Future Generation Computer Systems 117 (2021) 299–320

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Detecting impersonation attacks in cloud computing environments using a centric user profiling approach

Hisham A. Kholidy

The State University of New York (SUNY) Polytechnic Institute, College of Engineering, 100 Seymour Rd. Utica, NY 13502, United States

ARTICLE INFO

Article history:
Received 13 July 2019
Received in revised form 1 October 2020
Accepted 8 December 2020
Available online 13 December 2020

Keywords:
Impersonation detection
IDS
Audit correlation
Cloud computing
System calls
NetFlow
Feature extraction
Centric Profiling

ABSTRACT

Cloud computing has become the most needed technology for the IT industry. Impersonation attacks are among the most dangerous threats that Clouds face. In this paper, we present an approach to detect masquerade attacks in Clouds. The efficient detection of these attacks should correlate user behaviors in distinct environments and also should apply to several deployment models. We present and evaluate three approaches to detect Impersonation and masquerade attacks. The first approach analyzes sequences of correlated system calls from the VMs operating systems, while the second analyzes the NetFlow data from the network environment. The third approach integrates these two approaches by using a neural network that will produce better detections than any of the first two approaches. To simplify the testing and the evaluation of the three methods, the Cloud Intrusion Detection Dataset (CIDD) is used as a source for cloud audits data. The evaluation has considered alternative deployment models through our two intrusion detection frameworks, CIDS and CIDS-VIRT. The paper also shows that the proposed detection approaches are more accurate and outperform the SWAD-MMD, a recent masquerade detection framework that works in the cloud computing systems. Furthermore, the paper details our experimental results and evaluates the computational performance and the detection accuracy of these approaches.

1. Introduction

In impersonation or masquerade attacks, an attacker illegally uses the system resources after violating the account of a legal user. This class of attacks becomes a big challenge in cloud systems because the massive amount of system resources, alternative deployment models, and the distribution of user audits and activities across several VMs with distinct environments, strongly increase the complexity of detection.

Masquerade attacks are of two types, insiders and outsiders attacks [1]. The Cloud Security Alliance (CSA) released top threats to cloud computing, describing twelve threat areas considered most important to organizations using cloud services, including malicious insiders [2]. According to Cloud Security Alliance, the overall complexity increases in cloud systems where a user profile has to correlate distinct user activities in several virtual machines, VMs [1]. Any approach to detect masquerades in clouds should correlate all activities of the same user in different environments; i.e., host and network environments, and should apply to alternative deployment options such as private, hybrid, and public clouds. Instead, most of the proposed approaches analyze user behaviors only in just one environment without correlating these behaviors [3–8]. Furthermore, they do not apply to different cloud systems and that severely limits their adoption in cloud systems. Very few works studied this problem from the cloud perspective because of the lack of a real-time cloud dataset that provides real-time audits for masquerade attacks, a survey about the current state of the art is depicted in Section 2. To the best of our knowledge, this paper is premier research in the field of masquerade threat detection in real-time cloud systems. In this paper, we present three different methods to detect masquerader attacks in cloud systems. These methods utilize our approach, Data-Driven Semi Global Alignment (DDSGA) [9,10] to detect masquerade attacks in the cloud environment. The key notion underlying DDSGA is the alignment of the sequences of user behaviors against the user profile. The misalignment areas are labeled as anomalous and the presence of several anomalous areas is a strong indicator of masquerade attacks. The first method applies DDSGA to sequences of system calls from the host environment and while the second method uses NetFlow audits collected from the network environment. The third method uses a neural network to integrate the outputs of the first two methods and statistical information associated with active session e.g., the login period, user's source IP address, and login failure actions. The system calls and NetFlow data used to tune and evaluate the presented methods are those available in our CIDD dataset [11]. CIDD is the first publicly cloud-based dataset that enables us

E-mail address: hisham.kholidy@sunypoly.edu.

https://doi.org/10.1016/j.future.2020.12.009
0167-739X/

to practically study this problem. DDSGA is applied to the user audits in CIDD in a fully functional cloud system according to the distributed architecture of CIDD. To detect anomalies in the host environment, the approach applies the analysis of the system calls because it can result in a high degree of information assurance. These calls reflect all system activities and their monitoring is tightly integrated with the operating system. In this way, the monitoring process becomes more tamper-resistant for malicious updates of the monitored information. Most of the current IDSs consider all system calls categories which may result in a slow detection process and a high false alarm rate with a low hit ratio. Our analysis extracts specific features from the system calls through our "Behaviors Triangle Model". This model is focused on calls related to file access and to process activities. These calls are essential and unavoidable for any user and they reflect any regularity of the user behavior in audit data more than other calls. This regularity of audit data simplifies the labeling of abnormal behavior. In the network environment, a NetFlow profile is built for each source IP address based upon sequences of network actions captured by sniffing tools. For each action, the sequence of destination IPs that the user accessed successfully together with the protocols used for each access are recorded. To efficiently cover attacks in distinct cloud deployment models and improve the scalability, we introduced the CIDS-VIR (the full Virtualized CIDS) [12] intrusion detection framework that improves both the detection rates when compared to our previous CIDS framework [13]. CIDS-VIRT includes 24 VMs and simulates real-time cloud systems. Using CIDS and CIDS-VIRT, we have evaluated our three alternative correlation models, Audit Exchange, Independent, and Centralized-Backup. These three models are used to analyze and correlate the behavior of the same user in distinct cloud nodes. Our experiments and evaluation show that the Independent model works much better with CIDS than the Audit Exchange. The Independent model results are accurate and have acceptable performance in small and private cloud networks. The new Centralized-Backup model is firstly introduced in the current paper to evaluate the proposed approaches in a large-scale cloud system and works efficiently with the CIDS-VIRT in large clouds such as public and hybrid ones.

In summary, based on the state of the art [9,14–16], the most important factors to be considered in detecting masquerading attacks in cloud systems are that (1) the normal access control mechanisms like fully homomorphic encryption and countermeasures for data leakage and account hijacking are not sufficient for detecting a masquerade attack in cloud systems since the attacker has already gained access to the legitimate user's account and he/she does not try to exploit the access privileges of the user. (2) the behavior of a cloud user should be captured and analyzed at several locations in the cloud system (various host VMs that the user uses and in the cloud network) to build a consistent profile for each user. The main contribution of this work is to address the masquerade detection problem in the cloud by considering the following: (1) Addressing the too many features issue in the cloud datasets which leads to many false positives using our Behaviors Triangle Model (BTM) that is developed to extract important features in the dataset. One of the main advantages of the BTM is that it builds a consistent profile for each user by extracting important features from specific system calls that are related to specific security events audits, that are shared among almost all the operating systems. (2) Developing a dynamic user profiling approach that correlates the activity audits of each user in various VMs that the user uses and in the network to build a consistent profile for each user. The resultant user network profile helps in detecting potential masqueraders whenever host audit data is not accessible or some restrictions apply. (3) Developing a neural network model to integrate the host-based and network-based correlation detection models to improve both the masquerade detection accuracy and coverage. (4) Tuning and testing our proposed approaches to small and big cloud deployment models. The remainder of this paper is organized as follows: Section 2 highlights the current state of the art and briefly highlights the CIDD dataset and the DDSGA approach. It also discusses in detail the CIDS-VIRT Frameworks. Section 3 discusses the detection of masquerade attacks based on system calls audits. At first, the section outlines the main features extracted from the system calls to build a consistent user profile. Then, it discusses how DDSGA can be adapted to the extracted features. Lastly, it evaluates the security efficiency and the computational performance of the three implementation models and the adaptation of DDSGA. Section 4 discusses the detection of the masquerade attacks based on NetFlow data. Then, it outlines the main features extracted from the NetFlow data and discusses how the DDSGA approach has been adapted to work with these features. This section also evaluates both the security efficiency and the computational performance of these methods. Section 5 discusses the proposed neural network model and ends with the performance evaluation of that model. Section 6 compares our masquerade detection models and a recent model called the Sliding Window-based Anomaly Detection using Maximum Mean Discrepancy or SWAD-MMD framework [16]. Finally, Section 7 draws some concluding remarks and outlines future work, and in last section highlights the acknowledgment.

2. State of the art and related work

2.1. State of the art

Authors in [14] use a sequential rule mining algorithm to learn the behavior pattern of the user to build user profiles and it uses a matching algorithm to match the historical behavior of the user with the current behavior to detect a malicious user in the system. However, there are specific problems related to the application of sequential rule mining such as (1) it has too many parameters for somebody non-expert in data mining and (2) the obtained rules are far too many and most of them are non-interesting and with low comprehensibility. Furthermore, (3) this algorithm restricts ordering between items, because of which several rules may represent the same situation, similar rules are rated very differently and, rules may be too specific and less likely to be useful, sometimes none of the rules would match the new sequence. Authors in [17] proposed a combined attack-tree and kill-chain based method for identifying multiple indirect detection measures. Specifically, they use the attack trees to encapsulate all detection opportunities for masquerade attacks in cloud-service environments. However, their approach is not applicable in the real-time application of the cloud system because usually the fluctuation of the cloud users' audits is very high and the regularity of the user sessions is very low. This will result in large attack trees with a huge amount of data to be processed. Furthermore, the kill-chain and attack trees must be updated with all masqueraders' possible strategies that are frequently changed. Authors in [15] discuss how load balancing across availability zones may increase insider threat. They deal with insider threats in cloud relational database systems by revealing the flaws in cloud computing that insiders may use to launch attacks. They also proposed four models to detect insiders namely, the peer-to-peer model, centralized model, Mobile Knowledge-bases model, and Guided Mobile Knowledge bases model. They tested these models using a Cloud Simulator [18]. However, these models were not tested using real cloud datasets and are not suitable for real cloud deployment because they do not consider the correlation of the profiles of user behaviors at a different location

in the cloud. Authors in [19] proposed a model that consists of two policies for access control in Cloud computing namely, policy decision points (PDPs) and policy enforcement points (PEPs) is used. The model uses PEP-side caching to increase the availability and reduce the processing overhead on PDP. Authors show that using PEP-side caching can be exploited by insiders to bypass cloud access control mechanisms, which increases insider threat in cloud computing. To overcome this problem, the proposed manageable model detects and prevents insider threats at the PEP side with minimum overhead on the performance of PEP and PDP. Yet, the practical deployment of this model does not fit the real cloud systems and was not tested using a real cloud dataset. As apparent from the above survey, the state-of-the-art on anomaly detection does not address the detection of anomalies in a cloud collaboration system, by correlating user profiles at different locations of the cloud system. Additionally, researchers have utilized parametric and non-parametric strategies in various spaces, in any case, no work that has used real cloud datasets as a part of detecting anomalous insider activities. In [10] authors proposed a dynamic user profiling system for cloud users to detect masquerade attacks in the cloud computing systems. The User Profiling System monitors the user's behavior looking for the divergence of behavior from normal and analyzes the behavior of the users using the soft computing technique of neural networks and fuzzy logic. Recently, authors in [16] proposed an anomaly detection algorithm SWAD-MMD, Sliding Window-based Anomaly Detection using Maximum Mean Discrepancy, to detect anomalous insider requests using a non-parametric statistical technique called graph-based Maximum Mean Discrepancy. The proposed approach detects anomalous insider activities via an access network of users and objects. The MMD measures the distance between mean embedding of distributions into a Reproducing Kernel Hilbert Space (RKHS). The authors validated the proposed model using two publicly available datasets from Wikipedia [20] and present a performance evaluation in terms of the accuracy of the proposed model. Despite the detection accuracy of this algorithm in small datasets, the MMD has several drawbacks. For instance, it has a kernel bandwidth parameter that needs tuning [21], and the kernel can saturate so that the gradient vanishes [22] in a deep generative model. Furthermore, to have a reliable estimate of the distance, the mini-batch size must be large, e.g., 1000, which slows down the training by stochastic gradient descent [23]. However, we will compare our approach against the SWAD-MMD because it is the most recent and accurate approach that has been applied to a cloud collaboration system with a real cloud dataset.

2.2. Related work

In this section, we briefly describe the dataset that has been used to evaluate the proposed alternative detection and correlation models. Furthermore, we highlight the DDSGA approach and the security framework that underly the proposed solutions.

2.2.1. The Cloud Intrusion Detection Dataset (CIDD)

The lack of a public cloud dataset with a masquerade attack limited the study of this problem. The current public datasets, e.g., SEA [24], Greenberg [25], and Purdue University dataset [26] are not suitable to train and evaluate cloud IDSs, because they neglect the typical behaviors of a cloud user and cover audits in specific environments but do not correlate user behaviors in the host and network environments. Instead, CIDD [8] is one of the few datasets that can support the evaluation of a behavior-based IDS for a cloud because it integrates audit data associated with the user behaviors in both host and network environments and from different operating systems. CIDD audit data consists of two parts. In turn, each part is partitioned into training and test data. The first part is a collection of Unix Solari's audits and their corresponding TCP dump data. The second part includes Windows audits and their corresponding TCP dump data. The data include more than one hundred instances of attacks classified into several categories such as Denial Of Service (DOS), User to Root (U2R), remote to the user, surveillance probing, anomaly, real masquerade, and data attacks.

2.2.2. The Data-Driven Semi Global Alignment (DDSGA)

DDSGA [7] is a masquerade detection approach to improve both the computational and security efficiency of the Semi-Global Alignment (SGA) approach [3]. It can tolerate small mutations in the user sequences with small changes in the low-level representation of user behaviors. As shown in Fig. 1, DDSGA aligns the user active session sequences to earlier sequences of the same user and labels the misalignment areas as anomalous. A masquerade attack is signaled if the percentage of anomalous areas is larger than a dynamic threshold paired with each user. DDSGA can be decomposed into three phases: configuration, detection, and update, see Fig. 2. The configuration phase computes, for each user, the alignment parameters for both the detection and update phases i.e., gap and mismatch penalties, match score, and the threshold value. This phase is implemented by 5 modules namely, initialization, user's lexicon categorization, scoring parameters, average threshold, and maximum test-gaps module. The detection phase aligns the user's current session with the stored signature sequence. To improve the performance of this phase, two implementations are possible: Top-Matching Based Overlapping (TMBO) and the parallelized one [7]. The DDSGA update phase extends to both the user signatures and the user lexicon list with the new patterns. DDSGA also runs a long term update module to reconfigure the system parameters.

2.2.3. The CIDS-VIRT Framework and its Testbed

CIDS-VIRT [12,27–32] is a framework for intrusion detection that provides a defense strategy that deals with attacks against the most widely used cloud services: SaaS, PaaS, and IaaS. We primarily developed the CIDS-VIRT to work as a defense strategy for several cloud deployment models i.e., private, public and hybrid clouds. We have augmented the original CIDS framework [13] to develop the fully virtualized version CIDS-VIRT. The CIDS-VIRT framework offers a security solution for large cloud systems e.g., public and hybrid clouds because its scalability is much better than that of CIDS. Furthermore, CIDS-VIRT can be configured and managed in a simpler way than CIDS, because the administrators can access a central system backed up to other servers. This is important in public and hybrid cloud where the providers need to deploy and monitor the security solutions in a flexible way due to a large number of users. Even if CIDS-VIRT works with any deployment model because of its scalability and controllability, it is preferred to use it with public and hybrid clouds while the original CIDS is preferred for the private clouds. The first reason for this strategy is to get the benefit of the high performance and low network overhead of the Independent model that works with the original CIDS but not with CIDS-VIRT that is centralized. A second reason is that CIDS-VIRT does not optimize resource utilization because it introduces some management VMs to back up and reduce the computational overhead in the active one. This may not be acceptable in a small or private cloud. CIDS-VIRT improves the scalability of CIDS and achieves a reasonable performance in large clouds. We implemented three correlation models for this framework, the Independent, Audit Exchange, and Centralized-Backup Models. These three models are introduced

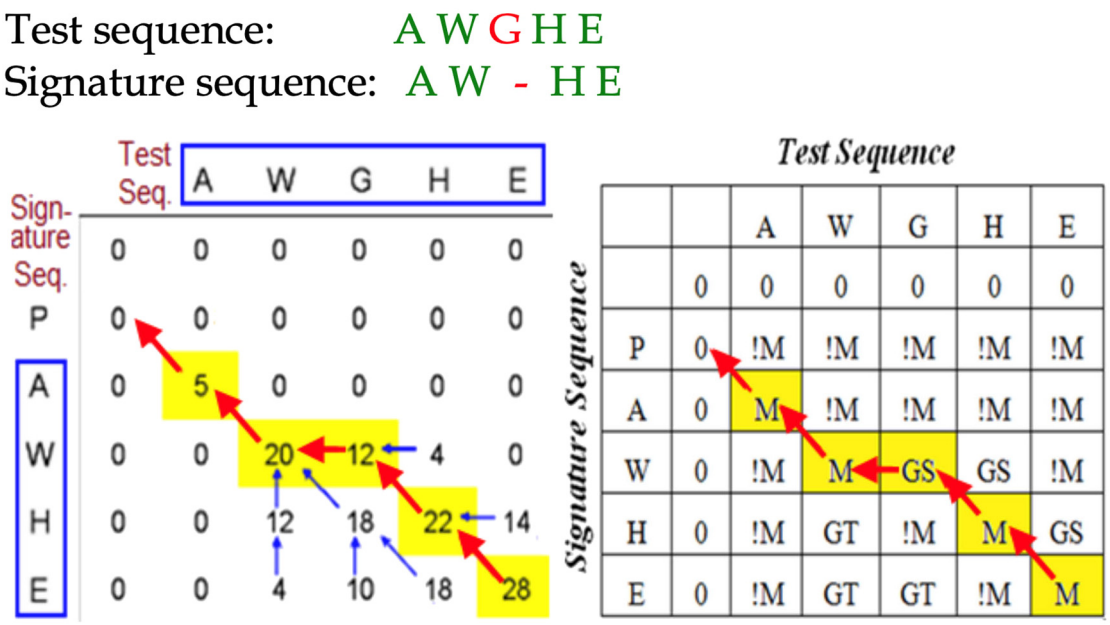

Fig. 1. An alignment example using DDSGA.

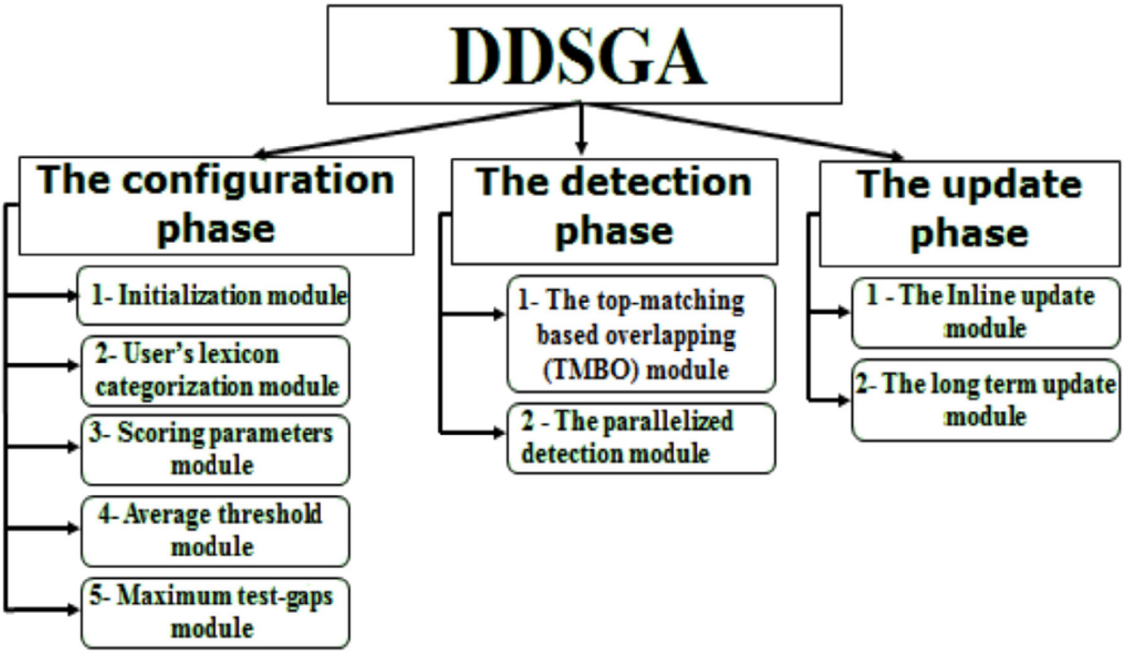

Fig. 2. The DDSGA three phases.

and evaluated in Section 3. In the following, we briefly describe the main components of the framework, see Fig. 3.

– Event collector: it collects logs, audit data, and sequence of user actions from both the HIDS sensor and the guest operating system. It also selects the most suitable management VM to analyze these audits and events.

– Event Correlator: it correlates the user logs and signatures collected from several sources according to the start and end time of the session and the source IP address of the user. Then, it sends a final list of network and VMs environments events to the event DB. This helps in correlating the normal behaviors of the same user in several VMs to detect a suspected behavior distributed among these VMs. Since a behavior deviation in one VM can be normal in another one, this reduces the false alarms rate in a cloud where a user may have different behaviors in several VMs.

– Behavior-based and Knowledge-based databases: The former is a profile history database for the behavior of cloud users. The latter stores a set of rules and signatures that describes known attacks.

– DDSGA Analyzer: it applies DDSGA to analyze user behaviors, e.g. sequence of commands or actions, collected from several sources. Whenever it detects a masquerade attack, it alerts the summarizer and reporter component.

HIDS Analyzer: it analyzes user behaviors to detect known trails left by attacks or predefined sequences of user actions that might represent an attack and is implemented by the OSSEC IDS

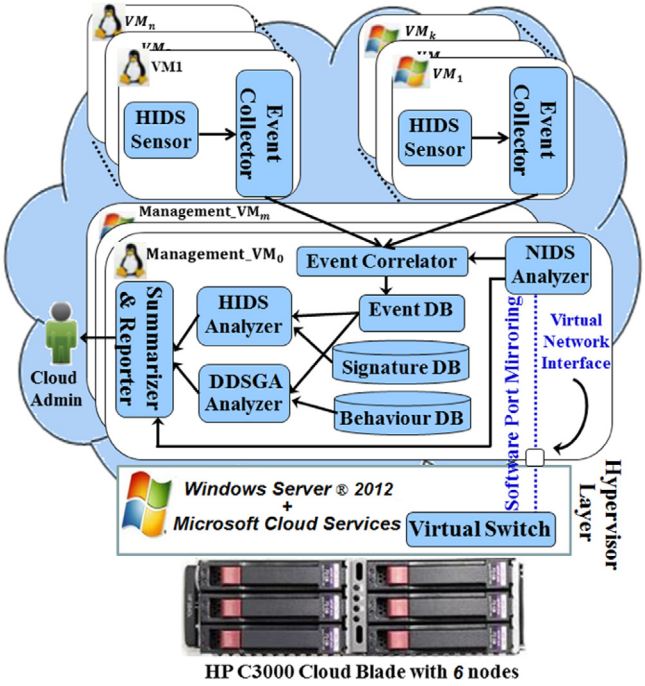

Fig. 3. CIDS-VIRT Architecture.

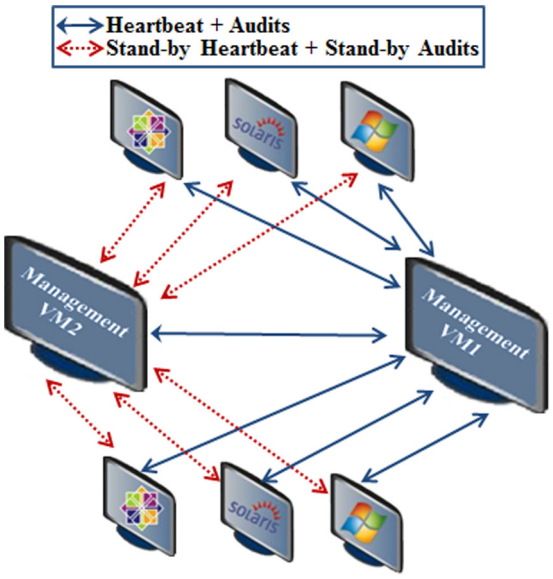

Fig. 4. Data exchange among the management VMs.

tool [33]. CIDS-VIRT uses OSSEC to perform log analysis, integrity checking, system activity monitoring, and time-based alerting.

NIDS Analyzer: it analyzes the VM network traffic to detect known trails that might represent an attack and is implemented by the SNORT IDS tool [34] which provides real-time traffic analysis and packet logging on IP networks.

– Summarizer & Reporter Component: It parses and summarizes the alerts fired by the HIDS and NIDS analyzers and correlates them. We use the Intrusion Detection Message Exchange Format (IDMEF) [35] as a standard data format that the host and network analyzers use to summarize, integrate, and report their alerts.

– Management VMs: These VMs run all the components previously described. They are reserved for the management of these components and only be accessed by the cloud administrators that can manage all these components from one place. To improve scalability and avoid a single point of failure, the cloud runs several management VMs with distinct operating systems. These VMs are fully connected to provide backup sources and to mutually exchange the detection task to avoid overloading the active one, see Fig. 4. The solid arrows in Fig. 4 represent the audits sent to the management VMs and the heartbeat message whereas the dotted line represents the interactions that occur if the active management VM fails or if it is highly overloaded.

Hypervisor Layer: It manages the creation of VMs, virtual networks, and virtual SAN drivers. Furthermore, it provides system security functions such as isolation, inspection, and interposition.

• CIDS-VIRT-Testbed:

The testbed consists of an HP c3000 Cloud blade with six nodes. The first node is the head node that works as a front side interface for the cloud blade and has a Quad-core 2.3 GHz CPUs, 32 GB RAM, 160 GB Hard drive, and a SmartArray P400 Controller for Storage Connect. The other five computing nodes have the same hardware as the head node. All nodes run a Microsoft core windows server 2016 instead of a VMware system. The head node runs a Microsoft GUI windows server 2016 with Microsoft cloud services and Microsoft Hypervisor manager. The testbed also includes one 24 port Procurve Switch (10/100/1000 ports) for data networks and another 24 port Procurve Switch (10/100 ports) for console management.

3. Detecting masquerades in host environment

This section describes masquerade detection in the host environment based on the anomalous analysis of system calls sequences. To evaluate this solution, we use the UNIX audits of the CIDD dataset that is described in Section 2.2.1. The detection in the network environment and the integrated solution are described in the following sections.

3.1. System calls feature extraction

Several system monitors and audit strategies of the activities of an operating system focused on system calls. These calls can be roughly grouped into five major categories: process control, file management, device management, information maintenance, and communication [36].

While current IDSs analyze calls in all categories, our solution only monitors two categories of system calls: file access and process activities. To explain the reasons behind our decision, we briefly discuss the disadvantage of considering all categories of system calls. First, an analysis that considers any call with its parameters and features often results in a slow detection process and produces very high false alarm rates with low hit ratio. This is due to a large number of system calls parameters and a large number of possible permutations of the calls. Another problem is that the training patterns for most calls are specific to the program versions and this affects the accuracy of detection anytime the version changes. Lastly, the basic premise for anomaly detection is the intrinsic regularity in the audit data that is consistent with the normal behavior and distinct from the abnormal one. Empirically, the more regular the data, the better the performance of anomaly detection [37].

We have focused on file access and process execution because they are essential and unavoidable for any user and can strongly reflect the user behavior with more intrinsic regularity than other

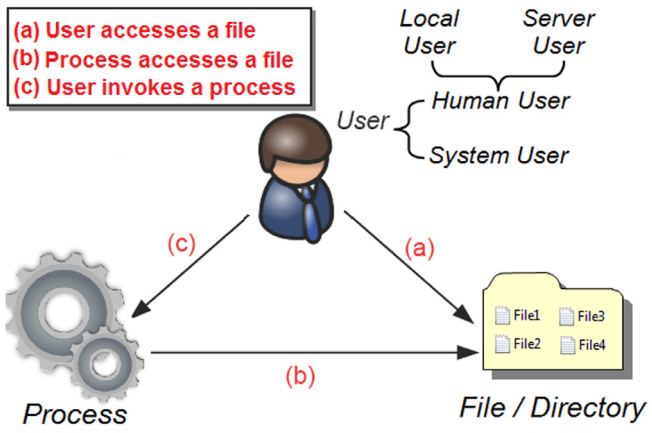

Fig. 5. The Behaviors Triangle Model.

activities. To analyze file access and process execution, we propose our Behaviors Triangle Model (BTM), see Fig. 5. The model builds a profile of the system activities of each user and reflects user behaviors in terms of three relationships, (a) user access a file, (b) process accesses a file, and (c) user invokes a process.

The Behaviors Triangle Model classifies users into human and system users. The human users are partitioned into local and server users. Users in these categories differ because of their nature. Local user activities are more complex and dynamic than those of server users e.g., database administration, word processing, web browsing, and miscellaneous activities such as command-prompt and window manager interaction. The activities of server users are more related to email services, file transfer, and web browsing. Unlike human users, system users are dedicated to a few tasks and have specific privileges, and likely to behave statically. These limited activities and privileges help in detecting any attempt to masquerade as a system user. DDSGA detects these masquerades by building a lexical list with common files and processes accessed or invoked by the system users. Any pattern outside this list is a strong indication of a masquerader. System users often interact with human processes in normal operations. We separate human user activities from system user processes and we then check each user separately. In multi-user systems like cloud systems, activities of distinct users are recognized in traces through the user ID paired with each system call. Using these IDs, we can easily define the activities of each human user. Instead, the system users' activities may interact with local and server users, but they usually do not refer to a specific user. Hence, we can detect a masquerader that is misusing the system user but without any information about the user that has executed the interaction that originated the misuse. To this purpose, a further detection process should be run for all human users. Fig. 6 shows three training sessions for the three kinds of users, according to the features extracted from the CIDD dataset. The UUCP system user can access to the /uccp directory, whereas the root user can access the /ufs and /etc files. As soon as a masquerader compromises them, the program tries to access files in other system directories such as /lib or /user. By exploiting the interactions between these system users and human processes in normal operation, the masquerader can escalate his/her access privileges to acquire some human user privileges. The last two masquerades in the sessions shown in Fig. 6 are executed by both users uccp and root but it is not known if they affected one or more of the human users, e.g., users 2060 or 2140. DDSGA detects these masquerades by checking the system users' profiles. If the detection rate is smaller than the detection threshold, DDSGA checks all the human users to discover if any of them were impersonated by that masquerader to use the privileges of the system users. The next section details how DDSGA detects masquerade attacks in UNIX system call traces.

As shown in Figs. 5 and 6, we build the user profile based on the sequence of patterns of the files the user has accessed and of the process being invoked. The arcs of the triangle define three relationships: (a) A user accesses a file: it includes the file access patterns that correspond to a given task. These access patterns represent a profile of normal user behavior. Therefore, any deviation in the current pattern with respect to this profile will be considered as an anomalous. (b) A process accesses a file: this relationship defines how the process accesses a file, a masquerader may use the user process privileges to access other important files. The analysis of the sequences of accessed files by the processes may result in a highly accurate detection with respect to the accessed files by the user because this relationship involves a fixed and well-defined list of files that each process can access. (c) A user invokes a process: the previous analyses for system calls in [38–40] consider the list of all the processes forked or executed by each program Instead, our analysis focuses on the sequence of processes the user has invoked and the program being executed. The main idea is that as each user has a characteristic working set of accessed files, the user also has a list of favorite programs. We record the sequences of these programs and consider as an anomalous the case where a test process or program either does not appear in these sequences or the process does not follow the normal sequences. The analysis neglects forked processes because we noticed that they are highly predictable and the HIDS component can easily detect if a process has forked one it should not have. Figs. 7, 8, and 9 show, respectively, the distribution of the access patterns and executed programs for the three user categories.

The previous figures show that the behaviors of both server and system users are highly predictable and that the detection of deviations in these behaviors may be more accurate than that for local users. We also notice that executed programs are more consistent and predictable and the corresponding access patterns can improve masquerade detection. In summary, the feature extraction from the system call data is implemented as follows:

(1) The auditor system in CIDS or the event correlator in CIDS-VIRT defines the start and the end of user sessions based on the Sliding Window Size (SWS) described in Section 3.2.1.
(2) The auditor in CIDS or the correlator in CIDS-VIRT filters the system calls data in the profile of system activity of each user that was built by the BTM and correlates all sessions' data of the same user.
(3) The auditor in CIDS or the correlator in CIDS-VIRT formats the extracted system calls features as a tuple of the form: $\{X\ (U, S, P, F, M, T), y_1(r_1), y_2(r_2) \ldots y_n(r_n)\}$ Where,

- "y_i" is the input parameter that can be a file or process name, and "r_i" is the return parameter of the executed pattern. r_i is zero if the pattern was successfully and one otherwise
- X is the session ID followed by its input parameters,

– "U" :the current user id.
– "S" : the login source IP address.
– "P" : the login period,
– "F" : a Boolean value to define whether there is a login failure in the session.
– "M" : a Boolean value to define if the session has a masquerade or not.
– "T" : the detection threshold for the session user.

Local User

= = = = = =

Session Head: Sess-ID, User-ID, SourceIP, Period, LoginFailure?, Masquerade?

368-VM2 , 2140, 194.007.248.153, Evening, 0, 1

Session Contents: (Path, Return-Value)

(/export/home/janes/.hushlogin, 0), (/opt/local/bin/tcsh, 0), (/usr/lib/fs/ufs/quota, 0), (/usr/bin/cat, 0), (/usr/bin/rm, 1), (/usr/bin/vi, 0), (/opt/local/lib/solaris/specs, 1) .....

Server user

= = = = = =

102-VM4, 2060, 172.016.112.207, Afternoon, 0, 0

(/opt/local/bin/tcsh, 0), (/usr/lib/fs/ufs/quota, 0), (/usr/bin/cat, 0, 0) ,(/usr/bin/ftp, 0), (/usr/bin/lynx, 0), (/usr/lib/sendmail, 0) .....

System Users

= = = = = = = =

68-VM1, uucp, 127.0.0.1, Afternoon, 0, 1

(/usr/bin/sh, 0), (/usr/bin/date, 0), (/usr/lib/uucp/uusched, 0), (/usr/lib/uuxqt, 1), (/usr/ bin/touch, 1), (/usr/bin/date, 0), (/usr/lib/uucp/uuxqt, 0) .....

73-VM1, root, 127.0.0.1, Morning, 0, 1

(/usr/lib/fs/ufs/ufsdump, 0), (/etc/dumpdates, 0), (/usr/ucb/whoami, 1).....

Fig. 6. Three training sessions with the extracted features for three types of users.

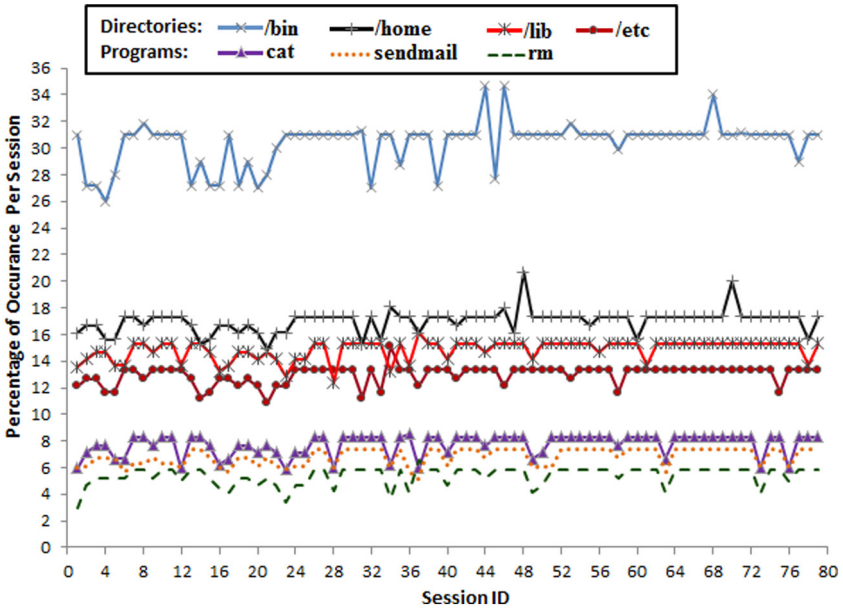

Fig. 7. The Local User with ID "2140" in CIDD.

– S, P, and F are the input of the neural network described in Section 5. For example, a valid tuple is: *{713-VM1 (2143, 135.013.216.191,"Afternoon", 0, 0, 0.72), /usr/lib/fs/ufs/(0), /usr/ucb/whoami(1), …..}*. Fig. 6 shows examples of these tuples.

(4) The DDSGA receives a tuple with *n+1* fields where "*n*" is the length of the session.

3.2. Applying DDSGA to correlated system calls

The main advantage of DDSGA is that it works efficiently with any sequence of patterns regardless of the running environment. To detect masquerades in UNIX audits, DDSGA computes the best alignment score by aligning the active session sequence e.g., the access patterns and executed processes as in Fig. 5, to the

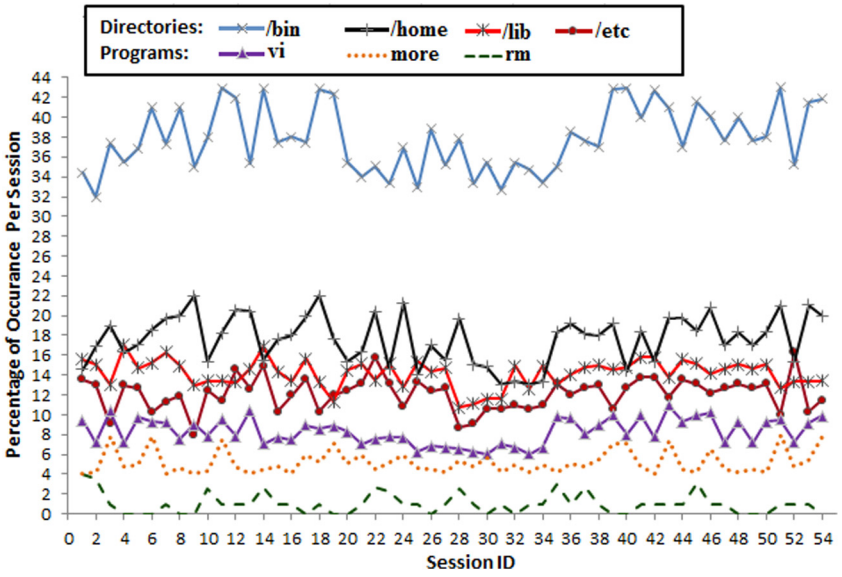

Fig. 8. The Server User with ID "2060" in CIDD.

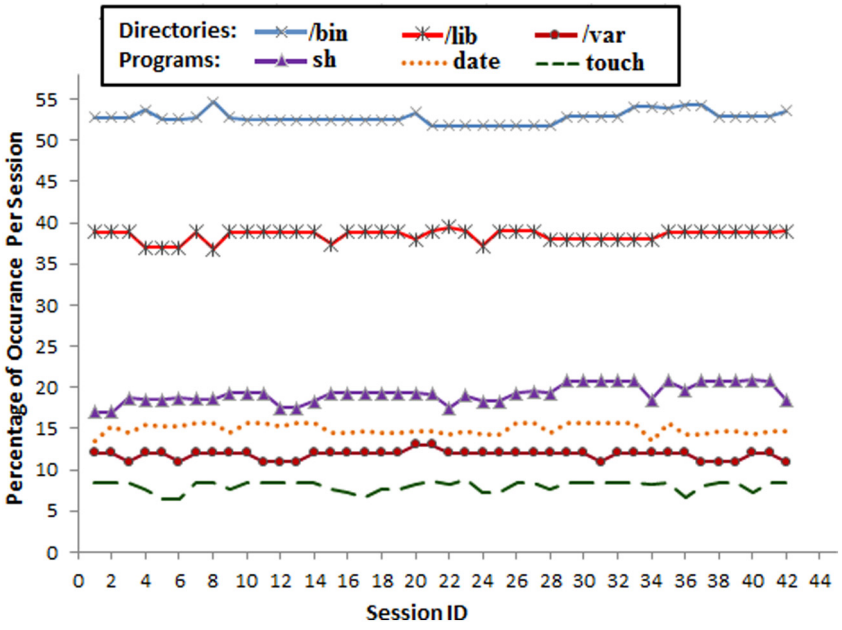

Fig. 9. The System User with ID "UUCP" in CIDD.

previous sequences of the same user. Some modules of the implementation have to be updated because the original DDSGA [41] was applied to the simulated SEA dataset [24] with overlapped sessions of fixed length. Instead, the CIDD dataset is distributed among the cloud VMs and has non-overlapped non-fixed length real-time sessions. To this purpose, we have to select the best Sliding Window Size (SWS) for each test session and to build a reasonable scoring system that adapts the previously extracted features of UNIX audits. Both the detection and update phases do not change if CIDD is considered, because they use the new parameters returned by the configuration phase. In the detection phase, we test three correlation models, Independent, Audit Exchange, and Centralized-Backup, to adapt to the characteristics of cloud systems. We detail these issues and discuss the effect of correlating the user audits in the following subsections.

3.2.1. Choosing the best Sliding Window Size (SWS)

The dynamic or Sliding Window Size (SWS) determines the length of the test sequence in the active session or, in other words, it determines when the detection phase should start the detection process. The SWS affects this process and helps in improving the system call modeling methods. Previous work [37] has chosen the optimal size based on an information-theoretic framework using two approaches based on the training data. The first one uses entropy modeling and it measures the regularity in the data. The second approach takes advantage of the context-dependency of the optimal size and estimates it according to the specific system calls in the training subsequences. The latter approach models the complete set of system call traces and it is not suitable in our framework where it focused on a subset of call traces (access patterns and the invoked processes). Furthermore, it uses the sparse Markov transducers [42] while DDSGA uses a

more flexible alignment technique. We have estimated the SWS factor using four approaches:

(a) Minimum Conditional Entropy (MCE),
(b) Test Session Length (TSL),
(c) Test Session Length with Sensitive Action (TSLSA),
(d) Average Signature Session Length (ASSL).

Choosing the best SWS is a data-driven process that runs once at the configuration step of the detection process. Thus the SWS is computed once for each dataset and is updated in the update phase. We have evaluated each approach through two main measures, the detection accuracy using the Receiver Operator Characteristic (ROC) curves [43] and the masquerader live time length. After highlighting the four approaches, we detail their evaluation.

(a) The Minimum Conditional Entropy (MCE):

The main idea underlying conditional entropy is to measure the regularity of the training data for each user using different *SWS* values and choose the one that results in the most regular data and corresponds to the lowest entropy. The definition of conditional entropy is recalled in Eq. (1).

$$H(X|Y) = -\sum_{x,y \in X,Y} P(x,y)\log_2 P(x|y) \quad (1)$$

where $P(x,y)$ is the joint probability of x and y, and $P(x|\ y)$ is the conditional probability of x given y.

To apply the entropy to our model, we introduce the following definitions:

- $X = S$: the set of system call patterns (access patterns and invoked processes).
- $Y = S_{SWS-1}$: the set of system calls patterns sequences of length *SWS-1*.
- S_{SWS} : a sequence of length *SWS*.
- *A:* the set of all sequences in the training data.
- $N(S_{SWS})$ *: the* number of times the sequence S_{SWS} appears in *A*.
- *N(A):* the total number of sequences in the training data.

If we define the joint probability $P(x,y)$ as follows:

$$P(x,y) = P(S_{SWS}) = \frac{N(S_{SWS})}{N(A)}$$

The conditional entropy of Eq. (1) for a window size *SWS* is:

$$H_{SWS}(X|Y) = -\sum_{x \in X, y \in Y} \frac{N(S_{SWS})}{N(A)} \log_2 P(x|y) \quad (2)$$

where the conditional probability $P(x|\ y)$ is the prediction of this entropy model and it means that the probability of system call *'x'* at a position (*SWS)* in the training sequences is estimated from *y*, the previous (*SWS-1)* system calls. To compute the prediction for each user training data we consider, for each (*SWS-1*) sequence of system call patterns in the training data of user *U*, we keep counts of the following system calls. Then, $P(x|\ y)$ the prediction for system call *'x'* given a sequence *'y'* of (*SWS-1*) preceding system calls is simply *p/t* where, *'p' is* the count of system call *'x'* in the calls in *'y'* and t is the total count of the system call *'x'* in the sequences that includes all the calls, even those after the *SWS-1* position.

If a sequence S_{SWS} does not occur in the system call patterns sequences, $P(S_{SWS}) = 0$. Therefore, we can set as in Eq. (3) the conditional entropy of Eq. (2) for a window size *SWS* to denote that at least one S_{SWS} sequence occurs in *'A'*, the set of all sequences in the training data.

$$H_{SWS}(X|Y) = -\sum_{S_{SWS} \in A} \frac{1}{N(A)} \log_2 P(x|y) \quad (3)$$

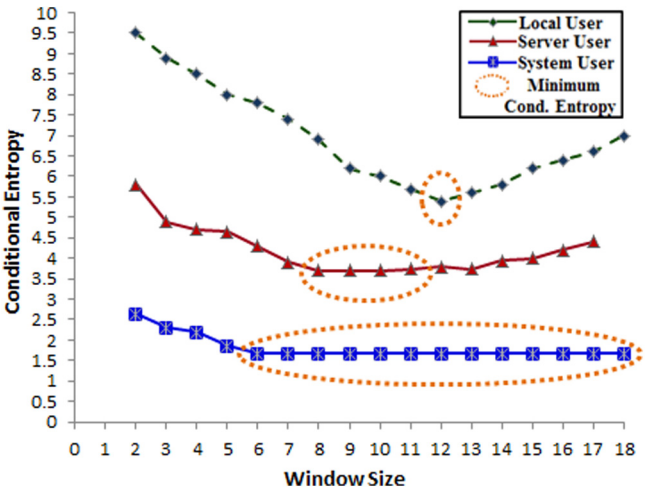

Fig. 10. The conditional entropy under different SWS values for local user "2140", server user "2060", and system user "UUCP".

Eq. (3) computes the conditional entropy $H_{SWS}(x|\ y)$ for the system call patterns sequences of each user using window size *SWS*. The most suitable windows size for each user is the one that corresponds to the minimum entropy and the more regular data.

To compute the conditional entropy for each user data, we use the cross-validation in [37] for the training sequences of each user extracted from the CIDD dataset. We train the prediction models with one part of the training data and apply Eq. (3) to compute the entropy over the second part of this data. Then, we repeat the computation after swapping the two parts. The total entropy is the sum of both entropies. Fig. 10 shows the conditional entropy for three users, each of a distinct kind.

Notice that the curves for server and system users do not have a specific minimum due to the high regularity of data for these users is very high. We have noticed that the lower the entropy, the more regular the data and the better the performance of detection. The accuracy and masquerader live time of the conditional entropy model are acceptable for server and system users because of their highly regular data. This solution does not achieve the same performance for local users whose data are less regular.

(b) Test Session Length (TSL): This approach sets the SWS to the length of the active test session. This results in the best accuracy among all the approaches because the test session is long enough to be compared against the previous training sequences. However, the masquerader's live time is longer than in the other approaches.

(c) Test Session Length with Sensitive Action (TSLSA):

The *SWS* is set to the length of the active test session but the detection process can start at any time a sensitive action occurs. Sensitive actions are a set of sensitive file access patterns or invoked programs predefined in the DDSGA database. An attacker exploits these patterns or invokes these programs to misuse cloud resources. Table 1 shows some examples of sensitive files and programs in the DDSGA database. The approach achieves a suitable detection accuracy and the shortest masquerader live time for all user categories except the system users. This may due to the fact that usually system users access sensitive files and execute sensitive commands to support operating system functionalities. As an example, when a user logs in, the operating system verifies the login by checking the */etc/passwd* file.

Table 1
Examples for sensitive files and programs.

File/Program	Pattern	Pattern task
File	*/etc/passwd*	Records users encrypted password
File	*/usr/adm/saveacct*	Records accounting information
File	*/usr/adm/wtmp*	Records all logins and logouts
File	*/etc/hosts*	List of IP hosts and hostnames
Program	*/bin/passwd*	Changes user password
Program	*/bin/yppasswd*	Changes NIS password
Program	*/etc/ttymon*	Monitors terminal ports
Program	*/sbin/disk*	Formats hard disk
Program	*/bin/chmod*	Changes file permissions

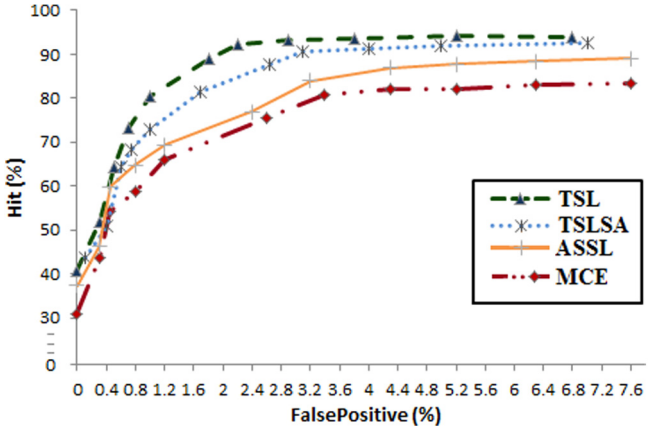

Fig. 11. The ROC for the three sliding window selection methods for local user "2140" in CIDD.

(d) Average Signature Session Length (ASSL):

The SWS is set to the average length of training sessions or the length of the current session if this session is shorter than the average one. ASSL increases both the accuracy and the masquerader live time for users with close training session lengths. To evaluate and compare the detection accuracy of each of the four approaches for local, server, and system users we use the ROC curves [43] in Figs. 11, 12, and 13 respectively. Each curve graphs the false positive rate versus the detection rate. We have also built the chart of the masquerader live time that shows some masqueraded sessions from the training data for the three users. To get distinct false-positive rates in the ROC curve, we have changed some of the DDSGA parameters such as the detection threshold and the scoring parameters. We have applied DDSGA using the Centralized-Backup model with the lexical and return checks as detailed in the next section. Other correlation models result in the same conclusions. In the ROC curves, the best detection is the one with the highest detection rate, minimum false positive rate, and the smallest Maxion–Townsend cost [44] that uses this equation (*Maxion-T. Cost=6*FP+(100-TP)*). Table 2 summarizes the detection results.

As shown in Fig. 14, both the SWS and the characteristic of the user activities affect the masquerader live time. As an example, a system user session is longer than those of the other users, because it reflects some operating system activities. Instead, server user sessions are among the shortest ones, because each reflects a small set of activities e.g., sending an email or transferring a file. The length of a local user session changes according to the user's behavior.

According to our analysis for detection accuracy and masquerade live time, we use TSLSA for local users, because of the low regularity of their data. Instead, we use MCE for both server and system users that have highly regular data.

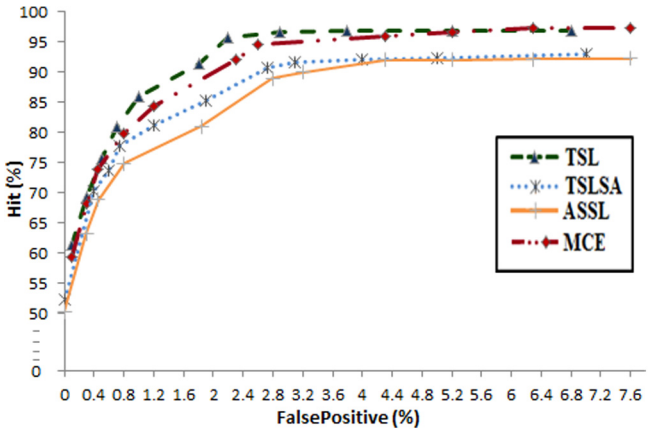

Fig. 12. The ROC for the three sliding window selection methods for server user "2060" in CIDD.

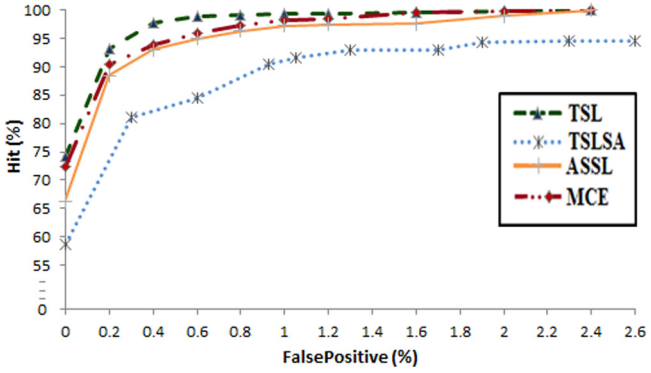

Fig. 13. The ROC curve for the three SWS approaches for system user "UUCP" in CIDD.

Table 2
A comparison between the best detection outputs for the previous four SWS approaches for 4 local, server, and system users sorted by user category and Maxion–Townsend cost.

SWS approach	Use category	False positive %	Hit %	Maxion-T. cost
TSL	Local	2.2	92.24	20.96
TSLSA	Local	3.1	90.49	28.11
ASSl	Local	3.2	84.01	35.19
MCE	Local	3.4	80.83	39.57
TSL	Server	2.2	95.74	17.46
MCE	Server	2.6	94.61	20.99
TSLSA	Server	2.73	90.71	25.67
ASSL	Server	2.8	88.98	27.82
TSL	System	0.6	98.94	4.66
MCE	System	0.8	97.31	7.49
ASSL	System	0.8	96.27	8.53
TSLSA	System	1.05	91.62	14.68

3.2.2. Scoring system

DDSGA computes the detection score for the session and compares it against the threshold to determine whether the session is a masquerade. Then, it compares this score against "M" to compute the false alarms and hit ratio. The scoring system of DDSGA can tolerate changes in user behaviors without significantly reducing the detection score. We have modified the original scoring system to work with the extracted features of the "Behaviors Triangle Model". The new scoring system, see Figure 15, rewards a mismatched pattern in the test sequence in two cases. In the first one, the test pattern has previously appeared in the user lexicon. This can tolerate various permutations of previously observed

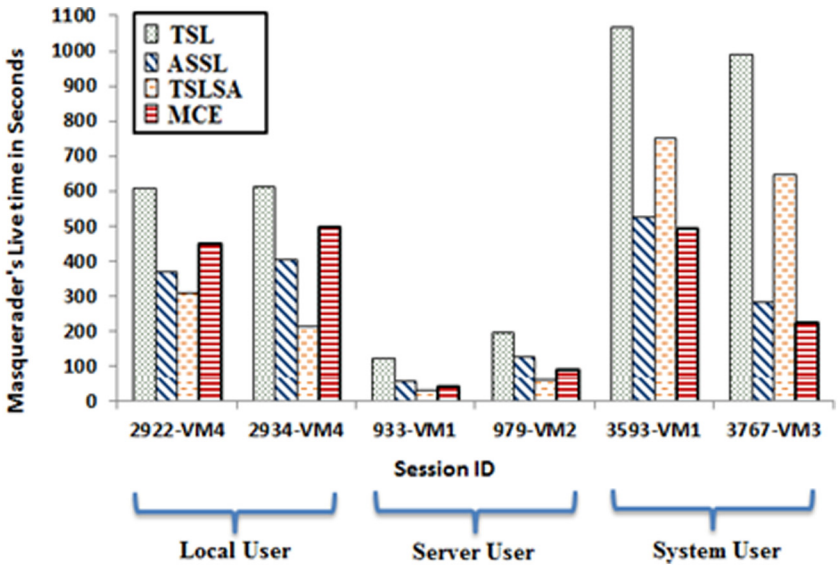

Fig. 14. The masquerader live time in seconds for local, server, and system users in some attached sessions.

patterns without reducing the detection score significantly. The second case occurs if r_i, the return parameter of the executed mismatched pattern, is zero, that is the access or executed pattern was successful. The scoring system rewards successful action and penalizes failed one because we noticed that most of the masqueraded patterns include failure attempts because a masquerader usually lacks some knowledge of the victim file system. The next section highlights the evaluation of the scoring system in terms of security and complexity for the three correlation models.

To address the recurrent activities that are less probable but still represent good behaviors of the cloud user. DDSGA runs two update-modules, the inline and long-term modules. The inline module updates the user signature patterns, the user lexicon list, and their corresponding command categories in a reconfiguration phase. The long-term module updates the system with the latest changes in the alignment parameters. It also updates the dynamic threshold values, scoring parameters, and the overlapping length i.e., the length of the overlapped signature subsequence according to the maximum number of inserted test gaps. The dynamic threshold, the scoring systems, and the two update-modules enable DDSGA to tolerate slight changes and recurrent activities in the user behavior over time.

To address the intrinsic dynamics of cloud user behaviors, (1) DDSGA considers not only lexical matching such as string matching or longest common substring searches but also by tolerating small mutations in the sequences with small changes in the low-level representation of the user commands. To this purpose, a command pattern can be aligned with one that implements the same functionalities. This helps in tolerating permutations of previously observed patterns with a low reduction of the alignment score. Furthermore, besides the new scoring systems described in this section (see Fig. 15), to increase the hit ratio and reduce both false positive and false negative rates, DDSGA pairs each user with distinct gap insertion penalties according to the intrinsic dynamics of user behavior. For more details about how DDSGA addresses such scenarios, see [41] (2) The BTM considers only the essential and unavoidable behavior actions for any user, the file access and process execution, that can strongly reflect the user behavior with more intrinsic regularity than other activities [36,37].

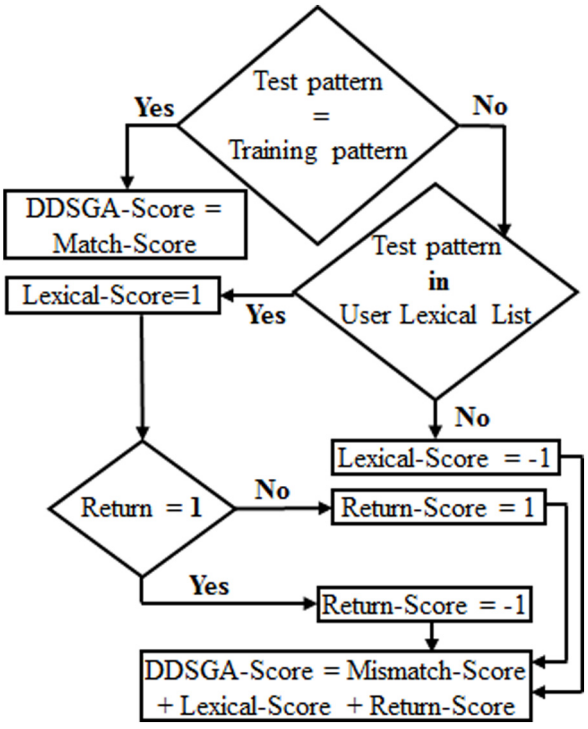

Fig. 15. A flowchart for the modified DDSGA scoring system.

3.3. The independent, audit exchange, and centralized-backup models

We have developed three correlation models for different deployment models namely, Audit Exchange, Independent, and Centralized-Backup. The first two models work with the original CIDS framework and they were briefly introduced in [13] without real-time evaluation with DDSGA or CIDD. The third model is firstly introduced in this paper and works with the improved framework, CIDS-VIRT. Section 3.4 discusses the experimental results of the three models. In the following, we briefly outline the three implementation models.

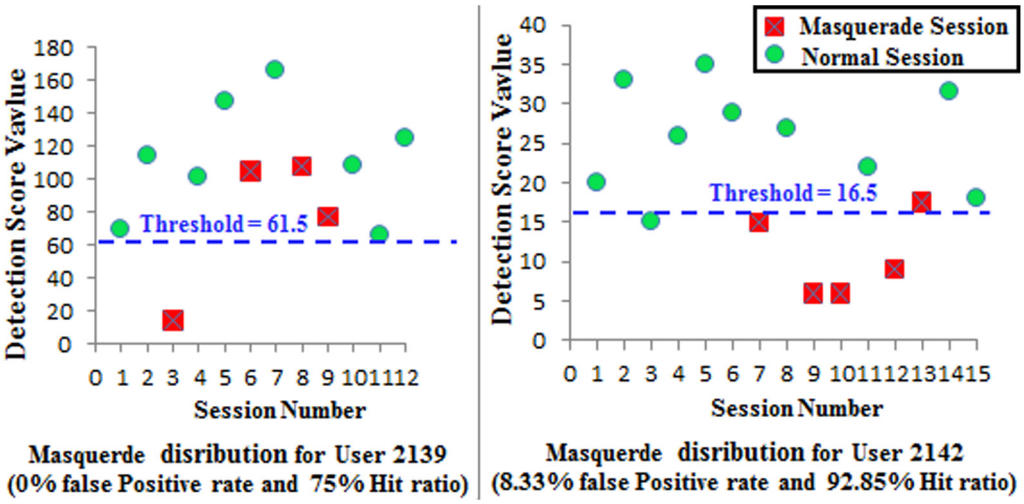

Fig. 16. DDSGA threshold and masquerades distribution in test sessions of CIDD users 2139 and 2142.

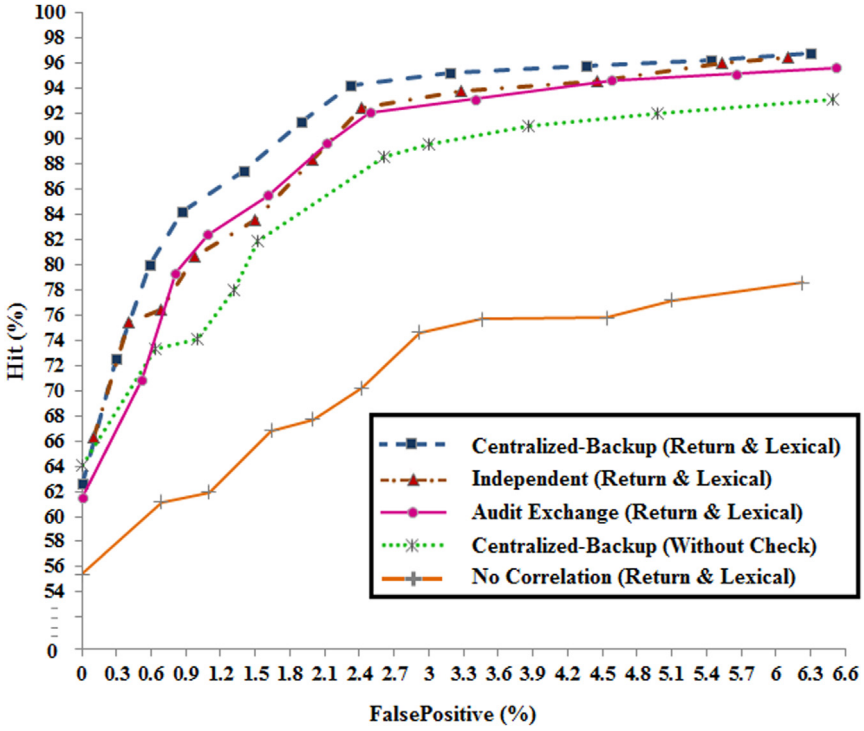

Fig. 17. The ROC curve for the three correlation models with and without the scoring system.

(A) Audit Exchange Model In this model, nodes exchange their audit data so that each one has a complete audit data for its current users. The detection phase depends upon two parameters: (1) The alignment score computed by the CIDS detector component, and (2) The alerts fired by the HIDS component. This balances the detection overhead among nodes with no single point of failure. The detection efficiency is high because the user audit is concentrated in one node and the masquerader surviving is much shorter than in model B, see Figs. 17 and 18. As a counterpart, the model needs a fast-periodic update, and nodes have to exchange the audit data of the same user. This increases the cloud network overhead and hinders scalability. Furthermore, the exchange may result in the loss of some audit data in highly overloaded networks, see Fig. 19.

(B) The Independent Model The detection phase depends upon the same two parameters of model A. A cloud node *CN* evaluates login usage patterns of a user *U* using both CIDS and HIDS detectors and by using the behavior-based and signature-based of *CN* without interacting with other nodes. If the CIDS detector of *CN* fires an alert, the current login usage patterns are checked against the audit data of *U* in the other nodes related to *U* until one of them accepts the current pattern. If no node is found, the current login session will be marked as a masquerade attack. The model advantages are:

(1) It does not require a periodic update of user audit data in each node. The regular periodic backup for VMs data is similar to that of other models,
(2) Very low overhead for the cloud network, as data is exchanged only if the detection score is less than the threshold. In this case, a node exchanges the test audit data produced by the user during the login session, see Fig. 19,

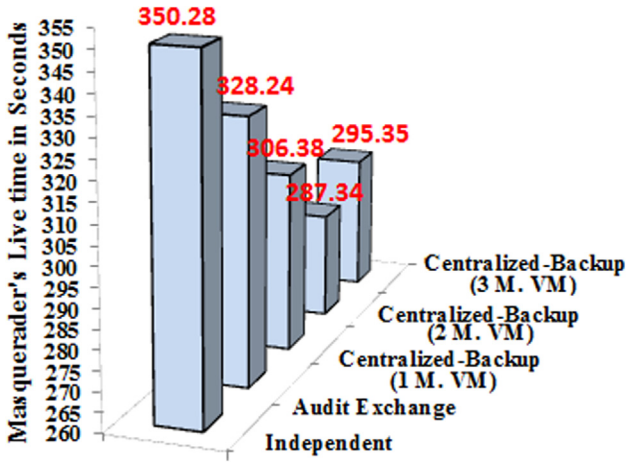

Fig. 18. Average Masquerader live Time per session in the three correlation models.

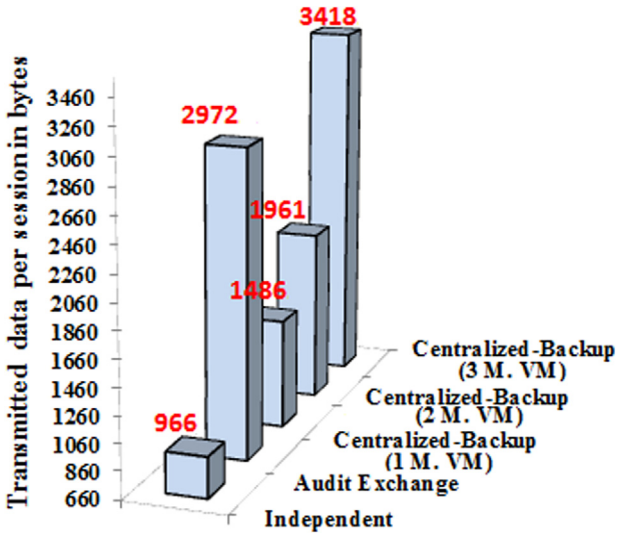

Fig. 19. Average transmitted data per session in bytes in the three implementation models.

(3) A high detection efficiency in terms of hit and the false alarm rates close to that of model A, see Fig. 17.
(4) A lower processing overhead in each cloud node than models A and C. This reduces the detection time because each node executes the DDSGA alignment only if the detection score is less than the threshold. The detection time is directly proportional to *NN*, the number of nodes that have audits for user *U*. See Fig. 20.

As a counterpart, the surviving of a masquerader is longer than in models A and C because as *CN* increases it also increases the time to analyze the audit data in all nodes, see Fig. 18. Hence, this model does not scale to large cloud networks that normally have a large number of VMs and users and consequently a larger *CN* value.

(C) The Centralized-Backup Correlation Model

In this model, users' VMs send their audit data to a reserved management VM that has a complete view of audit data for all users to analyze and report the final alerts. The management VM is backed up to some other VMs as explained in Section 2.2.3 to balance the detection overhead among the management VMs with no single point of failure. This model achieves the best detection efficiency because the user audit is concentrated in one place and there is no loss of the audit data. The masquerader surviving is very short comparing to model A and B, see Figs. 17 and 18 with low network overhead. The detection time is inversely proportional to the number of management VMs that reduce the processing overhead in the active management VM. This speeds up the detection phase and protects the IDS components from tampering by any attackers. On the other hand, the network overhead increases with the number of management VMs, see Figs. 19 and 20. Furthermore, the model requires several resources as it reserves some management VMs for detection.

3.4. A comparison of the three correlation models

We have applied DDSGA to all users in the CIDD dataset and focused our evaluation on local users that have a large deviation in their behaviors. The experiments compare the three models in terms of four values:

(1) Accuracy and efficiency using both the ROC curve and Maxion–Townsend cost,
(2) Average masquerader live time per session,
(3) Average transmitted data per session during the detection time,
(4) Average detection time per session.

(A) The accuracy and efficiency

To evaluate the accuracy and efficiency of the models, we focus on the effects of the computed DDSGA alignment parameters i.e., the detection threshold and the scoring system rewards and penalties, on the detection accuracy of the system. The false positives, false negatives, and hit ratios are computed for each user and then transformed into the corresponding rates that are summed and averaged over all users. Eqs. (4)–(6) show the metrics used by DDSGA.

$$Total\ False\ Positive = \left[\left(\sum_{k=1}^{nu}\left(\frac{fp_k}{n_k}\right)\right)/nu\right] * 100 \quad (4)$$

where
- *fp = No. of false positive alarms,*
- *n = No. of non-intrusion sessions,*
- *nu = No. of users in CIDD dataset (84 in our case)*

$$Total\ False\ Negative = \left[\left(\sum_{k=1}^{nui}\left(\frac{fn_k}{ni_k}\right)\right)/nui\right] * 100 \quad (5)$$

Where:
- *fn = No. of false negatives,*
- *ni = No. of intrusion command sequence blocks,*
- *nui = No. of users who have at least one intrusion block*

$$Total\ Hit\ Ratio = 100 - Total\ False\ Negative \quad (6)$$

Fig. 16 shows the masquerades distribution in the test sessions for some CIDD users and the detection threshold that DDSGA computes for each user in the training phase.

To plot the ROC curve, we use distinct values of the alignment parameters that result in false-positive rates in the *x*-axis and the corresponding hit ratios in the *y*-axis. Fig. 17 shows the ROC curves for each model with the scoring system, the Centralized-Backup model without the scoring system, and the No-Correlation model with the scoring system.

As shown in Fig. 17 and Table 3, the Centralized-Backup model with the scoring system results in the highest hit ratio with the corresponding lowest false positive rates. The flexibility of tolerating a large number of mutations and deviations in user behaviors enabled by the scoring system increases the hit ratio

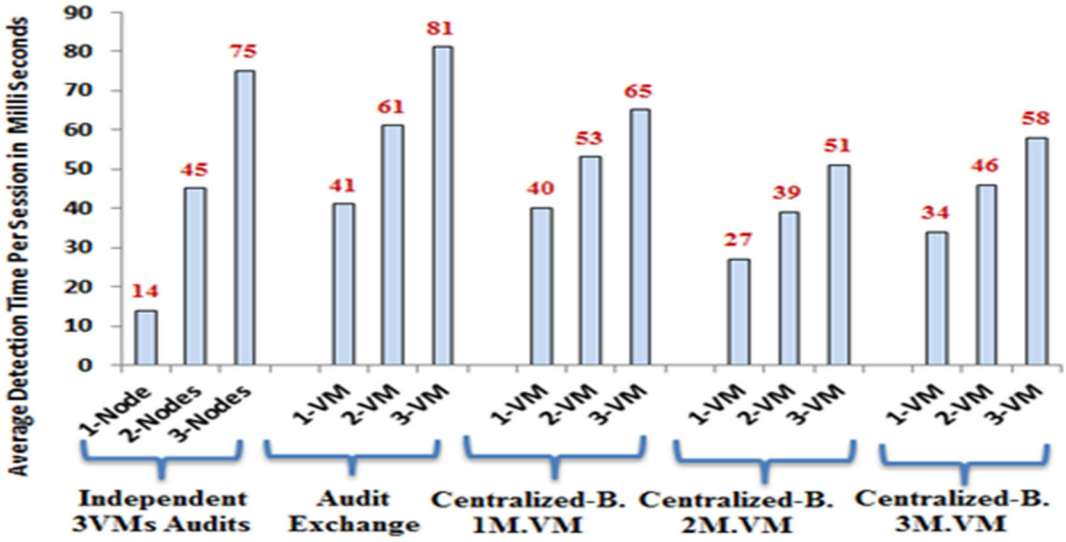

Fig. 20. Average detection time per session in milliseconds in the three models.

Table 3
The best accuracy of the three Correlation Models sorted by Maxion–Townsend cost.

Correlation model	False positive %	Hit %	Maxion-T. cost
Centralized-Backup model with return and lexical check	**2.32**	**94.24**	**19.68**
Independent model with return and lexical check	2.42	92.44	22.08
Audit Exchange model with return and lexical check	2.49	92.09	22.85
Centralized-Backup model without the scoring system (without check)	2.61	88.99	26.67
No correlation Centralized-Backup model with return and lexical check	2.92	74.60	42.92

by about 5.25% and reduces the Maxion–Townsend cost by 6.99. The correlation of the user audits is mandatory to build a consistent profile with all the audits from the VMs in the cloud. The correlation improves the hit ratio by about 19.64% and reduces the Maxion–Townsend cost by 23.24%.

(B) Average Masquerader live Time

We compute the average masquerader live time overall sessions for the three correlation models. In the case of the Centralized-Backup model, a larger number of management VMs reduces the computational overhead and, consequently the live time as well. As a counterpart, it increases the cloud network overhead. Therefore, we have to experimentally determine the optimal number of management VMs, as shown in Fig. 18, the shortest live time is achieved when using two management VMs.

(C) Average Network Overload per session.

We compute the overhead on the cloud network in terms of the average amount of data that each model transmits in a session. The data can be user audits or current active session according to the considered model and it is sent from the VM(s) that runs the detection task to the other VMs. As shown in Fig. 19, the Independent model is the most lightweight one. These curves confirm that the network overhead of the Centralized-Backup model is directly proportional to the number of management VMs.

(D) Average detection time per session

The average detection time is affected by the machine capabilities and available processing resources. The size of the test session, the corresponding training sessions, and the number of user VMs in the cloud system are further important factors. Fig. 20 shows that the detection time of the Independent model for the user who has audits distributed among three VMs depends upon, NN, the number of cloud nodes (CN) running these VMs. In this way, the Independent model distributes the training audits among the cloud nodes and each node independently runs the detection process using some training records. The shortest detection time is achieved if the detection score in the first node is larger than the detection threshold, and this time increases as NN increases. The 3VMs audits label of the independent model in Fig. 21 means that user audits are distributed across three nodes, each with one VM for that user. If we compare the 3-VM columns against the other models, we can notice that the Independent model may result in a noticeable improvement but the improvement is reduced as NN increases. If NN is equal to 3, this model results in the worst detection time. Therefore, this model is ideal with a small number of users and VMs and it cannot be adopted in the large cloud such as public or hybrid ones. On the other hand, the Centralized-Backup model has a reasonable detection time that is reduced as the number of management VMs increases. Hence, the model is more elastic and scalable and it may be adopted in large clouds.

4. Detecting masquerade in network environment based on NetFlow data analysis

Most of the current masquerade detection approaches are based on host-based profiles, such as command line, system call, and GUI interaction. All these approaches directly monitor the hosts and IDS components gather and process host data. However, massively distributed and interconnected systems, including cloud computing, distributed grid computing, and smart homes, require a network-centric approach to masquerade detection [6]. This approach preserves privacy by avoiding the data accessibility issues associated with host-based approaches as it only needs statistical information on the network traffic and it can replace user identifiers, source and destination IP addresses with encrypted or anonymous values. A NetFlow analysis, e.g., an analysis that considers information on the network activities of a user, can generate a unique and useful user network profile to detect

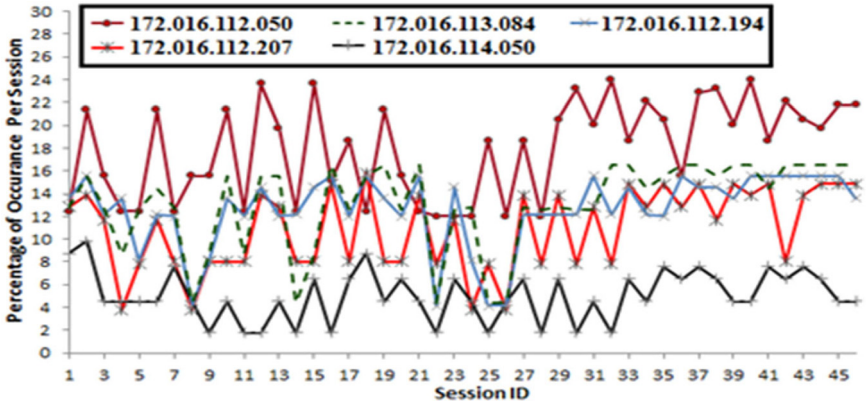

Fig. 21. The distribution of NetFlow destination IP addresses in local user sessions (user ID 2143 in CIDD).

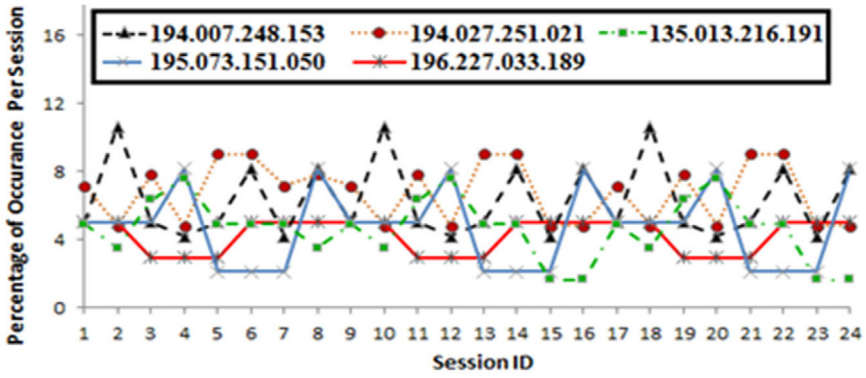

Fig. 22. The distribution of NetFlow destination IP addresses in server user sessions (user ID 2059 in CIDD).

potential masqueraders whenever the host data is not accessible or legal/ethical restrictions apply [6,45]. Furthermore, while most organizations may not routinely collect host audits on all users, by default they collect statistical information on network traffic for network administration, monitoring, and troubleshooting purposes.

As a counterpart, a NetFlow analysis follows the network activities of a user through the source IP address. Therefore, if several users log from the same address e.g., users share one server to remotely access their VMs in the cloud or more than one user is logged to a VM, the network activity profile will reflect the activity of the server or the VM connected to this source IP address rather than the activity of a single user. This is the reason why our analysis detects the masquerade IP source address. We propose three approaches to capture the NetFlow data namely, the log server, the inside–outside workstations, and the physical\virtual switch monitoring. The data of the log server approach consist of mail or FTP server logs. The log server associates each source IP address with its NetFlow traffic captured by the sniffer tool and it determines the start and end of user sessions. In the inside–outside workstations approach, TCPDUMP data can be captured by selecting two nodes, that are located, respectively, on the inside and the outside of a router. The TCPDUMP data includes any TCP/IP connection between the two nodes that would be captured by a sniffer located on the router. In the physical\virtual switch monitoring, a network IDS sensor like Snort is installed in one physical node or a VM and is connected to a physical switch port or to a promiscuous port on a virtual switch to mirror all traffic. In our CID-VIRT framework, virtual network traffic is forwarded to a management VM where Snort analyzes the traffic and gets the required NetFlow data. In the CIDS framework, network traffic is forwarded to one physical host that runs SNORT and it is attached to the auditor system to capture the network audits for user sessions as in Fig. 3.

4.1. Feature extraction from NetFlow data in the cloud network

Consider the sequence of destination IP addresses of the machines a user accesses and their corresponding protocol names. We are interested in using this sequence in the same way host-based masquerade detection uses system call access patterns. The correlator component of both the CIDS and the CIDS-VIRT frameworks defines the start and the end of user sessions as in the case of the log server. After that, the auditor system in CIDS or the event correlator in CIDS-VIRT filters the NetFlow data corresponding to the user host session according to the user source IP address. The NetFlow session length is equal to the host session length computed by one of the four SWS approaches discussed in 3.2.1. Figs. 21 and 22 show the distribution of the destination IP addresses in the network activity profiles of two source IP addresses that correspond to local and server user sessions, respectively.

In general, system users do not use the network environment because their activities are more related to some operating system tasks. These two figures show that server user sessions are more regular than local users' ones because their destination IP addresses are more specific and consistent than the corresponding ones in local user sessions. Besides, server users use just specific protocols to implement a specific task e.g., "FTP" to transfer files and "SMTP" to send emails, while local users use a larger number of protocols in their network activities. We also notice that NetFlow data is less regular than system call patterns because user network activities have a lower consistency than user host activities.

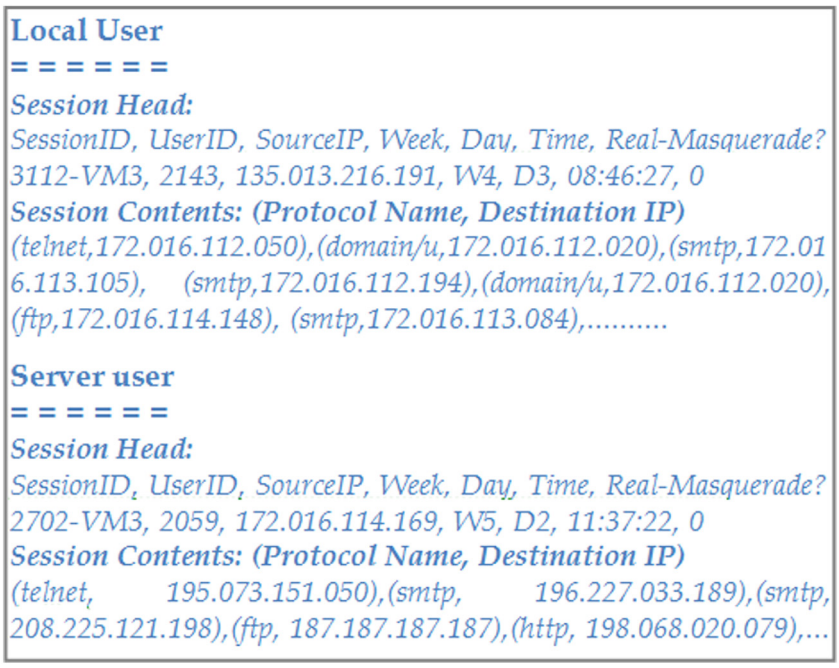

Fig. 23. Two training sessions with the extracted NetFlow features for local user 2143 and server user 2059.

A network activity profile reflects the behavior of a source IP address that may be shared among several users. This may reduce both the regularity and the consistency of data of the network activity profile that in turn, results in lower accuracy of detection. Therefore, we have to modify the DDSGA scoring system, see Section 4.2, to be more flexible and take into account all the previous features of NetFlow data.

In summary, the feature extraction from NetFlow data is implemented as follows:

(1) The auditor system in CIDS or the event correlator in CIDS-VIRT defines the start and the end of user sessions.
(2) The auditor in CIDS or the correlator in CIDS-VIRT filters the NetFlow data corresponding to the user host session according to the user source IP address.
(3) The auditor in CIDS or the correlator in CIDS-VIRT formats the extracted NetFlow features as a tuple of the form: *{X (U, S, DT, M, T), $y_1(p_1,d_1)$, $y_2(p_2,d_2)$... $y_n(p_n,d_n)$}*. Where "*n*" is the number of content patterns in the session and *X* is the session ID with the following input parameters:

– "*U*": the current user id.
– "*S*": the user source IP address.
– "*DT*": the date and time of the session in the form (Week Day Time).
– "*M*": Boolean value to define if the session has a masquerade or not.
– "*T*": the detection threshold for the session user.
– "y_i": a session content pattern that is composed of two parameters:
– The protocol name "p_n",
– The destination IP address "d_i".
For example, a valid tuple is: *{3112-VM3 (2143, 135.013.216.191, W4, D3, 8:56:47, 0, 0.64), (telnet, 172.016.112.050), (domain/u, 172.016.112.020)}*.

(4) Lastly, the DDSGA considers only two features of the Netflow tuples namely, the sequence of destination IP addresses of the machines a user accesses and their corresponding protocol names.

Again, DDSGA computes the detection score for the session and compares it against the threshold to determine whether the session has masquerade patterns. Then, it compares this output to "*M*" to compute the false alarms and hit ratio. Fig. 23 shows two training sessions with the extracted NetFlow features for a local user and a server one.

4.2. The netflow scoring system

Fig. 24 shows the scoring system for the extracted features of the NetFlow data. It has been defined by modifying the DDSGA scoring system.

The scoring system rewards a mismatched destination IP pattern in the test sequence in two cases. In the first case, the protocol name has previously appeared in the protocol list of the source IP in the test sequence. This tolerates various permutations of previously observed patterns of the combination (destination IP and protocol name) without reducing the detection score significantly. The destination IP and protocol name may be unrelated because a user can access the same destination with distinct protocols for distinct tasks. In the second case, the destination IP has previously appeared in the destination-IP list of the source IP. Again, this can tolerate various permutations of previously observed patterns because a user may access a set of destinations in distinct orders. We have applied DDSGA to the NetFlow data for each source IP sessions in CIDD in the same way described in Section 3 using the Centralized-Backup model of CIDS-VIRT. Fig. 25 and Table 4 show the detection accuracy of DDSGA in terms of both the ROC curve and Maxion–Townsend cost respectively.

As shown in Fig. 25 and Table 4, the NetFlow scoring system results in the highest hit ratio with corresponding lowest false positive rates. The scoring system increases the hit ratio by about 10.16% and reduces the Maxion–Townsend cost by 12.23.

5. A neural network model to integrate the host and network detection outputs

The integration of host and NetFlow detections improves the accuracy and the efficiency of the overall masquerade detection.

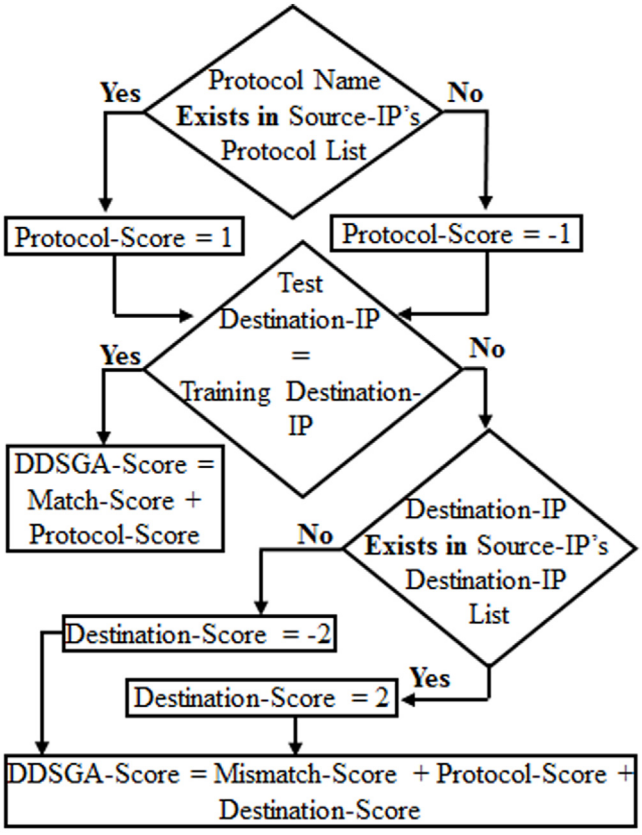

Fig. 24. A flowchart for the modified NetFlow scoring system.

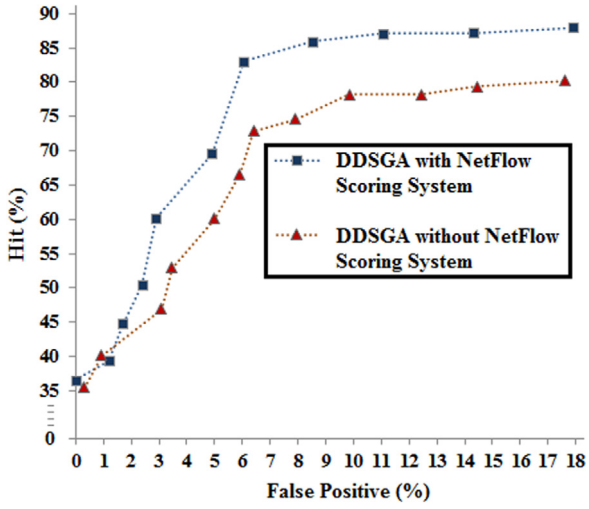

Fig. 25. The ROC curve for the DDSGA approach on the NetFlow audits with and without the scoring system.

Table 4
The best accuracy of the DDSGA approach on the NetFlow audits with and without the scoring system sorted by Maxion–Townsend cost.

Approach	False positive %	Hit %	Maxion-T. Cost
DDSGA with NetFow scoring system	6.05	83.05	53.253
DDSGA without NetFow scoring system	6.39	72.89	65.487

We have implemented the integration through a Threshold Logic Unit (TLU) [46] see Fig. 26. The TLU model is a vastly simplified model of the neuron network that has low computational complexity. The TLU works for each user independently to be adapted to the consistency of each user's behavior. A Group of TLUs consists of a complete neural network model for all users in the CIDD dataset. The TLU has 3 layers: one input layer of dimension $n = 4$ inputs, a hidden layer with 1 summing junction neuron, and an output layer with 1 neuron connected to an activation function. The activation function depends upon the TLU operational mode. The detection mode uses a threshold function which is equal to 0 if the sum of the input is less than a threshold (t_j), i.e., if there is a masquerade attack in session s, The function is equal to 1 if the sum of the inputs is larger than or equal to (t_j) i.e. if s is free of attacks. In learning mode, the neural network uses the sigmoid function to range the outputs between 0 and 1 to adapt to the training phase and the weight adjustments, see Section 5.2

5.1. The detection mode of the TLU

The mathematical model for the TLU output for a session s of user j *is* Y_{sj} and it is defined in Eq. (7):

$$Y_{sj} = F(X_{sj}), X_{sj} = \{DDSGA_{NetScore}, S, P, E\} \tag{7}$$

where F *is a* non-linear activation function implemented by the TLU. This function is denoted by $\varphi(.)$ and it acts as a squashing function, such that the output of a TLU belongs to a given range. In detection mode, the neural network uses the Threshold Function as a transformation function. X_{sj} is the TLU input parameters of session s of user j. The input parameters are:

– $DDSGA_{NetScore}$: is the overall detection score for the active user session according to user audits in both host and NetFlow data,

– S The login source IP address,

– P: The login period,

– E: A Boolean value to signal any login error/failure in the active session.

The last three parameters are collected from the host audits as in Section 3.1. An example of a valid input record is: {0.31, 172.016.114.169, Morning, 0}.

In order to compute the $DDSGA_{NetScore}$ parameter, we use Eqs. (8) and (9) below.

$$P_{Cmasq}(U_j) = \sum_{a=1}^{m} \left(\frac{P(U_j) * P(IP_a)}{\sum_{k=1}^{n} P(U_k)} + P(U_j) \right) \tag{8}$$

$$DDSGA_{NetScore}(U_j) = \theta_j - P_{Cmasq}(U_j) \tag{9}$$

Where:

– $P_{Cmasq}(U_j)$: the probability that the current active session of U_j has masquerade patterns according to U_j behaviors in all cloud VMs. It includes the probability that the masquerader can be detected by the behavior of its login IP(s).

– $P(U_j)$: the probability that the currently active host session of U_j has a masquerade behavior according to U_j behaviors in all cloud VMs as computed by the DDSGA as in Section 3.2. This probability does not include user IP behaviors.

– m: the number of IP(s) that U_j uses to login to the cloud.

– n: the number of cloud users who share the same IP_a of U_j.

– k: an index for the current user who shares the same IP of U_j.

– a: an index for the current IP address of U_j.

– $P(IP_a)$: the probability that IP_a reveals to be a masquerader. It is computed by DDSGA using the current session NetFlow audits as in Section 4.

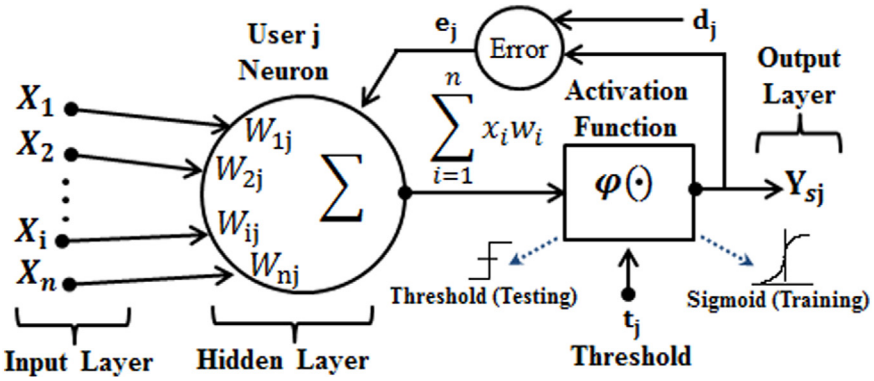

Fig. 26. The neural network model in the training mode for one user.

Consider, as an example, a simple case where U_1, U_2, and U_3 share IP_1 *and* IP_2.Also suppose that the probabilities that IP_1, *and* IP_2 could be used by a masquerader are: $P(IP_1) = 0.4$, and $P(IP_2) = 0.5$. The probabilities that U_1, U_2, and U_3 reveal to be masqueraders according to their behaviors in all the cloud VMs are: $P(U_1) = 0.4$, $P(U_2) = 0.3$, and $P(U_3) = 0.6$, and the detection threshold $\theta_j = 0.75$. We apply the previous equations to compute $P_{Cmasq}(U_j)$ for each U_j to determine which one is a real masquerader according to both the corresponding host and network audits.

$$P_{Cmasq}(U_1) = ((0.4 * 0.4)/(0.4 + 0.3 + 0.6) + (0.4 * 0.5)/(0.4 + 0.3 + 0.6)) + 0.4 = 0.6769 < \theta_j \quad \text{(not masquerader)}$$

$$DDSGA_{NetScore}(U_1) = 0.75 - 0.6769 = 0.0731.$$

$$P_{Cmasq}(U_2) = ((0.3 * 0.4)/(0.4 + 0.3 + 0.6) + (0.3 * 0.5)/(0.4 + 0.3 + 0.6)) + 0.3 = 0.5076 < \theta_j \quad \text{(not masquerader)}$$

$$DDSGA_{NetScore}(U_2) = 0.75 - 0.5076 = 0.2424.$$

$$P_{Cmasq}(U_3) = ((0.6 * 0.4)/(0.4 + 0.3 + 0.6) + (0.6 * 0.5)/(0.4 + 0.3 + 0.6)) + 0.6 = 1.0153 > \theta_j \quad \text{(masquerader)}$$

$$DDSGA_{NetScore}(U_3) = 0.75 - 1.0153 = -0.2653.$$

5.2. The training mode of the TLU

Unlike the unsupervised learning of DDSGA, all TLUs are trained using a supervised learning algorithm Since the TLU contains only one hidden layer with one neuron, a simple learning algorithm such as the Generalized Delta Procedure (GDP) [46] simplifies the learning that is performed off-line, during the training phase, to compute the input weights for each user independently.

The basic idea underlying the GDP is to compute the error distance between the desired response and the actual one. A fraction of this distance is backward propagated through the network. Each neuron in the network uses this fraction to tune its weights and threshold values to reduce the network error for the same inputs. This procedure is repeated until the individual or total errors in the responses become smaller than a specified value. At this point, the neural network has learned the training material and we can use it to produce responses to new input data. The sigmoid function used by the GDP has the benefit of being differentiable. This property is used to find the weight change rule. If the TLU result does not match the desired one for the given input, the weights are adjusted according to Eq. (10):

$$W_{ij} \leftarrow W_{ij} + C * e_j * f(1 - f) * X_{ij} \tag{10}$$

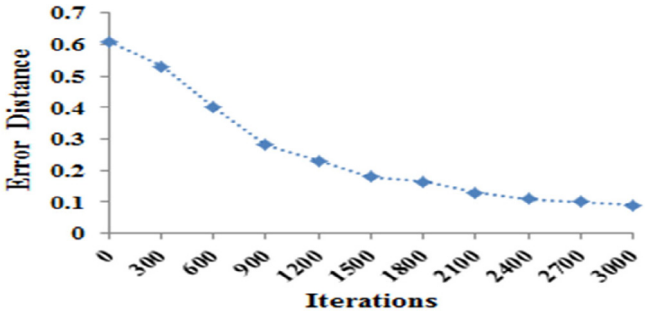

Fig. 27. The effect of learning iterations on the error distance.

$\leftarrow W_{ij} + C * e_j * f(1\text{-}f) * X_{ij}$
where,

- W_{ij}: The weight of input i to the hidden neuron j.
- C: A constant of learning. We take C to be universally 1.
- e_j: The error distance, and $e_j = d_j - f$, d_j is the desired result from the training set (0 or 1).
- f: The actual result from the TLU with the sigmoid function, and f (input) = 1 / (1+e-input), $f(1-f) \rightarrow 0$, where $f \rightarrow 0$ or $f \rightarrow 1$. This means that the weight change can occur only within the 'fuzzy' region surrounding the hyperplane near the point $f = 0.5$.
- X_{ij}: The input i to the hidden neuron j.

The threshold parameter t_j also needs to be tuned during training. By applying the concept of Augmented Vectors [47], we add a new input to the TLU, T_{n+1} and a new weight for this input, W_{n+1}. T_{n+1} always takes on a value 1 and W_{n+1} is adjusted like the other weights. The threshold value t_j is fixed at 0. After the training, we remove T_{n+1} and set t_j to the value of W_{n+1}. The weights and threshold values in the TLU are randomly initialized. We always set a larger weight value for the DDSGANetScore input because this is the input that mostly affects the TLU. The initial weight vector is {0.85, 0.05, 0.05, 0.05}.

Fig. 27 shows the effect of the number of learning iterations on the error distance for local user 2143 using 48 samples of training sessions.

5.3. Performance evaluation of the integrated approach

In the following, we compare the accuracy and the average detection time per session for the three detection approaches neural network, network, and host-based using the Centralized-Backup model with two management VMs. Fig. 28 shows that, as expected, the neural network model results in the longest average

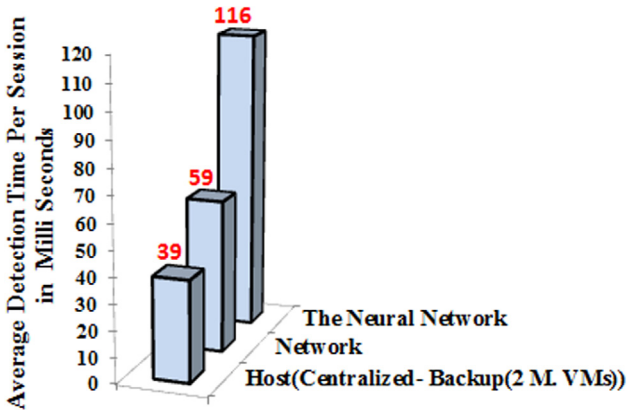

Fig. 28. Average detection time per session in milliseconds in host, network, and the neural network models.

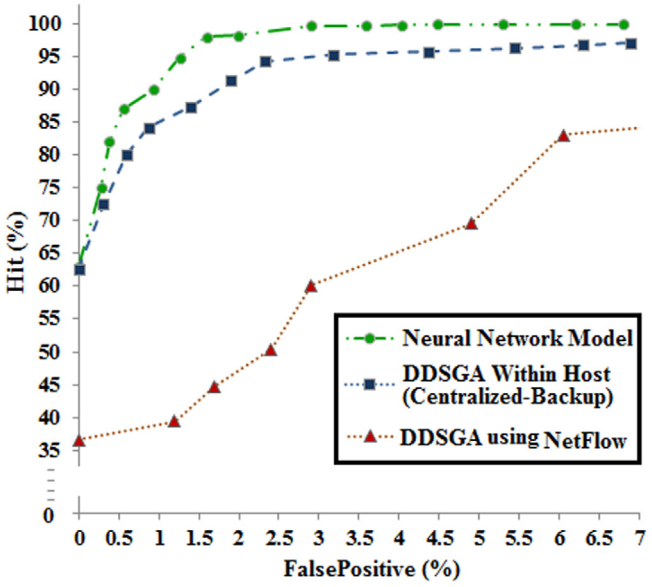

Fig. 29. The ROC curve for the DDSGA approach using the network, host, and neural network approaches.

Table 5
The best accuracy of the three detection approaches sorted by Maxion–Townsend cost.

Approach	False positive %	Hit %	Maxion-T. cost
Neural Network Model	**1.59**	**98.07**	**11.49**
DDSGA within Host system	2.32	94.24	19.68
DDSGA using NetFolw	6.05	83.04	53.25

detection time per session because it integrates the outputs of the other detections. However, the neural network model achieves better accuracy than the other models as shown in Fig. 29 and Table 5.

6. A comparison between our masquerade detection models and SWAD-MMD approach

In this section, we conduct experiments to compare our masquerade detection approaches against the SWAD-MMD approach [16]. The first experiment uses our CIDD dataset and the second uses the Wikipedia dataset [20].

6.1. A comparison between our detection approaches and the SWD-MMD framework using CIDD

The SWAD-MMD framework detects anomalous insider activities via an access network of users and objects using a non-parametric statistical technique called graph-based Maximum Mean Discrepancy. As explained in [16], the SWAD-MMD framework consists of two general modules namely 1) User–Object Relationship Construction Module (UORCM) and 2) Anomaly Detection Module (ADM). The SWAD-MMD framework maintains a buffer of the last n window sample metrics observed, this buffer is called a Sliding Look Back Window (SLBW) which slides window by window during the monitoring process. For example, SLBW = $(W_{t-n+1}, \ldots, W_t)$, where each W_i is an entry at time interval i in SLBW, W_{t-n+1} and W_t are the oldest and current entry in SLBW respectively and n is the size of SLBW. When a new window W_t is added to SLBW, the algorithm computes the statistic of the current window W_t with the remaining n-1 entries (windows) in SLBW. To apply this framework to our CIDD, we first used the UORCM to model the extracted access requests made by the users in our CIDD at any particular time interval from the access request log and build a Bipartite graph [48] (i.e. it represents the relationship between a set of users U and set of objects O). In the graph, users and objects are represented as vertices, and an edge between two vertices E<u_i, o_j, t_k> denotes that user u_i accessed object o_j at any time interval t_k.

To build the access request log linked to each user's profile, we use our BTM to model the three relations that are based upon the System Calls (SC) described at Section 3.1 between users and accessed objects namely: (a) user access a file, (b) process accesses a file, and (c) user invokes a process. The access requests that were made by users at any particular time interval are extracted based on the BTM and are stored in an access request log. After that, UORCM converts the transactions in the request log into a Bipartite graph of users and objects, see Fig. 30. The SWAD-MMD framework maintains an SLBW which slides window by window during the monitoring process. To model the network access requests that were made by users using the NetFlow audits, we use the sequence of destination IP addresses of user accessed machines and their corresponding protocols as access patterns as discussed in Section 4, see Fig. 30.

The second step in this experiment is to feed the output from UORCM into ADM as input. Based on graph-based Maximum Mean Discrepancy (MMD) [16], ADM detects anomalous windows at any time interval t. The ADM checks for deviation of the current window with the entries (recent n-1 entries) in the SLBW. The ADM classifies the current window as either benign or anomalous using a computed threshold. Table 6 shows a comparison between the SWAD-MMD and our detection models.

As shown in Table 6, our detection models outperform the SWAD-MMD framework. In our experiment, we test several SLBW of different sizes to choose the best among them. We noticed that the MMD kernel saturates quickly with large changes in data consistency when the size of input data increases, this results in a reduction in the detection accuracy (e.g., when it is used with NetFlow audits). The main reason is that the MMD kernel gradient vanishes quickly in a deep generative model. Furthermore, the performance of SWAD-MMD slows down when large input data are used. The accuracy of the MMD requires the mini-batch size to be large, which slows down the training and detection by stochastic gradient descent. Table 7 shows that the session length corresponding to the SLBW value is directly proportional to masquerade live time, network overhead, number of management VMs, and detection time.

As Table 7 shows, the best detection accuracy of the SWAD-MMD is achieved with large SLBW = 52. This means that this

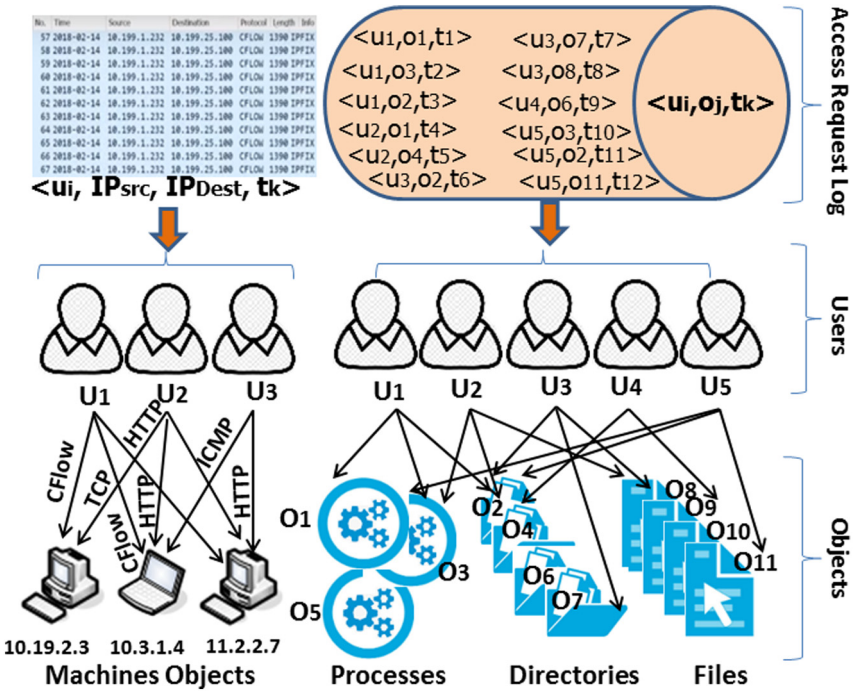

Fig. 30. The Bipartite Graph.

Table 6
A comparison between our masquerade detection models and SWAD-MMD.

Approach	False positive %	Hit %	Maxion-T. cost
Neural Network Model	**1.59**	**98.07**	**11.49**
DDSGA within a Host system	2.32	94.24	19.68
DDSGA using NetFlow	6.05	83.04	53.25
SWAD-MMD (Neural Network)	3.09	91.88	26.66
SWAD-MMD (Host (SC))	3.99	88.7	35.24
SWAD-MMD (NetFlow))	5.88	80.06	55.22

approach uses longer session length than our detection models and consequently, this results in longer masquerade live time, highest network overhead, large number of management VMs, and the longest detection time.

6.2. A comparison between our detection models and the SWAD-MMD approach using Wikipedia dataset

For evaluating the proposed work, we have used two publicly available datasets from Wikipedia networks, articles and metadata. The dataset 1 and dataset 2 conform to the logs of wiki-talk and wiki-talk temporal respectively from Wikipedia [20]. In Dataset 1, each registered user in Wikipedia has a talk page, which they and other users can edit to communicate and discuss updates to various articles. The dataset covers logs from the years 2002 to 2008. It contains 6, 482, 780 revisions, 2, 394, 385 users and 55, 200 articles. For our study, we analyzed revisions documented over 10 weeks during the year 2005. It consists of an average of 2592 users per week, 210 objects per week, and 26, 789 accesses per week. This dataset is called WT Dataset. In the WT dataset, articles represent objects and user revisions represent accesses. Dataset 2 is a temporal network data that represents Wikipedia users editing each other's Talk page. The dataset covers logs for the time span of 2320 days. It contains 7, 833, 140 revisions, 1, 140, 149 users and 64, 700 articles. For our study, we analyzed revisions documented over 70 days. It consists of an average of 452 users per day, 27 objects per day, and 3, 126 accesses per day. This dataset is called WTT Dataset. Where articles represent objects and user revisions represent accesses. In these experiments, we compare the accuracy of our Centralized Backup model using a host system with 4 management VMs against the SWAD-MMD framework. To simplify this comparison, we will only consider the best Precision results that SWAD-MMD achieves when SLBW = 250. To build the access request log linked to each user's profile, the sequences of accessed articles are extracted based on the BTM and are stored in an access request log as explained in Section 3.1. The WT and WTT datasets are used to test and train our model. Using the WT dataset, our model divides the dataset into a total of 2976 windows, where SWS = 90. Out of these 2976, only 477 windows are injected with anomalies (16%). Thus results in 2499 benign windows out of which 2488 have been correctly classified (True Negatives (TN) = 99.55% and FP = 0.45%) and 477 anomalous windows out of which 466 have been correctly classified (TP = 97.90% and False Negatives (FN) = 2.10%). Thus results in a Maxion-T. Cost = 4.80. Using the same WT dataset, SWAD-MMD considers a total of 1072 windows for an SLBW = 250. Out of these 1072, only 172 windows are injected with anomalies (16%). Thus results in 900 benign windows out of which 894 have been correctly classified (TN = 99.33% and FP = 0.67%) and 172 anomalous windows out of which 168 have been correctly classified (TP = 97.67% and FN = 2.33%). Thus results in a Maxion-T. Cost = 6.35.

On the other hand, using the WTT dataset, our model divides the dataset into a total of 2735 windows, where SWS = 80. Out of these 2735, only 390 windows are injected with anomalies (14.25%). Thus results in 2345 benign windows out of which 2339 have been correctly classified (TN = 99.74% and FP = 0.26%) and 390 anomalous windows out of which 383 have been correctly classified (TP = 98.20% and FN = 1.80%). Thus results in a Maxion-T. Cost = 3.36. Using the same WTT dataset, SWAD-MMD considers a total of 875 windows for an SLBW = 250. Out of these 875, only 125 windows are injected with anomalies (14.25%). Thus results in 750 benign windows out of which 746 have been correctly classified (TN = 99.47% and FP = 0.53%) and 125 anomalous windows out of which 122 have been correctly classified (TP

Table 7
A comparison between the performance of our masquerade detection models and SWAD-MMD.

Average detection time (MilliSeconds)					
SWAD-MMD (Neural Network)	149	SWAD-MMD (NetFlow)	86	SWAD-MMD (Host(3 M. VMs))	61
Neural Network (SC)	116	NetFlow (SE)	59	Host(2 M. VMs) (SC)	39
The SLBW and SWS for host Audits		Average Masquerader Live Time (Seconds) in host Audits		Average Transmitted Data Per Session (Bytes)	
SLBW 33		419.56		3632	
SWS (SC) 12		287.34		1961	

Table 8
A comparison between our detection models and the SWAD-MMD using WT and WTT datasets.

	Dataset	Window size	TP	FP	TN	FN	Maxion-T. cost
Centralized-Backup (4M. VMs)	**WT**	**SWS = 90**	**97.90%**	**0.45%**	**99.55%**	**2.10%**	**4.8**
SWAD-MMD	WT	SLBW = 250	97.67%	0.67%	99.33%	2.33%	6.35
Centralized-Backup (4M. VMs)	**WTT**	**SWS = 80**	**98.20%**	**0.26%**	**99.74%**	**1.80%**	**3.36**
SWAD-MMD	WTT	SLBW = 250	97.60%	0.53%	99.47%	2.40%	5.58

= 97.60% and FN = 2.40%). Thus results in a Maxion-T. Cost = 5.58. Table 8 summarizes these results.

The experiments show that our detection models outperform the SWAD-MMD framework as (1) they consider the fluctuating user access patterns in the audit profile of each user, (2) The MMD kernel saturates quickly with large changes in data consistency when the size of input data increases, this negatively impact the accuracy of the detection model. (3) unlike the SWAD-MMD framework that uses a heuristic approach to choose the best size of the SLBW, our detection models use a more efficient approach, the entropy, to choose the best window size to divide the active session. (4) The classification accuracy of the DDSGA approach is higher than the MDD approach.

7. Conclusion and future work

While masquerades attacks are among the top threats of cloud computing, current detection tools do not work efficiently for cloud systems. This is due to the fact that they do not correlate the behavior of a user in distinct environments, host and network, and in a distinct cloud node. To face these issues, first of all, we have extended DDSGA to define two subsystems that detect masquerade attacks through, respectively, system call sequences and NetFlow data. We built a consistent profile of system calls through a "Behaviors Triangle Model" that focuses on specific system calls activities namely, file accesses and process activities. A consistent profile for NetFlow data is built for each source IP address in terms of the sequences of machine accesses in the network and the names of the protocol used for each access. For each subsystem, we have tuned both the scoring system and the sliding window size of DSGGA to achieve better flexibility in the detection and to improve accuracy.

To efficiently define and tune a strategy to correlate the user behavior in distinct cloud nodes and cover distinct cloud deployment models, we have evaluated three correlation models namely, Audit Exchange, Independent, and Centralized-Backup using both CIDS and CIDS-VIRT, two distinct intrusion detection framework. Empirically we have verified that if the cloud user audits are not correlated, the detection accuracy will be very low. The correlation improves the hit ratio by about 19.64% and reduces the Maxion–Townsend cost by 23.24. The experiments prove that the Independent model works much better with CIDS than the Audit Exchange model. The Independent model is the ideal solution for small and private clouds. It achieves good accuracy and computational performance with low network overhead and short masquerade live time. Instead, the Centralized-Backup model works efficiently with CIDS-VIRT in the large cloud with good accuracy and computational performance, low network overhead, and short masquerade live time. After tuning and optimizing the correlation of user behavior in the two detection subsystems, we have integrated their results through a neural network. The resulting system achieves the highest accurate results, 98.07%, with respect to 94.24% of host-based detection and 83.04 of NetFlow based detection. Host-based detection results in the lowest computational overhead, while, as expected, the neural network model results in the largest one because it has to wait for the results of the two subsystems.

For future work, we plan to model user search behavior for masquerade detection, by identifying actions linked to search and information access activities, and use them to build user profiles. This helps in collecting user audits data from distinct cloud operating systems. Furthermore, we plan to use the Deep Learning Neural Network model using the TensorFlow [49] framework to integrate the host and network detections instead of the TLU neural network.

CRediT authorship contribution statement

Hisham A. Kholidy: Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing - original draft, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This research was generously supported in part by the SUNY Polytechnic Institute Research Seed Grant Program.

References

[1] Security guidance for critical areas of focus in cloud computing v4.0': Cloud security alliance, 2017, https://downloads.cloudsecurityalliance.org/assets/research/security-guidance/security-guidance-v4-FINAL.pdf.

[2] C.S. Alliance, Top threats to cloud computing, 2018, [Online]. Available: https://cloudsecurityalliance.org/articles/csa-releases-top-threats-to-cloud-computing-deep-dive/.
[3] M. Schonlau, W. DuMouchel, W. Ju, A.F. Karr, M. Theus, Y. Vardi, Computer intrusion: Detecting masquerades, Statist. Sci. 16 (1) (2001) 58–74.
[4] B. B. Szymanski, Y. Y. Zhang, Recursive Data Mining for Masquerade Detection and Author Identification, in: Proc. of the 5th IEEE System, Man and Cybernetics Information Assurance Workshop, West Point, 2004, pp. 424–431.
[5] S.E. Coull, J.W. Branch, B.K. Szymanski, E.A. Breimer, Sequence alignment for masquerade detection, J. Comput. Stat. Data Anal. 52 (8) (2008) 4116–4131.
[6] A. Garg, R. Rahalkar, S. Upadhyaya, K. Kwiat, Profiling Users in GUI Based Systems for Masquerade Detection, in: The 2006 IEEE Workshop on Information Assurance, pp. 48–54.
[7] E. Imsand, J. Hamilton, GUI Usage Analysis for Masquerade Detection, in: The 2007 IEEE Workshop on Information Assurance, June 20-22, West Point, NY, USA. pp. 270–277.
[8] C. Strasburg, S. Krishnan, K. Dorman, S. Basu, J. Wong, Masquerade Detection in Network Environments, in: The 2010 10th IEEE/IPSJ Int. Symp. on Applications and the Internet.
[9] Hisham A. Kholidy, Cloud Computing Security, an Intrusion Detection System for Cloud Computing Systems (Ph.D. Thesis), https://pdfs.semanticscholar.org/cf8a/14dc638480dbc5304824dd99a631d917d3fe.pdf.
[10] Chalonsh D'silva, Deepika Mulchandani, Rujuta Pimprikar, Sneha Nair, R. Priya, User profiling system for detection of masquerading attack on private cloud, Int. J. Tech. Res. Appl. (ISSN: 2320-8163) 4 (5) (2016) 7–14.
[11] Hisham. A. Kholidy, Fabrizio. Baiardi, CIDD: A Cloud Intrusion Detection Dataset For Cloud Computing and Masquerade Attacks, in: the 9th Int. Conf. on Information Technology: New Generations ITNG 2012, April 16-18, Las Vegas, Nevada, USA. http://www.di.unipi.it/~hkholidy/projects/cidd/.
[12] H.A. Kholidy, F. Baiardi, S. Hariri, et al., A hierarchical cloud intrusion detection system: design and evaluation, Int. J. Cloud Comput. Serv. Archit. (IJCCSA) 2 (2012) 1–24.
[13] Hisham. A. Kholidy, Fabrizio Baiardi, CIDS: A framework for Intrusion Detection in Cloud Systems, in: the 9th Int. Conf. on Information Technology: New Generations ITNG 2012, April 16–18, Las Vegas, Nevada, USA. http://www.di.unipi.it/~hkholidy/projects/cids/.
[14] L. Nkosi, P. Tarwireyi, M.O. Adigun, Detecting a malicious insider in the cloud environment using sequential rule mining, in: 2013 International Conference on Adaptive Science and Technology, P:1-10, 2013, pp. 25–27.
[15] Q. Yaseen, Q. Althebyan, B. Panda, Y. Jararweh, Mitigating insider threat in cloud relational databases, Secur. Commun. Netw. 9 (10) (2016) 11321145.
[16] G.S. Smrithy, Alfredo Cuzzocrea, Ramadoss Balakrishnan, Detecting insider malicious activities in cloud collaboration systems, Fundam. Inform. j. (2018) http://dx.doi.org/10.3233/FI-2018-1704, IOS Press, July 6.
[17] A. Duncan, S. Creese, M. Goldsmith, A combined attack-tree and kill-chain approach to designing attack-detection strategies for malicious insiders in cloud computing, in: 2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security), Oxford, United Kingdom, 2019, pp. 1–9, http://dx.doi.org/10.1109/CyberSecPODS.2019.8885401-.
[18] Cloud Simulator, https://www.superwits.com/library/cloudsim-simulation-framework.
[19] Q. Yaseen, Y. Jararweh, B. Panda, et al., An insider threat aware access control for cloud relational databases, Cluster Comput. 20 (2017) 2669, http://dx.doi.org/10.1007/s10586-017-0810-y, 2017.
[20] J. Leskovec, A. Krevl, SNAP Datasets: Stanford Large Network Dataset Collection, 2014, http://snap.stanford.edu/data.
[21] Arthur Gretton, Dino Sejdinovic, Heiko Strathmann, Sivaraman Balakrishnan, Massimiliano Pontil, Kenji Fukumizu, Bharath.K. Sriperumbudur, Optimal kernel choice for large-scale two-sample tests, Adv. Neural Inf. Process. Syst. (2012) 1205–1213.
[22] Martin Arjovsky, Soumith Chintala, Bottou Léon, Wasserstein gan, 2017, arXiv preprint arXiv:1701.07875.
[23] Y. Li, K. Swersky, R.S. Zemel, Generative Moment Matching Networks, In ICML, 2015, pp. 1718–1727.
[24] http://www.schonlau.net/intrusion.html.
[25] S. Greenberg, Using Unix: Collected Traces of 168 Users, Report 88/333/45, Univ. of Calgary, 1988.
[26] T. Lane, C.E. Brodley, An application of machine learning to anomaly detection, in: Proc. of the 20th National Information Systems Security Conference, 1997, pp. 366–380.
[27] Hisham A. Kholidy, Autonomous mitigation of cyber risks in the cyber–physical systems, Future Gener. Comput. Syst. (ISSN: 0167-739X) 115 (2021) 171–187, http://dx.doi.org/10.1016/j.future.2020.09.002, http://www.sciencedirect.com/science/article/pii/S0167739X19320680.
[28] Hisham A. Kholidy, Abdelkarim Erradi, Sherif Abdelwahed, Fabrizio Baiardi, A risk mitigation approach for autonomous cloud intrusion response system, J. Comput. (2016) 0495–0498, http://dx.doi.org/10.1007/s00607-016-.
[29] Hisham A. Kholidy, Abdelkarim Erradi, Abdelkarim Erradi A Cost-Aware Model for Risk Mitigation, in: Cloud Computing SystemsSuccessful accepted in 12th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA), Marrakech, Morocco, November, 2015.
[30] Hisham A. Kholidy, Ali Tekeoglu, Stefano Lannucci, Shamik Sengupta, Qian Chen, Sherif Abdelwahed, John Hamilton, Attacks Detection in SCADA Systems Using an Improved Non-Nested Generalized Exemplars Algorithm, in: The 12th IEEE International Conference on Computer Engineering and Systems (ICCES 2017), December 2017, pp.19–20.
[31] Qian Chen, Hisham A. Kholidy, Sherif Abdelwahed, John Hamilton, Towards Realizing a Distributed Event and Intrusion Detection System, the International Conference on Future Network Systems and Security (FNSS 2017), Springer, Industrial control system (ics) cyberattack datasets, Gainesville, Florida, USA, 2017, p. 31, http://www.ece.uah.edu/~thm0009/icsdatasets/PowerSystem_Dataset_README.pdf.
[32] Hisham A. Kholidy, Abdelkarim Erradi, Sherif Abdelwahed, Abdulrahman Azab, A Finite State Hidden Markov Model for Predicting Multistage Attacks in Cloud Systems, in: The 12th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC), Dalian, China, 2014.
[33] http://www.ossec.net/main/.
[34] http://www.snort.org/.
[35] H. Debar, D. Curry, B. Feinstein, The Intrusion Detection Message Exchange Format (IDMEF), RFC 4765exp, 2007.
[36] N. Nguyen, P. Reiher, G. Kuenning, Detecting Insider Threats by Monitoring System Call Activity, in: Proceedings of the 2003 IEEE Workshop on Information Assurance, NY, 2001.
[37] E. Eskin, W. Lee, S.J. Stolfo, Modeling system calls for intrusion detection with dynamic window sizes, in: Proceedings of DARPA Information Survivability Conference and Exposition II, Anaheim, CA, 2001.
[38] http://en.wikipedia.org/wiki/System_call.
[39] R. Sekar, T. Bowen, M. Segal, On Preventing Intrusions by Process Behavior Monitoring, in: The 1st USENIX Workshop on Intrusion Detection and Network Monitoring, 1999.
[40] D. Wagner, J. Foster, E. Brewer, A. Aiken, A First Step Towards Automated Detection of Buffer Overrun Vulnerabilities. in: Proceeding of the Year 2000 Network and Distributed Systems Security Symposium, 2000.
[41] Hisham A. Kholidy, Fabrizio Baiardi, DDSGA: A data-driven semi-global alignment approach for detecting masquerade attacks, IEEE Trans. Dependable Secur. Comput. (2015) 164–178, http://dx.doi.org/10.1109/TDSC.2014.2327966.
[42] E. Eskin, W.N. Grundy, Y. Singer, Protein family classification using sparse markov transducers, in: Proceedings of the Eighth Int. Conference on Intelligent Systems for Molecular Biology, Menlo Park, CA, 2000.
[43] Christopher D. Brown, Herbert T. Davis, Receiver operating characteristic curves and related decision measures: a tutorial, Chemometr. Intell. Lab. Syst. 80 (2006) 24–38.
[44] R.A. Maxion, T.N. Townsend, Masquerade Detection Using Truncated Command Lines, in: The Int. Conf. on Dependable Systems and Networks, Washington, D.C., 2002.
[45] A. Whisnant, S. Faber, Network Profiling using Flow, Technical Report, CMU/SEI-2012-TR-006, 2012.
[46] http://www.seeingwithc.org/topic5html.html.
[47] http://en.wikipedia.org/wiki/Augmented_matrix.
[48] Y. Chen, B. Malin, Detection of anomalous insiders in collaborative environments via relational analysis of access logs, in: Proceedings of the First ACM Conference on Data and Application Security and Privacy, ACM, 2011, pp. 63–74, http://dx.doi.org/10.1145/1943513.1943524.
[49] The tensorFlow framework, https://www.tensorflow.org/.

Hesham AbdElazim Ismail Mohamed (Hisham A. Kholidy) received his Ph.D. in Computer Science in a joint Ph.D. program between the University of Pisa in Italy and the University of Arizona in the USA. He works as an assistant professor at the Department of Networks and Computer Security (NCS), College of Engineering, State University of New York (SUNY) Polytechnic Institute. Prior to that, he worked as a postdoctoral associate at the University of Nevada, Reno and Mississippi State University. During his Ph.D., he worked as an associate researcher at the NSF Cloud and Autonomic Computing Center, Electrical and Computer Engineering Dept. at the University of Arizona. He holds several patents in Cybersecurity published by United States Patent and Trade Mark Office (USPTO), and he published more than 30 papers on high quality journals and conferences including IEEE transactions, FGCS, and Springer journals. He participated as PI, Co-PI, and senior personnel in several research projects and his research interests include Cybersecurity and SCADA systems security, 5G systems security, Cloud Computing and high-performance systems, service composition, and big data analytics.