Advances in Science, Technology and Engineering Systems Journal Vol. 2, No. 3, 449-459 (2017)
www.astesj.com
Special issue on Recent Advances in Engineering Systems

ASTESJ
ISSN: 2415-6698

Verifying the Detection Results of Impersonation Attacks in Service Clouds

Sarra Alqahtani*, Rose Gamble

Tandy School of Computer Science, University of Tulsa, Tulsa, OK, USA

ARTICLE INFO

Article history:
Received : 05 April, 2017
Accepted : 04 May, 2017
Online: 24 May, 2017

Keywords :
Impersonation
Cloud
Cybersecurity

ABSTRACT

A web service impersonation is a class of attacks in which an attacker poses as or assumes the identity of a legitimate service to maliciously utilize that service's privileges. Providing security for interacting cloud services requires more than user authentication with passwords or digital certificates and confidentiality in data transmission. In this paper, we focus on the service cloud model, which facilitates the composition and communication among web services owned by different cloud vendors. We develop a distributed mechanism to detect and mitigate impersonation attacks against web services in the cloud. The detection approach monitors the behavior of each service and identifies anomalies as a potential impersonation attack if it deviates significantly from the expected behavior. To verify the impersonation attack, we deploy a cloud-based verification technique, misleading suspicious services with useless responses. The experimental results show that modeling request behavior reliably detects a significant number of impersonation attempts, with a performance degradation that is a reasonable trade-off.

1. Introduction

A web service impersonation is a class of attacks in which an attacker poses as or assumes the identity of a legitimate service to maliciously utilize that service's privileges. This attack can cause a serious threat to the security of service clouds, i.e. those clouds that allow the provisioning of multiple vendor services to dynamically compose an application to answer a client request. By impersonating a legitimate service, the intruder can maliciously access the victimized a service's resources. For the service cloud, this attack class is a version of identity theft. Such attacks completely undermine traditional security mechanisms due to the trust imparted to service credentials once they have been authenticated. Many attempts have been made at detecting this kind of attack (also called masquerader or identity spoofing attacks) against legitimate users in clouds. In this dissertation, we have developed a distributed mechanism to detect and mitigate impersonation attacks against web services in the cloud [1].

When using the cloud, an end user's credentials to authenticate request messages between web services lacks the verification of the origin of the request (i.e. the requester service). Thus, this vulnerability can be used by the attacker after stealing the credentials to create fake services. Since the credentials represent a long-term authentication tool, the attacker can attack a web service for unlimited time and the owner of the credentials would not discover the problem until the damage is done.

In our proposed architecture (Figure 1) [1], we assume a secure session with a security token issued by a Security Token Service (STS) to mitigate the general vulnerability of spoofing the user credentials. The STS is trusted by both the client and the web service to provide interoperable security tokens. The client sends an authentication request, with accompanying credentials, to the STS. The STS verifies the credentials presented by the client, and then in response, it issues a security context token (SCT) that provides proof that the client has authenticated with the STS. The SCT is built on a SAML standard format for exchanging authentication and authorization data between different parties [1, 2]. The client presents the security token to the service. The service then verifies the token with the STS, which proves that the client has been successfully authenticated. For proper use of the token, it is expected that all composed, trusted services communicate with the STS to perform token delegation and validation at each service request and response, but only client services issue or cancel a token. The SCT holds information to specify its scope, creation, and expiration time to develop the basis for encrypting and signing subsequent message exchanges, which results in efficient and secure communications between services in an application. The cloud services and STS use X.509 certificates to sign and encrypt their messages. The scope of the issued SCT is limited to the designated STS regardless of whether the sender service specified

*Corresponding Author: Sarra Alqahtani, Tandy School of Computer Science, University of Tulsa, Tulsa, OK, USA | Email: sarra-alqahtani@utulsa.edu

the scope in the initial request. This prevents the sender from using the SCT to directly access a recipient without communicating first with STS. Therefore, a token's lifetime is limited to its session active time.

The proposed architecture introduces the concept of scoped distributed Security Management Databases (SMDBs) to store audit logs, meta-information about the infrastructure of each scope, history of security incidents, incident reports, related security policies, and SLAs [2]. Scoped detectors use SMDB content to identify security threats exploiting the scoped content. We have shown in [3] how CAPEC attack patterns fit into the evidence collection and evaluation strategy using the proposed architecture.

For the proposed system, we implement our solution using Apache Web Service Security for Java (WSS4J) to secure the deployed cloud services and their messages within a provisioned application given the primary security standards. WSS4J supports the encryption of messages using X.509 certificates of services. Furthermore, it prevents replay attacks that can happen when the attacker sniffs a message and resends it to the same recipient. Apache WSS4J makes all security tools available for developers to secure their services against impersonation attacks by using strict authentication standards. When a service passes the identity authentication it is fully trusted in the application. However, if the service identity has been compromised for any reason, WSS4J cannot detect the impersonation attack or even log its traces in the log file. Consequently, the impersonated service will be authenticated and the attacker can illegally gain access to potentially private information.

The impersonation attack is not only limited to the SOAP messaging protocol in service clouds. Any type of communications between services that requires request authentication is vulnerable to impersonation attacks. Authentication in REST services does not have a standard policy or rules. However, the most common used approaches for REST request authentication are HTTP AUTH and tokens. In HTTP AUTH, the client service needs to compute the Base64 encoding of its credentials and include them in each future HTTP request to the server using the "Authorization" HTTP header. This header can be encrypted using the server public key and the client private key. This approach is very similar to the approach used in the SOAP protocol by using the SOAP header to carry authentication information to the destination. Both HTTP and SOAP headers can be faked by the attackers who illegally obtain a copy of the legitimate client service's identity.

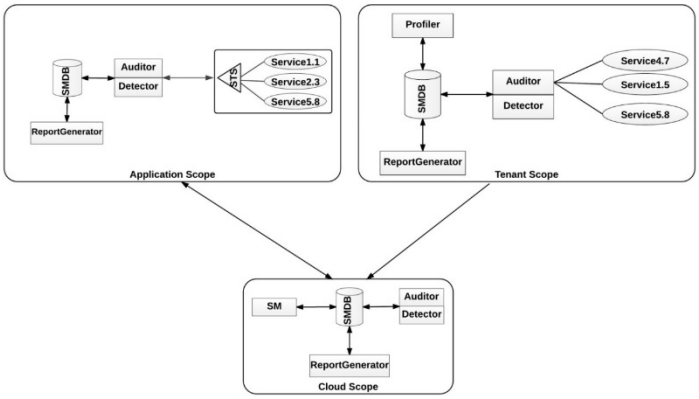
The second authentication approach used in RESTFUL clouds is to create a dedicated login service similar to a STS that accepts credentials and returns a token. This token should then be included, as a URL argument, to each following request. The attacker can also get a valid token by using the stolen credentials. Consequently, regardless of the type of communication messaging used between services, it is possible for an attacker who has a stolen identity to successfully pass the authentication step and get trusted in the target server.

The Amazon REST APIs use a custom HTTP scheme based on a keyed-HMAC (Hash Message Authentication Code) to authenticate REST requests. To authenticate a request, the client needs first to concatenate selected elements of the request to form a string. Then, it must use its AWS secret access key to calculate the HMAC of that string. Informally, this process is called "signing the request," and the output of the HMAC algorithm is called the signature, because it simulates the security properties of a real signature. Finally, the client, either a human user or another service, has to add this signature as a parameter of the request by using the syntax described by the server. The risk stems from the leaked AWS secret access key, which happened recently for many customers of Amazon clouds. Hence, the REST messaging architecture is also vulnerable to impersonation attacks if the authentication identity has been compromised. In this paper, though we will focus on detecting the attack by using SOAP protocol, the solution design can be extended to work with REST services.

In this paper, we hypothesize that the legitimate service can obtain sufficient information from the cloud regarding the successor service classes that it often communicates with when provisioned for certain application. An impersonated service, on the other hand, would likely have a more chaotic and random behavior by communicating more extensively with known successor services and more broadly to services not often communicated with in a manner that is different than the victim service being impersonated [4]. Thus, our detection approach focuses on monitoring a service's behavior in real time to determine whether current service actions are consistent with the service's profiled behavior.

To reduce the false positive rate, which is the main drawback of the anomaly detection systems, we combine the detection of anomalous behaviors with a cloud-based verification technique. The verification technique prevents the risk of revealing private data to impersonators and, at the same time, verifies the identity of the malicious service. When the verification algorithm positively verifies the detection results, it returns a fake response to delude the attacker.

The major contribution of this paper is our distributed framework for auditing and detecting impersonation attacks in service clouds. We design a distributed impersonation detection approach within the framework that is based on profiling each service's request behavior. This behavior reveals that monitoring the general features of the SOAP request messages between web services can achieve a reasonable detection rate with minimal false positive alarms. In order to increase the accuracy of the detection system, we design an aging method to determine when the profiles need to be updated and how to adjust them so they are more heavily influenced by the most recently observed behavior. Moreover, we manifest the false positives and decrease the impersonator live time inside the cloud by re-authenticating the suspicious service using a novel verification technique. Since the credentials of the suspicious service may be spoofed by the impersonator, our verification technique uses a predefined fake response to validate the identity of the suspicious service that exhibits a significant behavior deviation. If the suspicious service accepts the fake response by the suspicious service, then the detection system has correctly detected the impersonator. Otherwise, the suspicious service is considered to be a legitimate service that has been incorrectly classified as an impersonator because it changed its behavior without updating its profile.



Our experimental results show that modeling service behavior reliably detects a high percentage of impersonated services with no false positives. The limited set of statistical values used for service behavior modeling results in an acceptable performance. However, in the case of mimicry attacks where the impersonator fully mimics the victim service's behavior, the detection algorithm cannot detect the malicious requests. Resolving such attacks and improving the algorithm performance are future work.

The paper is organized as follows. The threat model is described in Section 2. The literature review of impersonation detection techniques is presented in Section 3. Section 4 demonstrate the detection approach used in this research. Section 5 discusses the detection results. Section 5 concludes the article with some suggestions.

2. Threat Model

2.1. Vulnerabilities

The standard approaches of identity verification including authentication and encryption have been shown to fail to detect impersonated services for a variety of reasons, including insider attacks, misconfigured services, faulty implementations, buggy code, and the creative construction of effective and sophisticated attacks not envisioned by the implementers of security procedures. Attack methods such as phishing, fraud, and exploitation of system vulnerabilities still achieve results in service clouds. Credentials and passwords are often reused, which amplifies the impact of such attacks.

The Heartbleed bug leaked private information such as encryption keys and credentials from the service's owner server. Leaked secret keys allow an attacker to decrypt any past and future traffic to the protected services and to impersonate the services as well. Therefore, any protection given by the encryption and the signatures in the X.509 certificates can still be bypassed. Mitigating bugs like Heartbleed is out of the service owner and cloud provider control, which means we need to design another line of defense to mitigate the risk of "bleeding" the encryption keys (X.509) and service credentials. In this article, we introduce a preventative approach that distinguishes between requests coming from legitimate services and those coming from fake services with impersonated or spoofed credentials to block them and alert the victim's owner to revoke the stolen keys.

Different incidents happened in Amazon cloud prove that the classical authentication and authorization techniques are not safe enough to fully trust all services holding valid identities. In April 2010, Amazon experienced a Cross-Site Scripting (XSS) bug that allowed attackers to hijack credentials from the site. In 2009, numerous Amazon systems were hijacked to run Zeus botnet nodes. Also, in April 2014, some of AWS clients claimed about very high bills due to their credentials were compromised.

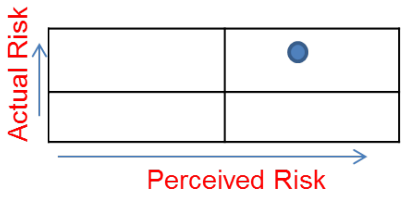
2.2. Target Assets

The expected target of the impersonation attack is the legitimate service's identity.

2.3. Impact

According to the CSA in [5], the risk analysis technique CIANA classifies the impact of service hijacking or impersonating as a combination of authenticity, integrity, confidentiality, non-repudiation, and availability threat. On the other hand, STRIDE risk analysis puts the service impersonation under tampering with Data, repudiation, information disclosure, elevation of privilege, and spoofing identity. Therefore, service impersonation attack causes a very high risk on the cloud from all aspects of security.

According to a survey conducted by CSA the extended CSA Top Threats Working Group [5] , the impersonation threat is still relevant to clouds by 87%. Also, the data leakage was ranked as the third top threat in 2013.The risk matrix for service impersonation is depicted in Figure 2.



2.4. Scenarios

The attacker who exploits a bug similar to Heartbleed and has a copy of the encryption public and private keys and the credentials of the victim service (we use identity to describe these information) can effectively perform a service impersonation attack against the victim service. Then, it can illegitimately use the stolen identity to gain access to other services in the service cloud by following the listed steps:

1. The attacker builds a fake web service by using the stolen identity to impersonate the victim service.
2. During an active session of an application that has the victim service provisioned within it, the attacker uses sniffer software to steal a valid token sent from the STS to the victim service. The intercepted token is encrypted by the public key of the victim service. Since the attacker has the encryption keys of the victim service, he can decrypt the token.
3. The attacker uses the stolen token along with the victim identity to ask the STS to delegate the token to the target service of the attack. The target service must be provisioned within the same application with the victim service.
4. The STS authenticates the impersonated service and replies with a delegated token.
5. The attacker finds the interface of the target web service and its entry points (i.e., methods) by discovering its public WSDL file.
6. The attacker uses the delegated token and the stolen identity to create and encrypt a request to the target service.

7. The target web service validates the token of the impersonated service with the STS.
8. The STS authenticates the token and sends the confirmation to the target service.
9. The target service decrypts and checks the structure of the message.
10. The target service responds to the impersonated service with a valid encrypted response SOAP.
11. The attacker decrypts the received response using the victim private key.

With this scenario, private information can be revealed to illegitimate services by trusting the presented identity without doing a more thorough check. The success of this attack can trigger several attacks, such as DDoS attacks as we have explained in [6], to abuse the availability of the target service and the integrity of the victim service.

3. Related Work

There are two general approaches for intrusion detection: misuse detection and anomaly detection [7]. The misuse detection uses the knowledge accumulated about attacks and checks for signatures of these attacks while the anomaly detection builds a reference model of the usual behavior of the system being monitored and checks for deviations from the observed usage. The false positive rates of misuse detection are lower than that of anomaly detection [8] but anomaly detection has the advantage of detecting previously unknown attacks. In impersonation attack detection, the intrusion detection system cannot get the signatures of attackers until the attackers have been detected by the system. Therefore, the attack model cannot be constructed in advance; hence we prefer to use anomaly detection approach in this context.

In the literature, anomaly detection was implemented in a variety of approaches. These approaches are usually categorized into two groups, i.e. statistical approaches and machine learning approaches [8]. In statistical approaches, anomaly detection systems usually watch behaviors of observed objects to comprise statistical distributions as a set of trained profiles during the training phase. These systems then apply the set of trained profiles by comparing them against a new set of activities of observed objects during runtime. An anomaly is detected if there is a significant deviation resulted from the observation. In general, any incident whose frequency goes beyond a predefined standard deviation from statistical normal ranges raises an intrusion alarm [7].

Machine learning based approaches tend to reduce the supervision costs during the training phase of statistical approaches by enabling detection systems to learn and improve their performance on their own. Neural networks and Hidden Markov Model have been proved to be useful techniques at the network traffic level as shown in in [9] and [10]. However, using ML algorithms in a highly dynamic environment like service clouds have several drawbacks such as increasing performance overhead, storage requirements, and computational expense [11].

In the context of this paper, we focus on statistical approaches for detecting impersonation attacks. In [8,12-15], statistical algorithms are applied to detect anomalous patterns in the system. They use the normal behavioral profiles of the monitored object, which is either a user or system process. Unlike these systems, this research applies anomaly detection at the web service level. Rather than profiling the normal behavior of users or systems, our approach profiles the normal request behavior of web services. Hence, the detection system deals with the web service as an independent object that can be impersonated irrespective of its owner's behavior.

Researchers in [16] provide an intrusion detection framework for cloud systems targeting masquerader attacks. Each user has a profile to model its behavior and make it available across the cloud. To measure the deviation between an activity initiated by a normal user and activity initiated by the masquerader, they use the Data-Driven Semi-Global Smith Waterman alignment algorithm. The objective is to compute the best alignment score, by aligning the active user's session sequence (e.g., mouse movements, system calls, opened windows titles, etc.) to the previous stored sequences for this user. It achieves good results in detecting masqueraders in clouds with detection accuracy of 88.4 % and a low false positive rate of 1.7 %. However, the focus on monitoring human users is insufficient for detecting service impersonations in service clouds. Service clouds are mainly composed of collaborative services from different administrative domains with different security policies. Researchers in [16] do not address attacks that might be triggered by impersonated services with legitimate identities.

The detection approach presented in [17] detects insider masqueraders in file systems. This approach tracks and measures changes in user behavior, alerting on any significant changes. They use one-class support vector machines to develop user behavior models and a set of data features related to the file systems including: the process name, the process path, the parent of the process, and the type of process action, the process command arguments. Their experiments show that modeling search behaviors of genuine users reliably detects all masqueraders with a very low false positive rate of 1.1%. Despite the fact that they do not test this approach on clouds, it may be applicable for only storage clouds. Hence, their detection system is not applicable for the service cloud model we work with. However, our verification technique is inspired by their proposed decoy technique, which launches disinformation attacks against malicious insiders, preventing them from distinguishing the real sensitive data from fake worthless data. They place traps within the file system which are documents downloaded from the Internet including several types of useless documents such as tax return forms, medical records, credit card statements, e-bay receipts, etc. The decoy files are downloaded by the legitimate user who owned the system and placed in highly-conspicuous locations that are not likely to cause any interference with the normal user activities on the system. A masquerader, who is not familiar with the file system and its contents, is likely to access these decoy files, if he or she is in search for sensitive information. Their decoy technology is more suitable for file systems or storage clouds more than service clouds. Hence, we design a service-cloud-based decoy technique as our verification technique to mislead the impersonators with fake information.

An anomaly based network intrusion detection system for Remote to Local (R2L) attacks is proposed in [15]. R2L attacks usually exploit a vulnerability in a service at the target machine to elevate the attacker's privileges. This attack is similar to the impersonation attacks occurring in clouds. The proposed detection system consists of two logical modules, the Packet Processing Unit (PPU) and the Statistical Processing Unit (SPU). The PPU has to extract service requests from the stream of packets on the wire to pass them to the SPU. The task of the SPU is to read the service

requests and extract suitable statistical input data. Requests are divided into several groups where each group contains request types with similar statistical properties such as Get and Post protocols. The requests of each group are then analyzed independently. The properties of a request used to determine its anomaly score include: type of request, length of request, and payload distribution. The anomaly score of a service request is the weighted sum of the three scores computed for each of the previously mentioned properties. Finally, the anomaly score is compared to a threshold that can be manually set by the security administrator. The presented detection system has detected all anomaly requests with a very slight false positive rate. Since the anomaly score technique proposed for network packets, we adjust it to detect impersonators in service clouds by computing the anomaly score of SOAP request messages exchanged between services.

In [14], the detection system concentrates on profiling the behavior of system processes by studying the set of system calls made by the program. The designed algorithm compares the sequence of system calls captured during online usage to the normal sequences profiles during the training phase. The detection accuracy was 72.2% with a false positive rate of 2.1%.

4. Impersonation Attack Detection Approach

When dealing with the impersonation attack detection, it is important to mention that we assume the impersonator has already obtained the required security parameters including the victim service's credentials, the security token, and the encryption keys to get involved in the cloud application and communicate with its services (explained above). We also assumed that the attack targets are only web services. Hence, SMDBs, detectors and other entities are assumed to be sufficiently secure.

When presenting the stolen identity, the impersonated service appears as a legitimate service with the same access rights as the victim service. Ideally, monitoring a service's behavior after being granted access is required in order to detect such attacks. We find that certain types of service activities can reveal the service intent. For instance, SOAP requests used for web service communication offer an interesting behavior to monitor. A service request contains the sender supplied data which is sent over the network to another provisioned service in the same application session to perform a single task on behalf of that sender, which is either a human user or another service. This data shows the intent of the service trying to gain secure information from different services in a very short period of time before the stolen security token expires and before the victim finds out about its stolen identity.

The developed approach composes of three steps. The output of each step represents the input to the next step (Figure 2):

A. Auditing services' messages by the auditing system during the training phase.

B. Profiling the request behavior of each service in a profile document before running the detection algorithms during runtime using log files.

C. Detecting any significant change between the profiled and live behaviors then verifying the identity of the malicious service by using the verification technique.

To assess the feasibility of our approach within the service cloud model, we craft an easy to understand case study with a deployed service cloud hosting 25 distinct web services with 15 tenants owning and offering these services. We implement a session manager randomly chooses among appropriate services and composes them to provision an application that satisfies a user's request. A service cloud receives the traveler requirements with the preferred maximum price and the cloud returns the most suitable travel package with allowed payment. Table 1 shows the services and tenants used in the case study to evaluate the proposed impersonation attack detection approach. The cloud employs different meta-services to manage the travel reservations. These services are a Travel Agency service, Transportation services, and Living services.

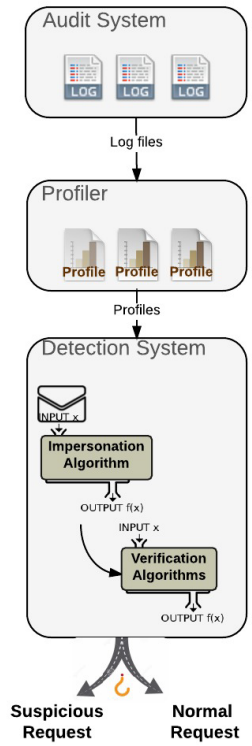


Figure 3: Impersonation detection approach

Table 1: The available services in the case study

Tenant Class	Web Services
Banks	CreditChecking, Payment, CreditCard, paymentService, CreditVerification,CreditPayment, CheckBalance
Airlines	FlightFinder, FlightService, CheapFlight, TicketFinder, FlightFinder
Car Rentals	RentalAgent, RentalService, CarRental, RentalService, RentCar
Hotels	RoomBooking, RoomLocator, Accom, AvaRoom, BookingService, FindRoom
Weather	WeatherRetriever, Forecast

The Session Manager (SM in Figure 1) dynamically provisions services from tenants into a service chain to simulate a travel management system as in Figure 4. Given a client request, the provisioned services buy a flight ticket to a specific destination at a specific time, reserve a hotel room, rent a car, retrieve the forecast for the desired travel dates, and pay the total price from the client's bank account. The resulting service chain consists of at least 9 services from at least 5 tenants.

4.1. Auditing System

In our service cloud architecture [3], the impersonation detection algorithm is distributed across a tenant scope, session or application scope, and cloud scope as depicted in Fig. 1. This distribution allows for load balancing among services logging events, tenants generating profiles of all of its hosted services, applications executing service chains to satisfy clients' requests and detect impersonation attempts, and the cloud verifying the extent of the attack.

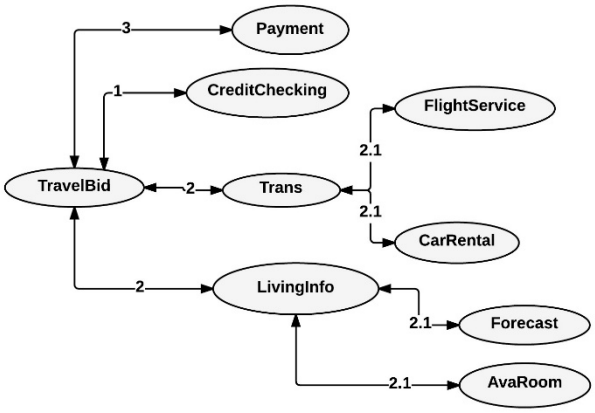


Figure 4: A scenario of Travel Package Bidder

Event logging at each service is done by its tenant's Auditor. The Auditor is a web service interceptor attached to each service to monitor its request and received messages. By intercepting SOAP requests, the Auditor is able to collect important information about the recipient to sufficiently check the sender's request behavior. Such information includes recipient services and their business classes, requested methods in these recipients, request size, and other critical details. The advantage of this approach is that the logging process created and managed independent of the service process code to protect the auditing process from intruders. Since the tenant auditor would not be able to log requests issued by the impersonated service, checking the victim service's log file for the suspicious events helps to detect the attack at early stages.

At the application scope, the STS log file retains messages communicated by the client and services as it manages the creation, delegation, validation, and cancellation of the security token. By logging these details, we can corroborate the STS log file with the victim service's log file (the service whose credentials have been stolen) to reveal any missing events [1] as significant evidence of an impersonation attack.

The cloud scope gathers centralized forensic investigation information and examines it for impersonation attacks. During the detection process, the cloud scope correlates alerts received from any affected application scope and compares the malicious request of the impersonated service with the victim service's log file. The final decision about the malicious request is logged by the cloud Auditor at the cloud scope.

4.2. Profiler

The detection system detects an impersonated service by comparing its observed request to its expected behavior using certain metrics that are defined in its profile. The expected behavior can be deduced during the training period. Since the manual creation of expected behavior is cumbersome, we use the training period for each service to collect information about its behavior. During training, the auditing system logs enough information about request events for each service. Extracted data about requests from a specific service's log file are not classified in advance, meaning that the frequent requests represent the general behavior of the service. Hence, during runtime, received requests are evaluated against a summary of the training data (i.e. profile) to measure the deviation from the normal behavior.

Profiling the services' behavior is done in two phases: profile generation and profile aging. Each phase is discussed in detail in next subsections.

4.2.1. Profile Generation

Instead of building the detection system by evaluating the received request during runtime against all requests in the sender's log file, we summarize these records in profile documents. Thus, the detection time dramatically decreases which leads to an improvement in the performance. The Profiler uses the logged data at the tenant scope to generate the service's request behavior profile. Then, SM makes the services' profiles available to the involved application detector.

The profiler depends on a statistical model to generate the request behavior profile from the service log file. This model consists of different attributes extracted from the SOAP request without processing its payload. Hence, the model reduces the cost of decrypting payloads of malicious requests that would be declined at the end. The used attributes are recipient's class, SOAP size, method, and historical incidents. We found these attributes sufficient to reveal any deviation from the normal request behavior.

1) Recipient's Class:

We hypothesize that each service has a logical "preferred" set of business classes where it sends requests based on its business purpose. For example, flight reservation services always communicate with different services in the banking and airline classes to buy a ticket for the client. In the statistical model, we compute how frequent the recipient's class has been accessed by the sender during the training phase. The probability of relevant frequency is used to compute the anomaly score (AS) of recipient's class as follows:

$$AS_{class} = - \log_2 (P[c]) \quad (1)$$

Equation (1) assigns higher anomaly scores to requests sent to recipients with less frequent classes. The probability P[c] represents the relative frequency that a request with class c has occurred during the training period of the sender. In order to prevent too high anomaly scores (or even infinite values) for the class that has been requested by the sender very infrequently (noise) or not at all during the training period, the probability of each request class needs to be set to a minimum value depends on the service characteristics and it might be adapted from time to time. Since this minimum value yields to a maximal anomaly score for classes that never received requests from the sender, it is necessary to maintain the same maximum anomaly score across all attributes of SOAP requests to keep the total anomaly score consistent.

2) SOAP Size:

The size of a SOAP request can be a good indicator of the reliability of its source. The structure of a SOAP request consists of optional elements that the service can include, such as SOAP Header and SOAP Fault. Usually, the size of the SOAP message does not vary much between requests that are generated by services in a certain class to a specific recipient. The situation looks different when requests carry input wanting to obtain information from the target recipient. For instance, the attacker may send a request with nested queries to get more information from the recipient's database. To assess this attribute properly, we correlate SOAP size with either the recipient service or class based on some properties of the observed request.

The sender's profile keeps track of all recipient services and their classes that have ever received requests from that sender along with the average size of these requests. If the current recipient of the observed request has been accessed by the sender over an acceptable rate of requests then the available information about the requests sent by the sender to this recipient would be sufficient to evaluate the size of the received request. Otherwise, the size of requests received by the current recipient's class (received by any service in this class) would be used to compute the size attribute.

We calculate the anomaly score for SOAP size as in [15], by using the mean (μ) and the standard deviation (σ) of the sizes of the requests that have been sent by the sender during its training period to either a specific recipient or class. These two values are stored in the sender's profile for each recipient and each class that has been previously accessed by the sender. The following formula is used to check the anomalousness of a request with size Z during run time. The anomaly score grows exponentially as the request size increases. In order to tolerate a reasonable amount of deviation of the size attributes, we set x variable as the base and multiply σ with a constant factor of y. The choice of x and y is the recipient's responsibility based on its features and needs. The x and y variables are stored in the application SMDB and available for the detection algorithm.

$$AS_{size} = x \wedge ((Z-\mu) / (y*\sigma)) \quad (2)$$

This equation assigns anomaly scores greater than x only to requests that have SOAP size longer than the average. This is consistent with our assumption that malicious request increases the total size of the soap message. We limit the maximum value of AS_{size} to the same value in AS_{class} to avoid having infinite anomaly scores.

3) Method

The WSDL file for a web service shows its business logic and policies. The business logic is represented by the operations or methods that are available for the client to call along with information about their required parameters, data types, maximum values etc. Since this information is publicly available, there is no reason for the attacker to restrict himself from calling any method while he has valid credentials. However, for the victim service, there should be a noticeable behavior towards the recipients' methods.

Assume the recipient is a credit card service with two methods; one to check the sufficiency of an input card number to cover a certain payment and another method to return the available balance of the received card number. If the recipient receives a call from the impersonated service to the balance returning method given information that the victim service is used to frequently call the sufficiency checking method then the detection system can consider this call as a serious flag of an impersonation attack.

To compute the anomaly score of the invoked method in the observed request, we correlate this method with the probability of relevant frequency of its service which is the current recipient of the request. However, the dynamic nature of the provisioning process in service clouds can lead to provisioning some services offering the same business service less frequent than others. For recipients that have been provisioned and accessed by the sender more than a predefined threshold, the anomaly score of their methods would be computed using their probabilities in equations (3) and (4) below.

The anomaly score (4) for the called method is calculated for each method in the recipient WSDL that was called by the sender using the conditional probability in (3) if P(rec) is greater than a preset threshold. The anomaly score is stored in the sender's profile. We also limit the maximum value of AS_{meth} to the same value in AS_{class} and AS_{size} to avoid having infinite anomaly scores.

$$P(meth|rec) = (P(meth \text{ and } rec))/(P(rec)) \quad (3)$$

$$AS_{meth} = - \log_2 (P[meth|rec]) \quad (4)$$

Other recipients with less provisioning probability do not have sufficient information about their methods in the sender's profile that can be used by the detection system to determine the anomaly scores of their methods. Hence, using equations (3) and (4) to compute the anomaly score of the invoked method for infrequent recipients would introduce a significant bias into the final anomaly decision. Instead, owners of infrequent recipients need to set anomaly scores to their methods representing their acceptable risk degree of revealing information by each method if it has been accessed by an impersonated sender.

4) Historical Impersonation Incidents

Since the cloud hosts many services owned and operated by different tenants, the cloud provider must not assume that all tenants are careful about the security configuration of their services. Thus, the detection system needs to consider the historical incidents of impersonations occurred for each service when examining its requests. The service profile provides the probability that the service historically reveals to be a victim of an impersonation attack.

To calculate the impersonation probability for each service, information about the historical incidents that occurred for the service is accumulated in the tenant SMDB. Then, Profiler exploits this information with the information extracted from the service log file about the total number of requests to compute the probability of the service being a victim of previous impersonation attacks:

$$P(s) = (\text{total impersonated requests}) / (\text{total requests}) \quad (5)$$

We use P(s) to compute the anomaly score for the service history in (6). We weight the probability to have a maximum value similar to other attributes. The higher the anomaly score of the service history the more likely that the service is currently involved in an impersonation attack.

$$5) \quad AS_{history} = P[s] * max \quad (6)$$

The behavior profile for each service has a summary of its request messages logged in its log file using equations (1)-(6). These values would be preserved in the tenant SMDB upon the service registration and provisioned to the relevant applications' SMDBs by the SM.

4.2.2. Profile Aging

When the detection system produces more false alarms than that logged for a previous period for a certain service, that means either the request behavior of this service has dramatically changed or the training data used to generate the profile became old and misrepresents the current behavior. Hence, we propose a strategy indicating when and how to update behavior profiles for services to increase the accuracy of the detection system.

We combine two conditions to determine the profile age and trigger an updating. The first one is a chronological condition for the profile to be updated periodically based on the service owner's policy. The updating process is automatically done at the tenant scope by Profiler using the service's log file. Secondly, when the detection system causes too many false alarms for a certain service before reaching the next updating cycle, the cloud scope (Figure 1) triggers the update at the tenant scope to use the most recent requests in the log file to model the service's profile.

During the updating period for a specific service, the service becomes unavailable for provisioning in cloud applications. Profiler checks the availability of enough requests in the service log file to replace the currently used training data since the last time this data has been collected. If there is enough data, the most recently logged requests are used to generate the profile. Otherwise all recent requests substitute the oldest requests in the training dataset. This method of aging has the effect of creating a moving time window for the profile data, so that the behavior is influenced strongly by the most recently observed behavior. Thus, the detection system adaptively learns services' behavior patterns; as services alter their behavior, their corresponding profiles change.

4.3. Detection System

Although any anomaly detection algorithm can be used to model normal behavior, the algorithm to be used for a significant amount of data in service clouds during real-time must be very efficient. Instead of ML algorithms, we develop an anomaly detection algorithm, which in our experiments leads to a high rate of accuracy and an acceptable performance overhead. In next sections, we explain in details our algorithms to detect suspicious requests and verify their owners' identities.

4.3.1. Detection Process

Inspired by the anomaly detection scoring system in [15], our algorithm does not detect data that represent an attack in the training period because it does not label such data as an anomaly; it assumes all data is normal. It is based on the premise that low probability values with a low likelihood of certain attributes are regarded as noise, and hence. Given that attack data is a minority of the training data then they would be recognized as attacks by the detection algorithm implicitly as low probability events and treated as anomalies at run time. "If attacks were prevalent and high probability events, then they are normal events, by definition" [18].

Upon each request, the application detector retrieves the sender's profile from the application SMDB to examine the impersonation attack. The deviation rate between the received request and the normal behavior is calculated by using the statistical model previously explained.

The total request anomaly score is a value that specifies the extent of the deviation of the received request from the expected values of anomaly scores specified by the sender's profile. It is a compound value derived from the attributes that have been computed in equations (1), (2), (4), and (6) and stored in the profile as follows:

$$AS = w_1\, AS_{class} + w_2\, AS_{size} + w_3\, AS_{meth} + w_4\, AS_{history} \quad (7)$$

The anomaly score for each attribute can be weighted to reflect its importance for the detection system using weight variables w1, w2, w3, and w4. This total score is compared to a threshold that can be chosen by the security administrator. A lower threshold means it is more likely that requests from impersonated services are detected with the disadvantage of an increasing number of false alarms. On the other hand, a higher threshold would allow for malicious requests to pass without detection.

ALGORITHM 1 performs the detection process. Line 24 indicates that the detector of the affected application sends the adequate information about the detected request to the cloud scope to take the proper action.

New services may not have a performance record in the cloud. If they are impersonated by attackers during the training phase; the detection system assumes their abnormal behaviors are normal behaviors. Therefore, once a new service arrives, the cloud must assume a certain amount of risk for some acceptable time if it provisions a new service into an application, given that the use of the new service can cause unrecoverable damage or too many false positives. Thus, the cloud needs to set constraints on the use of new services to determine their trustworthiness. In the proposed architecture (Figure 1), the SMDB of each tenant has an archive of the historical security incidents that have occurred on each service owned by this tenant. This history can be used by the cloud provider to evaluate the trustworthiness of the tenant in order to accept its new services in the provisioning process.

Each application has in its SMDB a white list of services that are within their training phase. The application detector skips checking this new service against ALGORITHM 1. Otherwise, the detector would generate an alert for each request instantiated by every new service which increases the false positive rate of the detection system.

```
ALGORITHM 1: IMPERSONATION_DETECTION
Input:
1.  Sender's profile (prof).
2.  Received request (req).
Output:
3.  Anomaly score (AS) for the received
request.
Begin:
4.  initialize soapSize, recipient, method,
class,μ,σ,AShist,ASmeth,ASclass,ASsize,
anomalyThreshold
5.  method =  req.getSoapAction()
```

```
6.  recipient = req.getRecipient()
7.  soapSize = req.getSoapSize()
8.  class = SMDB(recipient)
9.  if ( find(class in prof) {
10. ASclass= prof.get(ClassAS)          (1)
11. Ashist=prof.get(HistoryAS)          (6)
12. μ = get (Mean, prof)
13. σ = get (standardDeviation, prof)
14. ASsize=x^((soapSize-μ))/(y*σ)       (2)
15. if (find(recipient in prof)
16.     ASmeth=prof.get(MethodAS)       (4)
17. else
18.     ASmeth=max
19.     AS=w1*ASclass+w2*ASsize+w3*ASmeth+
        w4*AShist(7)
20.}
21.else {
22. AS= 0.4* 15 + 0.2* zero + 0.2* zero +
    0.2* zero }
23.if ( AS> anomalyThreshold)
24.  send alert to Cloud scope
25.else
26.  pass req
End
```

4.3.2. Verification Process

The impersonation detection algorithm gives an alert about a malicious request deviating from the normal behavior of the legitimate sender. To verify the identity of the suspicious sender, we use a verification technique. This technique confounds and confuses an impersonator into believing they have useful information when its identity is not verified. We integrate this technique with the service behavior profiling to secure the service in the cloud from being impersonated.

Upon registration, the verification technique requires each service to present a faked version of its response to the cloud SMDB. Then, the SMDB provisions this response to the applications that this service involved in. In addition, the cloud SMDB provisions a random code to each application for verification purposes. Hence, whenever a malicious request to the recipient service is detected by the application detector using ALGORITHM 1, the fake response is returned by the application detector in such a way as to appear completely legitimate and normal. The detector is the only party in the application has the privilege to access the code and fake responses reside in the application SMDB. Therefore, whenever a service has been impersonated, the attacker has no way to neither realize that the received response is faked nor use the application code to verify his identity.

When the sender is correctly identified as an impersonated service, it accepts the faked response since it does not have the ability to communicate with the application detector to distinguish between real and fake responses (Figure 5.A). As a result of this acceptance, the application detector terminates the application after a predefined period of time and delivers the information about the detected request as a security alert to the cloud scope for more investigation. The cloud scope then asks the tenant that owned the detected sender for its recent log file. If the detected request is not recorded in the original sender log file then the decision made by the application detector is correct otherwise it is a sign that the profile of the falsely detected needs to be updated.

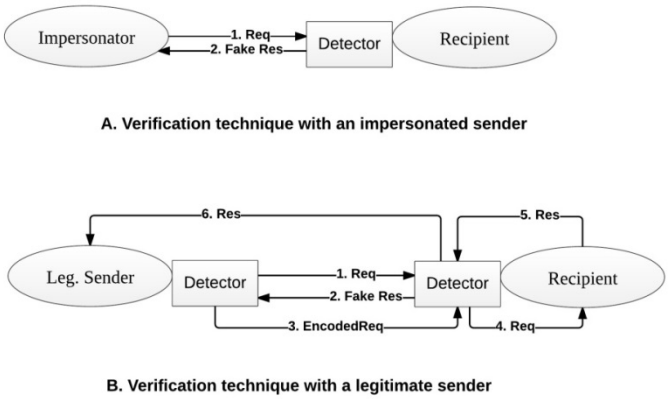


Figure 5: The verification process after detecting an impersonator

For the legitimate sender, the application detector would receive the fake response and readily recognize it and send the same request again along with the application random code to the same recipient (Figure 5.B). This encoded request is the application detector way to verify the sender's identity and issue a false positive alarm. The verification technique here serves two purposes: (1) validating whether the sender is authorized when abnormal request is detected, and (2) confusing the impersonator with useless information.

To incorporate the verification technique with the defined detection system, we need to change ALGORITHM 1 at line 24 to call ALGORITHM 2 instead of the cloud scope as follows:

VERIFICATION (req)

ALGORITHM 2 details how the detection system sends useless information as a response to malicious services. In line 5, the detector checks if the received request is a repeated request from a legitimate sender having the predefined code to verify its identity. In Line 7 the request is passed to the recipient after decoding it. Line 9 shows that the fake response for each service is extracted from the application SMDB. The fake response is passed in Line 10 to the sender.

```
ALGORITHM 2: VERIFICATION

Input:

1.   The malicious request (req).
Output:

2.   Detector's action against the malicious
  request.
Begin:

3.   sender = req.getSender()
4.   recipient = req.getRecipient()
5.   if (req.hasEncodedData) {
6.       Req.removeEncoded()
7.       pass req }
8.   else {
9.       fake_info= SMDB.getFake(recipient)
10.  pass fake_info  }
End
```

ALGORITHM 3 shows how the detector reacts when receiving a fake response by one of the application services. Line 4 declares that the detector uses a single code provisioned by the cloud scope to resend the legitimate requests. The detector deploys the verification technique either by sending or responding to the fake

response without getting the recipient service involved, which reduces the performance overhead and keeps the service black-boxed in the cloud without changing its internal code.

```
ALGORITHM 3: RECEIVING_FakeRes
Input:
1. fake response (res).
Output:
2. req: a new request holding the old request
and the code.
Begin:
3. if (res is fake){
4.   req.update(code)
5.   pass req }
6. else
7.   pass res
End
```

5. Results and Discussion

The developed framework is deployed in Azure cloud. JBoss server is used to deploy the web applications while Apache CXF Framework is used to develop web services and the STS. The application Detector and Auditor are implemented as CXF inbound and outbound interceptors attached to the STS in order to manipulate each SOAP request and response after being authenticated.

To generate behavior profiles for services in the Travel Agency case study, we deployed the framework 300 times to respond to client requests, randomly creating 300 service compositions (applications) from the available services. We used the resulting log files to generate profiles from while recording the generation time. The average time for each service to generate its behavior profile was about 3.2 seconds.

We have conducted three different tests on the designed case study: no attack check, anomaly check, and attack check. We have run the framework without attacks for legitimate services and anomalous services that are changed their behaviors without updating their profiles to measure the false positive rate. In addition, we designed several attacks to measure the detection accuracy rate of the detection algorithm. The designed tests have been deployed on randomly generated applications with a distinctive percentage of 94%. The anomaly threshold was ranging between 3 and 5. We used 0.4, 0.2, 0.2, and 0.2 to weight ASclass, ASsize, ASmeth, and AShistory respectively. The recipient's class anomaly score has slightly more weight since it primarily represents the business logic of the sender. The threshold is computed during the training phase and set to a value that would cause 10 false alarms per 100 applications when the system would receive the training data itself as input. A lower threshold would detect more attacks with the disadvantage of an increasing false positive rate. Hence, the threshold should be set to the lowest value possible provided the acceptable false positive rate. This decision depends on the type of traffic that is seen on the service clouds and SLAs between cloud providers and consumers deciding how many false alarms are considered acceptable [15].

For the test with no attacks, the framework responded with random applications to 100 client requests without involving any attacks. The services here behaved normally as in their profiles. The false positive rate was 1.7%. We have decreased the default threshold of 3 to 2.7 and the false positive rate slightly increases to 1.715%.

During the test with behavior anomalies, the same requests were tested on other random applications with some modifications on the legitimate services' behaviors. These legitimate services were enforced to dramatically change their behavior during runtime before the profile's update time. The changes included accessing new methods, sending large message sizes, and accessing rarely accessed classes. The false positive rate during this test was 28.75%. The false alarms have been exclusively caused by requests that called methods that had not been accessed during the training period. However, this result was generated without using the verification technique. With the verification technique, our detection system eliminated all false positives with a considerable performance overhead as depicted in Figure 6. The detection system needs to support variable security levels with different overhead for different applications. For example, if a banking application is running in the service cloud then there should be possibility to provide highly secure detection for this application by running the verification technique. In case of online games and voice over IP (VoIP) applications, performance and quality of service (QoS) could be of higher priority which requires disabling the verification technique.

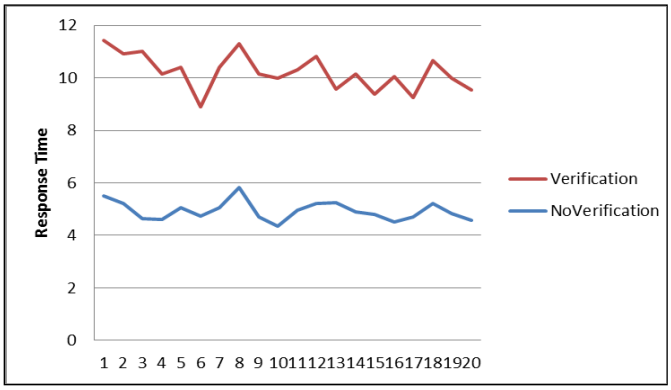


Figure6 : Verification technique overhead

After exploring the false positive rate, we attacked the case study in four distinct tests to measure accuracy. Each test had 100 client requests to initiate 100 different applications each has an impersonated service sending a malicious request to another service in the same application. The attacks mimicked the victim's behavior at different degrees. The victim services were meta-services since they have request behavior profiles while the tenants' services did not have profiles in this test. Any attack targeting a tenant's service would be detected immediately since the expected normal behavior for such services does not include sending requests to any service in the cloud. Table 2 lists the average anomaly scores for statistical properties of malicious requests and the detection accuracy of the impersonation attacks for all four cases. All tests were conducted twice; one with victim services having no historical impersonation attacks and one with a moderate history of attacks.

Table 2: The detection results for the attack tests

Test	Avg (ASclass)	Avg (ASsize)	Avg (ASmeth)	Avg (AShistory)	Accuracy
T1	6.6	1.1	1.9	0(1.5)	92%
T2	1.2	11.3	0.15	0(1.5)	67%(69%)
T3	1.2	11.3	6.2	0(1.5)	97%
T4	1.7	1.8	0.14	0(1.5)	0%

In the test T1, different impersonated services used the legitimate credentials of the victim services to send requests to services from classes that never been or rarely accessed by the victims before. The detection accuracy in this test was high (92%) because the Recipient's Class attribute has a highest weight in the anomaly detection equation among other attribute in the statistical model. However, any attribute can be differently weighted to reflect its security importance in the cloud.

T2 focused on revealing more information from the attacked services by embedding multiple queries in one request. 67% of the attacks have been correctly detected. On the other hand, the detection accuracy increased when the victim services had previously faced impersonation attacks. Furthermore, the detection accuracy reached 97% when the attacks targeted both the message size and the method at the same time as shown in T3 in Table 2. The last group of attacks fully mimicked the victim services which cannot be detected by the proposed detection system using the victim behavior profile. Thus, we need to extend the algorithm to cover such mimicry attacks in future work.

From the computational perspective, the presented algorithm accelerates the detection and profile update operations by implementing these operations in distinct scopes. The profile generation process is done before runtime at the tenant scope by the Profiler to reduce the computational load during the detection live time. The application scope holds the detection process during runtime by using the generated profiles. This results in shorter impersonator live time inside the cloud because it is limited to the active session lifetime at the application scope. As a counterpart, this approach needs a fast periodic update of the service's behavior profile and this introduces some overhead in the cloud network. Furthermore, the verification technique adds some overhead. We believe that updating the behavior profiles very frequently would decrease the need to trigger the verification technique which would eliminate much of the overhead.

Conclusions

The presented detection process depends on two parameters: the anomaly score computed by the detection algorithm using the behavior profiles of the services sending requests, and the fake response used by the verification technique to validate the malicious sender's identity.

In future work, some questions regarding the detection approach needs to be further investigated to improve the detection rate including: How much data is required in order to properly train each service? Should behavior profiles be based on individual service, or should services from the same class be grouped together?

Conflict of Interest

This manuscript is an extension of a conference paper that has been published in December 2016. All authors listed have contributed sufficiently to the project to be included as authors, and all those who are qualified to be authors are listed in the author byline. To the best of our knowledge, no conflict of interest, financial or other, exists.

References

[1] R. Gamble, S. AlQahtani, "Mitigating service impersonation attacks in clouds," Future Technologies Conference (FTC), San Francisco, CA, 2016.

[2] S. Alqahtani, R. Gamble, I. Ray, "Auditing requirements for implementing the Chinese Wall model in the service cloud," in World Congress on Services, San Jose, 2013.

[3] S. Alqahtani, R. Gamble, "Embedding a distributed auditing mechanism in the service cloud," in IEEE World Congress on Services,2014.

[4] M, Salem, S.J. Stolfo, "Modeling user search behavior for masquerade detection," in the the 14th International Conference on Recent Advances in Intrusion Detection, 2011.

[5] (CSA), Cloud Security Alliance, "The notorious nine: cloud computing top threats in 2013," Available from: http://www.cloudsecurityalliance.org/topthreats.

[6] S. Alqahtani, R. Gamble "DDoS attacks in service clouds", in 48th HICSS Conference, Hawaii, 2015.

[7] X. Zhao, G. Hu, Z. Wu Z, "Masquerade detection using support vector machines in the smart grid," in Computational Sciences and Optimization (CSO), 2014.

[8] W. Sha, Y. Zhu, M. Chen, T. Huang, "Statistical learning for anomaly detection in cloud server systems: A Multi-Order Markov Chain framework," in Cloud Computing, IEEE Transactions, 2015.

[9] S. Vanakumar, A. Kumar, S. Anandaraj, S. Gowtham, "Algorithms based on artificial neural networks for intrusion detection in heavy traffic computer networks," in the International Conference on Advancements in Information Technology With workshop of ICBMG 2011, Singapore.

[10] C.V. Raman, A. Negi, "A Hybrid method to intrusion detection systems using HMM," Springer Berlin Heidelberg, pp 389-396, 2005.

[11] S. Omar, A. Ngadi, H.H Jebur, "Machine learning techniques for anomaly detection: an overview," International Journal of Computer Applications (0975 – 8887) 79 (2), 2013.

[12] H.A, Kholidy, F. Baiardi, "CIDS: a framework for intrusion detection in cloud systems," in: Information Technology: New Generations (ITNG), 2012.

[13] D. Gao D, M.K. Reiter MK, D. Song, "Behavioral distance for intrusion detection," in the 8th international conference on Recent Advances in Intrusion Detection, Seattle, 2006.

[14] A.K. Ghosh, A. Schwartzbard, M.Schatz, "Learning program behavior profiles for intrusion detection," in the 1st conference on Intrusion Detection and Network Monitoring, Santa Clara, 1999.

[15] C. Krügel C, T. Toth, E. Kirda, "Service Specific Anomaly Detection for Network Intrusion Detection," in ACM symposium on Applied computing, 2002.

[16] H.A. Kholidy, F. Baiardi F, S. Hariri, "DDSGA: A Data-Driven Semi-Global alignment approach for detecting masquerade attacks," IEEE Transactions on Dependable and Secure Computing, **12**(2):164-178, 2015.

[17] S.J. Stolfo, M.B Salem, A.D Keromytis, "Fog Computing: Mitigating Insider Data Theft Attacks in the Cloud," in Security and Privacy Workshops (SPW), 2012.

[18] S.J. Stolfo, F. Apap,E. Eskin, K. Heller, S. Hershkop, A. Honig A, K. Svore, "A Comparative Evaluation of Two Algorithms for Windows Registry Anomaly Detection," Journal of Computer Security,659-693, 2005.