow from multiple hosts to the Spark cluster eventually. If an intrusion in an individual host is detected, the Spark master can notify the scheduler to take appropriate measures on that host.

6.3.3 CIDS for a Data Center. For the security of a large data center with massive hosts, to achieve more powerful detection, HIDS should articulate with NIDS to compose an effective and high-throughput CIDS. The system should consist of a central analytical unit plus sensors installed on each of the physical or virtual hosts and network devices for the collection of system call traces and network packets.

CIDS with MapReduce. Tan et al. provided a novel framework of CIDS with MapReduce for cloud computing systems [105]. Detection software sensors named cooperative agents are installed to collaborate with HIDS and NIDS, and anomalous behaviors detected by relevant agents are reported to the central coordinator for mining attack patterns of the whole system [105]. An alternative central coordinator will be used if the primary one fails. With the central coordinator, data

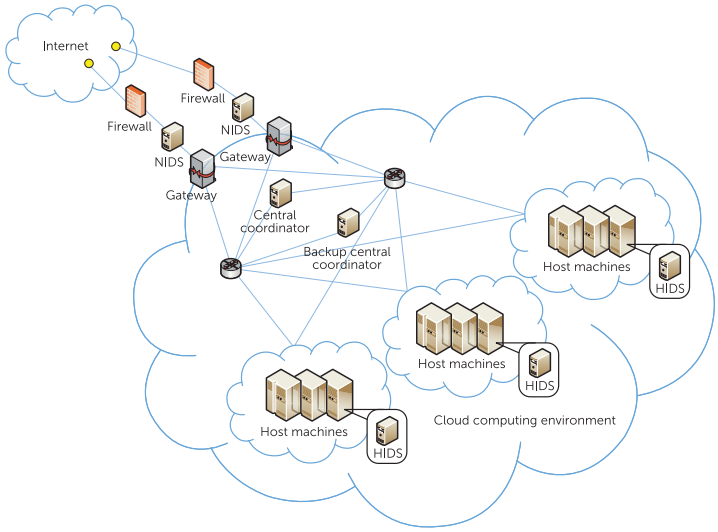

Fig. 6. A framework of CIDS [105].

collected by cooperative agents can be summarized for comprehensive analysis to resist cooperative intrusions [105]. Whenever anomalies are detected from cooperative agents or the central coordinator, the system administrator will be notified to take proper defensive measures. MapReduce is implemented and integrated with the proposed method for parallel summarization of data [105]. The master node works as the central coordinator, and worker nodes work as cooperative agents. Figure 6 demonstrates the proposed framework.

The Challenge. According to [109], CIDS with a centralized unit that analyzes a complete dataset is a dependable solution for data centers. However, the scalability is limited. It lacks scalable and applicable solutions to perform distributed intrusion detection in large networks. Meanwhile, intruders nowadays tend to employ a set of social engineering approaches to launch advanced persistent threats (APTs) toward the targeted systems, which make the security problems more sophisticated. New forms of attacks are continuously created by intruders, and intrusions can occur from both of the Intranet and the Internet. Meanwhile, protecting a system is always more difficult than making attacks. Traditionally, security specialists focus on security incidents that already occurred and make relevant incident responses. This kind of negative defensive approach is not robust when confronted with advanced attacks. In this case, traditional forms of IDS including CIDS offer inadequate information about attackers and potential attacks. Innovative CIDS updates and further comprehensive threat information-sharing methods are demanded [109].

6.3.4 Sharing Threat Information to Enhance the Collaborative Security. Currently, security specialists are seeking methods to block potential cyber attacks before they are launched to secure computer systems. Using cyber threat intelligence information sharing is effective to prevent attacks actively and can enhance the system security [11].

Structured Threat Information Feeds. The "threat information" is standardized information that can present attack patterns and relevant defensive strategies. Observation and analysis of attack patterns are necessary to prevent intrusions beforehand. Threat information for a system can be obtained from the Internet, the intranet, and trusted collaborators such as security specialists from one trusted community. Without collaboration, security specialists can hardly obtain adequate and

comprehensive threat information. Threat information to be consumed is commonly interpreted as structured feeds, which are ready to be integrated into security systems such as CIDS for updating. Security specialists from one trusted community can efficiently and collaboratively study complicated HIDS incidents together by sharing structured threat information feeds. The threat information feeds are composed based on some standards.

Threat Information Sharing → Standards. Recently, a set of standards has been created to facilitate automatic sharing of threat information [61]. The standards such as STIX, MAEC, OVAL, and CybOX attempt to describe the threat information in machine-readable formats for automatic integration.

Threat Information Sharing → Platforms. Threat information can be gathered, analyzed, and consumed automatically via some centralized platforms such as MITRE TAXII. Via such platforms, collaborators can make contributions to make threat information feeds more accurate and complete. For instance, if a host-based intrusion can be spotted in a system and related threat information feeds are shared with other collaborators, then similar attacks can be predicted. Submitting high-quality threat information feeds to platforms contributes to the community. If security specialists can integrate the available high-quality threat information feeds about HIDS properly, then the security of relevant computer systems can be enhanced.

Select the Most Valuable Feeds. Determining the most valuable feeds for integration is a significant task. For a particular consumer, some of the feeds available on the platforms may not be applicable for integration. Consuming redundant feeds from commercial platforms can cause additional expenses. Consumers need to understand their requirements of security and possible varieties of intrusions they may be confronted with and select the most valuable feeds for integration to maximize the effect of defense and minimize the cost. Usually, consumers can identify the origins of feeds according to the records of platforms. Even though the platforms provide authenticated threat information feeds, consumers still need to verify their accuracy and integrity before integration.

Challenges for Threat Information Sharing. Although taking advantage of threat information sharing can be an enhancement to HIDS and CIDS, some issues need to be considered and solved by the submitters and consumers of structured threat information feeds. Submitters need to consider what kind of threat information about HIDS or CIDS should be generated as relevant standardized threat information feeds and how to generate and submit those feeds to the sharing platform. Consumers need to consider what kind of HIDS- or CIDS-related feeds should be integrated from the sharing platform and how to integrate those feeds into the local HIDS or CIDS. Some other issues also have to be considered, such as how to develop a private threat information sharing platform for internal use to protect privacy.

6.3.5 Current Practices in the Industry Regarding This Trend. In the current industry, there are two significant improvements about HIDS: (1) The integration with other security capabilities. As HIDS usually cannot provide complete security protection to critical systems, in the current industry, HIDS is usually integrated with other essential security mechanisms, including NIDS, vulnerability assessment, and incident response. HIDS can be deployed as part of one unified security platform, on which security incidents can be aggregated and investigated. (2) The combination of the latest threat intelligence. Another significant improvement in the current industry is that HIDS is usually strengthened with the latest threat intelligence to keep up to date with emerging cyber threats. Actionable threat intelligence can be integrated into HIDS as continuous signature updates. Some typical HIDS systems deployed in the current industry are described below.

McAfee. The "McAfee host intrusion prevention for desktop" provides a dynamic and complete platform that protects the system security and data confidentiality [77]. The centralized console offers simple administration. With the increase in advanced threats, McAfee has integrated the cloud-based "Global Threat Intelligence" service to its HIDS to detect advanced cyber threats before attacks happen.

OSSEC. The HIDS solution "OSSEC (Open Source HIDS SEcurity)" provides services such as "file integrity checking, log monitoring, rootkit detection and active response" [107]. OSSEC can help the user to implement a comprehensive HIDS across multiple operating systems with a centralized management server.

OSSIM. The "OSSIM (Open Source Security Information and Event Management)" is an open-source threat management system that integrates HIDS and NIDS with other key threat detection capabilities [90]. It monitors the security of the local environment. OSSIM is a unified platform that supports multiple operating systems including Linux. OSSIM utilizes the capability of "AlienVault Open Threat Exchange (OTX)" by enabling collaborators to share the latest threat information of malicious hosts.

AlienVault USM. The "AlienVault Unified Security Managemen platform (USM)" is a commercial product that combines HIDS with NIDS and other security mechanisms in a unified platform to manage threats [6]. USM can monitor the security of both local and cloud environments. The information collected by HIDS agents will be sent to the unified platform for centralized threat detection. AlienVault USM receives continuous and automatic threat intelligence updates from the community of AlienVault OTX, where collaborators can share the latest threat information.

7 CONCLUDING REMARKS

We provide a survey of the host-based intrusion detection system with system calls, from the perspectives of its origin, algorithms, datasets, application areas, and future research trends. Instead of elaborating on every detail, the main aim of this article is trying to inspire future researchers about system-call-based HIDS and heading to CIDS with threat information sharing. When it comes to the current big data environment and the emergence of diversified cyber threats, combining multiple intrusion defense approaches to work collaboratively is the major trend for designing a robust threat-defensive infrastructure. Interdisciplinary research works can also be taken to augment the effectiveness of defense. We hope that high-quality datasets can be generated to guarantee that it is worth further investigation. Compared with choosing the optimal detection engines, data preparation and feature extraction are the decisive factors and therefore deserve more attention. Meanwhile, standardized evaluation metrics are suggested to be presented in each research work for easier comparison.

REFERENCES

[1] Mohamed Abdel-Azim, AI Abdel-Fatah, and Mohammed Awad. 2009. Performance analysis of artificial neural network intrusion detection systems. In *International Conference on Electrical and Electronics Engineering, 2009 (ELECO'09)*. IEEE, II–385–II–389.

[2] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. 2016. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications* 60 (2016), 19–31. DOI: https://doi.org/10.1016/j.jnca.2015.11.016

[3] Usman Ahmed and Asif Masood. 2009. Host based intrusion detection using RBF neural networks. In *International Conference on Emerging Technologies, 2009 (ICET'09)*. IEEE, 48–51.

[4] Suaad Alarifi and Stephen Wolthusen. 2013. Anomaly detection for ephemeral cloud IaaS virtual machines. In *International Conference on Network and System Security*. Springer, 321–335.

[5] Suaad S. Alarifi and Stephen D. Wolthusen. 2012. Detecting anomalies in IaaS environments through virtual machine host system call analysis. In *2012 International Conference for Internet Technology And Secured Transactions*. IEEE, 211–218.
[6] AlienVault. 2018. Host-based Intrusion Detection System. Retrieved from https://www.alienvault.com/solutions/host-intrusion-detection-system.
[7] Mohammed A. Ambusaidi, Xiangjian He, Priyadarsi Nanda, and Zhiyuan Tan. 2016. Building an intrusion detection system using a filter-based feature selection algorithm. *IEEE Transactions on Computers* 65, 10 (2016), 2986–2998.
[8] Austin Appleby. 2017. Murmurhash. Retrieved from https://sites.google.com/site/murmurhash/.
[9] Junaid Arshad, Paul Townend, and Jie Xu. 2013. A novel intrusion severity analysis approach for clouds. *Future Generation Computer Systems* 29, 1 (2013), 416–428. DOI: https://doi.org/10.1016/j.future.2011.08.009
[10] Chandrashekhar Azad and Vijay Kumar Jha. 2013. Data mining in intrusion detection: A comparative study of methods, types and data sets. *International Journal of Information Technology and Computer Science (IJITCS)* 5, 8 (2013), 75.
[11] Sean Barnum. 2012. Standardizing cyber threat intelligence information with the structured threat information eXpression (STIX). *MITRE Corporation* 11 (2012), 1–22.
[12] VMware Knowledge Base. 2017. esxtop. Retrieved from https://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1008205.
[13] Thomas Bläsing, Leonid Batyuk, Aubrey-Derrick Schmidt, Seyit Ahmet Camtepe, and Sahin Albayrak. 2010. An Android application sandbox system for suspicious software detection. In *2010 5th International Conference on Malicious and Unwanted Software (MALWARE'10)*. IEEE, 55–62.
[14] Andrei Broder and Michael Mitzenmacher. 2004. Network applications of bloom filters: A survey. *Internet Mathematics* 1, 4 (2004), 485–509.
[15] Bsd. 2017. Kernel Virtual Machine. Retrieved from http://www.linux-kvm.org/page/Main_Page.
[16] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. 2011. Crowdroid: Behavior-based malware detection system for android. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*. ACM, 15–26.
[17] Davide Canali, Andrea Lanzi, Davide Balzarotti, Christopher Kruegel, Mihai Christodorescu, and Engin Kirda. 2012. A quantitative study of accuracy in system call-based malware detection. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis*. ACM, 122–132.
[18] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection. *Computer Surveys* 41, 3 (2009), 1–58. DOI: https://doi.org/10.1145/1541880.1541882
[19] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2012. Anomaly detection for discrete sequences: A survey. *IEEE Transactions on Knowledge and Data Engineering* 24, 5 (2012), 823–839.
[20] Qiuwen Chen, Ryan Luley, Qing Wu, Morgan Bishop, Richard W. Linderman, and Qinru Qiu. 2018. AnRAD: A neuromorphic anomaly detection framework for massive concurrent data streams. *IEEE Transactions on Neural Networks and Learning Systems* 29, 5 (2018), 1622–1636.
[21] Qiuwen Chen, Qing Wu, Morgan Bishop, Richard Linderman, and Qinru Qiu. 2015. Self-structured confabulation network for fast anomaly detection and reasoning. In *2015 International Joint Conference on Neural Networks (IJCNN'15)*. IEEE, 1–8.
[22] The MITRE Corporation. 2017. Common Vulnerabilities and Exposures. Retrieved from https://cve.mitre.org/.
[23] Gideon Creech. 2014. *Developing a High-Accuracy Cross Platform Host-Based Intrusion Detection System Capable of Reliably Detecting Zero-Day Attacks*. Ph.D. Dissertation. PhD thesis, University of New South Wales.
[24] Gideon Creech and Jiankun Hu. 2013. Generation of a new IDS test dataset: Time to retire the KDD collection. In *2013 IEEE Wireless Communications and Networking Conference (WCNC'13)*. IEEE, 4487–4492.
[25] Gideon Creech and Jiankun Hu. 2014. A semantic approach to host-based intrusion detection systems using contiguous and discontiguous system call patterns. *IEEE Transactions on Computers* 63, 4 (2014), 807–819.
[26] Sanjeev Das, Yang Liu, Wei Zhang, and Mahintham Chandramohan. 2016. Semantics-based online malware detection: Towards efficient real-time protection against malware. *IEEE Transactions on Information Forensics and Security* 11, 2 (2016), 289–302.
[27] Jesse Davis and Mark Goadrich. 2006. The relationship between precision-recall and ROC curves. In *Proceedings of the 23rd International Conference on Machine Learning*. ACM, 233–240.
[28] Richard I. A. Davis, Brian C. Lovell, and Terry Caelli. 2002. Improved estimation of hidden Markov model parameters from multiple observation sequences. In *Proceedings of the 16th International Conference on Pattern Recognition, 2002.* Vol. 2. IEEE, 168–171.
[29] Huwaida Tagelsir Elshoush and Izzeldin Mohamed Osman. 2011. Alert correlation in collaborative intelligent intrusion detection systems-A survey. *Applied Soft Computing* 11, 7 (2011), 4349–4365.

[30] Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, and Ainuddin Wahid Abdul Wahab. 2015. A review on feature selection in mobile malware detection. *Digital Investigation* 13 (2015), 22–37.
[31] Stephen Feldman, Dillon Stadther, and Bing Wang. 2014. Manilyzer: Automated android malware detection through manifest analysis. In *2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems (MASS'14)*. IEEE, 767–772.
[32] Shai Fine, Yoram Singer, and Naftali Tishby. 1998. The hierarchical hidden Markov model: Analysis and applications. *Machine Learning* 32, 1 (1998), 41–62.
[33] Stephanie Forrest, Steven Hofmeyr, and Anil Somayaji. 2008. The evolution of system-call monitoring. In *Annual Computer Security Applications Conference, 2008 (ACSAC'08)*. IEEE, 418–430.
[34] Stephanie Forrest, Steven A. Hofmeyr, Aniln Somayaji, and Thomas A. Longstaff. 1996. A sense of self for unix processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy, 1996*. IEEE, 120–128.
[35] Alluxio Open Foundation. 2017. Alluxio. Retrieved from http://www.alluxio.org/.
[36] Apache Software Foundation. 2017. Apache Hadoop YARN. Retrieved from https://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/YARN.html.
[37] Apache Software Foundation. 2017. Apache Kafka a distributed streaming platform. Retrieved from https://kafka.apache.org/.
[38] The Apache Software Foundation. 2017. Apache Flume. Retrieved from https://flume.apache.org/.
[39] The Apache Software Foundation. 2018. Spark. Retrieved from http://spark.apache.org/.
[40] Anup K. Ghosh, Aaron Schwartzbard, and Michael Schatz. 1999. Learning program behavior profiles for intrusion detection. In *Workshop on Intrusion Detection and Network Monitoring*, Vol. 51462. 1–13.
[41] Sebastien Godard. 2017. mpstat. Retrieved from http://linuxcommand.org/man_pages/mpstat1.html.
[42] Ye Gu, Weihua Sheng, Yongsheng Ou, Meiqin Liu, and Senlin Zhang. 2013. Human action recognition with contextual constraints using a RGB-D sensor. In *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO'13)*. IEEE, 674–679.
[43] Sanchika Gupta and Padam Kumar. 2015. An immediate system call sequence based approach for detecting malicious program executions in cloud environment. *Wireless Personal Communications* 81, 1 (2015), 405–425.
[44] Waqas Haider, Gideon Creech, Yi Xie, and Jiankun Hu. 2016. Windows based data sets for evaluation of robustness of host based intrusion detection systems (IDS) to zero-day and stealth attacks. *Future Internet* 8, 3 (2016), 29.
[45] Waqas Haider, Jiankun Hu, Jill Slay, Benjamin Turnbull, and Yi Xie. 2017. Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling. *Journal of Network and Computer Applications* 87, C (2017), 185–192.
[46] Waqas Haider, Jiankun Hu, and Miao Xie. 2015. Towards reliable data feature retrieval and decision engine in host-based anomaly detection systems. In *2015 IEEE 10th Conference on Industrial Electronics and Applications (ICIEA'15)*. IEEE, 513–517.
[47] Waqas Haider, Jiankun Hu, Yi Xie, Xinghuo Yu, and Qianhong Wu. 2017. Detecting anomalous behavior in cloud servers by nested arc hidden SEMI-Markov model with state summarization. *IEEE Transactions on Big Data* (2017).
[48] Waqas Haider, Jiankun Hu, Xinghuo Yu, and Yi Xie. 2015. Integer data zero-watermark assisted system calls abstraction and normalization for host based anomaly detection systems. In *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing (CSCloud'15)*. IEEE, 349–355.
[49] Fabian Frederick Henry Ware. 2017. vmstat. Retrieved from http://www.linuxcommand.org/man_pages/vmstat8.html.
[50] Xuan Dau Hoang, Jiankun Hu, and Peter Bertok. 2003. A multi-layer model for anomaly intrusion detection using program sequences of system calls. In *The 11th IEEE International Conference on Networks, 2003 (ICON'03)*. Citeseer.
[51] Xuan Dau Hoang, Jiankun Hu, and Peter Bertok. 2009. A program-based anomaly intrusion detection scheme using multiple detection engines and fuzzy inference. *Journal of Network and Computer Applications* 32, 6 (2009), 1219–1228.
[52] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780.
[53] Steven A. Hofmeyr, Stephanie Forrest, and Anil Somayaji. 1998. Intrusion detection using sequences of system calls. *Journal of Computer Security* 6, 3 (1998), 151–180.
[54] Jiankun Hu. 2010. Host-based anomaly intrusion detection. *Handbook of Information and Communication Security* (2010), 235–255.
[55] Jiankun Hu, Xinghuo Yu, Dong Qiu, and Hsiao-Hwa Chen. 2009. A simple and efficient hidden Markov model scheme for host-based anomaly intrusion detection. *IEEE Network* 23, 1 (2009), 42–47.
[56] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. 2006. Extreme learning machine: Theory and applications. *Neurocomputing* 70, 1 (2006), 489–501.
[57] Ixia. 2017. PerfectStorm. Retrieved from https://www.ixiacom.com/products/perfectstorm.

[58] Gaya K. Jayasinghe, J. Shane Culpepper, and Peter Bertok. 2014. Efficient and effective realtime prediction of drive-by download attacks. *Journal of Network and Computer Applications* 38 (2014), 135–149.
[59] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM International Conference on Multimedia*. ACM, 675–678.
[60] Guofei Jiang, Haifeng Chen, Cristian Ungureanu, and Kenji Yoshihira. 2007. Multiresolution abnormal trace detection using varied-length n-grams and automata. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 37, 1 (2007), 86–97.
[61] Panos Kampanakis. 2014. Security automation and threat information-sharing options. *IEEE Security & Privacy* 12, 5 (2014), 42–51.
[62] Parmeet Kaur and Shikha Mehta. 2017. Resource provisioning and work flow scheduling in clouds using augmented shuffled frog leaping algorithm. *Journal of Parallel and Distributed Computing* 101 (2017), 41–50.
[63] W. Khreich, E. Granger, R. Sabourin, and A. Miri. 2009. Combining hidden Markov models for improved anomaly detection. In *IEEE International Conference on Communications*. 1–6.
[64] Wael Khreich, Syed Shariyar Murtaza, Abdelwahab Hamou-Lhadj, and Chamseddine Talhi. 2017. Combining heterogeneous anomaly detectors for improved software security. *Journal of Systems and Software* 137 (2018), 415–429.
[65] Andrea Kovács and Tamás Szirányi. 2013. Improved harris feature point set for orientation-sensitive urban-area detection in aerial images. *IEEE Geoscience and Remote Sensing Letters* 10, 4 (2013), 796–800.
[66] Manish Kulariya, Priyanka Saraf, Raushan Ranjan, and Govind P. Gupta. 2016. Performance analysis of network intrusion detection schemes using Apache Spark. In *2016 International Conference on Communication and Signal Processing (ICCSP'16)*. IEEE, 1973–1977.
[67] Aron Laszka, Waseem Abbas, S. Shankar Sastry, Yevgeniy Vorobeychik, and Xenofon Koutsoukos. 2016. Optimal thresholds for intrusion detection systems. In *Proceedings of the Symposium and Bootcamp on the Science of Security*. ACM, 72–81.
[68] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.
[69] Wenke Lee and Salvatore J. Stolfo. 1998. Data mining approaches for intrusion detection. In *Usenix Security*.
[70] Wenke Lee, Salvatore J. Stolfo, and Philip K. Chan. 1996. Learning patterns from unix process execution traces for intrusion detection. In *AAAI Workshop on AI Approaches to Fraud Detection and Risk Management*. 50–56.
[71] Wenke Lee and Dong Xiang. 2001. Information-theoretic measures for anomaly detection. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy, 2001 (S&P'01)*. IEEE, 130–143.
[72] Yihua Liao and V. Rao Vemuri. 2002. Using text categorization techniques for intrusion detection. In *USENIX Security Symposium*, Vol. 12. 51–59.
[73] Peter Lichodzijewski, A. Nur Zincir-Heywood, and Malcolm I. Heywood. 2002. Host-based intrusion detection using self-organizing maps. In *IEEE International Joint Conference on Neural Networks*. 1714–1719.
[74] Federico Maggi, Matteo Matteucci, and Stefano Zanero. 2010. Detecting intrusions through system call sequence and argument analysis. *IEEE Transactions on Dependable and Secure Computing* 7, 4 (2010), 381–395.
[75] Matthew V. Mahoney and Philip K. Chan. 2003. An analysis of the 1999 DARPA/lincoln laboratory evaluation data for network anomaly detection. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 220–237.
[76] Matthew V. Mahoney and Philip K. Chan. 2003. Learning rules for anomaly detection of hostile network traffic. In *3rd IEEE International Conference on Data Mining, 2003 (ICDM'03)*. IEEE, 601.
[77] McAfee. 2018. McAfee Host Intrusion Prevention for Desktop. Retrieved from https://www.mcafee.com/uk/products/host-ips-for-desktop.aspx.
[78] John McHugh. 2000. Testing intrusion detection systems: A critique of the 1998 and 1999 Darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security (TISSEC)* 3, 4 (2000), 262–294.
[79] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. 2016. Mllib: Machine learning in apache spark. *Journal of Machine Learning Research* 17, 34 (2016), 1–7.
[80] Enza Messina and Daniele Toscani. 2008. Hidden Markov models for scenario generation. *Ima Journal of Management Mathematics* 19, 4 (2008), 379–401(23).
[81] Aleksandar Milenkoski, Marco Vieira, Samuel Kounev, Alberto Avritzer, and Bryan D. Payne. 2015. Evaluating computer intrusion detection systems: A survey of common practices. *ACM Computing Surveys (CSUR)* 48, 1 (2015), 12.
[82] Philipp Moritz, Robert Nishihara, Ion Stoica, and Michael I. Jordan. 2015. Sparknet: Training deep networks in spark. *arXiv Preprint arXiv:1511.06051* (2015).

[83] Syed Shariyar Murtaza, Abdelwahab Hamou-Lhadj, Wael Khreich, and Mario Couture. 2014. Total ADS: Automated software anomaly detection system. In *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation (SCAM'14)*. IEEE, 83–88.

[84] Syed Shariyar Murtaza, Wael Khreich, Abdelwahab Hamou-Lhadj, and Mario Couture. 2013. A host-based anomaly detection approach by representing system calls as states of kernel modules. In *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE'13)*. IEEE, 431–440.

[85] Syed Shariyar Murtaza, Wael Khreich, Abdelwahab Hamou-Lhadj, and Stephane Gagnon. 2015. A trace abstraction approach for host-based anomaly detection. In *2015 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA'15)*. IEEE, 1–8.

[86] Seyyedeh Atefeh Musavi and Mehdi Kharrazi. 2014. Back to static analysis for kernel-level rootkit detection. *IEEE Transactions on Information Forensics and Security* 9, 9 (2014), 1465–1476.

[87] Darren Mutz, Fredrik Valeur, Giovanni Vigna, and Christopher Kruegel. 2006. Anomalous system call detection. *ACM Transactions on Information and System Security (TISSEC)* 9, 1 (2006), 61–93.

[88] Mohammad Nauman, Nouman Azam, and JingTao Yao. 2016. A three-way decision making approach to malware analysis using probabilistic rough sets. *Information Sciences* 374 (2016), 193–209.

[89] University of New Mexico. 2017. Sequence-Based Intrusion Detection. Retrieved from http://www.cs.unm.edu/immsec/systemcalls.htm.

[90] OSSIM. 2018. AlienVault OSSIM: The World's Most Widely Used Open Source SIEM. Retrieved from https://www.alienvault.com/products/ossim.

[91] Pingbo Pan, Zhongwen Xu, Yi Yang, Fei Wu, and Yueting Zhuang. 2015. Hierarchical recurrent neural encoder for video representation with application to captioning. *arXiv Preprint arXiv:1511.03476* (2015).

[92] Jonas Pfoh, Christian Schneider, and Claudia Eckert. 2011. Nitro: Hardware-based system call tracing for virtual machines. In *International Workshop on Security*. Springer, 96–112.

[93] Yan Qiao, X. W. Xin, Yang Bin, and S. Ge. 2002. Anomaly intrusion detection method based on HMM. *Electronics Letters* 38, 13 (2002), 1.

[94] Lawrence R. Rabiner. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE* 77, 2 (1989), 257–286.

[95] Rapid7. 2017. Metasploit. Retrieved from https://www.metasploit.com/.

[96] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The Hadoop distributed file system. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST'10)*. IEEE, 1–10.

[97] Mohiuddin Solaimani, Mohammed Iftekhar, Latifur Khan, and Bhavani Thuraisingham. 2014. Statistical technique for online anomaly detection using spark over heterogeneous data from multi-source VMware performance data. In *2014 IEEE International Conference on Big Data (Big Data'14)*. IEEE, 1086–1094.

[98] Mohiuddin Solaimani, Mohammed Iftekhar, Latifur Khan, Bhavani Thuraisingham, Joe Ingram, and Sadi Evren Seker. 2016. Online anomaly detection for multi-source VMware using a distributed streaming framework. *Software: Practice and Experience* 46, 11 (2016), 1479–1497.

[99] Robin Sommer and Vern Paxson. 2010. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE Symposium on Security and Privacy*. IEEE, 305–316.

[100] Xin Su, M. Chuah, and Gang Tan. 2012. Smartphone dual defense protection framework: Detecting malicious applications in android markets. In *2012 8th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN'12)*. IEEE, 153–160.

[101] Michio Sugeno and Takahiro Yasukawa. 1993. A fuzzy-logic-based approach to qualitative modeling. *IEEE Transactions on Fuzzy Systems* 1, 1 (1993), 7–31.

[102] Kymie M. C. Tan and Roy A. Maxion. 2002. "Why 6?" Defining the operational limits of Stide, an anomaly-based intrusion detector. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy, 2002*. IEEE, 188–201.

[103] Zhiyuan Tan, Aruna Jamdagni, Xiangjian He, Priyadarsi Nanda, and Ren Ping Liu. 2014. A system for denial-of-service attack detection based on multivariate correlation analysis. *IEEE Transactions on Parallel and Distributed Systems* 25, 2 (2014), 447–456.

[104] Zhiyuan Tan, Aruna Jamdagni, Xiangjian He, Priyadarsi Nanda, Ren Ping Liu, and Jiankun Hu. 2015. Detection of denial-of-service attacks based on computer vision techniques. *IEEE Transactions on Computers* 64, 9 (2015), 2519–2533.

[105] Zhiyuan Tan, Upasana T. Nagar, Xiangjian He, Priyadarsi Nanda, Ren Ping Liu, Song Wang, and Jiankun Hu. 2014. Enhancing big data security with collaborative intrusion detection. *IEEE Cloud Computing* 1, 3 (2014), 27–33.

[106] Gaurav Tandon. 2008. *Machine Learning for Host-Based Anomaly Detection*. Thesis.

[107] OSSEC Project Team. 2017. OSSEC Open Source HIDS SECurity. Retrieved from http://ossec.github.io/.

[108] Julien Thomas, Cédric Rose, and François Charpillet. 2006. A multi-HMM approach to ECG segmentation. In *18th IEEE International Conference on Tools with Artificial Intelligence, 2006 (ICTAI'06)*. IEEE, 609–616.

[109] Emmanouil Vasilomanolakis, Shankar Karuppayah, Max Muhlhauser, and Mathias Fischer. 2015. Taxonomy and survey of collaborative intrusion detection. *ACM Computing Surveys (CSUR)* 47, 4 (2015), 55.
[110] vmware. 2017. vSphere Guest SDK. Retrieved from https://www.vmware.com/support/developer/guest-sdk/.
[111] David Wagner and Paolo Soto. 2002. Mimicry attacks on host-based intrusion detection systems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*. ACM, 255–264.
[112] Rasna Rani Walia. 2014. Sequence-based prediction of RNA-protein interactions. *Dissertations and Theses - Gradworks* (2014).
[113] Ke Wang, Janak J. Parekh, and Salvatore J. Stolfo. 2006. Anagram: A content anomaly detector resistant to mimicry attack. In *International Workshop on Recent Advances in Intrusion Detection, 2006*. Springer, 226–248.
[114] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. 1999. Detecting intrusions using system calls: Alternative data models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy, 1999*. IEEE, 133–145.
[115] Michael R. Watson, Angelos K. Marnerides, Andreas Mauthe, and David Hutchison. 2016. Malware detection in cloud computing infrastructures. *IEEE Transactions on Dependable and Secure Computing* 13, 2 (2016), 192–205.
[116] Xi Xiao, Zhenlong Wang, Qi Li, Qing Li, and Yong Jiang. 2015. ANNs on co-occurrence matrices for mobile malware detection. *KSII Transactions on Internet and Information Systems (TIIS)* 9, 7 (2015), 2736–2754.
[117] Miao Xie and Jiankun Hu. 2013. Evaluating host-based anomaly detection systems: A preliminary analysis of ADFA-LD. In *2013 6th International Congress on Image and Signal Processing (CISP'13)*, Vol. 3. IEEE, 1711–1716.
[118] Miao Xie, Jiankun Hu, and Jill Slay. 2014. Evaluating host-based anomaly detection systems: Application of the one-class SVM algorithm to ADFA-LD. In *2014 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD'14)*. IEEE, 978–982.
[119] Miao Xie, Jiankun Hu, Xinghuo Yu, and Elizabeth Chang. 2014. Evaluating host-based anomaly detection systems: Application of the frequency-based algorithms to ADFA-LA. In *International Conference on Network and System Security*. Springer, 542–549.
[120] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. *arXiv Preprint arXiv:1502.03044* 2, 3 (2015), 5.
[121] Lifan Xu, Dongping Zhang, Marco A. Alvarez, Jose Andre Morales, Xudong Ma, and John Cavazos. 2016. Dynamic android malware classification using graph-based representations. In *2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud'16)*. IEEE, 220–231.
[122] Chi Yang and Jinjun Chen. 2017. A scalable data chunk similarity based compression approach for efficient big sensing data processing on cloud. *IEEE Transactions on Knowledge and Data Engineering* 29, 6 (2017), 1144–1157.
[123] Chi Yang, Chang Liu, Xuyun Zhang, Surya Nepal, and Jinjun Chen. 2015. A time efficient approach for detecting errors in big sensor data on cloud. *IEEE Transactions on Parallel and Distributed Systems* 26, 2 (2015), 329–339.
[124] Chi Yang, Deepak Puthal, Saraju P. Mohanty, and Elias Kougianos. 2017. Big-sensing-data curation for the cloud is coming: A promise of scalable cloud-data-center mitigation for next-generation IoT and wireless sensor networks. *IEEE Consumer Electronics Magazine* 6, 4 (2017), 48–56.
[125] Chi Yang, Xuyun Zhang, Changmin Zhong, Chang Liu, Jian Pei, Kotagiri Ramamohanarao, and Jinjun Chen. 2014. A spatiotemporal compression based approach for efficient big data processing on cloud. *Journal of Computer and System Sciences* 80, 8 (2014), 1563–1583.
[126] Zhilin Yang, Ye Yuan, Yuexin Wu, Ruslan Salakhutdinov, and William W. Cohen. 2016. Encode, review, and decode: Reviewer module for caption generation. *arXiv Preprint arXiv:1605.07912* (2016).
[127] Qing Ye, Xiaoping Wu, and Bo Yan. 2010. An intrusion detection approach based on system call sequences and rules extraction. In *2010 2nd International Conference on E-business and Information System Security*. IEEE, 1–4.
[128] Dit-Yan Yeung and Yuxin Ding. 2003. Host-based intrusion detection using dynamic and static behavioral models. *Pattern Recognition* 36, 1 (2003), 229–243.
[129] Ding Yuxin, Yuan Xuebing, Zhou Di, Dong Li, and An Zhanchao. 2011. Feature representation and selection in malicious code detection methods based on static system calls. *Computers & Security* 30, 6 (2011), 514–524.
[130] Matei Zaharia. 2016. *An Architecture for Fast and General Data Processing on Large Clusters*. Morgan & Claypool.
[131] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. USENIX Association, 2–2.
[132] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. 2013. Discretized streams: Fault-tolerant streaming computation at scale. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles*. ACM, 423–438.
[133] Xuyun Zhang, Wanchun Dou, Jian Pei, Surya Nepal, Chi Yang, Chang Liu, and Jinjun Chen. 2015. Proximity-aware local-recoding anonymization with mapreduce for scalable big data privacy preservation in cloud. *IEEE Transactions on Computers* 64, 8 (2015), 2293–2307.

[134] Xuyun Zhang, Chang Liu, Surya Nepal, Chi Yang, and Jinjun Chen. 2014. Privacy preservation over big data in cloud systems. In *Security, Privacy and Trust in Cloud Systems*. Springer, 239–257.

[135] Xuyun Zhang, Chang Liu, Surya Nepal, Chi Yang, Wanchun Dou, and Jinjun Chen. 2014. A hybrid approach for scalable sub-tree anonymization over big data using MapReduce on cloud. *Journal of Computer and System Sciences* 80, 5 (2014), 1008–1020.

[136] Xuyun Zhang, Laurence T. Yang, Chang Liu, and Jinjun Chen. 2014. A scalable two-phase top-down specialization approach for data anonymization using Mapreduce on cloud. *IEEE Transactions on Parallel and Distributed Systems* 25, 2 (2014), 363–373.

[137] Richard Zuech, Taghi M. Khoshgoftaar, and Randall Wald. 2015. Intrusion detection and big heterogeneous data: A survey. *Journal of Big Data* 2, 1 (2015), 1.

Received July 2017; revised April 2018; accepted April 2018