UNIVERSITY OF TECHNOLOGY SYDNEY
Faculty of Engineering and Information Technology

SCALABLE PROCESSING METHODS FOR HOST-BASED INTRUSION DETECTION SYSTEMS

by

Ming Liu

A Thesis Submitted
in Fulfillment of the
Requirements for the Degree

Doctor of Philosophy

Sydney, Australia

2019

CERTIFICATE OF ORIGINAL AUTHORSHIP

I, Ming Liu declare that this thesis, is submitted in fulfillment of the requirements for the award of Doctor of Philosophy, in the School of Electrical and Data Engineering at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise reference or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This thesis is the result of a research candidature conducted with another University as part of a collaborative Doctoral degree.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Signature:

Production Note:
Signature removed prior to publication.

Date: 28/01/2019

ABSTRACT

SCALABLE PROCESSING METHODS FOR HOST-BASED INTRUSION DETECTION SYSTEMS

by

Ming Liu

Host-based intrusion detection system (HIDS) is renowned for the fine-grained analysis and the capability of discovering internal malicious behaviors. HIDS monitors logs from operating systems, whereas network-based intrusion detection system (NIDS) focuses on the data flow of network traffic. 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 system 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 update of 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.

First, considering these limitations of current system call-based HIDS, this thesis provides a review of the development of system call-based HIDS. 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. Modern application of system call-based HIDS on embedded systems is summarized, and related works are investigated.

Second, this thesis forecasts the future research trends of HIDS regarding three aspects, namely, the reduction of false positive rate, the improvement of detection

efficiency, and the enhancement of collaborative security; then a real-time scalable HIDS framework with big data tools in cloud for a data center is proposed to enhance the collaborative security. The framework is comprised of three layers, namely, data collection layer, data analytics layer, and data storage layer. The framework is deployed in an open-source private cloud computing environment, and this framework is easily scalable to fulfill the requirement of new hosts set up in the data center.

Third, this thesis presents SCADS, a corresponding scalable HIDS solution using Apache Spark in the Google cloud environment. A set of Spark algorithms are developed to achieve the computational scalability. The experimental results demonstrate that the efficiency of intrusion detection can be enhanced, which indicates that the proposed method is applicable to the design of next-generation host-based intrusion detection systems with system calls.

Fourth, in the current industry, there are two significant improvements about HIDS, i.e., the integration with other security capabilities, and the combination of the latest threat intelligence (CTI). Therefore, to design a comprehensive HIDS under the current sophisticated threat environment, traditional HIDS should combine with other security capabilities and the latest CTI. The key component of CTI is the sharing of threat information. This thesis briefly introduces the cyber threat intelligence and the threat information sharing; and proposes a scalable real-time threat information sharing framework in cloud, based on some recently leading platforms, such as MITRE TAXII, IBM X-Force, WEBROOT BrightCloud, EclecticIQ platform, and AlienVault OTX platform.

Fifth, this thesis provides a private and scalable online virus detection system. The system is expected to be integrated into the forecast scalable real-time threat information sharing system, which is helpful to the design of a comprehensive HIDS. The system incorporates multiple anti-virus engines and a web interface. The proposed system can perform the "isolated detection and update", which guarantees that the uploaded confidential samples are not exposed to the Internet, during either virus detection or system upgrade. Furthermore, the low-coupling design of this system is scalable to support the distributed deployment mode. The system is tested with benign

files, EICAR (European Institute for Computer Antivirus Research) Standard Anti-Virus Test File, and other suspicious samples. The system testing results demonstrate that the proposed mechanisms are pragmatic.

Dedication

To my family

Acknowledgements

I would like to sincerely thank my supervisor Prof. **Jinjun Chen** for his constant assistance and helpful suggestions. His great supervision makes a remarkable PhD journey for me. I am grateful for all of his help. I would like to sincerely thank my supervisor Prof. **Xiangjian He** for his continuous assistance and helpful guidance. His support and encouragement make me going forward steadfastly.

I would like to sincerely thank Prof. **Renping Liu** for his valuable suggestions during candidate assessment. I would like to sincerely thank Dr. **Priyadarsi Nanda** for his valuable suggestions and English corrections. I would like to sincerely thank Dr. **Deepak Puthal** for his suggestions and helps for the publication of academic papers. I would like to sincerely thank Dr. **Chang Liu** for his suggestions and help about the procedures of pursuing the PhD degree in UTS. I would like to sincerely thank **Yuxuan He** for his help in chapter 6.

I would like to sincerely thank my colleagues and friends for their continuous help and encouragements. They are A/Prof. Yuhong Zhang, Dr. Christy Liang, Dr. Chi Yang, Dr. Xuyun Zhang, Dr. Wenjing Jia, Dr. Zhiyuan Tan, Dr. Pakawat Pupatwibul, Dr. Khaled Aldebei, Dr. Muhammad Usman, Xiaochen Fan, Hanhui Li, Xiaohua Wei, Saeed Amirgholipour, Vinh Tung Le, Farhan Mohd, Ashish Nanda, Haihan Sun, and Haimin Zhang, etc.

I would like to sincerely thank the staff of the school and UTS for the help. I would like to sincerely thank the China Scholarship Council for financial assistance.

Last but not least, I would like to sincerely thank my parents for the continuous assistance and understanding.

Ming Liu

Sydney, Australia, 2019.

List of Publications

Journal Papers

J-1. **Ming Liu**, Zhi Xue, Xianghua Xu, Changmin Zhong, Jinjun Chen. Host-based Intrusion Detection System with System Calls: Review and Future Trends. ACM Computing Surveys. Accepted.

J-2. **Ming Liu**, Zhi Xue, Xiangjian He, Jinjun Chen. Enhancing the Collaborative Security with Cyber Threat Intelligence Information Sharing. IEEE Consumer Electronics. Accepted.

J-3. **Ming Liu**, Yuxuan He, Zhi Xue, Xiangjian He, Jinjun Chen. MultiScan: a Private Online Virus Detection System with Multiple Anti-virus Engines. IEEE Consumer Electronics. Accepted.

J-4. **Ming Liu**, Zhi Xue, Xiangjian He, Jinjun Chen. SCADS: A Scalable Approach Using Spark in Cloud for Host-based Intrusion Detection System with System Calls. To be submitted.

J-5. Pengze Guo, **Ming Liu**, Jun Wu, Zhi Xue, Xiangjian He. Energy-Efficient Fault-Tolerant Scheduling Algorithm for Real-Time Tasks in Cloud-Based 5G Networks. IEEE Access. Accepted.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis is composed of multiple aspects relating to host-based intrusion detection system (HIDS). First, this thesis provides a review of the development of system call-based HIDS. Second, this thesis forecast the future research trends of HIDS regarding three aspects, namely, the reduction of false positive rate, the improvement of detection efficiency, and the enhancement of collaborative security; then a real-time scalable HIDS framework with big data tools in cloud for a data center is proposed to enhance the collaborative security. Third, this thesis presents SCADS, a corresponding scalable HIDS solution using Apache Spark in the Google cloud environment to improve the detection efficiency of HIDS. Fourth, as future HIDS should work collaboratively with cyber threat intelligence (CTI), this thesis briefly introduces CTI and threat information sharing; and proposes a framework of a scalable real-time threat information sharing system in cloud. Fifth, this thesis provides a private and scalable multi-engine online virus detection system to contribute to the scalable real-time threat information sharing system in cloud.

In this chapter, section 1.1 provides the background of HIDS. Section 1.2 discusses the motivations of this research. Section 1.3 discusses the research objectives. Section 1.4 discusses the contributions of this study. Section 1.5 provides the organization of the rest of the thesis.

1.1 Background

Host-based intrusion detection system (HIDS) has been gaining attention in the community of cybersecurity for over two decades. Compared with 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 suffers 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 this kind of challenges according to their practical experience, and they listed some currently deployed systems and their performance in specific numbers for comparison [57]. Traditional database management systems and data mining methods on a single host may be not able to handle massive system call traces efficiently. Processing large amount 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 are kept 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 be not 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.

1.2 Research Motivations

The motivation of chapter 2 is that to the best of the author's knowledge, there are limited number of surveys discussing host-based intrusion detection system 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 [43]. This work was composed years ago, yet many new techniques and applications have been generated since that time. Hu [65] composed an introductory article about host-based anomaly intrusion detection. This article emphasized the application of Hidden Markov Model, 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 ADFA-LD dataset [3]. This survey was mostly about network-based anomaly detection system. Chandola et al. proposed a comprehensive taxonomy to classify studies of anomaly detection for discrete sequences into three distinct categories [21]. 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. Therefore, it is necessary to propose a survey to fill these voids.

The motivation of chapter 3 is that first of all, 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 a HIDS and has to be lowered down to a minimal level to save resources. Second, detection efficiency is another significant issue in real-time intrusion detection. Detection speed and accuracy are usually difficult to be well-balanced. Third, to enhance the collaborative security, it is a significant trend that HIDS should combine with NIDS to form future CIDS. Therefore, it is necessary to study these three future trends to improve future HIDS, and propose a new real-time scalable HIDS framework with big data tools in cloud for a data center to enhance the collaborative security.

The motivation of chapter 4 is that recently cloud computing presents extensive computational capability and massive storage capacity that can facilitate security specialists to implement data-intensive projects with manageable expenditure. Meanwhile, a set of modern frameworks such as Apache Hadoop and Apache Spark are specifically developed for stable and scalable processing of big data. Combining those big data pro-

cessing frameworks and the capability of cloud computing can provide an opportunity to improve the detection efficiency of traditional system call-based HIDS. However, to the best of the author's knowledge, there are limited research works concerning applying big data tools such as Apache Spark to system call-based HIDS. Therefore, it is necessary to propose a scalable HIDS approach using Spark in cloud to improve the detection efficiency and the scalability for a new-generation system call-based HIDS.

The motivation of chapter 5 is that to prohibit cyber attacks before they are launched, not only the vulnerabilities should be fixed, but the attack patterns should be perceived to optimize the effect of defense. Although attacks cannot be monitored before they are launched, the threats can be analyzed. A threat has the potential of exploiting vulnerabilities to compromise IT systems. The threats can be classified as low-leveled threats such as threats related to system and software vulnerabilities, and high-leveled threats such as APT. The active threat analysis approaches are helpful to gain the information about sophisticated novel attack methods. Utilizing cyber threat intelligence (CTI) can assist the deep threat analysis. CTI can facilitate the processing of the large quantity of threat information that industries confronted nowadays. Therefore, it is necessary to study the sharing of threat information, which is the key component of CTI and then propose a real-time scalable threat intelligence information sharing framework to enhance the collaborative security.

The motivation of chapter 6 is that current computer systems are constantly confronted with new kinds of computer viruses, which have evolved to more sophisticated and stealthy forms to hide from the detection of anti-virus systems [110]. Recently, some anti-virus systems are composed of multiple anti-virus engines. This kind of system allows users to upload suspicious samples online and then presents the users with multiple independent detection results. But these systems may save user-uploaded suspicious samples and share them to the security enterprises that own the anti-virus engines. Currently, some anti-virus engines provide cloud-based anti-virus services, and user files may be uploaded to the cloud for analysis. These factors may cause some privacy-leakage problems [163][166][162], particularly for the samples with high confidentiality. Therefore, it is necessary to construct a multi-engine system that can provide accurate and private virus detection services to those users with high privacy requirements.

1.3 Research Objectives

i. This thesis will provide a review of the development of HIDS with system calls.

ii. This thesis will forecast three future research trends of HIDS in the current big data and cloud computing environment, and then propose a real-time scalable HIDS framework with big data tools in cloud to enhance the collaborative security.

iii. This thesis will contribute to the community of HIDS by proposing a scalable HIDS approach using Spark in the Google cloud, endeavoring to improve the detection efficiency and the scalability for a new-generation system call-based HIDS.

iv. This thesis will study the issues related to threat intelligence information sharing, and then propose a real-time scalable threat intelligence information sharing framework to enhance the collaborative security.

v. This thesis will propose a private online virus detection system, which incorporates multiple anti-virus engines.

1.4 Research Contributions

i. In chapter 2, the main contributions are:

   (a) An overview of the development of system call-based HIDS is presented.

   (b) Algorithms and techniques relevant to system call-based HIDS are investigated, including various data mining methods.

   (c) The application of system call-based HIDS on embedded systems is studied, and related works are investigated.

   (d) The HIDS dataset issues are analyzed; currently available datasets with system calls and approaches for researchers to generate new datasets are discussed.

ii. In chapter 3, limitations relating to current system call-based HIDS are discussed, and future research trends are forecast regarding three aspects, namely, the reduction of false positive rate, the improvement of detection efficiency, and the

enhancement of collaborative security. Then a real-time scalable HIDS framework with big data tools in cloud for a data center is proposed to enhance the collaborative security, and this framework is easily scalable to fulfill the requirement of new hosts set up in the data center.

iii. In chapter 4, a HIDS solution using Apache Spark in the Google cloud environment for HIDS with system calls (SCADS) is presented. SCADS shows the ability to improve the detection efficiency and the scalability for a new-generation system call-based HIDS.

iv. In chapter 5, the main contributions are:

(a) The current threat information sharing ecosystem is investigated, including the famous three models of the threat information sharing process, the drawbacks of conventional sharing methods, and some new standards and platforms for the automatic exchange of structured threat information feeds.

(b) The necessity of constructing a scalable real-time threat information sharing system in the cloud is analyzed.

(c) A real-time scalable threat intelligence information sharing framework is proposed to enhance the collaborative security. The system design specifications are provided based on some recent notable platforms.

(d) The issue that demands to be solved in the future is addressed.

v. In chapter 6, the main contributions are:

(a) A private online virus detection system is presented. The system incorporates multiple anti-virus engines. The proposed system can perform the "isolated detection and update" of the anti-virus engines. This mechanism guarantees that the uploaded confidential samples are not exposed to the Internet, during either virus detection or system upgrade.

(b) A web interface is provided so that the system users can upload suspicious samples via web browsers and the detection results from multiple anti-virus engines can be displayed on the web interface.

(c) The low-coupling design of this system is scalable to support the distributed deployment mode.

1.5 Thesis Organization

The rest of this thesis is composed as follows.

- *Chapter 2*: This chapter presents the review of host-based intrusion detection system with system calls.
- *Chapter 3*: This chapter presents three future research trends of host-based intrusion detection system with system calls; then a real-time scalable HIDS framework with big data tools in cloud for a data center is proposed to enhance the collaborative security.
- *Chapter 4*: This chapter presents SCADS, a HIDS solution using Apache Spark in the Google cloud environment for HIDS with system calls.
- *Chapter 5*: This chapter introduces the cyber threat intelligence and the threat information sharing; and proposes a framework of scalable real-time threat information sharing system in cloud.
- *Chapter 6*: This chapter presents a private and scalable multi-engine online virus detection system to contribute to the scalable real-time threat information sharing system in cloud.
- *Chapter 7*: This chapter provides the conclusion of this thesis and discusses the potential future research works.

Chapter 2

Review of Host-based Intrusion Detection System with System Calls

2.1 Introduction

To the best of our knowledge, there are limited number of surveys discussing host-based intrusion detection system 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 [43]. This work was composed years ago, yet many new techniques and applications have been generated since that time. Hu [65] composed an introductory article about host-based anomaly intrusion detection. This article emphasized the application of Hidden Markov Model, 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 ADFA-LD dataset [3]. This survey was mostly about network-based anomaly detection system. Chandola et al. proposed a comprehensive taxonomy to classify studies of anomaly detection for discrete sequences into three distinct categories [21]. 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. This chapter endeavors to present a clear overview of the development for HIDS with system calls. The main contributions of this chapter are described below.

i. An overview of the development of system call-based HIDS is presented.

ii. Algorithms and techniques relevant to system call-based HIDS are investigated, including various data mining methods.

iii. The HIDS dataset issues are analyzed; currently available datasets with system calls and approaches for researchers to generate new datasets are discussed.

iv. The application of system call-based HIDS on embedded systems is studied, and

related works are investigated.

The rest of this chapter is composed as follows. Section 2.2 presents an overview of intrusion detection systems. In Section 2.3, algorithms and techniques of system call-based HIDS are outlined. In Section 2.4, HIDS dataset issues are analyzed, and dataset generation methods are discussed. In Section 2.5, the application of HIDS on embedded systems is studied. Section 2.6 gives summary of this chapter.

2.2 An overview of intrusion detection systems

An intrusion detection system, or IDS, monitors the real-time data of a network or a host/hosts to detect malicious behaviors [37]. 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) [99].

Types of attacks → Misuse detection system and anomaly detection system [99].

2.2.1 Categorization based on the types of analyzed data

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 is a kind of NIDS. 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 denial-of-service (DoS) attacks for interconnected server systems by applying multivariate correlation analysis [127]. 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 [128]. Network traffic entries are considered as corresponding images by taking multivariate correlation analysis. Those images that used

Table 2.1 : A typical system call sequence [44].

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

as observed objects are composed with Earth Mover's Distance. The experiment with ten-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 be detected by NIDS in this case.

Host-based intrusion detection system

HIDS monitors activities such as system or shell logs within a host/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 chapter. 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 2.1 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 recent twenty 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 time 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 [103]. 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.

Collaborative intrusion detection system

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, i.e., centralized CIDS, decentralized CIDS, and distributed CIDS [133].

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

2.2.2 Categorization based on the types of attacks

Misuse detection system

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, misuse detection system can effectively discover attack methods that are already identified.

Limitation of misuse detection system Misuse detection system is often criticized with 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.

Anomaly detection system

Anomaly detection system (ADS) requires no knowledge of known intrusions. Chandola et al. provided a comprehensive survey of anomaly detection system [20]. 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 a host/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 the criteria for anomaly detection. Therefore, anomaly detection system can detect zero-day attacks.

Challenge for host-based anomaly detection system High false alarm rate (FAR) is the challenge. Novel system call traces are being generated along with the 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.

2.2.3 Combination of misuse detection with anomaly detection

Although IDS can be categorized into these two groups, the combination of misuse detection system with 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, firstly 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 [10].

2.3 Algorithms and techniques of HIDS

Intrusion detection system was briefly introduced from different perspectives in the previous section. In this section, algorithms and techniques that mainly used in the

area of system call-based HIDS will be discussed.

2.3.1 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 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 as a contiguous sequence of n system calls extracted from a system call trace within a particular time interval [150]. 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 [140][72][158]. Creech et al. [30] 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 [126]. 6-gram and 7-gram are claimed to have better performance in UNM and ADFA-LD datasets, respectively [126][81]. In [6], 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.

2.3.2 Enumerating sequences

Enumerating sequences-based method, or known as the "sequence time-delay embedding (STIDE)" [106], is the first kind of system call-based HIDS method introduced by Forrest et al. [44][141][63][43]. STIDE is inspired by the natural immune systems of organisms. This approach is claimed simple and efficient to be deployed for possible real-time implementation. Parameters of system calls are removed, endeavoring to reduce system load and obtain the best result with system call numbers 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 databases construction. During this stage, sliding window algorithm is applied to scan normal system call traces and generate short sequences, 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 Hamming distance, which is acquired by counting different positions of two system call sequences, to get the number of mismatches [63]. 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. There-

fore, the efficient methods of building, updating, and maintaining those normal databases need to be designed.

2.3.3 Rule learning

Lee et al. implemented a rule learning-based method with continuous system call sequences to describe normal and abnormal kernel behaviors [84][83]. During the data preprocessing stage, 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 deviate 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. [72] 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 [130] designed some variants of the LEARD rule algorithm [91] to generate rules with system call sequences and their arguments. Qing et al. proposed an anomaly detection approach based on rough set theory [156]. A minimized set of rules is extracted from normal system call sequences to define a normal behavior model [156].

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 large data center.

2.3.4 Bloom filter

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 [9], 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, firstly 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 semi-supervised training method on system call sequences [140]. Bloom filters are implemented in this method to save space and enhance privacy-preserving ability. Anagram can detect anomalies and model malicious behaviors with "high detection rate and low false positive rate" [140], and shows robustness to some mimicry attacks [137].

Limitation of Bloom filter-based methods Bloom filter has shown the simplicity and effectiveness when computational space is inadequate. However, the limitation of Bloom filter is that it allows false positives [15], yet it is expected that the FPR of a HIDS can be minimized. Moreover, considering the improvements of 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.

2.3.5 Classification and clustering

The classification and clustering based methods here are referred to a set of commonly used machine learning algorithms that have been applied to system call-based HIDS. Well-known models such as k-nearest neighbor algorithm (kNN) [87][158], k-means clustering algorithm (KMC) [146][148], decision trees [158][10], support vector

machine (SVM) [158][30][147][142][8][77] and Bayesian networks [105] have been implemented on system call-based HIDS.

Yuxin et al. proposed a behavior-based detection method with semantic analysis approach [158]. 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 n-gram approach with 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 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.

2.3.6 Hidden Markov Model

Hidden Markov Model (HMM) is a doubly stochastic model which contains a finite number of hidden states [114]. Yeung et al. verified that applying HMM training on system call sequences operates better than modeling frequency distributions of shell command logs [157]. Hu [65] provided a detailed tutorial about how to apply HMM on system calls. HMM is widely known for its three fundamental problems shown in table 2.2, 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 [141]. 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 [98]. States are completely connected and reachable from each other. The probabilities

Table 2.2 : Three fundamental problems of HMM in HIDS.

The evaluation problem
Given an HMM $\lambda = (\Lambda, B, \pi)$, obtain $P(O\|\lambda)$, the probability of $O = \{O_1, O_2, ..., O_t, ..., O_T\}$, $1 \leq t \leq T$ [65]. Λ is the state transition probability matrix, B is the observation probability distribution, π is the initial state distribution, O is the observation sequence with time t [52]. A high $P(O\|\lambda)$ indicates that O is normal whereas a low $P(O\|\lambda)$ indicates that O is abnormal. $\rightarrow$ *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 $Q = \{q_1, q_2, ..., q_t, ..., q_T\}$, $1 \leq t \leq T$ to maximize the joint probability $P(O, Q\|\lambda)$ [132].
The learning problem
Given an observation sequence O and a model λ, obtain the parameters (Λ, B, π) to maximize $P(O\|\lambda)$ [65]. $\rightarrow$ *In HIDS this problem is related to training a model with the input observation sequence, i.e., benign system call trace.*

of states 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 sequences-based approaches. Alert is raised if the probability of an incoming system call is under a particular threshold.

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 [113]. 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 [85]. 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 fifteen. 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 [57].

2.3.7 Neural networks

Many HIDS works are related to neural networks, including multi-layer perceptron [1][33], self-organizing maps neural network [88], radial basis functions (RBF)-based neural networks [4], extreme learning machine [30], self-structuring confabulation network [23][22], etc. [102]. Ghosh et al. applied Elman neural network on DARPA dataset for anomaly detection, and the performance of the Elman nets is better than the back-propagation neural network in the experiments [50]. 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 [82].

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

2.3.8 Validation method and evaluation metrics

Cross-validation K-fold cross-validation is widely used to validate data mining algorithms [138]. Although k can be any integer number, ten-fold cross-validation is often applied. In ten-fold cross-validation, the original dataset is divided into ten partitions with the same size, nine partitions are used for training, and one partition is used for testing [70]. Then another nine partitions and the leftover one partition will be used for training and testing. This cross-validation approach will repeat for ten times with

different training and testing data, ten results will be generated and averaged to get the final result [75]. 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, thus the performance can be evaluated using the Precision/Recall criteria [34],

$$\begin{aligned} Precision &= \frac{TP}{TP+FP} \qquad Recall = \frac{TP}{TP+FN} \\ FPR &= \frac{FP}{FP+TN} \qquad TPR = \frac{TP}{TP+FN} \end{aligned} \tag{2.1}$$

where TP, FP, TN, and FN denote True Positive, False Positive, True Negative, False Negative, respectively [78]. The meaning of these terms for system call-based HIDS are described below:

• True Positive (TP) → The label of a trace is positive, and the prediction is also positive. The anomalous trace is alarmed properly.

• False Positive (FP) → The label of a trace is negative, but the prediction is positive. A FP or a false alarm in system call-based HIDS indicates that a benign system call trace is treated as an anomaly.

• True Negative (TN) → The label of a trace is negative, and the prediction is also negative. The incoming system call trace is treated as benign properly.

• False Negative (FN) → The label of a trace is positive, but the prediction is negative. A FN or a missed alarm indicates that the system fails to raise the alarm for an anomalous system call trace.

Table 2.3 shows the confusion matrix of these terms. FPR denotes False Positive Rate and TPR denotes True Positive Rate. In the area of IDS, True Positive Rate (TPR) is equal to detection rate (DR) and is equal to *Recall*, the False Positive Rate is equal to False Alarm Rate (FAR) [34]. F-measure is often adopted to combine Precision

Table 2.3 : Confusion matrix.

Predicted/Actual label	Actual intrusive	Actual benign
Predicted intrusive	TP	FP
Predicted benign	FN	TN

with Recall for evaluation,

$$F_\beta = \left(\beta^2 + 1\right) \cdot \left(\frac{Precision \cdot Recall}{\beta^2 \cdot Precision + Recall}\right) \tag{2.2}$$

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

$$F_1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{2.3}$$

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 [34].

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 [34]. 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 value is given, and AUC value is not shown. Therefore, standardized evaluation metrics are recommended so that comparing different research outcomes can be easier.

2.4 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

Table 2.4 : 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

studies describing the limitations of current HIDS datasets available on the Internet [89][167][122][11][94][90][3]. Some widely used system call-based HIDS datasets are described below and compared in table 2.4.

2.4.1 Current datasets for HIDS with system calls

DARPA and UNM DARPA and University of New Mexico dataset are two most commonly used datasets for evaluation of HIDS [107]. DARPA and UNM datasets were collected many years ago and may be outdated. They may contain errors and lack of volume, variety, and veracity from the perspective of real-world big data [153]. Those two datasets only have a few kinds of attacks with small data scale. Therefore, they

are not representative of current diversified attack methods. DARPA dataset consists of both network and host auditing data. Maggi et al. provided a thorough analysis of flaws and shortcomings of the DARPA dataset [89]. System call traces of DARPA dataset are relatively simple, and UNM dataset consists of system call identifiers only. Other information such as system call arguments is all removed. Therefore, traditional HIDS datasets with system calls cannot represent contemporary Linux systems.

Firefox-DS Firefox dataset [102] 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 ADFA Linux Dataset (ADFA-LD) [28][30] was created on 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, 4372 traces for analyzing 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 with system call arguments cannot be applied. Xie et al. [146][148] 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. [27] for Win-

dows, namely, "Australian Defence Force Academy Windows Dataset (ADFA-WD)" and "Australian Defence Force Academy Windows Dataset with a Stealth Attacks Addendum (ADFA-WD: SAA)" [55]. ADFA-WD involves acknowledged "Windows-based vulnerability-oriented zero-day attacks" [55] 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" [55]. These two datasets are not designed for Linux system call analysis.

NGIDS-DS Haider et al. proposed a synthetically high-quality and realistic IDS dataset named the next-generation IDS dataset (NGIDS-DS) using advantageous Ixia PerfectStorm commercial hardware platform [69] and infrastructure in ADFA [54]. 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" [125].

2.4.2 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 [115] on different kinds of vulnerabilities according to their Common Vulnerabilities Exposures (CVE) ID [25]. Table 2.5 compares typical system call tracing tools. Table 2.6 provides common tools for penetration testing.

Table 2.5 : Comparison of system call tracing tools.

Tracing tools	**Functional summaries**	**Other features**	**Platforms**
strace	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	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	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	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	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	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	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	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	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

Table 2.6 : Tools for HIDS dataset generation.

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

2.4.3 The challenge

This section 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 should have high quality, which ensure that relevant studies will not be misled by the datasets.

2.5 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 Linux kernel, which is similar to Linux operating systems installed on desktops [145]. 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, light-weight 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.

2.5.1 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 [39]. Features are categorized into four types, namely, "static features, dynamic features, hybrid features and applications metadata" [39]. System calls and network traffic are "two main types of dynamic features" [39]. Many researchers used system calls as dynamic features as applications use system calls to communicate with the mobile operating system [104]. They surveyed 100 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 [40]. 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 [14]. Statical 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 [18]. They indicate that monitoring system calls can provide detailed low-level information, which is feasible to determine malware behaviors. The system includes an overall framework for the collection of system call traces and the analysis of data; and the two-means clustering algorithm is applied to classify benign and intrusive applications [18]. 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 [18]. They also provide a brief survey, demonstrating that many malware detection approaches are based on NIDS and HIDS techniques.

2.5.2 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 [33]. 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 multi-layer 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 [33].

2.5.3 Cloud-based HIDS with system calls for embedded systems

Deploying system call-based HIDS as a two-tier platform shown in figure 2.1 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 feedbacks processed by the centralized unit can be returned to users' devices timely,

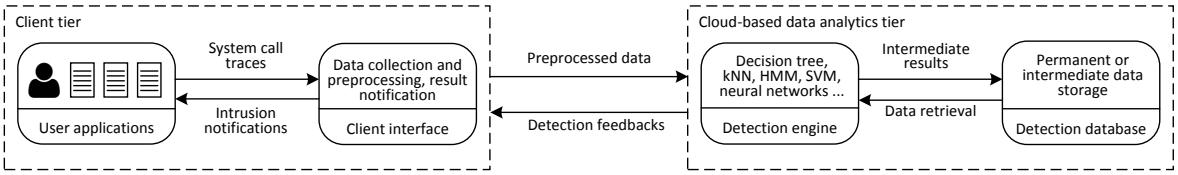

Figure 2.1 : A two-tier HIDS platform for mobile devices.

and further defensive actions can be launched.

Dual defense protection Su et al. proposed a "dual defense protection framework" for Android malware detection [124]. Applications can be uploaded to servers in the cloud for verification. The framework has two main components, i.e., system call monitoring tool and network monitoring tool. "System call monitoring tool" is utilized for evaluation of the "new application for potential malware", and "network monitoring tool" is applied to solve the false negatives problem [124]. The experiments show that the framework produces high detection accuracy with machine learning classifiers. The Weka tool is used for experiments, random forest classifier in the tool performs better than J.48 classifier, achieving the 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 timely, and further responses to the malware will not be delayed.

2.6 Summary

This chapter provides a survey of host-based intrusion detection system with system calls, from the perspectives of its origin, algorithms, datasets and application areas. Instead of elaborating every detail, the main aim of this chapter is trying to provide researchers a clear overview of the development of system call-based HIDS. Compared with choosing the optimal detection engines, data preparation and feature extraction are the decisive factors and therefore desire more attention. It is expected that high-quality datasets can be generated to guarantee that it is worth for further in-

vestigation. Meanwhile, standardized evaluation metrics are suggested to be presented in each research work for easier comparison.

Chapter 3

Future Trends of Host-based Intrusion Detection System and Constructing a Real-time Scalable HIDS in Cloud

3.1 Introduction

The previous chapter concentrated on the review of HIDS development. This chapter provides inspirational future research trends in the current big data and cloud computing environment. 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 chapter aims to inspire researchers in the community to collaboratively push forward the frontier of HIDS.

The rest of this chapter is composed as follows. Section 3.2 analyzes the reduction of false alarm rate. Section 3.3 discusses the improvement of detection efficiency. Section 3.4 studies the enhancement of collaborative security. Section 3.5 gives summary of this chapter.

3.2 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, high false alarm rate can affect the performance of a 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.

3.2.1 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 [105]. Results have shown that taking system call arguments to train Bayesian networks not only can improve the detection accuracy, but computation and memory resources can be saved as well. Maggi et al. [89] 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 [33].

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

3.2.2 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 missed alarm rate and false alarm rate [30]. Word and phrase dictionaries are formed with various-length system call numbers. They trained extreme learning machine (ELM) [67] for anomaly detection and achieved notable detection rate and false alarm rate. The result shows the effectiveness to handle 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 ADFA-LD and DARPA datasets [56][58]. The proposed method in conjunction with

multiple machine learning algorithms show 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 classification result, Ambusaidi et al. [8] 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 data and can deal with linearly as well as non-linearly 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 3.1 represents their proposed system.

3.2.3 Refine the decision-making process

Nauman et al. proposed a "three-way decision-making approach" to reduce false positive rate [106]. 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)" [106] are exploited to construct the model, and UNM dataset is taken for the experiment. A minimum 8.5% false positive rate is achieved with the proposed approach.

3.2.4 Threshold optimization

The threshold of a 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 high missed alarm rate, yet high sensitivity may cause 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. [81] discussed related problems of generating the optimal threshold and built an algorithm for threshold selection. ADFA-LD is used for evaluation, showing that the

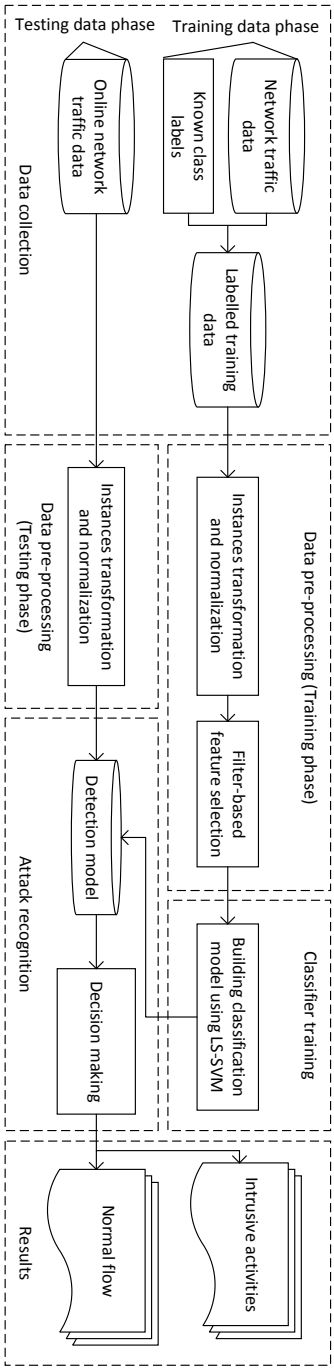

Figure 3.1 : LS-SVM-based intrusion detection system, modified from [8].

proposed algorithm achieves a better result than the "optimal uniform strategy" and "locally optimal strategy" [81].

3.2.5 The integration of decision engines

Fuzzy inference engine Hoang et al. built a "multi-layer model of program behaviors" [60], based on HMM and enumerating sequences method using the temporal characteristics. They further proposed a hybrid HIDS scheme [61] that integrates HMM with enumerating sequences method using a "fuzzy inference engine" to decrease false alarm rate. HMM is trained by adopting a modified HMMMOSA scheme [35] 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" [61] 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 which 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" [77]. Experiments on the Linux dataset (ADFA-LD) and the Window dataset (CANALI-LD [19]) show that the proposed method can reduce the false alarm rate and substantially increase the true positive rate on both of the two datasets [77]. 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.

3.2.6 Long short-term memory

Long short-term memory (LSTM) is an architecture of recurrent neural network (RNN) firstly presented by Hochreiter et al. [62]. LSTM network can lengthily persist temporal information and therefore 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 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 be captured 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 granularity [109].

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 exceed the size of most training traces. Instead of compressing all information of system call traces into fixed-length vectors, the attention model [149] 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 review network, which is an extension of the encoder-decoder framework, to augment global modeling capability [155].

The challenge of LSTM LSTM has gained notable advances on 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.

3.2.7 Challenges regarding this trend

Controlling the false alarm rates is still the most significant challenge and the critical point for both of 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 are kept being developed, the generated system call traces can be unstable and miscellaneous.

Therefore, how to preprocess the raw dataset and select right features can influence significantly on the performance of 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 training dataset.

3.3 Improvement of detection efficiency

Detection efficiency is another significant issue in real-time intrusion detection. Detection speed and accuracy are usually difficult to be well-balanced. When there is a heavy load of 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.

3.3.1 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 HMM incremental training process which reduces half of the training time for the UNM dataset [66]. 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.

3.3.2 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 [102][103]. 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 lower false alarm rate and less execution time. Their later work presented a creative and extensible Eclipse-based open-source HADS framework named automated anomaly detection framework (Total ADS) [101], 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) [102] 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 [23][22]. They aimed to exploit the computational capability of parallel computing platforms efficiently. The feature dependencies of 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 cloud environment by integrating "state summarization" and the novel "nested-arc hidden semi-Markov model (NAHSMM)" [57]. Designed according to the hierarchical HMM [41] 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.

3.3.3 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 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 3.1 provides an overview and comparison of public/open-source cloud services.

Table 3.1 : 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)" [74]: • *SaaS* utilizes the Internet to provide services and applications. Most of 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 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 some time. • *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 the 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.

3.3.4 Open-source big data tools

The development of scalable big data processing tools such as Hadoop shows high scalability for big data processing [166][165][162][154][152][151]. These tools are designed to perform underlying resources 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 compute 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 3.2.

Apache Hadoop Hadoop is a popular open-source distributed big data processing and storage framework [118]. 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 [96]. 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 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 [160], 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 [159]. Therefore, it only needs to re-compute that partition according to its lineage, instead of re-computing the whole DAG again. Figure 3.2 shows the structure of a standard Spark cluster. Spark currently can run on three kinds of cluster manager, namely, Standalone, Mesos, and YARN. Recently, for deep learning acceleration, utilizing Spark in cloud environment shows more flexibility compared with traditional methods. Philipp et al. introduced

Table 3.2 : 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 processing • 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*

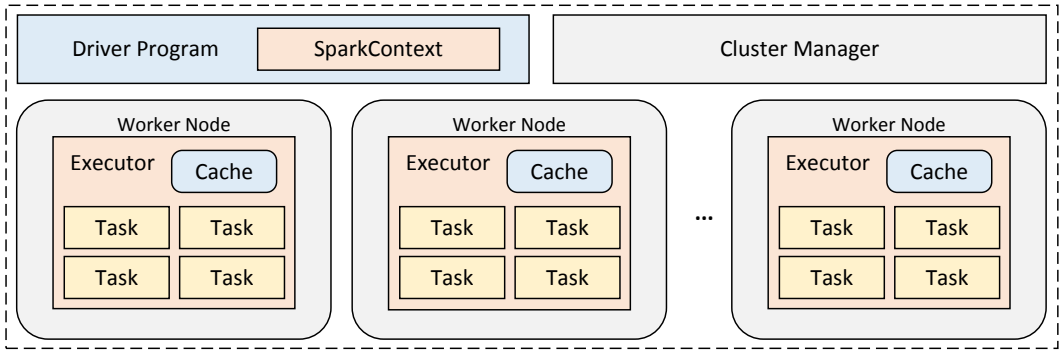

Figure 3.2 : A standard Spark cluster [49].

SparkNet [100] that implements a deep neural networks training framework on Spark, including interfaces of loading data from other RDDs and the connection to Caffe [71].

Spark Streaming Spark Streaming is a scalable fault-tolerant streaming data processing framework [161]. 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 which has a master-workers structure similar to HDFS [45]. 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 [47]. In real-time HIDS data processing framework, Kafka can operate as an intermediary between multiple hosts and the Spark cluster [120], propagating system call streams from origin to destination.

Table 3.3 : 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 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, random forest and clustering models such as k-means, latent Dirichlet allocation, Gaussian Mixture Model can be trained and predicted using MLlib [80]. • 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.

Scalability of algorithms in big data tools 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 3.3 discusses the scalability of a set of data mining algorithms in big data tools, particularly for system call-based HIDS.

3.3.5 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.

3.4 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 [120]. 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, 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 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 operating systems of physical machines.

3.4.1 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 [111]. This framework is based on modified KVM and is segregated from guest OS. Approaches for trapping desired events of system calls to the hypervisor are designed. Interrupt-based, syscall-based and sysenter-based system calls are supported [111]. Users can control the granularity of tracing the system call data.

Bag of system calls As 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 [6]. 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 [5] uses HMM with predefined capacity as the normal profile. Virtual machines are firstly 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 [10]. 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" [10], which assesses intrusion severity, and the result is transferred to an "intrusion response system" [10] to make appropriate responses.

Structure-based approach Gupta et al. developed a structure-based approach for anomalous processes detection in a private cloud environment [53]. 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 UNM dataset

due to the complexity of collection and maintenance of real-world system call data generated from the cloud. System call database of key-value pairs is built based on programs of 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 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. firstly 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 [120]. 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 Spark cluster for analytics. vSphere Guest SDK [136] is used for continuously monitoring streaming data with Kafka API integrated. CPU and memory usage percentages are gathered by mpsat [51] and vmstat [59] 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 resources allocation. Their next experiment was conducted in a VMware cluster, which involves five VMware hypervisor server systems, and each host has three virtual machines [121]. 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 Kafka cluster and Spark cluster. The interactive esxtop [13] utility of 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 Kafka cluster for further use. VMware resources 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

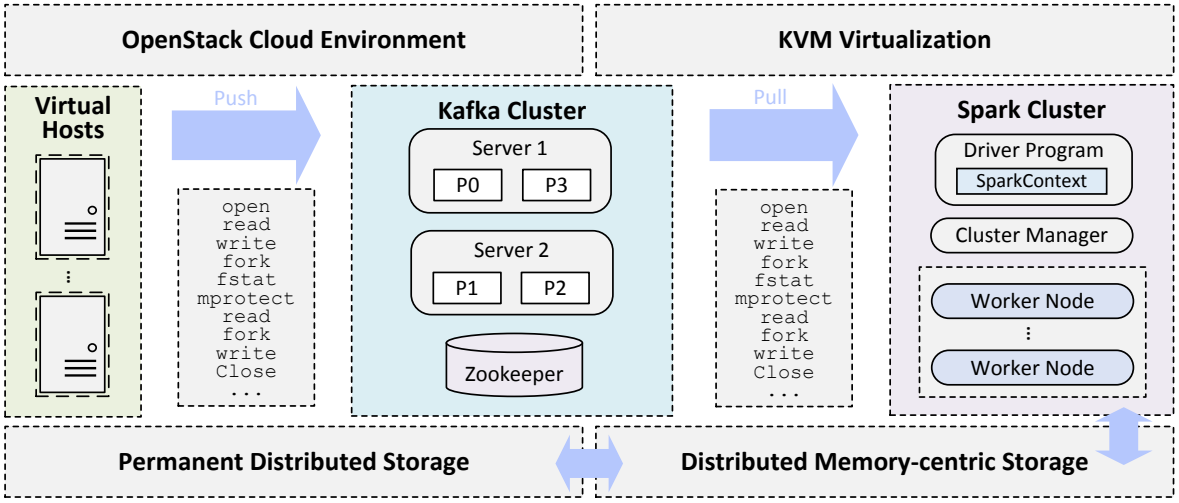

Figure 3.3 : A preliminary real-time scalable HIDS framework with big data tools in cloud. The system call traces are from [44].

anomaly, would be rendered in the performance metrics. The non-updatable 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.

3.4.2 Constructing a real-time scalable HIDS with big data tools in cloud

Inspired by current research works, a preliminary real-time scalable HIDS framework with big data tools in cloud for a data center is proposed. The framework is comprised of three layers described in table 3.4. Figure 3.3 demonstrates the framework deployed in an open-source private cloud computing environment. Computing facilities such as servers, routers, and switches are basic hardware components of the cloud. Kernel-based Virtual Machine (KVM) [16] open-source software included in 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, OpenStack open-source cloud operating system is installed for resource management and interactions with applications of users [164]. Administrators can control all resources via the provided dashboard. Hadoop with HDFS and YARN [46] 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

Table 3.4 : 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 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.*

by Apache Flume [48] from hosts in DMZ or Intranet are pushed into Kafka cluster for caching. Spark Streaming pulls system call traces accordingly from Kafka server to analyze anomalies in a fine-grained manner. This framework is easily scalable to fulfill the requirement of new hosts set up in the data center. With heavier data load as a result of 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 Spark cluster eventually. If an intrusion in an individual host is detected, Spark master can notify the scheduler to take appropriate measures on that host.

3.4.3 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 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 [129]. 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 [129]. An alternative central coordinator will be used if the primary one fails. With the central coordinator, data collected by cooperative agents can be summarized for comprehensive analysis to resist cooperative intrusions [129]. 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 [129]. The master node works as the central coordinator, and worker nodes work as cooperative agents. Figure 3.4 demonstrates the proposed framework.

The challenge According to [133], 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 approaches is not robust when confronted with advanced attacks. In this case,

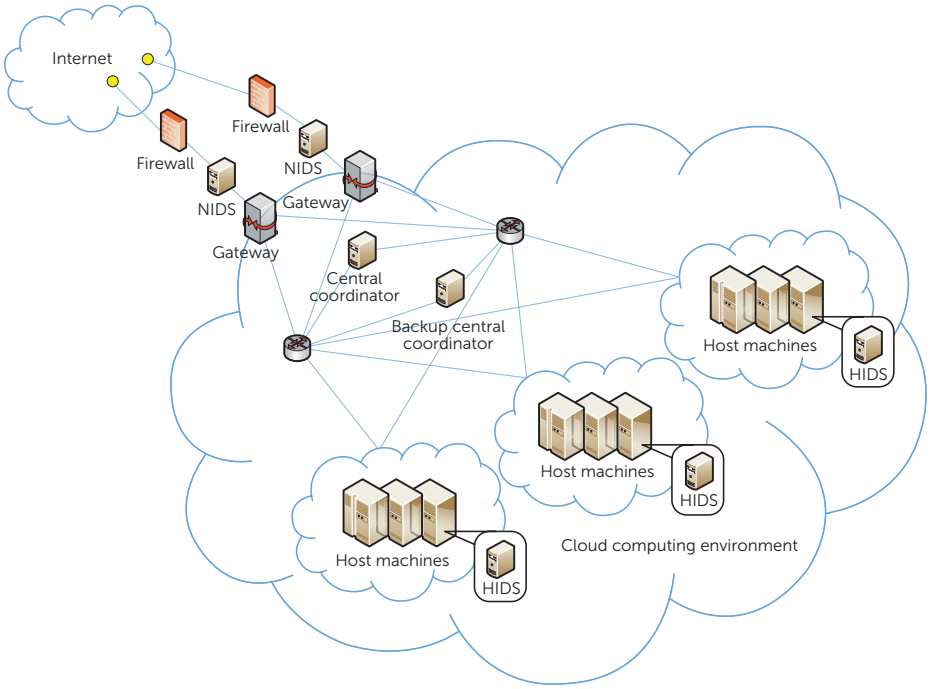

Figure 3.4 : A framework of CIDS [129].

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 [133].

3.4.4 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 [12].

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 upon 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 have been created to facilitate automatic sharing of threat information [73]. 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 that may confront 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 the privacy.

3.4.5 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 [93]. 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" [131]. 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 [108]. It monitors the security of 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 latest threat information of malicious hosts.

AlienVault USM The "AlienVault Unified Security Management platform (USM)" is a commercial product that combines HIDS with NIDS and other security mechanisms in a unified platform to manage threats [7]. USM can monitor the security of both local and cloud environments. The information that 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.

3.5 Summary

This chapter provides the future research trends of host-based intrusion detection system with system calls. This chapter aims to inspire future researchers about the three trends of HIDS, namely, the reduction of false positive rate, the improvement of detection efficiency, and the enhancement of collaborative security. This chapter also proposes a real-time scalable HIDS framework with big data tools in cloud for a data center to enhance the collaborative security. The framework is easily scalable to fulfill the requirement of new hosts set up in the data center. When it comes to current big data environment and the emerging of diversified cyber threats, combining multiple intrusion defense approaches to work collaboratively is the dominant trend for designing a robust threat-defensive infrastructure. Meanwhile, taking advantage of threat information sharing can be an enhancement to HIDS and CIDS.

Chapter 4

SCADS: A Scalable Approach Using Spark in Cloud for Host-based Intrusion Detection System with System Calls

4.1 Introduction

Recently, cloud computing presents extensive computational capability and massive storage capacity that can facilitate security specialists to implement data-intensive projects with manageable expenditure. Users can focus on their works using a group of flexible IT services, with less concern about the purchase and maintenance of physical devices. Therefore, various security enterprises have moved their projects to the cloud. Moreover, a set of modern frameworks such as Apache Hadoop and Apache Spark are specifically developed for stable and scalable processing of big data. These frameworks enable the processing and storage of massive datasets among clusters that have the "master-workers" structure. Clusters can be created with multiple common computers, which provide local computation and storage capability. With these big data processing frameworks, computational resources in clusters can be scheduled, hardware failures can be handled, and functional user interfaces can be provided. Therefore, combining those big data processing frameworks and the capability of cloud computing can provide an opportunity to improve the detection efficiency of traditional system call-based HIDS.

To the best of our knowledge, there are limited research works concerning applying big data tools such as Apache Spark to system call-based HIDS. Motivated by this issue, in this chapter, we contribute to the community of HIDS by proposing a scalable HIDS approach using Spark in the Google cloud, endeavoring to improve the detection efficiency and the scalability for a new-generation system call-based HIDS.

The rest of this chapter is organized as follows. In Section 4.2, we provide related works. In Section 4.3, we introduce the design of SCADS, the scalable approach using

Spark in the Google cloud for HIDS with system calls. In Section 4.4, we demonstrate the experimental results. In Section 4.5, the summary of this chapter is provided.

4.2 Related works

4.2.1 Public cloud

Contemporary public cloud computing services can offer load-balanced IaaS (Infrastructure as a Service) for scalable data analytics. Based on the information provided by Amazon EC2 [139] and Google Compute Engine [79], IaaS cloud computing services can be scaled from single to multiple virtual machines (instances) running in data centers with consistent performance. Users can manage and configure those instances on web interfaces. The pricing of those public cloud computing services is flexible. According to the scale their projects, users choose and pay for the size of processors, the volume of storage, and the time of computation. Moreover, renowned enterprises such as Amazon and Google may provide more reliable and less vulnerable cloud computing services, considering intruders may attack the intrusion detection software based on the vulnerabilities. Therefore, system call-based HIDS can be deployed in public cloud. General acceleration methods used in the community for system call-based HIDS can also be integrated into public cloud platforms.

4.2.2 Apache Spark

Contemporary big data processing frameworks (or big data tools) refer to a group of open-source software that can execute on clusters built with common hosts for distributed, scalable, and reliable processing of large-scale datasets. These frameworks provide functionalities such as distributed storage and processing of big data, and they usually have user-friendly APIs and web interfaces. These frameworks can deliver high-availability services, and frequent hardware failures can be automatically handled and restored. Distributed algorithms can be passed to worker nodes to process the local data. Therefore, with big data tools deploy on clusters, datasets can be efficiently processed.

Apache Spark is an open-source in-memory cluster computing framework [117]. Figure 4.1 demonstrates a simplified structure of Spark cluster. Spark can exchange data with various distributed storage systems such as the Hadoop Distributed File

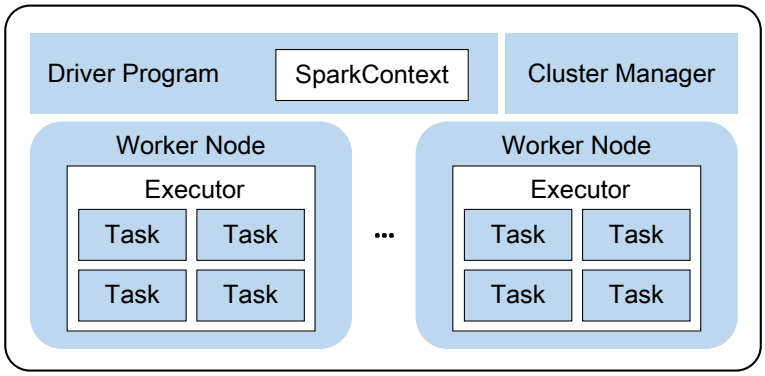

Figure 4.1 : A simplified structure of Spark cluster, modified from [123].

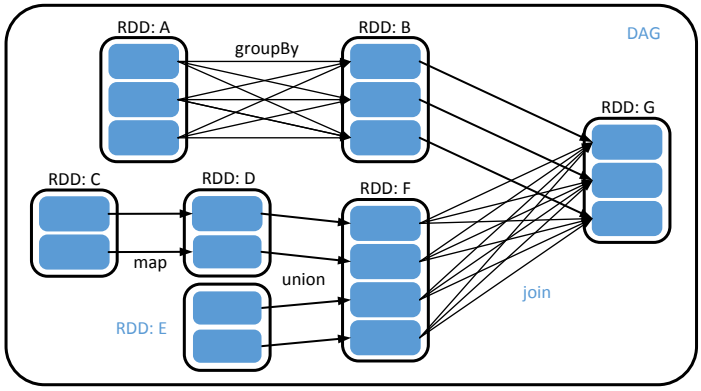

Figure 4.2 : An example of RDD and DAG. Unshaded rounded rectangles are RDDs. Shaded rounded rectangles are partitions. Modified from Zaharia's article [160].

System (HDFS), which splits files into redundant blocks that are distributed among worker nodes. Spark facilitates the implementation of iterative algorithms (such as the training of machine learning models). The intermediate datasets can be cached in the distributed memory. The distributed and fault-tolerant computing paradigm of Spark is based on the Resilient Distributed Dataset (RDD) and the relevant directed acyclic graph (DAG) (depicted in figure 4.2). An RDD consists of multiple partitions. The dependencies between RDDs can be classified into two types, namely, narrow dependencies and wide dependencies. Narrow dependencies allow pipelined executions in one cluster node with no shuffling, whereas wide dependencies require data shuffling across nodes [160]. Some examples of narrow dependencies and wide dependencies are depicted in figure 4.3. As the process of data shuffling in wide dependencies may require additional processing time, unnecessary data shuffling should be avoided.

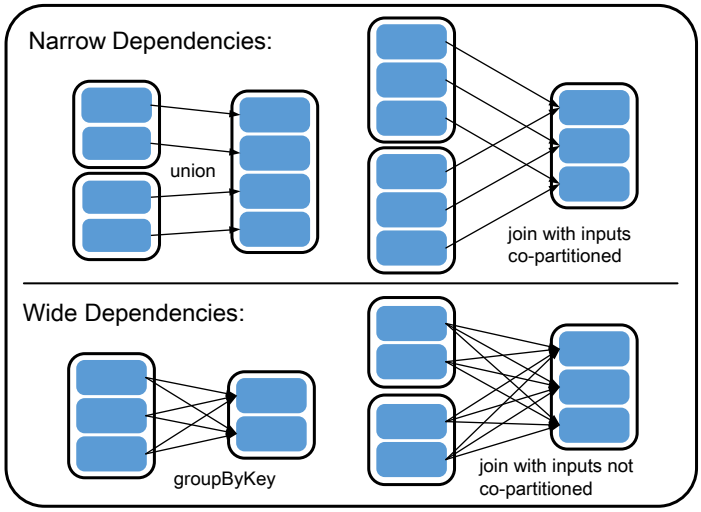

Figure 4.3 : Examples of narrow dependencies and wide dependencies. Modified from Zaharia's article [160].

4.3 Scalable Approach Using Spark in Cloud for system call-based HIDS

The intrusion detection procedure for SCADS is depicted in figure 4.4. There are two main phases in this procedure, i.e., the training phase and the testing phase. There are several steps for each phase.

• **The training phase**. Firstly, the ADFA-LD dataset is prepared and stored in the cloud storage system. For the preprocessing and feature extraction of the training phase, vectors of words are generated from raw system call traces with multiple-length n-gram method. Single-length n-gram method is also applied for comparison. TF-IDF feature vectors are generated from the vectors of words and labeled as normal or attack. The detection model is trained using logistic regression with limited-memory BFGS (LR-LBFGS).

• **The testing phase**. The preprocessing and feature extraction of the testing phase are same as the training phase. Then for the generation of TF-IDF feature vectors, it is assumed that in the real-world scenario of intrusion detection, real-time system call traces are gathered by the IDS trace by trace. Therefore, the IDF model of the testing dataset cannot be one-time obtained. In this case, for each testing trace, the TF vector is generated, then the TF-IDF feature vector is generated using IDF model

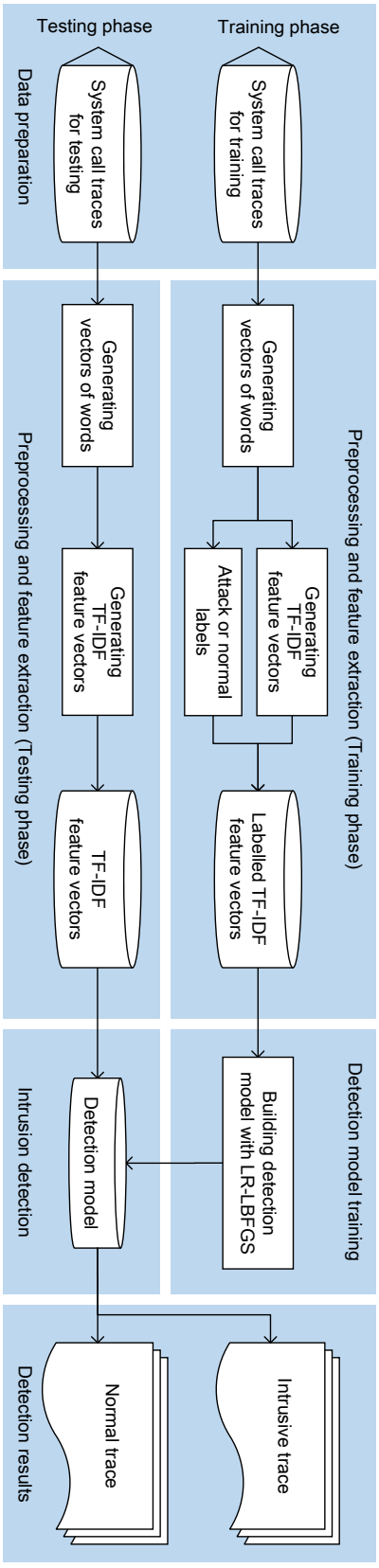

Figure 4.4 : The training and detection processes of SCADS.

Table 4.1 : Symbols and description.

Symbols	Description
$\mathbb{H}$	The complete dataset of system call traces $\mathcal{T}$
$\mathcal{T}$	A trace of system calls S
S	A system call
$\mathbb{D}$	The complete corpus of documents $\mathcal{D}$
$\mathcal{D}$	The generated document that contains the extracted words W of systems calls S
W	An extracted word of systems calls S

of the training dataset and then passed to the detection model for the prediction of either intrusive or normal trace. Finally, the value of AUC is returned to evaluate the detection accuracy. We have also used TF feature vectors only in this phase for comparison.

The detailed steps are elaborated in the following subsections.

4.3.1 Symbols

Symbols used for description of the proposed approach are listed and described in table 4.1.

4.3.2 Preprocessing and feature extraction

RDD repartitioning

To achieve the detection efficiency and scalability for a HIDS using Spark, the design of the intrusion detection algorithms should conform to the internal mechanism of Spark. To achieve parallel processing, the raw system call traces should be firstly distributed evenly among worker nodes of the Spark cluster, and then be processed by appropriate algorithms in parallel. In the proposed approach, after eliminating redundant traces, the raw system call traces for training and testing are treated as Spark RDDs, each RDD has several partitions. When the system call traces are firstly loaded from the storage system and cached as RDDs, they may only have a few partitions and are not distributed evenly among worker nodes of the cluster. Therefore, we have designed an RDD repartitioning method using Spark API to transform those RDDs of raw system call traces into new RDDs with more partitions; the new partitions are distributed evenly to the worker nodes.

Algorithm 1: Generate vectors of words from an RDD of raw system call traces using Spark [123]

Input: An RDD of system call traces *traces*
Output: An RDD of vectors of words *vects*

```
1 def VectsGen(traces : RDD[String]) : RDD[Seq[String]]=
2     val tHashed = traces.map(x => (x.hashCode, x))
3     val rangePtnr = new RangePartitioner(traces.count.toInt, tHashed)
4     val tPtned = tHashed.partitionBy(rangePtnr)
5     val input = tPtned.values.flatMap(_.split("\\s+")).cache
6     val output = input.mapPartitions(WordsGen).cache
7     val vects = output.mapPartitions(iter =>
        Iterator(iter.toArray.toSeq)).coalesce(8, false)
8     return vects
9 end
```

The RDD repartitioning method for system call traces is implemented in algorithm 1, and figure 4.5 provides a simplified demonstration of the RDD repartitioning process using *RangePartitioner* in Spark. In algorithm 1, for an RDD of system call traces *traces*, every trace is assigned a hashCode value by calling the transformation *map*. Then the RDD *tHashed* is repartitioned by the *RangePartitioner*, which can partition traces with hashCode values into roughly equal ranges. Then a Spark *RangePartitioner* *rangePtnr* is constructed with two parameters. Since *traces*.count.toInt counts the number of system call traces in the dataset, the number of partitions of the new RDD *tPtned* is roughly equal to the number of system call traces. Thus each partition of *tPtned* may represent a system call trace in this case. Then for the repartitioned RDD *tPtned*, after processed by the transformation *flatMap* on the values to split by one or more whitespaces, the transformation *mapPartitions* operates on every partition of the RDD *input* to extract words of system calls by calling function *WordsGen*. Then the RDD *output* is coalesced to eight partitions by using the transformation *coalesce*, which is "useful for running operations more efficiently after filtering down a large dataset" [123]. The number of the coalesced partitions in the proposed approach is selected based on the number of worker nodes. The second parameter of *coalesce* is set to "false" in the proposed approach, as it is expected that the coalescence of partitions only occurs within the local worker nodes with no data shuffling, i.e., the coalescence conforms to the narrow dependencies depicted in figure 4.3. Using the parameter of

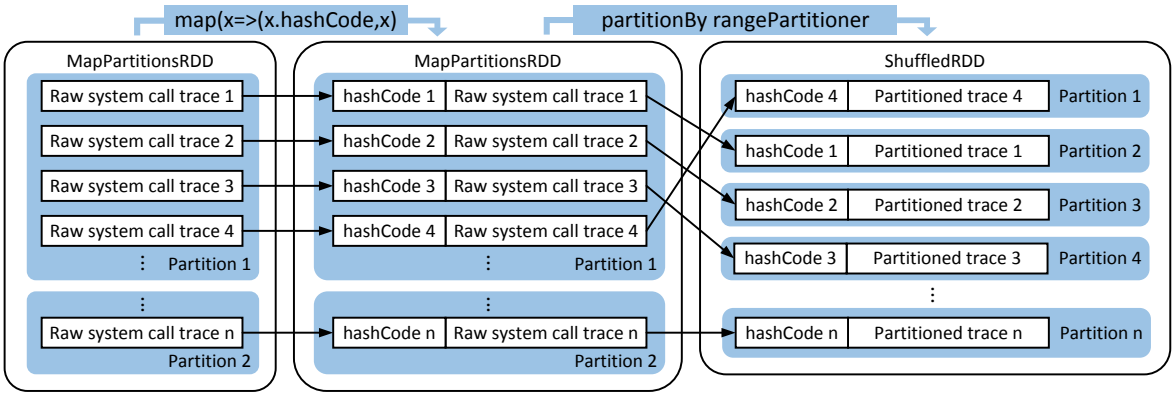

Figure 4.5 : A simplified demonstration of the RDD repartitioning process by *RangePartitioner* for the preprocessing of raw system call traces.

"true" may cause additional shuffling between worker nodes.

Single-length n-grams

In the area of system call-based HIDS, an n-gram is referred to a contiguous sequence of n system calls extracted from a system call trace within a time interval [150]. The sliding window algorithm with window size n can be taken to scan a system call trace to generate n-grams of system calls, and the generated n-grams can be used for training detection models. Using the n-gram method for preprocessing has been widely adopted by HIDS researchers [140][158]. The term "single-length n-gram method" is used to compare with the "multiple-length n-gram method".

The process of feature extraction with the single-length n-gram method is described as follows. Let $\mathbb{H} = \{\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_m\}$ be the complete dataset of m system call traces $\mathcal{T}$; and let $\mathcal{T} = \{S_1, S_2, ..., S_k\}$ be a trace of k system calls S. Then for a trace $\mathcal{T}$, one sliding window of length n are applied to traverse from the beginning to the end of trace $\mathcal{T}$ to extract contiguous system call sequences. The maximum window size n allowed in this case is k, which is the length of trace $\mathcal{T}$. The contiguous system call sequences extracted are treated as words W of length n. Therefore, if n is equal to six, for trace $\mathcal{T}$ the generated document $\mathcal{D}$ that contains the extracted words W of systems calls is,

$$\mathcal{D} = \{S_1S_2S_3S_4S_5S_6,\ S_2S_3S_4S_5S_6S_7,\ S_3S_4S_5S_6S_7S_8,\ ...,$$
$$S_{k-5}S_{k-4}S_{k-3}S_{k-2}S_{k-1}S_k\}$$

Algorithm 2 implements the method of single-length n-grams using Spark. It takes the sliding window algorithm with one window size n. In Algorithm 2, n is set to six. Thus the length of the extracted words W is six.

Algorithm 2: Generate words of system calls with the single-length n-gram method from one RDD partition using Spark [123]

```
1 def WordsGen[T : ClassTag](iter : Iterator[T])=
2 |   var words = List[String]()
3 |   var arr = iter.toArray
4 |   val n = 6
5 |   for j = 0 → (arr.length − n) do
6 |   |   words ::= arr.slice(j, j + n).mkString(" ")
7 |   end
8 |   words.iterator
9 end
```

Multiple-length n-grams

The multiple-length n-gram method has been proven to be effective to increase the detection accuracy in the area of system call-based HIDS [144][92][76]. For instance, Creech et al. used multiple sliding windows with various sizes to scan the complete system call trace; the generated multiple-length n-grams were used for feature extraction, and notable detection accuracy was achieved[30]. Therefore, we have implemented the multiple-length n-gram method with Spark for feature extraction.

The process of feature extraction with the multiple-length n-gram method is described as follows. Let $\mathbb{H} = \{\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_m\}$ be the complete dataset of m system call traces $\mathcal{T}$; and $\mathcal{T} = \{S_1, S_2, ..., S_k\}$ be a trace of k system calls S. Then for a trace $\mathcal{T}$, a set of sliding windows ranging from size one to n are applied to traverse from the beginning to the end of trace $\mathcal{T}$ to extract contiguous system call sequences. In this case, the maximum window size n allowed is k, which is the length of trace $\mathcal{T}$. The contiguous system call sequences extracted at this stage form words W of length one to n. Therefore, if n is equal to six, for trace $\mathcal{T}$ the generated document $\mathcal{D}$ that contains the extracted words W of system calls is,

$\mathcal{D} = \{S_1, S_2, S_3, ..., S_k,$
$S_1S_2, S_2S_3, ..., S_{k-1}S_k,$

$S_1 \rightarrow S_3$, $S_2 \rightarrow S_4$, ..., $S_{k-2}S_{k-1}S_k$,
$S_1 \rightarrow S_4$, $S_2 \rightarrow S_5$, ..., $S_{k-3}S_{k-2}S_{k-1}S_k$,
$S_1 \rightarrow S_5$, $S_2 \rightarrow S_6$, ..., $S_{k-4}S_{k-3}S_{k-2}S_{k-1}S_k$,
$S_1 \rightarrow S_6$, $S_2 \rightarrow S_7$, ..., $S_{k-5}S_{k-4}S_{k-3}S_{k-2}S_{k-1}S_k\}$

Algorithm 3 implements the method of multiple-length n-grams with the sliding window algorithm with multiple window sizes. In Algorithm 3, the maximum window size n is set to six, thus the length of the extracted words can range from one to six. When the maximum window size n is set to 1, the algorithm is the same with the single-length n-gram method.

Algorithm 3: Generate words of system calls with the multiple-length n-gram method from one RDD partition using Spark [123]

```
1  def WordsGen[T : ClassTag](iter : Iterator[T])=
2      var words = List[String]()
3      var arr = iter.toArray
4      val n = 6
5      for i = 0 → (n − 1) do
6          for j = 0 → (arr.length − i − 1) do
7              words ::= arr.slice(j, j + i + 1).mkString
8          end
9      end
10     words.iterator
11 end
```

Constructing feature vectors with TF-IDF

In this chapter, the description of feature extraction using TF-IDF is mainly based on [123]. "Term frequency-inverse document frequency (TF-IDF)" is a feature extraction scheme commonly used in text-based information retrieval to measure the importance of a word in a document and the relevant corpus [86]. Term frequency (TF) is the number of times that the word appears in the document; document frequency (DF) is the number of documents that contain the word [64]. Using only TF may over-emphasize frequent words, as a frequent word in the corpus may carry little special information about a particular document. Inverse document frequency (IDF) can measure the quantity of information that a word provides about a particular document. Thus the TF-IDF value of a word increases according to its frequency in the

document and is adjusted by the frequency in the corpus. We have implemented TF-IDF with Spark to construct feature vectors. The process is described as follows. Let $\mathbb{D} = \{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_n\}$ denote the complete corpus of n documents $\mathcal{D}$. Term frequency $TF(W, \mathcal{D})$ is the number of times that a word W appears in document $\mathcal{D}$, while document frequency $DF(W, \mathbb{D})$ is the number of documents that contain the word W [32]. In this case, the IDF is defined as,

$$IDF(W, \mathbb{D}) = \log \frac{n+1}{DF(W, \mathbb{D}) + 1} \tag{4.1}$$

The TF-IDF is defined as,

$$TFIDF(W, \mathcal{D}, \mathbb{D}) = TF(W, \mathcal{D}) \cdot IDF(W, \mathbb{D}) \tag{4.2}$$

Algorithm 4: Count the number of distinct words for an RDD of vectors of words using Spark [123]

Input: An RDD of vectors of words *vects*
Output: The number of distinct words *wordsNum*

```
1 def NumOfDistinctWords(vects : RDD[Seq[String]]) : Int=
2 |   val wordsNum =
  |    vects.flatMap{identity}.map(x => (x, 1)).reduceByKey(_ + _).count.toInt
3 |   return wordsNum
4 end
```

Algorithm 5: Generate TF feature vectors from an RDD of vectors of words as sparse vectors using Spark [123]

Input: An RDD of vectors of words *vects*, the number of dimensions *num*
Output: Sparse TF feature vectors *tf*

```
1 def TfGen(vects : RDD[Seq[String]], num : Int) :
   RDD[org.apache.spark.mllib.linalg.Vector]=
2 |   val hashingTF = new HashingTF(num)
3 |   val tf = hashingTF.transform(vects).cache()
4 |   return tf
5 end
```

TF and IDF are implemented in *HashingTF* and *IDF* in Spark. The TF-IDF feature vectors generated from the training and testing system call traces are also treated as

Spark RDDs. The *HashingTF* utilizes feature hashing in machine learning [143]. In the proposed approach, to reduce the chance of hash collisions, the dimension number of the target feature vectors is set to the number of distinct words in the corpus of the training dataset. Algorithm 4 is implemented to count the number of distinct words for an RDD of vectors of words. Algorithm 5 is implemented to generate TF feature vectors from an RDD of vectors of words as sparse vectors, and algorithm 6 is implemented to generate TF-IDF feature vectors from an RDD of vectors of words as sparse vectors. The real-time system call traces are assumed to be gathered by the IDS trace by trace. Thus the whole testing dataset cannot be one-time processed, instead only one single trace can be handled each time. In this case, for the testing traces, only TF feature vectors are generated from them, and the IDF model is unobtainable. Therefore, algorithm 6 is applied to the training dataset only. For the testing dataset, the TF feature vectors generated are adjusted by the Spark IDF model generated from the training dataset.

Algorithm 6: Generate TF-IDF feature vectors from an RDD of vectors of words as sparse vectors using Spark [123]

Input: An RDD of vectors of words $vects$, the number of dimensions num
Output: Sparse TF-IDF feature vectors $tfidf$

```
1 def TfidfGen(vects : RDD[Seq[String]], num : Int) :
    RDD[org.apache.spark.mllib.linalg.Vector]=
2   val hashingTF = new HashingTF(num)
3   val tf = hashingTF.transform(vects).cache()
4   val idf = new IDF().fit(tf)
5   val tfidf = idf.transform(tf)
6   return tfidf
7 end
```

4.3.3 Classifier training and prediction

Logistic regression is a widely used linear method for binary classification. In the proposed approach, logistic regression with limited-memory BFGS (LR-LBFGS) is utilized for classifier training and prediction, as it is recommended in Spark for faster convergence. By default, standard feature scaling and L2 regularization are utilized for LR-LBFGS in Spark MLlib. The detailed classifier training and prediction steps are described below.

i. In the process of classifier training and prediction, firstly the attack traces and normal traces of ADFA-LD are loaded from the Google cloud storage system. The *distinct* transformation, which can produce a new RDD with only distinct elements, is called to eliminate repetitive traces for both of attack traces and normal traces. For both of the attack traces and normal traces, 60% of them are randomly selected as the training data, and the other 40% of them are selected as the testing data.

ii. To get the number of dimensions *num* for functions *TfGen* and *TfidfGen*, the attack training traces and normal training traces are combined using the *union* method in Spark, followed by applying the *distinct* transformation to eliminate repetitive traces. Then the function *VectsGen* is applied to the combined traces to generate vectors of words, followed by applying the function *NumOfDistinct-Words* to get the number of distinct words, i.e., the number of dimensions.

iii. For both of the attack training data and normal training data, the function *VectsGen* is called to generate vectors of words, followed by applying the function *TfidfGen* to generate TF-IDF feature vectors. The attack feature vectors are labeled point 1, and the normal feature vectors are labeled point 0. Then the labeled feature vectors are combined using the *union* method in Spark to form *trainingData*, which is the dataset for training the LR-LBFGS classifier.

iv. As the real-time system call traces are assumed to be gathered by the IDS trace by trace, thus the whole testing dataset cannot be one-time processed, instead only one single trace can be processed every time. Therefore, for both of the attack testing data and normal testing data, the function *VectsGen* is called to generate vectors of words, followed by applying the function *TfGen* to generate TF feature vectors. Then the function *TfGen* is applied to the vectors of words formed in step 2 to generate TF feature vectors from the training dataset, followed by generating the relevant Spark IDF model of the training dataset. Then this IDF model is used to adjust the attack testing TF feature vectors and normal testing TF feature vectors. The attack feature vectors are labeled point 1, and the normal feature vectors are labeled point 0. These label points are used for evaluation of the system only. Thus the labeling process is not shown in figure 4.4. Finally, the labeled feature vectors are combined using the *union* method

in Spark to form *testingData*, which is the dataset for testing the LR-LBFGS classifier.

v. Algorithm 7 implements the classifier training and prediction with LR-LBFGS using Spark. As implemented in algorithm 7, the *trainingData* and the *testingData* are used for training and prediction, respectively. The LR-LBFGS classifier is trained using the *LogisticRegressionWithLBFGS* method in Spark. For prediction, the labels predicted by the classifiers are compared with the labels of the testing feature vectors to evaluate the detection accuracy. Finally, the AUC value *AUC* is returned using Spark *BinaryClassificationMetrics* and the *areaUnderROC* method.

Algorithm 7: Classifier training and prediction with LR-LBFGS using Spark [123]

Input: The dataset for training *trainingData*, the dataset for testing *testingData*
Output: The value of AUC *AUC*

```
1 val model = new LogisticRegressionWithLBFGS().run(trainingData)
2 model.clearThreshold
3 val predictionAndLabels = testingData.map
    {case LabeledPoint(label, features) =>
    val prediction = model.predict(features) (prediction, label)}
4 val metrics = new BinaryClassificationMetrics(predictionAndLabels)
5 val AUC = metrics.areaUnderROC
6 return AUC
```

4.4 Experiments

4.4.1 The computational environment

The experiments are launched with Dataproc on the Google Compute Engine [79] cloud computing environment. Dataproc provides pre-installed and administrated Apache Hadoop and Spark services with a user-friendly API for configuration. Dataproc can automatically and quickly create and manage clusters of virtual machines. There are three modes to deploy a cluster, i.e., "single node (one master and zero workers)", "standard (one master and multiple workers)", or "high availability (three masters and numerous workers)". Virtual machines of a cluster can connect with each

other with internal IP networking. Users can customize the number of virtual CPUs; the memory capacity; the sizes and types of disks; and the region/zone where the cluster to be deployed. The created master node contains the HDFS NameNode and the YARN ResourceManager, and can be accessed with SSH; each of the worker nodes includes an HDFS DataNode and a YARN NodeManager. Spark jobs can be submitted via the Dataproc API, where the jobs' output can be accessed; the graphs of the network, disk, and CPU utilization can also be viewed. Via the API, the created cluster can be scaled up or down regarding the number of worker nodes. Therefore, a cluster can be scaled multiple times with simple operations, which is flexible to test the scalability of the proposed approach. In our experiments, a free account has been firstly set up. Standard clusters (one master, N workers) are created within the *global* region and *us-central1-c* zone. For a cluster, the master node has one vCPU with 3.75GB memory, and the primary disk size is 100GB; each of the worker nodes has the same machine type with the master node. Spark version 2.2.1 on Dataproc is used for the experiments, and the number of Spark executors can be flexibly extended from one to six.

4.4.2 The ADFA-LD dataset

As most of the datasets utilized for measuring system call-based HIDS were created in the last century and cannot represent modern attack approaches, Creech et al. compiled the new publicly available ADFA-LD dataset [30]. In their method, Ubuntu Linux is the host operating system, which represents a contemporary Linux server and provides multiple functionalities with a few vulnerabilities. The dataset has three subsets of raw system call traces, including 833 traces of normal data, 4372 traces of validation data, and 746 traces of attack data. Training and validation traces were gathered during normal activities of the operating system. The attack traces were gathered under the cyber attack environment. Attack methods include "brute force password guessing", "add new superuser", "Java Meterpreter payload", "Linux Meterpreter payload", and "C100 webshell" [29]. The ADFA-LD dataset was claimed to be more challenging to investigate than the traditional datasets since the modern Linux environment has become more complicated than before [29]. Thus, ADFA-LD is a new benchmark dataset for analyzing system call-based HIDS and is utilized in our experiments. In our experiments, the attack traces and normal traces of ADFA-LD

are utilized. Multiple files in the dataset are combined into single files for experiments. The dataset is prepared and stored in a bucket of the Google's cloud storage system. With Google Cloud Storage, users can store and retrieve large-scale data flexibly, regarding the time and the location. The default storage class of the bucket used for experiments is Multi-Regional, and the location is ASIA. During an experiment, after logging into the master node with SSH and launching the Spark shell, the attack traces and normal traces of ADFA-LD are firstly retrieved from the bucket of Google's cloud storage system, followed by performing the complete training and testing phases for intrusion detection.

4.4.3 Performance evaluation

Evaluation regarding the detection accuracy

The detection accuracy of a system call-based HIDS can be evaluated with the True Positive Rate (TPR) and False Positive Rate (FPR) criteria, which are defined as [2],

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad \text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{4.3}$$

• True Positive (TP): The label of a system call trace is attack and the prediction is also attack. The attack trace is detected correctly.

• False Positive (FP): The label of a trace is normal but the prediction is attack. A normal system call trace is falsely detected as an attack trace.

• True Negative (TN): The label of a trace is normal and the prediction is also normal. The testing system call trace is correctly predicted as normal.

• False Negative (FN): The label of a trace is attack but the prediction is normal. The system omits an attack system call trace.

The Receiver Operating Characteristic (ROC) curve can provide an intuitive view of the relation between the false positive rate (FPR) and the true positive rate (TPR) [116]. In the proposed approach, the performance regarding detection accuracy is evaluated using values of the area under the ROC curve (AUC). AUC can be a simple metric to provide an overall evaluation of the detection accuracy. Using AUC values, we have evaluated the performance regarding two aspects, i.e., the effectiveness of using

the multiple-length n-gram method for feature extraction, and the effectiveness of using the TF-IDF method to construct feature vectors.

i. To test the effectiveness of using the multiple-length n-gram method for feature extraction, the AUC values achieved using the multiple-length n-gram method are compared with the AUC values achieved using the single-length n-gram method. For the single-length n-gram method, we have tested the performance with the length of n from 1 to 10. For each length of n, the AUC values of 10 experiments are recorded and averaged in table 4.2. For the multiple-length n-gram method, we have tested the performance with the maximum length of n from 1 to 10. For each of the maximum length of n, the AUC values of 10 experiments are recorded and averaged in table 4.3. The averaged AUC values for both of the single-length n-gram method and the multiple-length n-gram method are depicted in figure 4.6 for comparison. Based on figure 4.6, for the multiple-length n-gram method, the AUC values reach a plateau when the maximum length of n is greater than 5. For the single-length n-gram method, the AUC values reach the maximum when n is equal to 5. Based on our implementation methods and experimental results, the multiple-length n-gram method outperforms the single-length n-gram method using the LR-LBFGS classifier when the sizes of n (or maximum n) range from 1 to 10.

Table 4.2 : AUC values obtained from ten experiments using the single-length n-gram method.

n	E.1	E.2	E.3	E.4	E.5	E.6	E.7	E.8	E.9	E.10	Avg.
1	0.937	0.939	0.908	0.954	0.945	0.896	0.939	0.944	0.958	0.943	0.936
2	0.979	0.968	0.956	0.952	0.964	0.982	0.975	0.970	0.970	0.967	0.968
3	0.978	0.982	0.968	0.975	0.979	0.972	0.977	0.981	0.974	0.987	0.977
4	0.978	0.977	0.982	0.970	0.976	0.965	0.979	0.976	0.981	0.979	0.976
5	0.979	0.978	0.986	0.977	0.983	0.975	0.975	0.987	0.986	0.983	0.981
6	0.981	0.968	0.947	0.978	0.983	0.985	0.978	0.977	0.978	0.983	0.976
7	0.987	0.976	0.977	0.979	0.978	0.974	0.975	0.976	0.972	0.975	0.977
8	0.956	0.965	0.955	0.966	0.972	0.971	0.981	0.988	0.966	0.979	0.970
9	0.972	0.967	0.963	0.967	0.969	0.943	0.958	0.952	0.970	0.968	0.963
10	0.936	0.958	0.964	0.962	0.947	0.965	0.954	0.959	0.966	0.960	0.957

ii. To test the effectiveness of using the TF-IDF method to construct feature vectors, for the testing traces, we have compared the detection accuracies of using only the

Table 4.3 : AUC values obtained from ten experiments using the multiple-length n-gram method.

Max. n	E.1	E.2	E.3	E.4	E.5	E.6	E.7	E.8	E.9	E.10	Avg.
1	0.885	0.940	0.936	0.945	0.956	0.945	0.951	0.922	0.935	0.943	0.936
2	0.978	0.957	0.966	0.963	0.976	0.981	0.965	0.976	0.973	0.974	0.971
3	0.970	0.986	0.978	0.974	0.986	0.967	0.985	0.968	0.973	0.978	0.977
4	0.977	0.979	0.986	0.969	0.973	0.975	0.988	0.979	0.985	0.957	0.977
5	0.988	0.984	0.989	0.987	0.982	0.980	0.990	0.978	0.985	0.981	0.984
6	0.981	0.992	0.987	0.980	0.982	0.987	0.990	0.985	0.979	0.983	0.985
7	0.986	0.985	0.985	0.987	0.981	0.985	0.992	0.984	0.982	0.980	0.985
8	0.981	0.977	0.989	0.988	0.983	0.989	0.986	0.981	0.989	0.981	0.984
9	0.980	0.981	0.980	0.987	0.977	0.985	0.986	0.979	0.991	0.983	0.983
10	0.980	0.979	0.987	0.989	0.979	0.993	0.980	0.992	0.983	0.983	0.985

Table 4.4 : AUC values obtained with the multiple-length n-gram method and only the TF feature vectors for testing.

Max. n	E.1	E.2	E.3	E.4	E.5	E.6	E.7	E.8	E.9	E.10	Avg.
1	0.807	0.841	0.818	0.739	0.756	0.935	0.770	0.658	0.813	0.817	0.795
2	0.836	0.878	0.859	0.890	0.929	0.886	0.844	0.906	0.850	0.860	0.874
3	0.928	0.926	0.938	0.947	0.973	0.948	0.957	0.925	0.938	0.935	0.942
4	0.913	0.973	0.940	0.904	0.974	0.913	0.957	0.964	0.946	0.944	0.943
5	0..912	0.971	0.874	0.895	0.979	0.970	0.970	0.912	0.981	0.928	0.939
6	0.968	0925	0.941	0.902	0.955	0.953	0.907	0.938	0.961	0.980	0.943
7	0.901	0.957	0.881	0.920	0.972	0.936	0.984	0.916	0.943	0.920	0.933
8	0.975	0.964	0.983	0.966	0.976	0.907	0.968	0.978	0.982	0.975	0.967
9	0.978	0.974	0.951	0.954	0.942	0.944	0.982	0.962	0.981	0.978	0.965
10	0.959	0.970	0.973	0.959	0.958	0.970	0.916	0.966	0.958	0.981	0.961

TF feature vectors with using the TF-IDF feature vectors adjusted by the Spark IDF model generated from the training traces. The multiple-length n-gram method is used. The AUC values of using only the TF feature vectors for testing traces are recorded in table 4.4. For each of the maximum length of n, the AUC values of 10 experiments are recorded and averaged. The averaged AUC values of these two cases (TF only and TF-IDF) are depicted in figure 4.7. Based on our implementation methods and experimental results, using only the TF feature vectors for testing shows unstable performance, and using Spark IDF model generated from the training traces to adjust the TF feature vectors of the testing traces is effective to improve the detection accuracy.

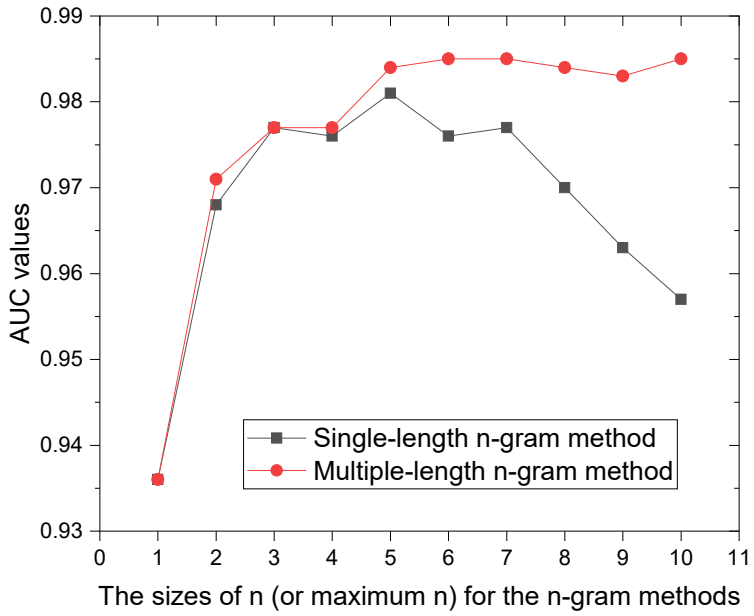

Figure 4.6 : AUC values of the single-length n-gram method and the multiple-length n-gram method.

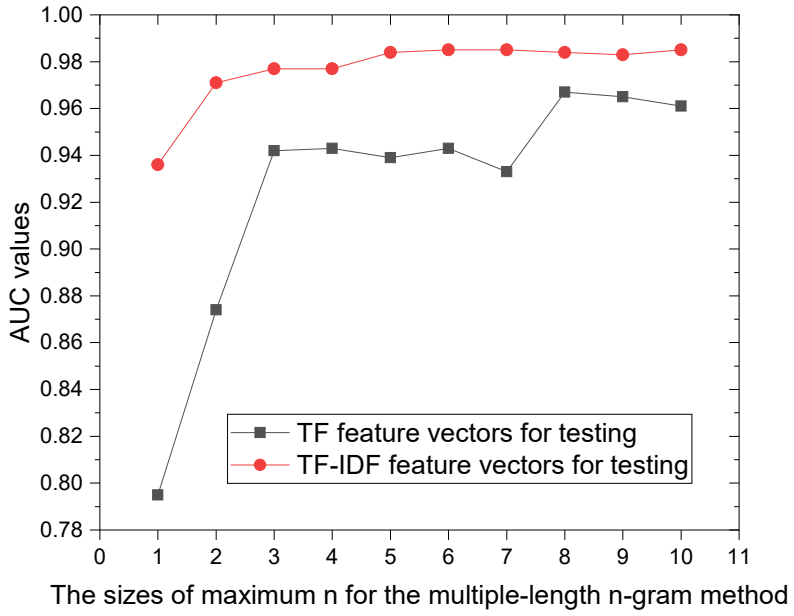

Figure 4.7 : AUC values of using TF feature vectors only and TF-IDF feature vectors for testing traces.

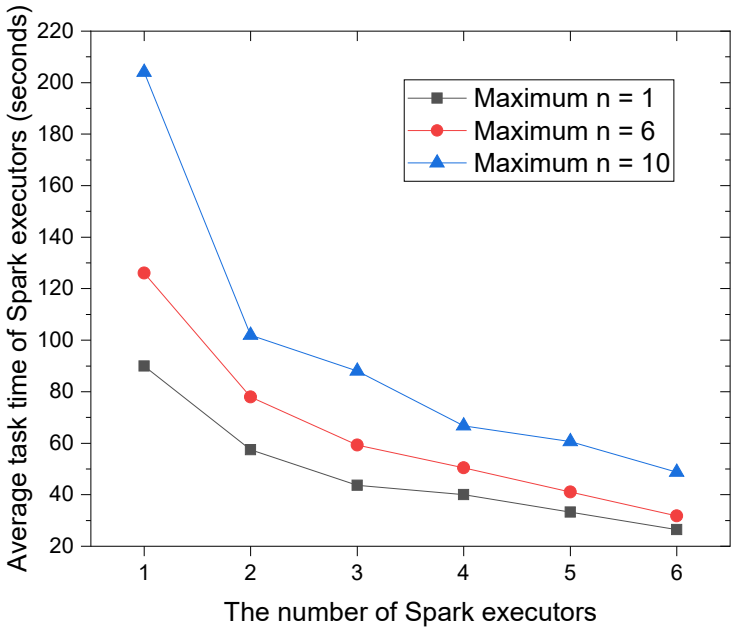

Figure 4.8 : The scalability of SCADS. The number of Spark executors ranges from 1 to 6.

Evaluation of the scalability

Via the Dataproc API, the created Spark cluster can be flexibly scaled up or down to test the scalability of the proposed approach. As we are using a free account of the Google Compute Engine, the number of Spark executors in our experiments ranges from 1 to 6. In each case of the number of Spark executors, we have executed the system three times, applying the multiple-length n-gram method with the maximum length of n=1, 6, and 10, respectively. The relevant average task time of Spark executors is recorded in each case. As depicted in figure 4.8, regarding the average task time of Spark executors, the approach of SCADS shows acceptable scalability. With the increase of number of Spark executors, the average task time of Spark executors decreases apparently.

4.5 Summary

In this chapter, we have proposed SCADS, a scalable approach using Spark in the Google cloud for host-based intrusion detection system with system calls. Based on the experimental results, the proposed framework with Apache Spark and Google cloud

computing services can improve the detection efficiency of traditional system call-based HIDS. Besides the algorithms used in our experiments, other existing intrusion detection algorithms may also be incorporated into this proposed framework with Apache Spark and Google cloud computing services. However, as the current experiments are conducted with a free account, the scale of computation is relatively limited. Therefore, in the future experiments, we will keep tunning the system to achieve better system performance, and we will also consider upgrading the account to a paid one for the processing of real-time large-scale system call traces.

Chapter 5

Enhancing the Collaborative Security with Cyber Threat Intelligence Information Sharing

5.1 Introduction

The term big data is prevalent in the research area of cybersecurity. Big data commonly refers to datasets with large size, regarding the volume, variety, and velocity. Currently, the relationship between cybersecurity and big data has become more sophisticated. For enterprises, big data may bring threats as well as opportunities to them. Although attackers have the chance to obtain the private and secret information of the enterprises via the utilization of modern attack approaches with the analysis of big data, the defenders can also analyze the large-scale data about attackers and generate defensive methods to defend future attacks.

5.1.1 Motivation of attackers

A cyber attack refers to a set of intrusive methods toward some particular computer systems or networks, manipulated by some people or associations, resulting in malicious consequences, such as system damage, financial loss or information stealing. For a particular enterprise, attacks may come from the adversaries or the internal network. The attackers' hacking demands and interests, such as distributing virus and Trojans towards the targeted organizations to get the financial benefits, never terminate. Attackers prefer to install malware into the targeted systems to obtain high system authorities and steal private information of enterprises. Beyond traditional attack strategies, the attack approaches are becoming more advanced and sophisticated nowadays. Innovative attack patterns are kept being generated by adversaries. Profit is not the only reason why attackers launch malicious attacks. Some attacks come from well-organized terrorists and professional cybercriminals with sufficient funding. This kind of attackers prefers to perform long-term social engineering methods to launch complicated and advanced attacks toward large IT systems, endeavoring to obtain

some special purposes. Attacks are prepared thoughtfully and launched quickly, resulting in severe private data leakage, financial loss or other serious consequences.

The "advanced persistent threat" [31], or APT, is the definition of this kind of behavior. An increasing number of APT incidents have been witnessed during the past few years. Attackers nowadays prefer to adopt APT approaches to hack IoT devices and systems [110][112]. Absolute isolation from APT is impractical, as organizations are inevitable to connect to the Internet. Small and medium enterprises, large corporations, security companies, and government agencies are all confronted with this kind of challenge. Revolutionary defensive approaches are necessitated to be developed to conform to the new attack patterns of APT and keep up with the pace of advanced attackers.

5.1.2 Conventional defensive approaches

Defense is much more challenging than attack, as intrusions can happen by exploiting only a few vulnerabilities, whereas defenders have to ensure the whole system they protect is impeccable. Conventional defensive methods concentrate on scanning and mining the system and software vulnerabilities. Vulnerabilities can have various types, such as system or software backdoors, faulty administration of IT facilities, or other sorts of human errors that are often exploited by social engineering strategies. This kind of concentration is negative and obsolete when confronted with recent comprehensive attacks. Although these methods are still valid for some kinds of attacks, however, they cannot prevent most kinds of APT and present no information about attackers for long-term defense. For instance, a system administrator may spot a malware in an operating system and uses the antivirus software to remove that malware and fixes related system and software vulnerabilities. This process is considered to be a full traditional incident response activity. However, a severe security incident may still be ignored in this case. The system administrator perceives no information about the origin of the malware and the related attacker. Possibly, the installation of the malware is just one of the steps of an advanced persistent threat. Even if the malware is eliminated, the attacker may seek other vulnerabilities and reinstall a modified version of the malware.

5.1.3 Using cyber threat intelligence to analyze the attack patterns of adversaries

Although there are methods to alleviate the impact caused by attacks, dealing with occurred attacks is still cumbersome, especially for the recovery of loss. Therefore, optimally, enterprises should prevent intrusive behaviors beforehand, instead of just making responses or analysis after attacks have already occurred. The best time of defense is the time when attacks are not happened yet, followed by the time when attacks have just happened. The quicker that an attack is spotted, the more effective that the incident response will be. E.g., if information about the domain names that contain fraud content can be immediately shared with other enterprises, then the relevant websites can be blocked by them timely to prevent future visits. Consequently, the concept of actively detecting potential attacks is currently emerging. To prohibit attacks before they are launched, not only the vulnerabilities should be fixed, but the information about attackers, or the attack patterns, should be perceived to optimize the effect of defense. Attack patterns are convincing to represent essential information about attack techniques utilized by adversaries. Consequently, by analyzing attack patterns, there is a larger possibility for enterprises to predict future attacks and take proper actions to secure the systems in advance and avoid compromises. Attack patterns are usually generated from deep threat analysis of the data about attackers.

Although attacks cannot be monitored before they are launched, the threats can be analyzed. A threat has the potential of exploiting vulnerabilities to compromise IT systems. The threats can be classified as low-leveled threats such as threats related to system and software vulnerabilities, and high-leveled threats such as APT. The active threat analysis approaches are helpful to gain the information about sophisticated novel attack methods. Utilizing cyber threat intelligence can assist the deep threat analysis. The term "Cyber threat intelligence" or "CTI" was firstly defined by Gartner in 2013, and the meaning of "Intelligence" is referred as both of the information about enemies and the computational ability:

• According to Gartner's definition in 2013, "cyber threat intelligence" is "evidence-based knowledge, including context, mechanisms, indicators, implications and actionable advice, about an existing or emerging menace or hazard to assets that can be used to inform decisions regarding the subject's response to that menace or hazard." [95]

• Based on the definition of Merriam-Webster dictionary, "intelligence is the ability to learn or understand or to deal with new situations; information concerning an enemy or possible enemy; the ability to perform computer functions." [97]

CTI can facilitate the processing of the large quantity of threat information that industries confronted nowadays. The sharing of threat information is the key component of CTI, which will be elaborated in the rest of this chapter. The contribution of this chapter is that the issues related to threat intelligence information sharing are studied, and then a real-time scalable threat intelligence information sharing framework is proposed to enhance the collaborative security.

The rest of this chapter is composed as follows. In section 5.2, we investigate the current threat information sharing ecosystem, including the famous three models of the threat information sharing process, the drawbacks of conventional sharing methods, and some new standards and platforms for the automatic exchange of structured threat information feeds. In section 5.3, we analyze the necessity of constructing a scalable real-time threat information sharing system in the cloud. In section 5.4, we provide the system design specifications based on some recent notable platforms. In section 5.5, we present the summary of this chapter.

5.2 The threat information sharing ecosystem

The functionality of CTI can only be maximized via the "sharing" or "exchange" of threat information. Many organizations are rapidly becoming active consumers of CTI, as they can obtain the threat information that is associated with APT. They can understand the behaviors of attackers thoroughly with the threat information and then customize their cybersecurity systems. The "threat information" is the investigated and interpreted information that is practical and actionable to the defense of threats. Complete threat information includes multiple factors, such as the attacker's identification, attack approaches, previously launched attacks, attack targets, vulnerabilities of the targeted systems, and possible solutions. According to "Cybersecurity Information Sharing Act of 2015 (CISA)" [36], threat information is comprised of two phases, namely, Threat Indicator and Defensive Measure:

• **Threat Indicator**: The threat indicator, or the indicator of compromise (IOC),

represents information about malicious reconnaissance, attack methods, vulnerabilities, internal threats, malicious cyber commands, actual or potential harm, and other attributes.

• **Defensive Measure**: The defensive measure is related to how an information system should cope with threats and vulnerabilities.

Internet users and companies can use the generated threat information to judge the risk of opening some particular websites or running some specific executable files. Therefore Internet frauds can be relatively prevented. Some characteristics of threat information are listed below:

• The threat information is usually generated from analyzing the threat data from heterogeneous sources, and available as feeds for integration; the generated threat information can either be short-term or long-term;

• The threat information does not present information about vulnerabilities only; it mainly provides attack patterns of adversaries so that the defense can be more active;

• The threat information feeds can be integrated into organizations' defense systems, such as malware protection systems, firewalls or intrusion prevention systems, either as the system update or for the incident response purposes.

For a particular organization, its threat information mainly comes from three origins, namely, the intranet, the public and the collaborators:

• **The intranet**: The threat information of intranet includes records and logs from defensive tools and devices such as antivirus software, intrusion detection systems (IDS), and the firewalls.

• **The public**: The public refers to sources out of the community. The public threat information is usually free and easy to obtain. However, the quality cannot be ensured.

• **The community collaborators**: Community collaborators for threat information sharing are other participants within a community that has a mutual defensive aim. Joining an active community that expedites structured threat information sharing can help enterprises to grasp a broad and quick view of the cyber threat environment. The

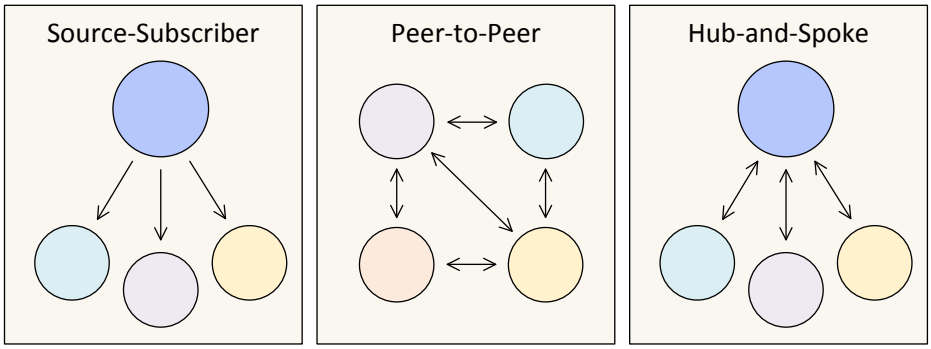

Figure 5.1 : Three models of the threat information sharing process. Modified from [24].

participants of the community can be individuals such as students, threat analysts, and academic researchers. They also can be groups such as private businesses, government organizations, and research institutes.

It is difficult for enterprises to maintain security without collaboration, as an isolated enterprise cannot acquire sufficient, reliable, and multi-scope threat information. The process of sharing can enhance the diversity, quantity, and quality of the threat information and hence can produce an all-around description of attackers. Via threat information sharing, threat analysts from different enterprises within the community can work together to solve the most complex security problems. Contributions from various enterprises can accomplish a set of comprehensive and multi-scope threat information, which in turn can bring benefits to each of those enterprises. E.g., if an attack is discovered in one enterprise, relevant threat information can be shared with other enterprises in time to prevent more attacks. In general, as shown in figure 5.1, the process of threat information sharing can be defined by three models, namely, Peer-to-Peer, Source-Subscriber, and Hub-and-Spoke [24]:

• **Peer-to-Peer**: A couple of collaborators share threat information with each other according to some standards and agreements.

• **Source-Subscriber**: A couple of consumers receive the threat information distributed from a single producer.

• **Hub-and-Spoke**: A centralized unit receives and analyzes the threat information contributed from multiple collaborators, and then the processed threat information can

be consumed by the collaborators.

Conventional threat information sharing methods are mostly based on the Peer-to-Peer and Source-Subscriber models, whereas the Hub-and-Spoke model has been broadly accepted for the development of the future-generation threat information sharing systems.

5.2.1 Drawbacks of conventional sharing methods

Conventionally, the sharing of threat information was a manual process. Threat information is not shared in a real-time manner. Threat analysts exchange their threat information via emails, forums, and web pages. They publish information regarding the encountered threats and their actions online. This mechanism may be practical when there was a small amount of threat information. However, as the volume of threat information grows larger than before, this mechanism encounters the dilemma, which has three major drawbacks:

• **Security and privacy**: The privacy of the shared threat information cannot be guaranteed, as the information is directly shared on the Internet forums and web pages without encryption, the attackers can obtain them easily. Due to this reason, many enterprises do not prefer to share their threat information online.

• **Authenticity and dependability**: The authenticity and dependability of the threat information may not be well-checked before they are submitted online. E.g., some organizations may encounter false alarms on their defense systems but share the false information online, which may mislead other organizations.

• **Efficiency and efficacy**: The more time it takes to update the defensive deployment with the threat information, the more possibility for the system to be compromised by new attacks. Regarding the variety of attack approaches, the volume and the generation velocity of related threat information because of the increasing amount of the cyber attacks, analyzing the information by human resources is an arduous work that could take excessive time, and it may be tedious for anyone to grasp a complete view of the threat environment by browsing forums or web pages. Important and newly updated threat information may be easily ignored in this case. Moreover, the

threat information for sharing is usually not in a standardized structure, which makes it extremely cumbersome for organizations to understand and integrate.

Trust is the dominant factor regarding the secure sharing of threat information. To preserve the data privacy and trust for the sharing of threat information, Fisk et al. specially defined three principles, namely, "Least Disclosure, Qualitative Evaluation, and Forward Progress", and implemented them with engineering methods [42]. The system of an enterprise may be compromised if its sensitive threat information is presented to untrusted people or groups. E.g., if the threat information about some kinds of attacks is uncovered, the attacker may realize that the information has been investigated, and then does modifications to the attack approaches, resulting in ineffectiveness of the defense system. Therefore, threat information cannot be easily shared with the untrusted public and organizations should consolidate with a trusted community for the threat information exchange.

5.2.2 New standards and platforms for the automatic sharing of structured threat information feeds

Obviously, manual and unorganized threat information sharing methods are gradually outdated. Currently, some innovative platforms and standards that are constructive to the automatic sharing of threat information have been developed. Those normalized languages, dictionaries, and platforms facilitate standardized representation and automatic integration of the threat information:

- The standardized languages and dictionaries endeavor to define all of the threat information about computer systems, software weaknesses, vulnerabilities, configurations, platforms, attack patterns, etc. in a machine-understandable manner.

- The platforms incorporate those languages and dictionaries to perform efficient collection, processing, and sharing of threat information.

With these standards and platforms, organizations can produce their standardized threat information feeds for the automatic sharing within the community. The integration of the feeds can be efficient concerning computational resources. Table 5.1 summarizes the commonly used standards and platforms for threat information sharing. Beside of the standards and platforms listed in the table, there are some other

Table 5.1 : A summary of the standards and platforms for threat information sharing.

CTI tools	**Category**	**Threat information presented or functionalities**
CVE	Dictionary	Identifiers of vulnerabilities
CWE	Dictionary	Software security weaknesses
CPE	Dictionary	Description of IT platforms
CCE	Dictionary	Identifiers of system configuration issues
CAPEC	Dictionary	Identified attack patterns
MAEC	Language	Machine-understandable malware information
CybOX	Language	Information about cyber observables
OVAL	Language	System information and machine states
STIX	Language	Standardized threat information
XCCDF	Language	Security checklists
CVSS	Framework	Provides severity scores of vulnerabilities
TAXII	Framework	Facilitates secure and automatic threat information sharing
OpenIOC	Framework	Facilitates quick information sharing with an XML schema

studies related to automatic threat information sharing. E.g., ENISA provides an analysis of the standards and platforms for threat information sharing, including their pros and cons [38]. Costa et al. proposed a standardized ontology that facilitates the sharing of threat indicators among multiple participants while protecting the sensitive information [26]. Burger et al. proposed a taxonomy of threat information sharing approaches with an agnostic framework [17]. Iannacone et al. proposed an ontology for cyber threat knowledge graphs to standardize threat information from heterogeneous sources [68].

5.3 Constructing a scalable real-time threat information sharing system in the cloud

Within a trusted community, to promote the automatic sharing of threat information, not only a set of standards should be utilized to compose threat information as structured feeds, but a scalable real-time system should be implemented. An ideal threat information sharing system should promote the automatic integration of threat information, and keep the insight with the latest global cyber threat environment and the threat situation in the local area. Currently, large-scale threat data streams are continuously being generated from heterogeneous sources. These threat data streams may be essential resources for the threat analysis. However, due to their high volume, velocity, variety, and veracity, relevant technologies need to be employed to process

them. The sharing of the generated threat information feeds should also be conducted timely.

Currently, many academic, government or commercial CTI providers choose to maintain a centralized system for the gathering, processing, and sharing of real-time threat information, such as MITRE TAXII, IBM X-Force, WEBROOT BrightCloud, EclecticIQ platform, and AlienVault OTX platform. Deploying the threat information sharing system in the cloud has been widely adopted. The elasticity, computational ability, and storage capacity of cloud computing ensure that the computational resources can flexibly match the demand of the threat information sharing system. Meanwhile, the development of the big data tools also brings new resources to the construction of the information sharing system. The tools described below have notable scalability:

• **Hadoop**: Apache Hadoop is a cluster-based big data processing and storage framework. A master node manages multiple worker nodes for computation. Next generation MapReduce on YARN is the computational framework, and HDFS is the storage framework. HDFS can be applied as a long-term storage for the generated threat information feeds. However, analyzing the large volume of real-time threat data with MapReduce may cause additional I/O time among hard disks because of shuffling.

• **Kafka**: Apache Kafka is a modern distributed streaming platform. Kafka was initially designed for the gathering and caching of data streams. Recently, besides these functions, the function of processing data streams has been developed as well in Kafka. In the real-time threat information sharing system, Kafka can work as a message broker between sources of threat data and the processing system. Kafka cluster can persist streams of threat data as topics, each of which can have multiple consumers include Spark Streaming.

• **Spark**: Apache Spark is a distributed in-memory big data processing framework. In Spark, intermediate datasets can be persisted into distributed memory to save the cost of I/O. Spark also has a master-workers deployment mode. The Resilient Distributed Dataset (RDD) is Spark's fault-tolerant distributed dataset abstraction. The fault-tolerant mechanism of Spark is based on the DAG of RDDs. Spark Streaming is an ideal solution for processing real-time threat data.

• **Alluxio**: Alluxio is an open source distributed in-memory storage system, which also has a master-workers deployment mode. It can be deployed in the same cluster within which Spark and Hadoop are deployed, as an off-heap storage solution for the intermediate datasets of the threat information sharing system.

In general, deploying the threat information sharing system as the "Software as a Service (SaaS)" mode in the cloud with big data tools has the following aspects of advantages:

• **Elasticity**: As the system is deployed in the cloud instead of physical machines, less space and devices are required, which save computational resources. Plenty of cloud computing resources can be elastically allocated to scale instantly with the demand, and they can be utilized flexibly. The contemporary IaaS, SaaS, and PaaS in the cloud are user-friendly, and the consoles can be easily accessed via web browsers.

• **Scalability**: The rapid online analysis of real-time threat data streams can be realized by crafting the scalable processing functionalities of big data tools, whose scalability ensures the computational capability and storage capacity of cloud computing can be scaled simultaneously with the increasing amount of threat data. Therefore this is a practical solution of CTI in the big data environment.

• **Robustness**: The combination of cloud computing and big data tools facilitates the resilient and reliable sharing of the threat information. Occasionally, the threat information sharing system may encounter an abrupt increase of user accesses due to the burst of new malware. High-throughput cloud computing services can address this kind of issues related to data communications. The big data tools such as Hadoop and Spark also have their particular fault-tolerance schemes to handle errors encountered when processing threat data streams. Moreover, employing services in the cloud can avoid all of the misuses as when operating physical machines, for cloud computing services are mainly accessible via secured APIs.

5.4 System design specifications

Figure 5.2 sketches the framework of a scalable real-time threat information sharing system in the cloud. The system consists of five main units and four groups of security

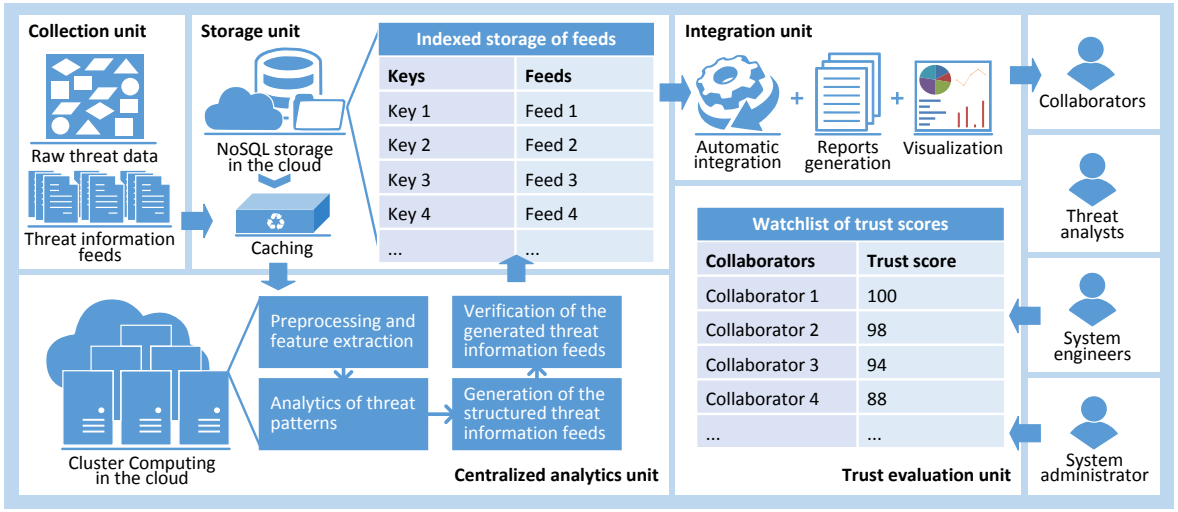

Figure 5.2 : The scalable threat information sharing system in the cloud.

personnel shown in figure 5.3. Each of the five main units achieves a major and particular feature and cooperates with other units simultaneously:

• **Trust evaluation unit**: This unit evaluates the trustworthiness of each collaborator by maintaining a watchlist of trust scores for all collaborators.

• **Threat data collection unit**: This unit performs the gathering of raw threat data from heterogeneous sources; this unit also collects structured threat information feeds submitted by collaborators.

• **Centralized analytics unit**: This unit performs the centralized preprocessing and analytics of the threat data; and the generation, interpretation, and investigation of the structured threat information feeds. Ideally, the centralized analytics unit should be able to perform deep packet inspection (DPI) and APT analysis.

• **Storage unit**: This unit performs caching of the collected threat data; this unit also performs indexed and searchable storage of the generated standardized threat information feeds.

• **Integration unit**: This unit achieves the automatic and secure sharing, distribution, and integration of the structured threat information feeds for collaborators. A user-friendly API is equipped for the enhancement of this unit.

Four groups of security personnel related to the system are described below:

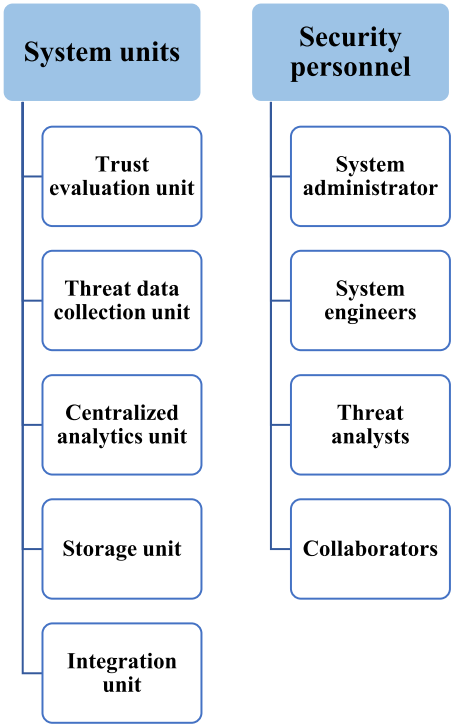

Figure 5.3 : Five system units and four groups of security personnel.

• **System administrator**: The system administrator is the chief operator of the system, who has the highest system authority and supervises all of the system operations. The system administrator should be the organization with the highest trust evaluation such as the government.

• **System engineers**: The system engineers mainly focus on hardware. They monitor and guarantee the normal system operations, and maintain the watchlist of the trusted cooperators.

• **Threat analysts**: The threat analysts design the algorithms and techniques to process the threat data. They should keep informed of emerging security incidents, discover the most vulnerable incidents, and provide broader views of specific attacks. They also make adjustments to the generated structured threat information feeds.

• **Collaborators**: The collaborators can contribute to the system by submitting raw threat data or standardized threat information, and they can consume generated threat information feeds.

5.4.1 Trust evaluation

The watchlist of the trusted collaborators is maintained by the trust evaluation unit; and supervised by the system engineers and the system administrator. Only the collaborators in the watchlist can submit and consume the threat information feeds. Each of the collaborators in the watchlist owns its trust score, which is used to judge whether a collaborator should be kept in the watchlist or removed from it. The trust score of a collaborator is judged by multiple factors, such as the quantity and quality of the collaborator's threat information feeds submitted to the system and other collaborators' remarks. The collaborators are encouraged to submit more high-quality threat information feeds to get higher trust scores. The collaborators with higher trust scores can consume the more sensitive threat information feeds from the system and assist the system operations.

5.4.2 Threat data collection and caching

The threat data is mainly gathered by the collection unit from two categories of sources:

i. Structured threat information feeds submitted by collaborators. The threat information feeds for submission are recommended to be organized into a standardized structure such as STIX to save the computational resources of the centralized analytics unit. Each feed contains the name and ID of the threat information and the content. An application programming interface (API) is provided to collaborators for the submission of their threat information feeds. The collection unit classifies the submitted threat information feeds into groups based on the names and IDs for further processing.

ii. Structured and unstructured raw threat data related to security incidents from heterogeneous sources and various IoT platforms [119], such as IP addresses, URLs, MD5 hashes, Internet files, web pages, vulnerabilities, images, network traffic data, the content of webmails and domain names. IP addresses that are related to hacking behaviors, URLs of websites that contain Trojans, or emails that have phishing content can be discovered by analyzing these threat data. Collected raw threat data are cached in the storage unit.

5.4.3 Centralized threat data analytics and generation of the threat information feeds

In the centralized analytics unit, the large-scale threat data that transferred from the collection unit are analyzed. The execution can be accelerated by the batch processing methods with executable scripts. Threat analysts design and implement the algorithms and techniques, and take the responsibility of regularly inspecting the quality and dependability of the intermediate datasets and the generated threat information feeds. As the threat data are categorized into "structured threat information feeds from collaborators" and "raw threat data from heterogeneous sources", they will be processed separately according to their categories:

i. For those structured threat information feeds submitted by collaborators, the centralized analytics unit will verify the quality, integrity, authenticity, and availability of the feeds, assisted by the threat analysts. Unqualified feeds will be discarded, and qualified feeds will be classified, organized, and passed to the storage unit for sharing. As the format of feeds should conform to one sharing standard such as STIX, those feeds that conform to other standards should be reformatted or restructured before the storage.

ii. For those raw threat data in inconsistent and diverse formats, the data have to be preprocessed, and important features have to be extracted. Thus the centralized analytics unit will apply modern data mining techniques to them. Automatic preprocessing and feature selection techniques will be utilized firstly, followed by the classification or clustering with machine learning models.

Raw threat data are mostly unstructured and usually have numerous flaws and duplication. Data preprocessing and feature extraction techniques are helpful to normalize and convert raw threat data into the formats that are machine-readable and can adapt to the machine learning models. After preprocessing and the selection of the representative and standardized features, the threat data are assumed to be flawless and consistent in the formats. These techniques have been realized in some of the latest big data tools. E.g., in Apache Spark, these techniques are realized with MLlib in a scalable manner and are categorized into three collections, namely, Extraction, Transformation, and Selection. Scalable feature extractors, transformers, and selectors

are applicable with Spark. E.g., TF-IDF and n-gram are applicable for the threat data in the text format, PCA is applicable for the threat data in the image format such as human face images.

Using machine learning techniques can achieve the more practical understanding of threat patterns. Machine learning models such as Bayesian networks, decision trees, k-nearest neighbor classification, k-means clustering, hidden Markov model (HMM), support vector machine (SVM), and artificial neural networks (ANN) can be used in this process. The latest Spark MLlib implements various machine learning algorithms as pipelines composed of DataFrames. A pipeline is a flow of directed stages. The DataFrames support many data types and can enable users to build and tune machine learning pipelines quickly. A generated threat information feed can be represented as a DataFrame at the final stage of a pipeline, and the DataFrame can contain multiple columns for the threat information feed, such as its name, ID, type, and description. The finally generated threat information feeds should be automatically integrated into the collaborators' defensive measures and platforms. Therefore, to ensure the feeds are understandable for both computer and threat analysts, the structure of the threat information feeds should be standardized by the centralized analytics unit for automatic and time-efficient sharing and integration. The format should conform to one sharing standard such as STIX, and the feeds will be passed to the storage unit for sharing. Each of the feeds contains the name and ID of the threat and the content. To ensure the best quality of the threat information feeds, the threat analysts can make auxiliary adjustments to them.

5.4.4 The indexed storage of the threat information feeds

The storage unit takes the responsibility to maintain an indexed and searchable database that contains the generated threat information feeds for sharing. As cyber attacks are kept being generated on a daily basis, the total amount of threat data is tremendous. Therefore, a massive amount of threat information feeds may be generated, imposing a heavy load on the traditional relational databases. Thus the latency can be high when a collaborator intends to search and integrate a threat information feed.

In this case, open source and distributed NoSQL databases such as Apache HBase

and Apache Cassandra are taken into consideration. These cloud-based NoSQL databases have their fault-tolerance measures and easy-to-use APIs and show the linear scalability, efficiency, and robustness when deployed on cloud computing infrastructures. They provide real-time queries based on the key-value pairs, and the values can be timely returned with the input keys. The keys can be the IDs of the threat information feeds, and the values can be the contents. The XML format is supported, which is suitable for some threat information languages such as OVAL. The database can be deployed on a distributed storage system such as HDFS for the permanent and fault-tolerant storage of the threat information feeds. The feeds are ranked and grouped in the database based on some criteria, for example:

- The generated threat information feeds have different levels of confidentiality. Only those collaborators who have higher trust scores can get access to the threat information feeds with high confidentiality levels.

- The threat information feeds for integration are classified in the database, e.g., feeds that are associated with Distributed Denial of Service (DDoS) attack and botnet can be generalized into the same group, and feeds that are related to spear phishing and watering hole attacks can be categorized into another group, so that collaborators can choose relevant category of threat information accordingly for easier integration.

5.4.5 The API for automatic and secure integration of the threat information feeds

An API with secure access enables the automatic integration of the latest, real-time and actionable threat information feeds into the collaborators' defensive systems to save the cost of human resources. The collaborators can get access to the complete series of available and searchable threat information feeds via the API. With the API, the system can enable customized integration services to highlight the most important threat information feeds and facilitate the optimized integration of the threat information, to guarantee that the feeds integrated into a particular collaborator's defensive system conform to the current and forthcoming threat situation of that collaborator.

Recommendations for the integration of the threat information feeds are provided according to the type of that collaborator, the statistics of threat information feeds that the collaborator integrated, and other information based on the footprints of

that collaborator in the sharing system. The recommendations can either be machine generated or provided by threat analysts. Whenever new security incidents occur, the system can also notify and warn the trusted collaborators in the watchlist through the API, for the resistance of future identified or zero-day attacks. Scalable data privacy preservation techniques [166] can be utilized in the sharing process and incorporated into the API to prevent eavesdropping during the communication of data streams on the Internet.

Currently, many CTI providers provide report and visualization services of the threat information via the APIs. A report is a detailed and comprehensive summarization of the threat information for a particular collaborator, and the visualization is a set of methods in a visual manner that helps human analysts to understand the text-based data easily. The visualization is a powerful combination of art, computer science, and statistics. Compared with detailed yet complicated reports, visualization can bring multi-scope views of the latest anomalies in the big data environment to the threat analysts. The collaborators can understand their potential threats confronted within the local area via a visualization of the time and locations of the occurred attacks. Therefore, the visualization is one of the essential parts of the sharing system.

5.4.6 Choosing the most valuable threat information feeds for integration

Although plenty of relevant threat information feeds are available for sharing in the database, not all of the threat information feeds provided are suitable for a particular collaborator. Some issues have risen during the process of sharing and integration:

- **The cost**: Some commercial CTI providers charge the collaborators for using their services, the integration of additional feeds may bring the extra cost for collaborators.

- **The latency**: It may be a repetitive and time-consuming process to choose the most appropriate feeds from the massive database.

- **The defensive outcome**: Collaborators may mainly concentrate on those popular yet unnecessary feeds for integration, but neglect the vital information for them conveyed by uncommon feeds.

Therefore, the collaborators should understand their defensive demands and choose the most valuable threat information feeds for integration to save the time and cost as

well as maximize the defensive outcome. The following aspects may be considered by the collaborators:

- The potential types of threats and attacks that they may encounter, according to the varieties of the IT systems and the vulnerabilities.
- The most vulnerable and private data that should be protected.
- The severity of financial or privacy loss that the collaborators may suffer if their systems were attacked.

For instance, if an enterprise possesses many private and worthy documents, then one possible threat is the encryption caused by the ransomware. Then the enterprise should focus more on the feeds that are linked to this threat. The origins of the threat information feeds should be marked by the system, thus via proper source investigations, the collaborators can know which source that each threat information feed comes from, and make proper responses. When it comes to the threat information conveyed by different feeds, possibly there are some overlaps and duplication. Therefore, even though the qualifications of the threat information feeds in the database have been checked by the system, the collaborators still have to analyze the dependability, novelty, quality, and integrity of the feeds.

5.5 Summary

In this chapter, we have briefly introduced cyber threat intelligence and the significance of the automated threat information sharing. We have introduced a scalable real-time threat information sharing system in the cloud, based on current researchers' and commercial companies' achievements. Fortunately, it is evident that various new standards are being released, and numerous researchers and commercial enterprises have been endeavoring to seek the approaches to incorporate these standards into their threat information sharing platforms to promote the automatic formation and integration of the threat information feeds. Therefore, the computational cost and human resources can be saved, and the defensive outcome can be maximized. However, the amount of available standards and platforms for threat information sharing is excessive, where a new dilemma has arisen. Within a particular community that utilizes

the same sharing standard, the exchange process may be efficient, yet the communications between different communities that utilize different sharing standards may be complicated. Therefore, we hope that in the future, the reformatting or restructuring methods between various standards should be enhanced to make the exchange processes more efficient, so that different communities can work together seamlessly as a broad community to defend cyber threats.

Chapter 6

A Private and Scalable Online Virus Detection System with Multiple Anti-virus Engines

6.1 Introduction

The battle between computer viruses and anti-virus systems has last for decades. Nowadays, the intrusive cyber behaviors have become more complicated and diversified in the current big data environment [112][153]. Computer systems are constantly confronted with new kinds of computer viruses, which have evolved to more sophisticated and stealthy forms to hide from the detection of anti-virus systems [110]. Anti-virus organizations currently may own a variety of advanced detection technologies and large-scale repositories of virus samples. However, for one particular anti-virus engine, false alarms or missed alarms during virus detection can still occur occasionally. Recently, some anti-virus systems are composed of multiple anti-virus engines. This kind of system allows users to upload suspicious samples online and then presents the users with multiple independent detection results. Via summarization and comparison of these results, both of the missed alarm rate and the false alarm rate can be reduced. VirusTotal [135] and VirSCAN [134] are two signature systems that provide multi-engine virus detection. But according to the information provided on the official websites of VirusTotal and VirSCAN, these systems may save user-uploaded suspicious samples and share them to the security enterprises that own the anti-virus engines. Currently, some anti-virus engines provide cloud-based anti-virus services, and user files may be uploaded to the cloud for analysis. These factors may cause some privacy-leakage problems [163][166][162], particularly for the samples with high confidentiality. Therefore, it is necessary to construct a multi-engine system that can provide accurate and private virus detection services to those users with high privacy requirements.

Motivated by this issue, in this chapter, we provide a private online virus detection system, which incorporates 32 anti-virus engines. Our contributions are: (1) The proposed system can perform the "isolated detection and update" of the anti-virus

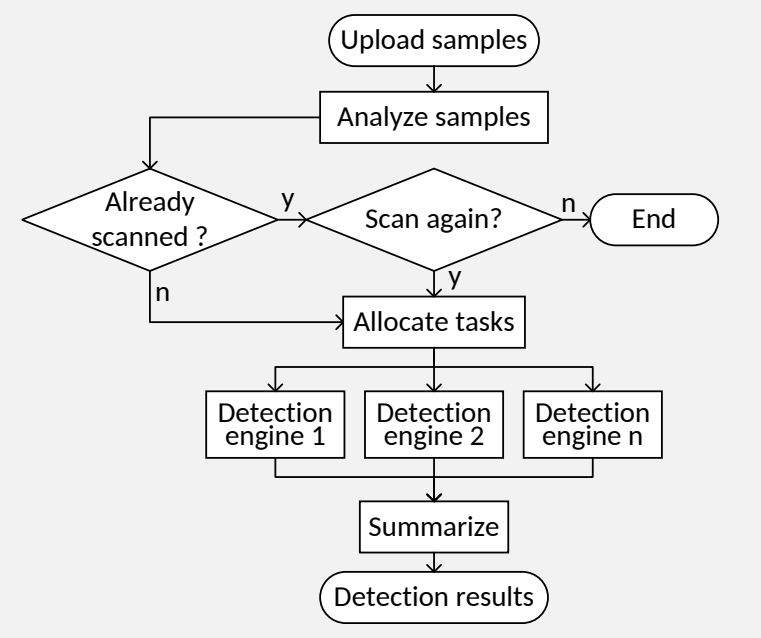

Figure 6.1 : The virus detection flowchart.

engines. This mechanism guarantees that the uploaded confidential samples are not exposed to the Internet, during either virus detection or system upgrade. (2) A web interface is provided so that the system users can upload suspicious samples via web browsers and the detection results from multiple anti-virus engines can be displayed on the web interface. (3) The low-coupling design of this system is scalable to support the distributed deployment mode.

The rest of this chapter is composed as follows. In section 6.2, we describe the virus detection process of the proposed system. In section 6.3, we introduce the system design specifications. In section 6.4, we present the system testing process. In section 6.5, we provide the summary of this chapter.

6.2 The virus detection process

The system performs the virus detection process according to the flowchart illustrated in figure 6.1. After a sample is uploaded, the system will perform an immediate analysis. If the sample has been scanned before, the system can inform the user and provide the relevant detection history; the user can also request the system to scan the sample again. For the virus detection, the SHA-1, SHA-256 and MD5 hash values of the sample are generated, then the detection tasks are created and allocated to multiple

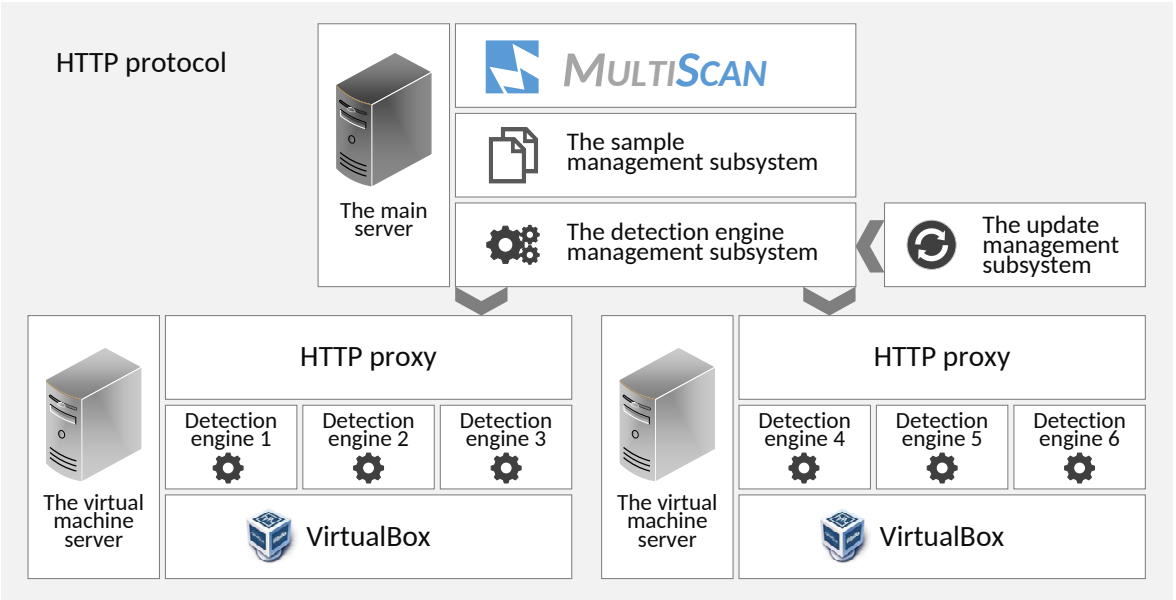

Figure 6.2 : The system architecture.

anti-virus engines based on their execution status. Finally, multiple detection results are summarized and displayed on the web interface.

6.3 System design specifications

In overall, the system is composed of five core modules, including the web interface, the sample management subsystem, the engine management subsystem, the packaging scripts, and the update management subsystem. The system architecture is depicted in figure 6.2. The system design is low-coupling, for the communications among the system modules are launched with HTTP. The system is scalable with this kind of design. Ubuntu Linux is the primary operational environment of the proposed system. Apache2 is utilized for building the web server, accompanied with Laravel PHP and MySQL. The open-source software VirtualBox is utilized for virtualization.

6.3.1 The web interface

The design style of the web interface is concise and similar to VirusTotal. Figure 6.3 represents the webpage for uploading suspicious samples. The web interface is a unified API of the system for the interactions with the system administrator, system users, and third-party applications. Compatibility is the main advantage of building the system interface on web browsers. The browser-server design ensures that the accesses of users

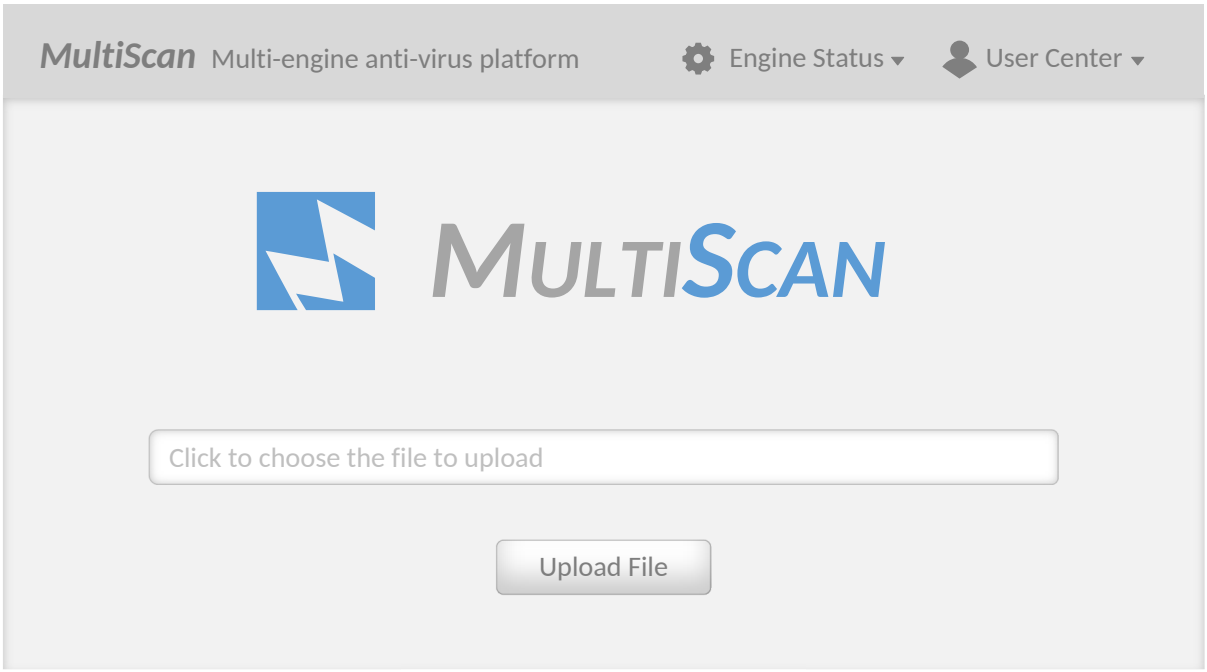

Figure 6.3 : The webpage for uploading suspicious samples.

are not limited by hardware and operating systems. Users can also get access to the system using mobile devices. E.g., if a user has an Android smartphone, the newly downloaded APK files can be uploaded to the system immediately for virus detection. As the web interface is the entry of the whole system and the control terminal for both of regular users and the system administrator, the relevant access control mechanism is essential. In the proposed system, the web interface is composed of multiple webpages, which have various functionalities. The accessible webpages for regular users and the system administrator are different. After logging into the system with username and password, on the upload page, a user can upload a single suspicious sample or multiple samples compressed in an archive. The detection results of multiple anti-virus engines can be summarized and displayed on the result page for the user. On the management pages of the web interface, the system administrator can check the system execution status and conduct user management. Table 6.1 describes the webpages of the interface and their functionalities.

JSON API is provided in the system to facilitate third-party applications uploading samples and retrieving detection results automatically. The information is transmitted in JSON format. As the proposed system utilizes RESTful API, third-party applications can upload samples, check the anti-virus engine status, and retrieve the detection

Table 6.1 : Webpages of the interface and their functionalities.

Class	**Webpage**	**Functionalities**	**User Types**
The webpages for virus detection	Home page	Introduction of the system.	All users
	Upload page	Upload suspicious samples.	Logged-in users, system administrator
	Result page	View the detection results of one user's uploaded samples.	The logged-in user who uploaded the samples, system administrator
The webpages for system and user management	User center page	Modify the password, renew the access token, etc.	Logged-in users, system administrator
	Upload history page	View all uploaded samples of one user and the relevant detection results.	Logged-in users, system administrator
	Full history page	View all uploaded samples of all users and the relevant detection results.	System administrator
	User management page	Add or delete users, modify the passwords of all users, etc.	System administrator
	System status page	Check the status of CPU, RAM, and network, etc.	System administrator
	Statistics page	Check the statistics, such as the number of samples uploaded daily; the average detection time of all anti-virus engines, etc.	System administrator

results. After logging in, on the user center page, a user can generate the access token required for JSON API access. The code below represents the status of anti-virus engines in JSON format.

```
{
"result": "success",
"engines": [
{
"name": "f-prot",
```

```
"full_name": "F-Prot",
"platform": "Windows",
"library_date": "01-05-2017",
"status": "online"
},
...
{
"name": "symantec",
"full_name": "Symantec Endpoint Protection",
"platform": "Windows",
"library_date": "02-05-2017",
"status": "online"
}]
}
```

The "engines" part represents the status of anti-virus engines in the system, including the name, the operational status (online, offline, or updating), the operating system, and the update date of virus library. The code below represents the standardized detection results returned in JSON format.

```
{
"results": {
"symantec": {
"result": "OK",
"full_name": "Symantec Endpoint Protection",
"library_date": "02-05-2017"
},
...
"f-prot": {
"result": "Virus",
"full_name": "F-Prot",
"library_date": "01-05-2017"
}
```

```
},
"total": 32,
"nothreatsNum": 20,
"errorNum": 0,
"detectedNum": 12
}
```


The "results" part represents the results of the anti-virus engines after a sample has been uploaded to the system. If a result is "OK", then the sample is benign for the relevant anti-virus engine; if a result is "TIMEOUT" or "ERROR", then the corresponding anti-virus engine encounters errors or malfunctions; if a result is "Virus", then the sample is malicious. The "nothreatsNum" represents the number of engines that report benign. The "errorNum" represents the number of engines that encounter errors or malfunctions. The "detectedNum" represents the number of engines that report malicious.

6.3.2 The sample management subsystem

The sample management subsystem manages the uploaded samples from users. After the samples are uploaded, the sample management subsystem will analyze them instantly. For an over-sized sample, the subsystem can prompt an error message on the web interface, informing the maximum permitted size. For an archive of samples, the subsystem can unzip it and create a batch task; if the archive contains an excessive number of samples, the subsystem can also prompt an error message on the web interface, informing the permitted number of maximum. If a sample has already been scanned, then the relevant detection history can be retrieved, or the sample can be re-scanned.

6.3.3 The engine management subsystem

The most unstable factor of the system is the condition of 32 anti-virus engines. Some of the anti-virus engines are free to use without commercial redistribution, while for some other engines the services have to be purchased. The anti-virus engines are developed by various companies, the functionalities are different, and they require different operational environments. Uncertain detection results may be generated due to

the instability of execution or update. Therefore, it is critical for the status monitoring and error handling of these anti-virus engines to make sure that they can execute and update efficiently and proper detection results can be returned.

Therefore, each of the 32 anti-virus engines is installed in an independent virtual machine and isolated with others so that they can execute independently and conflicts can be avoided. The system can still execute normally even if some anti-virus engines fail. The virtual machines can be distributed among multiple physical hosts, and thus the scalability can be achieved. Operating systems such as Windows XP 32bit, Windows Thin PC 32bit, CentOS 6.6 64bit are installed into the virtual machines, according to the anti-virus engines' requirements of operating systems. The 32 anti-virus engines are enclosed in packages with customized and executable scripts. The engine management subsystem manages these engines and communicates with the scripts with HTTP. Information such as the status of anti-virus engines and the versions of virus libraries are collected by this subsystem.

6.3.4 The packaging scripts

Composed with Python, the packaging scripts can monitor the anti-virus engines in real-time and report their status including any errors occurred to the engine management subsystem. The scripts can request virus detection tasks from the engine management subsystem regularly. When a detection task is allocated to one of the anti-virus engines, the script of that engine downloads the sample to the local virtual machine and calls the engine to scan. The detection results are collected using APIs of the anti-virus engines. The scripts can also update the anti-virus engines regularly. As the implementation methods are various for different anti-virus engines, customized scripts are developed and summarized in table 6.2.

Engine calling methods

• *Command line.* In Windows and Linux, some engines support command line interfaces for calling.

• *ShellMenu.* In Windows system resource manager, each file has its shell context menu, and an anti-virus engine often adds in its menu item. Windows provides calling functions to a file's shell context menu. Therefore, for some engines which do not

Table 6.2 : Packaging methods of 32 anti-virus engines.

Anti-virus Engines	**Operating System**	**Engine Calling Method**	**Result Acquisition Method**	**Library Update Date Acquisition Method**
Kingsoft, Rising	Windows	Command line	GUI software log	Configuration file
360	Windows	Command line	GUI software log	VDF file modification date
AVG, DrWeb	Windows	Command line	Detection report	Detection report
McAfee	Windows	Command line	Detection report	Configuration file
Norman	Windows	Command line	Detection report	VDF file modification date
BitDefender, eScan	Linux	Command line	Command line output	Command line output
Avast, ClamAV, Emsisoft, NOD32, F-Prot, F-Secure, IKARUS, Kaspersky	Windows	Command line	Command line output	Command line output
G-DATA	Windows	Command line	Command line output	Detection report
Comodo	Linux	Command line	Command line output	VDF file modification date
Microsoft	Windows	Command line	System event log	VDF file modification date
ZoneAlarm	Windows	Command line	Software log	VDF file modification date
Agnitum, Defenx	Windows	Trigger real-time protection	SQLite file	VDF file modification date
Avira	Windows	Trigger real-time protection	System event log	VDF file modification date
Symantec	Windows	Trigger real-time protection	System event log	Configuration file
K7, TrendMicro	Windows	Trigger real-time protection	Software log	VDF file modification date
Fortinet, Panda, Tencent	Windows	ShellMenu	GUI software log	VDF file modification date
Baidu	Windows	ShellMenu	GUI detection report	Configuration file
MalwareSecure	Windows	ShellMenu	SQLite file	VDF file modification date

support command line interfaces, the ShellMenu method is utilized for calling.

• *Trigger real-time protection.* Most of the engines have real-time protection mechanisms. Whenever a file is saved into a hard disk, it will trigger real-time protection. Therefore, for those engines that do not support the command line or ShellMenu methods, the real-time protection triggering method is utilized for calling.

Result acquisition methods

• *Command line output.* Most engines can return standardized detection results in command lines with the command line calling method.

• *System event log.* For some engines, the detection results are recorded into system event logs. This method is often applied to those engines that utilize the calling method of real-time protection triggering.

• *Software log.* Some engines save the detection results into software logs.

• *SQLite file.* Some engines save the detection results into SQLite databases.

• *Detection report.* Some engines generate the detection results as official detection reports.

• *GUI software log.* For some engines, instead of outputting text-based detection results directly, graphical windows are displayed for the results. For those engines, the scripts acquire the results by simulating manual manipulating processes. The acquired results are text-based software logs.

Virus library update methods

• *Automatic update.* Most engines can check the network connections and communicate with official update servers. When there are available update packages, the virus libraries will be automatically updated.

• *Command line calling.* Some engines have independent functions for calling to start the update processes.

• *Offline update package installation.* Due to the network conditions, the online update of some engines may fail. In this case, the relevant offline update packages need to be downloaded from relevant official websites.

Acquisition methods of the virus library update dates

• *Command line output.* Some engines have command line interfaces, from which the dates can be acquired.

• *Configuration file.* Some engines write the dates into configuration files.

• *Detection report.* Some engines record the dates into official detection reports.

• *VDF file modification date.* For those engines whose update dates cannot be acquired from above methods, the modification dates of VDF files are regarded as the relevant virus library update dates.

6.3.5 The update management subsystem

The technique of "isolated detection and update" guarantees that when scanning samples, the anti-virus engines are isolated from the Internet; and when updating, the anti-virus engines are isolated from the samples. In this case, the samples are isolated from the Internet at all times. Therefore, the privacy can be maintained. This technique is monitored by the update management subsystem, which controls the operational status of virtual machines, such as: (1) Boot up and shut down; (2) Network connection (Internet or host only); (3) Creation/deletion of snapshots. The detailed process of "isolated detection and update" is explained as follows:

i. The original virtual machines are "updaters", which have Internet connection. However, the samples are never uploaded to "updaters".

ii. Snapshots named "scanners" are created from the original virtual machines. The "scanners" have no Internet connection since their network configurations are set to "host only". During detection, all of the uploaded samples are only scanned by anti-virus engines in these "scanners".

iii. After the anti-virus engines in the "updaters" are updated, new "scanners" are created from these "updaters". The old "scanners" are then replaced by the new ones.

Figure 6.4 demonstrates the process of "isolated detection and update".

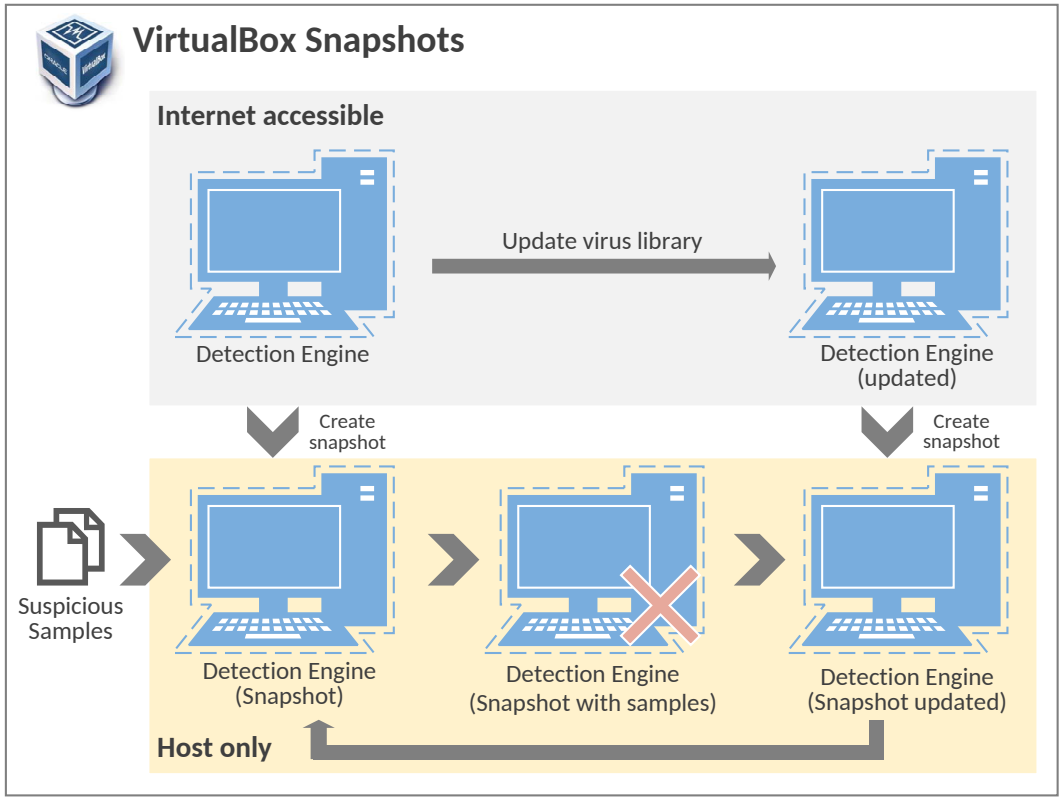

Figure 6.4 : Isolated detection and update.

6.4 System testing

The aim of testing is to guarantee that the system can work according to the requirements analysis. Via long-term testing, stability of the anti-virus engines can be enhanced. The hardware requirement of the system can also be obtained via testing; thus the recommended hardware configuration can be provided for deployment in real enterprises. The system should present acceptable performance of fault-tolerance and privacy-preservation, and it should be defensive when confronted with regular cyber attacks. We use unit testing and end-to-end testing to guarantee the testing comprehensiveness. We test the system under the following hardware environment:

- Server: *Lenovo ThinkServer TD340*
- CPU: *Xeon E5-2420 v2 2.2GHz 4 cores*2*
- RAM: *32GB DDR3 1600MHz*
- Hard-drive: *Samsung 256GB SSD*2; 1TB 7200rpm HDD*2*

Detection complete

MD5 Hash:	69630e4574ec6798239b091cda43dca0
SHA1 Hash:	cf8bd9dfddff007f75adf4c2be48005cea317c62
SHA256 Hash:	131f95c51cc819465fa1797f6ccacf9d494aaaff46fa3eac73ae63ffBdfd8267
Detection result:	24/30

Anti-virus engine	Detection result	Library date
K7	found the Trojan (000139291)	30-04-2017
Defenx	EICAR_Test_File	30-04-2017
IKARUS	EICAR_Test_File	30-04-2017
ZoneAlarm	EICAR_Test_File	03-05-2017
Emsisoft	EICAR_Test_File (not a virus) (B)	30-04-2017
Agnitum Outpost	EICAR_Test_File	29-04-2017

Figure 6.5 : The detection result of an uploaded sample.

For testing of the web interface, we have simulated user activities such as logging in, uploading samples, viewing the detection process, visiting the user center, and retrieving the detection history. The results have shown that the web interface layout is correct and the website can normally be accessed; all other functions can execute normally. We use Apache Benchmark to test the performance of web interface. The result has shown that the website is available for hundreds of user accesses simultaneously. Figure 6.5 represents a screenshot of the detection result page.

For testing of the sample management subsystem, we have uploaded repeated samples, over-sized samples, and archived samples to the system. The results have shown that the system can correctly handle these samples.

For testing of the engine management subsystem, we have designed multiple testing cases based on the anti-virus engines' status (online, updating, or offline). The results have shown that the subsystem can correctly obtain the anti-virus engines' status. Via collaboration with the sample management subsystem, the engine management subsystem can accurately assign detection tasks to the anti-virus engines, according to their status. Figure 6.6 represents the status of anti-virus engines collected by this subsystem.

Status of Engines

Engine Name	Operating System	Date of Virus Library	Status
360 Antivirus	Windows	20/04/2017	Online
Avira Server Security	Windows	26/04/2017	Online
Malware Secure	Windows	26/04/2017	Online
TrendMicro PC-cilin	Windows	24/04/2017	Online
Rising Antivirus	Windows	25/04/2017	Online
Microsoft Security Essentials	Windows	27/04/2017	Online
F-Prot	Windows	24/04/2017	Online
Baidu	Windows	22/04/2017	Online
Tencent	Windows	21/04/2017	Online
Fortinet	Windows	22/04/2017	Online
Norman Antivirus	Windows	22/04/2017	Online
Panda Antivirus	Windows	21/04/2017	Online
Avast	Windows	22/04/2017	Online

Figure 6.6 : The status of anti-virus engines.

For testing of the update management subsystem, a set of operations have been applied to the virtual machines, such as generation and deletion of virtual machine snapshots. The results have shown that the snapshots can be generated and deleted properly.

For testing of the virtual machines, anti-virus engines and their packaging scripts, a set of single samples and archives (ZIP or RAR) with more than one hundred samples are uploaded to the system for testing. We have uploaded benign files (such as Windows system files), EICAR (European Institute for Computer Antivirus Research) Standard Anti-Virus Test File, and other suspicious samples collected from the Internet to the system. The results have shown that the anti-virus engines can normally work in regular conditions. The scripts can detect errors of anti-virus engines and return the error messages to the engine management subsystem for troubleshooting. The average overall detection time for a single sample is 60 seconds, and 70 percent of the anti-virus engines can return the results in 30 seconds. For an archive with 119 samples, it takes 48 minutes to complete the task. The average detection speed is 24 seconds per sample. Figure 6.7 demonstrates the average detection time of 32 engines.

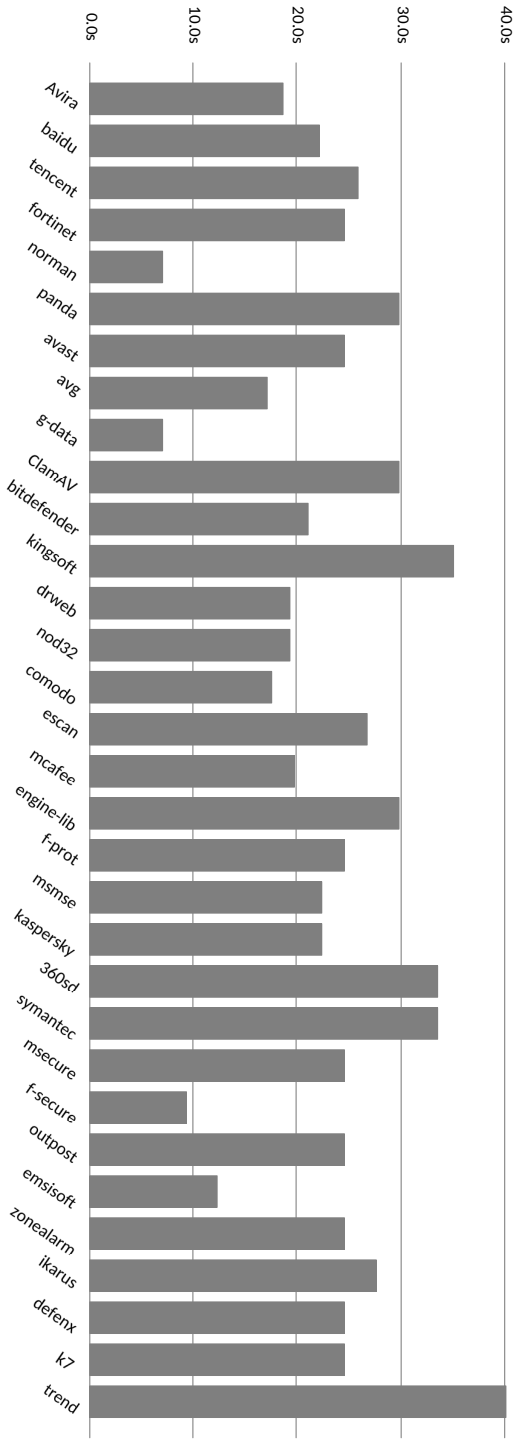

Figure 6.7 : The average detection time of detection engines.

For testing of the system security, we use Acunetix Web Vulnerability Scanner to scan the system to guarantee that the system has no high-risk vulnerabilities. Besides, we have audited all codes to avoid SQL injection and XSS vulnerabilities. We have simulated unauthorized accesses of attackers. We also have simulated XSS and SQL injection attacks to the system and tried to obtain the uploaded samples from the Internet. The results have shown that the system is defensive to these attacks.

According to the testing results, privacy of system users can be effectively enhanced with the "isolated detection and update". However, for the overall performance, the detection speed of our system is slower than VirusTotal and VirSCAN. The hardware environment for testing is limited. Another limitation is the speed of each engine. Various companies have different engine design. Some anti-virus engines use GUI interaction methods, which lower the speed. Therefore, the system performance can be improved in three aspects, i.e., the improvement of web interface; the upgrade of system hardware; and the selection of more stable enterprise-edition anti-virus engines.

6.5 Summary

In this chapter, we have proposed a multi-engine online virus detection system that can perform "isolated detection and update". This mechanism guarantees that the uploaded confidential samples are not exposed to the Internet. As the system can collect and store the suspicious samples, in the future, we will further improve the system functionalities so that the system can analyze the distribution patterns of different kinds of malicious samples. Moreover, as the low-coupling design of the system ensures scalability, we will modify and build the system among virtual machines in an IaaS cloud with cluster-based big data tools to further improve the system performance.

Chapter 7

Conclusion and Future Works

7.1 Summary of Contributions

This thesis has studied multiple aspects relating to host-based intrusion detection system (HIDS). First, this thesis has proposed a review of the development of HIDS with system calls. Second, this thesis provides inspirational future research trends in the current big data and cloud computing environment. Third, this thesis contributes to the community of HIDS by proposing a scalable HIDS approach using Spark in the Google cloud, endeavoring to improve the detection efficiency and the scalability for a new-generation system call-based HIDS. Fourth, this thesis has studied issues relating to threat intelligence information sharing, and has proposed a real-time scalable threat intelligence information sharing framework to enhance the collaborative security. Finally, this thesis has proposed a private online virus detection system, which incorporates multiple anti-virus engines. The system is applicable to be integrated into the forecast scalable real-time threat information sharing system.

Chapter 2 provides a review of host-based intrusion detection system with system calls, from the perspectives of its origin, algorithms, datasets and application areas. Instead of elaborating every detail, the main aim of this chapter is trying to provide researchers with a clear overview of the development of system call-based HIDS.

Chapter 3 provides the future research trends of host-based intrusion detection system with system calls. This chapter aims to inspire future researchers about the three trends of HIDS, namely, the reduction of false positive rate, the improvement of detection efficiency, and the enhancement of collaborative security. This chapter also proposes a real-time scalable HIDS framework with big data tools in cloud for a data center to enhance the collaborative security. The framework is comprised of three layers, namely, data collection layer, data analytics layer, and data storage layer. The framework is deployed in an open-source private cloud computing environment.

Chapter 4 proposes SCADS, a scalable approach using Spark in the Google cloud for host-based intrusion detection system with system calls. Based on the experimental results, the proposed framework with Apache Spark and Google cloud computing services can improve the detection efficiency of traditional system call-based HIDS. Besides the algorithms used in our experiments, other existing intrusion detection algorithms may also be incorporated into this proposed framework.

Chapter 5 briefly introduced cyber threat intelligence and the significance of automated threat information sharing. To design a comprehensive HIDS under the current sophisticated threat environment, traditional HIDS should combine with other security capabilities and the latest CTI. This chapter has introduced a scalable real-time CTI information sharing system in the cloud, based on current researchers' and commercial companies' achievements.

Chapter 6 proposed a multi-engine online virus detection system that can perform "isolated detection and update". This mechanism guarantees that the uploaded confidential samples are not exposed to the Internet. Meanwhile, a web interface is provided so that the system users can upload suspicious samples via web browsers and the detection results from multiple anti-virus engines can be displayed on the web interface. Furthermore, the low-coupling design of this system is scalable to support the distributed deployment mode.

7.2 Future works

Future works will concentrate on the three future research trends of HIDS, i.e., the reduction of false alarm rate, the improvement of detection efficiency, and the enhancement of collaborative security.

For the reduction of false alarm rate, future works will concentrate on the data preprocessing and feature extraction procedures. New datasets will be used for experiments and system call arguments will be used for modeling. Meanwhile, standardized evaluation metrics will be provided in every research work.

For the improvement of detection efficiency, future works will continue combining cloud computing and big data tools to improve the detection efficiency of HIDS. The DataFrame-based API in Spark MLlib will be used to construct and tune machine learning pipelines for HIDS.

For the enhancement of collaborative security, future works will integrate HIDS with other security mechanisms and the latest threat intelligence to design a complete threat-defensive infrastructure. The reformatting methods between various information sharing standards will be developed to make the exchange processes between different communities more efficient.

Bibliography

[1] M. Abdel-Azim, A. Abdel-Fatah, and M. Awad, "Performance analysis of artificial neural network intrusion detection systems," in *Electrical and Electronics Engineering, 2009. ELECO 2009. International Conference on.* IEEE, 2009, Conference Proceedings, pp. II–385–II–389.

[2] A. S. Abed, T. C. Clancy, and D. S. Levy, "Applying bag of system calls for anomalous behavior detection of applications in linux containers," in *Globecom Workshops (GC Wkshps), 2015 IEEE.* IEEE, 2015, pp. 1–5.

[3] M. Ahmed, A. Naser Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *Journal of Network and Computer Applications*, vol. 60, pp. 19–31, 2016.

[4] U. Ahmed and A. Masood, "Host based intrusion detection using rbf neural networks," in *Emerging Technologies, 2009. ICET 2009. International Conference on.* IEEE, 2009, Conference Proceedings, pp. 48–51.

[5] S. Alarifi and S. Wolthusen, "Anomaly detection for ephemeral cloud iaas virtual machines," in *International Conference on Network and System Security.* Springer, 2013, Conference Proceedings, pp. 321–335.

[6] S. S. Alarifi and S. D. Wolthusen, "Detecting anomalies in iaas environments through virtual machine host system call analysis," in *Internet Technology And Secured Transactions, 2012 International Conference for.* IEEE, 2012, Conference Proceedings, pp. 211–218.

[7] AlienVault, "Host-based intrusion detection system," https://www.alienvault.com/solutions/host-intrusion-detection-system, 2018.

[8] M. A. Ambusaidi, X. He, P. Nanda, and Z. Tan, "Building an intrusion detection system using a filter-based feature selection algorithm," *IEEE transactions on computers*, vol. 65, no. 10, pp. 2986–2998, 2016.

[9] A. Appleby, "Murmurhash," https://sites.google.com/site/murmurhash/, 2017.

[10] J. Arshad, P. Townend, and J. Xu, "A novel intrusion severity analysis approach for clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 416–428, 2013.

[11] C. Azad and V. K. Jha, "Data mining in intrusion detection: a comparative study of methods, types and data sets," *International Journal of Information Technology and Computer Science (IJITCS)*, vol. 5, no. 8, p. 75, 2013.

[12] S. Barnum, "Standardizing cyber threat intelligence information with the structured threat information expression (stix)," *MITRE Corporation*, vol. 11, pp. 1–22, 2012.

[13] V. K. Base, "esxtop," https://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1008205, 2017.

[14] T. Bläsing, L. Batyuk, A.-D. Schmidt, S. A. Camtepe, and S. Albayrak, "An android application sandbox system for suspicious software detection," in *Malicious and unwanted software (MALWARE), 2010 5th international conference on*. IEEE, 2010, pp. 55–62.

[15] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet mathematics*, vol. 1, no. 4, pp. 485–509, 2004.

[16] Bsd, "Kernel virtual machine," http://www.linux-kvm.org/page/Main_Page, 2017.

[17] E. W. Burger, M. D. Goodman, P. Kampanakis, and K. A. Zhu, "Taxonomy model for cyber threat intelligence information exchange technologies," in *Proceedings of the 2014 ACM Workshop on Information Sharing & Collaborative Security*. ACM, 2014, pp. 51–60.

[18] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "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, 2011, pp. 15–26.

[19] D. Canali, A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda, "A quantitative study of accuracy in system call-based malware detection," in *Proceedings of the 2012 International Symposium on Software Testing and Analysis*. ACM, 2012, pp. 122–132.

[20] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection," *ACM Computing Surveys*, vol. 41, no. 3, pp. 1–58, 2009.

[21] ——, "Anomaly detection for discrete sequences: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 5, pp. 823–839, 2012.

[22] Q. Chen, R. Luley, Q. Wu, M. Bishop, R. W. Linderman, and Q. Qiu, "Anrad: A neuromorphic anomaly detection framework for massive concurrent data streams," *IEEE Transactions on Neural Networks and Learning Systems*, 2017.

[23] Q. Chen, Q. Wu, M. Bishop, R. Linderman, and Q. Qiu, "Self-structured confabulation network for fast anomaly detection and reasoning," in *Neural Networks (IJCNN), 2015 International Joint Conference on*. IEEE, 2015, pp. 1–8.

[24] J. Connolly, M. Davidson, and C. Schmidt, "The trusted automated exchange of indicator information (taxii)," *The MITRE Corporation*, 2014.

[25] T. M. Corporation, "Common vulnerabilities and exposures," https://cve.mitre.org/, 2017.

[26] D. L. Costa, M. L. Collins, S. J. Perl, M. J. Albrethsen, G. J. Silowash, and D. L. Spooner, "An ontology for insider threat indicators development and applications," CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, Tech. Rep., 2014.

[27] G. Creech, "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, 2014.

[28] G. Creech and J. Hu, "Generation of a new ids test dataset: Time to retire the kdd collection," in *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*. IEEE, 2013, Conference Proceedings, pp. 4487–4492.

[29] ——, "Generation of a new ids test dataset: Time to retire the kdd collection," in *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*. IEEE, 2013, pp. 4487–4492.

[30] ——, "A semantic approach to host-based intrusion detection systems using contiguous and discontiguous system call patterns," *Computers, IEEE Transactions on*, vol. 63, no. 4, pp. 807–819, 2014.

[31] M. K. Daly, "Advanced persistent threat," *Usenix, Nov*, vol. 4, no. 4, pp. 2013–2016, 2009.

[32] A. Danesh, B. Moshiri, and O. Fatemi, "Improve text classification accuracy based on classifier fusion methods," in *Information Fusion, 2007 10th International Conference on*. IEEE, 2007, pp. 1–6.

[33] S. Das, Y. Liu, W. Zhang, and M. Chandramohan, "Semantics-based online malware detection: Towards efficient real-time protection against malware," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 2, pp. 289–302, 2016.

[34] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 233–240.

[35] R. I. Davis, B. C. Lovell, and T. Caelli, "Improved estimation of hidden markov model parameters from multiple observation sequences," in *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, vol. 2. IEEE, 2002, Conference Proceedings, pp. 168–171.

[36] T. F. Duffy, "Cybersecurity information sharing act of 2015," https://www.cisecurity.org/newsletter/cybersecurity-information-sharing-act-of-2015/, 2015.

[37] H. T. Elshoush and I. M. Osman, "Alert correlation in collaborative intelligent intrusion detection systems-a survey," *Applied Soft Computing*, vol. 11, no. 7, pp. 4349–4365, 2011.

[38] ENISA, "Detect, share, protect - solutions for improving threat data exchange among certs," https://www.enisa.europa.eu/, 2013.

[39] A. Feizollah, N. B. Anuar, R. Salleh, and A. W. A. Wahab, "A review on feature selection in mobile malware detection," *Digital Investigation*, vol. 13, pp. 22–37, 2015.

[40] S. Feldman, D. Stadther, and B. Wang, "Manilyzer: automated android malware detection through manifest analysis," in *Mobile Ad Hoc and Sensor Systems (MASS), 2014 IEEE 11th International Conference on*. IEEE, 2014, pp. 767–772.

[41] S. Fine, Y. Singer, and N. Tishby, "The hierarchical hidden markov model: Analysis and applications," *Machine learning*, vol. 32, no. 1, pp. 41–62, 1998.

[42] G. Fisk, C. Ardi, N. Pickett, J. Heidemann, M. Fisk, and C. Papadopoulos, "Privacy principles for sharing cyber security data," in *Security and Privacy Workshops (SPW), 2015 IEEE*. IEEE, 2015, pp. 193–197.

[43] S. Forrest, S. Hofmeyr, and A. Somayaji, "The evolution of system-call monitoring," in *Computer Security Applications Conference, 2008. ACSAC 2008. Annual*. IEEE, 2008, Conference Proceedings, pp. 418–430.

[44] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for unix processes," in *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*. IEEE, 1996, Conference Proceedings, pp. 120–128.

[45] A. O. Foundation, "Alluxio," http://www.alluxio.org/, 2017.

[46] A. S. Foundation, "Apache hadoop yarn," https://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/YARN.html, 2017.

[47] ——, "Apache kafka a distributed streaming platform," https://kafka.apache.org/, 2017.

[48] T. A. S. Foundation, "Apache flume," https://flume.apache.org/, 2017.

[49] ——, "Spark," http://spark.apache.org/, 2018.

[50] A. K. Ghosh, A. Schwartzbard, and M. Schatz, "Learning program behavior profiles for intrusion detection," in *Workshop on Intrusion Detection and Network Monitoring*, vol. 51462, 1999, Conference Proceedings, pp. 1–13.

[51] S. Godard, "mpstat," http://linuxcommand.org/man_pages/mpstat1.html, 2017.

[52] Y. Gu, W. Sheng, Y. Ou, M. Liu, and S. Zhang, "Human action recognition with contextual constraints using a rgb-d sensor," in *Robotics and Biomimetics (ROBIO), 2013 IEEE International Conference on.* IEEE, 2013, pp. 674–679.

[53] S. Gupta and P. Kumar, "An immediate system call sequence based approach for detecting malicious program executions in cloud environment," *Wireless Personal Communications*, vol. 81, no. 1, pp. 405–425, 2015.

[54] W. Haider, J. Hu, J. Slay, B. Turnbull, and Y. Xie, "Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling," *Journal of Network and Computer Applications*, 2017.

[55] W. Haider, G. Creech, Y. Xie, and J. Hu, "Windows based data sets for evaluation of robustness of host based intrusion detection systems (ids) to zero-day and stealth attacks," *Future Internet*, vol. 8, no. 3, p. 29, 2016.

[56] W. Haider, J. Hu, and M. Xie, "Towards reliable data feature retrieval and decision engine in host-based anomaly detection systems," in *Industrial Electronics and Applications (ICIEA), 2015 IEEE 10th Conference on.* IEEE, 2015, pp. 513–517.

[57] W. Haider, J. Hu, Y. Xie, X. Yu, and Q. Wu, "Detecting anomalous behavior in cloud servers by nested arc hidden semi-markov model with state summarization," *IEEE Transactions on Big Data*, 2017.

[58] W. Haider, J. Hu, X. Yu, and Y. Xie, "Integer data zero-watermark assisted system calls abstraction and normalization for host based anomaly detection systems," in *Cyber Security and Cloud Computing (CSCloud), 2015 IEEE 2nd International Conference on.* IEEE, 2015, pp. 349–355.

[59] F. F. Henry Ware, "vmstat," http://www.linuxcommand.org/man_pages/vmstat8.html, 2017.

[60] X. D. Hoang, J. Hu, and P. Bertok, "A multi-layer model for anomaly intrusion detection using program sequences of system calls," in *The 11th IEEE Interna-*

tional Conference on Networks, 2003. ICON2003. Citeseer, 2003, Conference Proceedings.

[61] ——, "A program-based anomaly intrusion detection scheme using multiple detection engines and fuzzy inference," *Journal of Network and Computer Applications*, vol. 32, no. 6, pp. 1219–1228, 2009.

[62] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[63] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," *Journal of computer security*, vol. 6, no. 3, pp. 151–180, 1998.

[64] V. Hristidis, Y. Hu, and P. G. Ipeirotis, "Ranked queries over sources with boolean query interfaces without ranking support," in *Data Engineering (ICDE), 2010 IEEE 26th International Conference on.* IEEE, 2010, pp. 872–875.

[65] J. Hu, "Host-based anomaly intrusion detection," *Handbook of Information and Communication Security*, pp. 235–255, 2010.

[66] J. Hu, X. Yu, D. Qiu, and H.-H. Chen, "A simple and efficient hidden markov model scheme for host-based anomaly intrusion detection," *Network, IEEE*, vol. 23, no. 1, pp. 42–47, 2009.

[67] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.

[68] M. Iannacone, S. Bohn, G. Nakamura, J. Gerth, K. Huffer, R. Bridges, E. Ferragut, and J. Goodall, "Developing an ontology for cyber security knowledge graphs," in *Proceedings of the 10th Annual Cyber and Information Security Research Conference.* ACM, 2015, p. 12.

[69] Ixia, "Perfectstorm," https://www.ixiacom.com/products/perfectstorm, 2017.

[70] G. K. Jayasinghe, J. S. Culpepper, and P. Bertok, "Efficient and effective realtime prediction of drive-by download attacks," *Journal of Network and Computer Applications*, vol. 38, pp. 135–149, 2014.

[71] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 675–678.

[72] G. Jiang, H. Chen, C. Ungureanu, and K. Yoshihira, "Multiresolution abnormal trace detection using varied-length n-grams and automata," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 37, no. 1, pp. 86–97, 2007.

[73] P. Kampanakis, "Security automation and threat information-sharing options," *IEEE Security & Privacy*, vol. 12, no. 5, pp. 42–51, 2014.

[74] P. Kaur and S. Mehta, "Resource provisioning and work flow scheduling in clouds using augmented shuffled frog leaping algorithm," *Journal of Parallel and Distributed Computing*, vol. 101, pp. 41–50, 2017.

[75] W. Khreich, E. Granger, R. Sabourin, and A. Miri, "Combining hidden markov models for improved anomaly detection," in *IEEE International Conference on Communications*, 2009, pp. 1–6.

[76] W. Khreich, B. Khosravifar, A. Hamou-Lhadj, and C. Talhi, "An anomaly detection system based on variable n-gram features and one-class svm," *Information and Software Technology*, vol. 91, pp. 186–197, 2017.

[77] W. Khreich, S. S. Murtaza, A. Hamou-Lhadj, and C. Talhi, "Combining heterogeneous anomaly detectors for improved software security," *Journal of Systems and Software*, 2017.

[78] A. Kovács and T. Szirányi, "Improved harris feature point set for orientation-sensitive urban-area detection in aerial images," *IEEE Geoscience and Remote Sensing Letters*, vol. 10, no. 4, pp. 796–800, 2013.

[79] S. Krishnan and J. L. U. Gonzalez, "Google compute engine," in *Building Your Next Big Thing with Google Cloud Platform*. Springer, 2015, pp. 53–81.

[80] M. Kulariya, P. Saraf, R. Ranjan, and G. P. Gupta, "Performance analysis of network intrusion detection schemes using apache spark," in *Communication and*

Signal Processing (ICCSP), 2016 International Conference on. IEEE, 2016, pp. 1973–1977.

[81] A. Laszka, W. Abbas, S. S. Sastry, Y. Vorobeychik, and X. Koutsoukos, "Optimal thresholds for intrusion detection systems," in *Proceedings of the Symposium and Bootcamp on the Science of Security.* ACM, 2016, pp. 72–81.

[82] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[83] W. Lee and S. J. Stolfo, "Data mining approaches for intrusion detection," in *Usenix security*, 1998, Conference Proceedings.

[84] W. Lee, S. J. Stolfo, and P. K. Chan, "Learning patterns from unix process execution traces for intrusion detection," in *AAAI Workshop on AI Approaches to Fraud Detection and Risk Management*, 1996, Conference Proceedings, pp. 50–56.

[85] W. Lee and D. Xiang, "Information-theoretic measures for anomaly detection," in *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on.* IEEE, 2001, Conference Proceedings, pp. 130–143.

[86] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of massive datasets.* Cambridge university press, 2014.

[87] Y. Liao and V. R. Vemuri, "Using text categorization techniques for intrusion detection," in *USENIX Security Symposium*, vol. 12, 2002, Conference Proceedings, pp. 51–59.

[88] P. Lichodzijewski, A. N. Zincir-Heywood, and M. I. Heywood, "Host-based intrusion detection using self-organizing maps," in *IEEE international joint conference on neural networks*, 2002, Conference Proceedings, pp. 1714–1719.

[89] F. Maggi, M. Matteucci, and S. Zanero, "Detecting intrusions through system call sequence and argument analysis," *Dependable and Secure Computing, IEEE Transactions on*, vol. 7, no. 4, pp. 381–395, 2010.

[90] M. V. Mahoney and P. K. Chan, "An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2003, Conference Proceedings, pp. 220–237.

[91] ——, "Learning rules for anomaly detection of hostile network traffic," in *Third IEEE International Conference on Data Mining, 2003. ICDM 2003.* IEEE, 2003, Conference Proceedings, p. 601.

[92] C. Marceau, "Characterizing the behavior of a program using multiple-length n-grams," in *Proceedings of the 2000 workshop on New security paradigms*. ACM, 2001, pp. 101–110.

[93] McAfee, "Mcafee host intrusion prevention for desktop," https://www.mcafee.com/uk/products/host-ips-for-desktop.aspx, 2018.

[94] J. McHugh, "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)*, vol. 3, no. 4, pp. 262–294, 2000.

[95] R. McMillan, "Definition: threat intelligence," *Gartner*, 2013.

[96] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *Journal of Machine Learning Research*, vol. 17, no. 34, pp. 1–7, 2016.

[97] Merriam-Webster, "Definition of intelligence," https://www.merriam-webster.com/dictionary/intelligence, 2017.

[98] E. Messina and D. Toscani, "Hidden markov models for scenario generation," *Ima Journal of Management Mathematics*, vol. volume 19, no. 4, pp. 379–401(23), 2008.

[99] A. Milenkoski, M. Vieira, S. Kounev, A. Avritzer, and B. D. Payne, "Evaluating computer intrusion detection systems: A survey of common practices," *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, p. 12, 2015.

[100] P. Moritz, R. Nishihara, I. Stoica, and M. I. Jordan, "Sparknet: Training deep networks in spark," *arXiv preprint arXiv:1511.06051*, 2015.

[101] S. S. Murtaza, A. Hamou-Lhadj, W. Khreich, and M. Couture, "Total ads: Automated software anomaly detection system," in *Source Code Analysis and Manipulation (SCAM), 2014 IEEE 14th International Working Conference on*. IEEE, 2014, pp. 83–88.

[102] S. S. Murtaza, W. Khreich, A. Hamou-Lhadj, and M. Couture, "A host-based anomaly detection approach by representing system calls as states of kernel modules," in *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*. IEEE, 2013, Conference Proceedings, pp. 431–440.

[103] S. S. Murtaza, W. Khreich, A. Hamou-Lhadj, and S. Gagnon, "A trace abstraction approach for host-based anomaly detection," in *Computational Intelligence for Security and Defense Applications (CISDA), 2015 IEEE Symposium on*. IEEE, 2015, Conference Proceedings, pp. 1–8.

[104] S. A. Musavi and M. Kharrazi, "Back to static analysis for kernel-level rootkit detection," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 9, pp. 1465–1476, 2014.

[105] D. Mutz, F. Valeur, G. Vigna, and C. Kruegel, "Anomalous system call detection," *ACM Transactions on Information and System Security (TISSEC)*, vol. 9, no. 1, pp. 61–93, 2006.

[106] M. Nauman, N. Azam, and J. Yao, "A three-way decision making approach to malware analysis using probabilistic rough sets," *Information Sciences*, vol. 374, pp. 193–209, 2016.

[107] U. of New Mexico, "Sequence-based intrusion detection," http://www.cs.unm.edu/~immsec/systemcalls.htm, 2017.

[108] OSSIM, "Alienvault ossim: The world's most widely used open source siem," https://www.alienvault.com/products/ossim, 2018.

[109] P. Pan, Z. Xu, Y. Yang, F. Wu, and Y. Zhuang, "Hierarchical recurrent neural encoder for video representation with application to captioning," *arXiv preprint arXiv:1511.03476*, 2015.

[110] J. Park and A. Tyagi, "Using power clues to hack iot devices: The power side channel provides for instruction-level disassembly." *IEEE Consumer Electronics Magazine*, vol. 6, no. 3, pp. 92–102, 2017.

[111] J. Pfoh, C. Schneider, and C. Eckert, "Nitro: Hardware-based system call tracing for virtual machines," in *International Workshop on Security*. Springer, 2011, Conference Proceedings, pp. 96–112.

[112] D. Puthal, S. P. Mohanty, P. Nanda, and U. Choppali, "Building security perimeters to protect network systems against cyber threats [future directions]," *IEEE Consumer Electronics Magazine*, vol. 6, no. 4, pp. 24–27, 2017.

[113] Y. Qiao, X. Xin, Y. Bin, and S. Ge, "Anomaly intrusion detection method based on hmm," *Electronics Letters*, vol. 38, no. 13, p. 1, 2002.

[114] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[115] Rapid7, "Metasploit," https://www.metasploit.com/, 2017.

[116] C. Sacca, S. Teso, M. Diligenti, and A. Passerini, "Improved multi-level protein–protein interaction prediction with semantic-based regularization," *BMC bioinformatics*, vol. 15, no. 1, p. 103, 2014.

[117] J. G. Shanahan and L. Dai, "Large scale distributed data science using apache spark," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 2323–2324.

[118] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*. Ieee, 2010, pp. 1–10.

[119] K. J. Singh and D. S. Kapoor, "Create your own internet of things: A survey of iot platforms." *IEEE Consumer Electronics Magazine*, vol. 6, no. 2, pp. 57–68, 2017.

[120] M. Solaimani, M. Iftekhar, L. Khan, and B. Thuraisingham, "Statistical technique for online anomaly detection using spark over heterogeneous data from multi-source vmware performance data," in *Big Data (Big Data), 2014 IEEE International Conference on.* IEEE, 2014, Conference Proceedings, pp. 1086–1094.

[121] M. Solaimani, M. Iftekhar, L. Khan, B. Thuraisingham, J. Ingram, and S. E. Seker, "Online anomaly detection for multi-source vmware using a distributed streaming framework," *Software: Practice and Experience*, 2016.

[122] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE symposium on security and privacy.* IEEE, 2010, Conference Proceedings, pp. 305–316.

[123] Spark, "Spark programming guides," https://spark.apache.org/docs/latest/index.html, 2018.

[124] X. Su, M. Chuah, and G. Tan, "Smartphone dual defense protection framework: Detecting malicious applications in android markets," in *Mobile Ad-hoc and Sensor Networks (MSN), 2012 Eighth International Conference on.* IEEE, 2012, pp. 153–160.

[125] M. Sugeno and T. Yasukawa, "A fuzzy-logic-based approach to qualitative modeling," *IEEE Transactions on fuzzy systems*, vol. 1, no. 1, pp. 7–31, 1993.

[126] K. M. Tan and R. A. Maxion, "" why 6?" defining the operational limits of stide, an anomaly-based intrusion detector," in *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on.* IEEE, 2002, Conference Proceedings, pp. 188–201.

[127] Z. Tan, A. Jamdagni, X. He, P. Nanda, and R. P. Liu, "A system for denial-of-service attack detection based on multivariate correlation analysis," *IEEE transactions on parallel and distributed systems*, vol. 25, no. 2, pp. 447–456, 2014.

[128] Z. Tan, A. Jamdagni, X. He, P. Nanda, R. P. Liu, and J. Hu, "Detection of denial-of-service attacks based on computer vision techniques," *IEEE transactions on computers*, vol. 64, no. 9, pp. 2519–2533, 2015.

[129] Z. Tan, U. T. Nagar, X. He, P. Nanda, R. P. Liu, S. Wang, and J. Hu, "Enhancing big data security with collaborative intrusion detection," *IEEE cloud computing*, vol. 1, no. 3, pp. 27–33, 2014.

[130] G. Tandon, "Machine learning for host-based anomaly detection," Thesis, 2008.

[131] O. P. Team, "Ossec open source hids security," http://ossec.github.io/, 2017.

[132] J. Thomas, C. Rose, and F. Charpillet, "A multi-hmm approach to ecg segmentation," in *Tools with Artificial Intelligence, 2006. ICTAI'06. 18th IEEE International Conference on.* IEEE, 2006, pp. 609–616.

[133] E. Vasilomanolakis, S. Karuppayah, M. Muhlhauser, and M. Fischer, "Taxonomy and survey of collaborative intrusion detection," *ACM Computing Surveys (CSUR)*, vol. 47, no. 4, p. 55, 2015.

[134] VirSCAN, "Virscan-on-line scan service," http://www.virscan.org, 2017.

[135] VirusTotal, "Virustotal-free online virus, malware and url scanner," https://www.virustotal.com, 2017.

[136] vmware, "vsphere guest sdk," https://www.vmware.com/support/developer/guest-sdk/, 2017.

[137] D. Wagner and P. Soto, "Mimicry attacks on host-based intrusion detection systems," in *Proceedings of the 9th ACM Conference on Computer and Communications Security.* ACM, 2002, Conference Proceedings, pp. 255–264.

[138] R. R. Walia, "Sequence-based prediction of rna-protein interactions," *Dissertations and Theses - Gradworks*, 2014.

[139] E. Walker, "Benchmarking amazon ec2 for high-performance scientific computing," *USENIX login*, vol. 33, no. 5, pp. 18–23, 2008.

[140] K. Wang, J. J. Parekh, and S. J. Stolfo, "Anagram: A content anomaly detector resistant to mimicry attack," in *International Workshop on Recent Advances in Intrusion Detection, 2006.* Springer, 2006, Conference Proceedings, pp. 226–248.

[141] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting intrusions using system calls: Alternative data models," in *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on.* IEEE, 1999, Conference Proceedings, pp. 133–145.

[142] M. R. Watson, A. K. Marnerides, A. Mauthe, and D. Hutchison, "Malware detection in cloud computing infrastructures," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 2, pp. 192–205, 2016.

[143] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning," in *Proceedings of the 26th Annual International Conference on Machine Learning.* ACM, 2009, pp. 1113–1120.

[144] A. Wespi, M. Dacier, and H. Debar, "Intrusion detection using variable-length audit trail patterns," in *International Workshop on Recent Advances in Intrusion Detection.* Springer, 2000, pp. 110–129.

[145] X. Xiao, Z. Wang, Q. Li, Q. Li, and Y. Jiang, "Anns on co-occurrence matrices for mobile malware detection," *KSII Transactions on Internet and Information Systems (TIIS)*, vol. 9, no. 7, pp. 2736–2754, 2015.

[146] M. Xie and J. Hu, "Evaluating host-based anomaly detection systems: A preliminary analysis of adfa-ld," in *Image and Signal Processing (CISP), 2013 6th International Congress on*, vol. 3. IEEE, 2013, pp. 1711–1716.

[147] M. Xie, J. Hu, and J. Slay, "Evaluating host-based anomaly detection systems: Application of the one-class svm algorithm to adfa-ld," in *Fuzzy Systems and Knowledge Discovery (FSKD), 2014 11th International Conference on.* IEEE, 2014, pp. 978–982.

[148] M. Xie, J. Hu, X. Yu, and E. Chang, "Evaluating host-based anomaly detection systems: Application of the frequency-based algorithms to adfa-ld," in *International Conference on Network and System Security.* Springer, 2014, pp. 542–549.

[149] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," *arXiv preprint arXiv:1502.03044*, vol. 2, no. 3, p. 5, 2015.

[150] L. Xu, D. Zhang, M. A. Alvarez, J. A. Morales, X. Ma, and J. Cavazos, "Dynamic android malware classification using graph-based representations," in *Cyber Security and Cloud Computing (CSCloud), 2016 IEEE 3rd International Conference on.* IEEE, 2016, pp. 220–231.

[151] C. Yang and J. Chen, "A scalable data chunk similarity based compression approach for efficient big sensing data processing on cloud," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 6, pp. 1144–1157, 2017.

[152] C. Yang, C. Liu, X. Zhang, S. Nepal, and J. Chen, "A time efficient approach for detecting errors in big sensor data on cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 2, pp. 329–339, 2015.

[153] C. Yang, D. Puthal, S. P. Mohanty, and E. Kougianos, "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*, vol. 6, no. 4, pp. 48–56, 2017.

[154] C. Yang, X. Zhang, C. Zhong, C. Liu, J. Pei, K. Ramamohanarao, and J. Chen, "A spatiotemporal compression based approach for efficient big data processing on cloud," *Journal of Computer and System Sciences*, vol. 80, no. 8, pp. 1563–1583, 2014.

[155] Z. Yang, Y. Yuan, Y. Wu, R. Salakhutdinov, and W. W. Cohen, "Encode, review, and decode: Reviewer module for caption generation," *arXiv preprint arXiv:1605.07912*, 2016.

[156] Q. Ye, X. Wu, and B. Yan, "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, 2010, Conference Proceedings, pp. 1–4.

[157] D.-Y. Yeung and Y. Ding, "Host-based intrusion detection using dynamic and static behavioral models," *Pattern recognition*, vol. 36, no. 1, pp. 229–243, 2003.

[158] D. Yuxin, Y. Xuebing, Z. Di, D. Li, and A. Zhanchao, "Feature representation and selection in malicious code detection methods based on static system calls," *Computers & Security*, vol. 30, no. 6, pp. 514–524, 2011.

[159] M. Zaharia, *An architecture for fast and general data processing on large clusters*. Morgan & Claypool, 2016.

[160] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "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, 2012, Conference Proceedings, pp. 2–2.

[161] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: Fault-tolerant streaming computation at scale," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013, Conference Proceedings, pp. 423–438.

[162] X. Zhang, W. Dou, J. Pei, S. Nepal, C. Yang, C. Liu, and J. Chen, "Proximity-aware local-recoding anonymization with mapreduce for scalable big data privacy preservation in cloud," *IEEE transactions on computers*, vol. 64, no. 8, pp. 2293–2307, 2015.

[163] X. Zhang, C. Liu, S. Nepal, S. Pandey, and J. Chen, "A privacy leakage upper bound constraint-based approach for cost-effective privacy preserving of intermediate data sets in cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1192–1202, 2013.

[164] X. Zhang, C. Liu, S. Nepal, C. Yang, and J. Chen, "Privacy preservation over big data in cloud systems," in *Security, Privacy and Trust in Cloud Systems*. Springer, 2014, pp. 239–257.

[165] X. Zhang, C. Liu, S. Nepal, C. Yang, W. Dou, and J. Chen, "A hybrid approach for scalable sub-tree anonymization over big data using mapreduce on cloud," *Journal of Computer and System Sciences*, vol. 80, no. 5, pp. 1008–1020, 2014.

[166] X. Zhang, L. T. Yang, C. Liu, and J. Chen, "A scalable two-phase top-down specialization approach for data anonymization using mapreduce on cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 2, pp. 363–373, 2014.

[167] R. Zuech, T. M. Khoshgoftaar, and R. Wald, "Intrusion detection and big heterogeneous data: a survey," *Journal of Big Data*, vol. 2, no. 1, p. 1, 2015.