An Enhanced Cloud-Based Secure Authentication (ECSA) Protocol Suite for Prevention of Denial-of-Service (DoS) Attacks

Marwan Darwish
Department of Electrical and Computer Engineering,
University of Western Ontario, London, Canada
mdarwis3@uwo.ca

Dr. Abdelkader Ouda
Department of Electrical and Computer Engineering,
University of Western Ontario, London, Canada
aouda@uwo.ca

***Abstract*—**Cloud systems are currently one of the primary solutions used in the information technology (IT) domain, also known as cloud services. Cloud services are accessed via an identity authentication process. These authentication processes have become gradually vulnerable to aggressive attackers who may perform Denial of Service (DoS) attacks to keep cloud services inaccessible. Several strong authentication protocols have been employed to protect traditional network systems and verify the identity of the users. Nevertheless, these authentication protocols could cause a DoS threat when implemented in the cloud-computing system. This is because the comprehensive verification process may exhaust the clouds' resources and shut their services down. In this work, we propose an enhanced cloud-based secure authentication protocol suite to operate as DoS resistance on multiple cloud layers. Our proposed solution utilizes multi-technique to prevent external and internal risks of DoS attacks. These techniques can distinguish legitimate a user's requests from an attacker's requests and then direct the legitimate user to the requested service(s). The cloud's servers in the proposed authentication process become imprint-free servers, and fully aware of DoS attacks. To validate the proposed solution, an experiment is conducted using state-of-the-art cloud simulation (GreenCloud). The experimental results verify that the proposed solution is practically applicable as a lightweight authentication protocol suite in multiple cloud layers in terms of reliability and scalability.

Keywords*-Cloud computing; DoS; Security; Authentication; Protocol*

*****

I. Introduction and Related Works

Cloud computing is an operation of software and hardware to deliver solutions to the end users throughout a networking system like the Internet. It consists of a collection of virtual machines that models actual computers and provides solutions such as software applications and operating systems. A cloud-computing system has three layers: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) [1] (Fig. 1). SaaS offers clients with access to the application, PaaS provides clients accessibility to the operating systems to develop the applications, and IaaS gives access to physical components of the system [1].

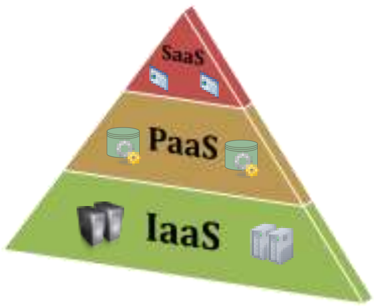


Figure 1. Cloud Computing Architecture

DoS attacks are serious security threats for cloud computing systems as cloud components are used by many consumers. A DoS attack focuses on cloud components and cloud services in order to make the cloud inaccessible by flooding the cloud's resources with significant amounts of forged requests. The purpose of DoS attacks is to exhaust the cloud's resources like network bandwidth, CPUs, memory, or storage systems to become unreachable to the end users.

Handling DoS attacks in different cloud-computing layers is a difficult technique due to the process of differentiating valid user requests from an attacker's requests [2]. In addition, DoS attacks in cloud-computing environments can be initiated externally or internally [3], as shown in Fig. 2. An external cloud-based DoS attack launches from outside the cloud system and focuses on the cloud's services to disturb the availability of these services. Therefore, an external DoS attack can affect the SaaS and PaaS layers. On the other hand, internal cloud-based DoS attack originates inside the cloud system, mainly in the IaaS and PaaS layers. Examples may present themselves in a number of different ways; for instance, an attacker can take advantage of free trial periods of some cloud services' providers. Hence, an authorized client inside the cloud system may initiate a DoS attack to the targeted services internally.

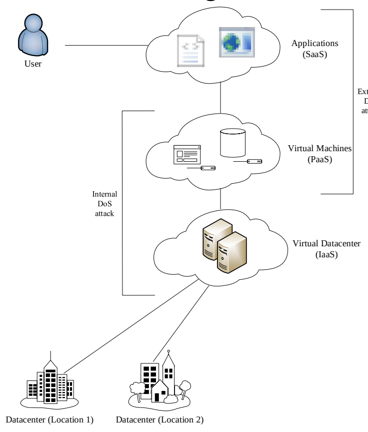


Figure 2. External and internal cloud-based DoS attacks

One of the most commonly implemented authentication protocols in cloud-computing systems is an OAuth 2.0 protocol [4]. This protocol conducts the access to a HTTP service through a third-party application. One implementation of OAuth 2.0 was developed by Google for web server applications [5]. This implementation allows web server frameworks and languages to use Google's application program interface (API). The authorization process of this implementation starts, as shown in Fig. 3, when the web server application does token request. To request the token, web server application redirected user's browser to Google's uniform resource locator (URL) that includes some parameters such as the requested type of access. Google servers controls the authentication process and then sends the authorization code to the web server application. The webserver application will exchange the received authorization code and a user's information with Google servers to obtain an access token or refresh token in a particular case. Consequently, the webserver application can access a Google API using the received token. In the case of "offline access", the webserver application will exchange the authorization code with Google servers for a refresh token. For instance, in case that the user is not available on the browser during the process of issuing a new access token. Therefore, the refresh token will be provided to Google servers to get new access token. In this implementation of OAuth 2.0 for web server application, and in particular the "offline access" case, the webserver application is required to store the refresh token in long-term storage for future use. This refresh access token will be stored as long as the user does not revoke the granted access to the web server application [5]. Therefore, the attacker may take an advantage of this implementation to perform a DoS attack due to storage process as shown in sub-section 4.1.

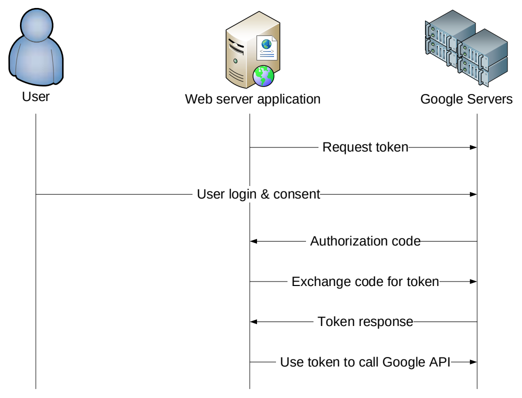


Figure 3. Authorization process of Google's OAuth 2.0 for web server application

Host Identity Protocol (HIP) [6] is a type of authentication protocol that considers DoS attacks in traditional network systems. In spite of this, it is extremely hard to apply it in the cloud-computing system because it depends on the host identity of the network layers in the OSI reference model, and its configuration and management work at an operating system level. Furthermore, the use of any authentication protocol that is dependent on IP address verification (e.g., the IPSec protocol) makes it hard to hide the identity of the contributors.

Several authentication protocols protect against DoS attack on cloud computing have been proposed. Choudhury et al. [7] have proposed an authentication protocol that is aware of DoS attacks for cloud-computing; they use a smart card reader as an additional physical device in the authentication process. Kim et al. [8] proposed a secure authentication protocol for hybrid cloud-computing architecture. Their proposed protocol relies on a 2-Factor authentication service with an existing remote authentication dial in user service (RADIUS) [9]. However, this protocol by itself is in risk to DoS attack because it depends on the IP address and the MAC address of the contributor. An attacker can easily forge an IP address as well as a MAC address. Therefore, the attacker can take advantage of this threat and send many forged requests to an external server.

The Meadows cost-based model approach is an approach to analyze the computation process of an authentication protocol when it comes to its vulnerability to DoS attack [10]. This technique is designed to avoid DoS attacks through the authentication operation. This cost-based model relies on exhausted resources' costs of the protocol's contributors. The cost-based model approach practically demonstrates the ability of the protocol in avoiding DoS attacks. In this approach, the computation cost is described as the overall resource consumption cost of the user and the server when they get involved in the authentication process. The cost is computed throughout the authentication process prior to the process of detecting DoS attacker and then prevented from completing the authentication process. The user's total cost is the total cost of every single operation in the authentication process from the user's part up until the authentication process completes. Additionally, the servers' total costs are the total cost of every single operation throughout the authentication process up until the user is set to appear either a legitimate user or an attacker. The following categorizations are proposed by Meadows [10] for an operation's cost: expensive, medium, and inexpensive. The Meadows approach considers that the signature, a check signature, and exponential operations that are executed throughout the authentication process are expensive. The decryption, encryption, and pre-calculated exponential value operations have medium cost. Every other operation is inexpensive.

In other work, we proposed a Cloud-based Secure Authentication (CSA) protocol suite [11] to authenticate the cloud user in a secure way and to prevent DoS attacks in an early stage on the SaaS layer. Due to its limitation, it is difficult to implement CSA protocol suite on PaaS and IaaS to defend against risks of internal cloud-based DoS attacks. Therefore, this work enhances our previously proposed CSA protocol suite to work as an authentication protocol suite in different cloud layers including PaaS and IaaS.

The rest of this paper is structured as follows. Section 2 explains the research methodology of this work. Section 3 describes the proposed Enhanced Cloud-based Secure Authentication (ECSA) protocol suite. Section 4 validates ECSA protocol suite. Section 5 discusses the proposed work. Finally, section 6 briefly summarizes the work.

II. Research Methodology

Due to the existing of cloud-based DoS attacks in both forms externally or internally, different cloud layers SaaS, PaaS, and IaaS are vulnerable to DoS attacks. This work proposes ECSA protocol suite that is aware of DoS risks in the different cloud layers; at the same time, it is able to securely authenticate cloud clients and identify their requests. ECSA forces the cloud user to do a high computation process while the cloud server does an inexpensive task. In addition, the cloud server is not required to store new additional records or to use extra physical devices during the authentication process.

A well-known "Client puzzle" process is usually applied to achieve the cost-based model approach in authentication protocols [12]. This work considers a cryptographic knapsack [13] as a client puzzle for two reasons. First, it is infeasible one-way function since it is a type of a NP-complete problem [14]. Second, it can be adaptable so that the cloud server can adjust the difficulty level of solving the puzzle according to the required work from the participated user. The difficulty of the knapsack problem depends on the object capacity (quantity of its items) and on the subset size. Notice that when the number of objects is small, a comprehensive attempt to find a solution is functional. Also, when the subset size is small compared to the quantity of knapsack items, the puzzle can be solved in an acceptable time by applying dynamic programming algorithms. Thus, changing the quantity of knapsack items and the subset size identifies the complexity degree of the knapsack problem, and therefore the cost-based model approach can implemented in an adaptive way.

A banker's algorithm approach [15] is also implemented in this protocol to control the service allocation process and to prevent risk of internal DoS attack to a specific service host or specific VM.

A GreenCloud simulator [16] is used as a tool to assess currently available technique, and the implementation of ECSA in terms of their venerability to DoS attacks.

III. Enhanced Cloud-based Secure Authentication (ECSA) Protocol Suite

This work proposes an enhanced cloud-based secure authentication (ECSA) protocol suite to prevent DoS attacks in different cloud-computing layers. In addition, it securely authenticates and identifies cloud users who wish to use cloud services. The ECSA protocol suite can be described in the following 4 phases, as shown in Fig.4. These phases progress consecutively such that each phase process begins based on the output of the previous phase. In the registration phase, the cloud user follows the registration process in the cloud server until the user has confirmed to be as a registered and activated user. Once the user is registered, the cloud server in the identification and authentication phase will determine whether the cloud user who requests a cloud service is a legitimate user or is a DoS attacker. Once the external DoS attack is prevented, the service management and allocation phase directs the cloud user to the requested service with awareness to risks of internal DoS attacks. Finally, once the legitimate user is directed to the host of the requested service, the host session and authentication phase authenticates the cloud user to access the requested service.

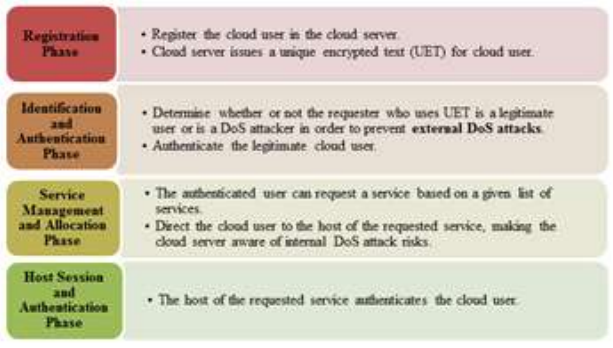

Registration Phase
• Register the cloud user in the cloud server.
• Cloud server issues a unique encrypted text (UET) for cloud user.
Identification and Authentication Phase
• Determine whether or not the requester who uses UET is a legitimate user or is a DoS attacker in order to prevent external DoS attacks.
• Authenticate the legitimate cloud user.
Service Management and Allocation Phase
• The authenticated user can request a service based on a given list of services.
• Direct the cloud user to the host of the requested service, making the cloud server aware of internal DoS attack risks.
Host Session and Authentication Phase
• The host of the requested service authenticates the cloud user.

Figure 4. The ECSA protocol suite phases

The notations of the ECSA protocol suite are shown in Table 1.

TABLE I. NOTATIONS IN THE ECSA PROTOCOL SUITE

Notation	Description
cloud_user	The cloud user/client
cloud_server	The cloud server/service provider
v_service_host	The virtual host of the requested service
CUID	Cloud user ID
SVID	Service ID
ACL	Access control list; an information set issued by **cloud_server**to allow **cloud_user** access to the requested services
UET	Unique encrypted text, the key of which is known only by **cloud_server**
SK	Session key
A	A set of random integers of the server challenge function
S	A subset sum of the server challenge function
B	A vector representing the challenge function solution
R_{cloud_server}	The nonce generated by **cloud_server**
T	Time stamp
K_X	Secret key of *X*
MK	Master secret key of **cloud_server**
Tag	An encryption of time stamp and cloud user ID by master secret key of **cloud_server**
$Tag_{v\_service\_host}$	An encryption of time stamp and cloud user ID by secret key of **v_service_host**

A. Registration phase

In this phase, the cloud server registers and activates the cloud user in its own database. In addition, the cloud server will issue a UET for the cloud user to use it in future protocols' processes. Therefore, a user registration protocol is proposed so that both participants (cloud_user and cloud_server) communicate to share mandatory identification information in order to register cloud_user in the cloud_server's database. To achieved this goal the user registration protocol is designed as the first protocol in ECSA protocol suite.

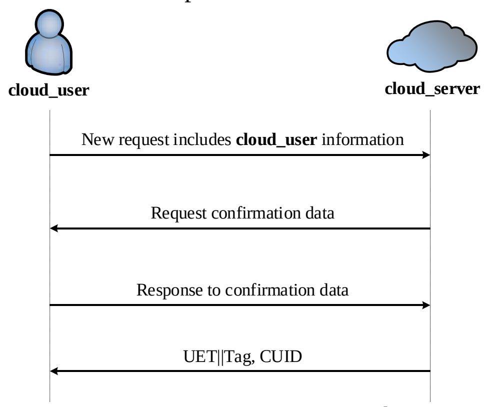

cloud_user
cloud_server
New request includes **cloud_user** information
Request confirmation data
Response to confirmation data
UET||Tag, CUID

Figure 5. User registration protocol

In the first line of the user registration protocol, cloud_user initiates a request to cloud_server that includes cloud_user's information, as shown in Fig 5. This information contains, but is not limited to, cloud_user's name, phone number, email address, and other information that cloud_server should maintain in its database. Cloud_server stores the information within its database and then sends an email message to cloud_user to validate the email address as shown in the second line of the protocol. In the third line of the protocol, once cloud_user responds to the validation message, it is designated an activated user. On the other hand, if cloud_user does not respond to the message within a certain period, the cloud_user information is deleted from cloud_server. Cloud_server then issues cloud_user an identification, or ID (CUID), and a UET that is encrypted by its own master key (MK).

In order to maintain a map between cloud_users and cloud_server that is aware of DoS attack, cloud_server generates a tag by encrypting the timestamp (T) and CUID by its own MK. Cloud_server then inserts the CUID and tag into its database, particularly in the cloud_user's information table (called lookup table in this work), as shown in Fig. 6. Note that the number of entities in the lookup table depends on the number of confirmed registered cloud_users. Note also that cloud_server uses the tag to identify cloud_user based on the information in the lookup table and to avoid any risk of DoS attacks.

CUID	**Tag** = E (T + CUID, MK)	**PSK**	Other **cloud_user**'s i

Figure 6. **Cloud_user** (lookup) database table structure

Then, cloud_server sends the CUID to cloud_user along with the UET and tag as shown in the fourth line of the protocol. This UET includes the CUID, last SK, last T and service request status, as shown in Fig. 7(a). In case that there were any services assigned to the cloud_user previously, the UET includes additionally other information required for other ECSA protocols such as direct and indirect accessible SVIDs list, as shown in Fig. 7(b). Note also that the UET was sent to cloud_user but was not saved on cloud_server.

CUID	Last SK	Last T	Service request status

(a)

CUID	Last SK	Last T	Service request status	Direct accessible SVIDs list	Indirect accessible SVIDs list

(b)

Figure 7. (a) UET structure with no service assigned to **cloud_user**
(b) UET structure after at least one service assigned to **cloud_user**

Both participants use a shared secret and a key derivation function in a very restricted, secure environment for a pre-shared key (PSK) agreement. This agreement process is similar to those used in WPA2 and UMTS protocols [17]; cloud_server then stores the PSK in the lookup table.

As a result, cloud_user in the registration phase has registered and activated in cloud_server; furthermore, cloud_server sent a UET to cloud_user. In the following phase, cloud_server will identify and authenticate the requester who uses the UET; at the same time, it will aware of external cloud-based DoS attack.

B. Identification and authentication phase

The goal of the identification and authentication phase is to make the cloud_server be able to determine whether the requester is a legitimate cloud_user or not, and then authenticates the legitimate cloud_user. As such, an adaptive based identification and authentication protocol is proposed. In this protocol, cloud_server forces cloud_user to perform a computational process before cloud_server is involved in any computational power.

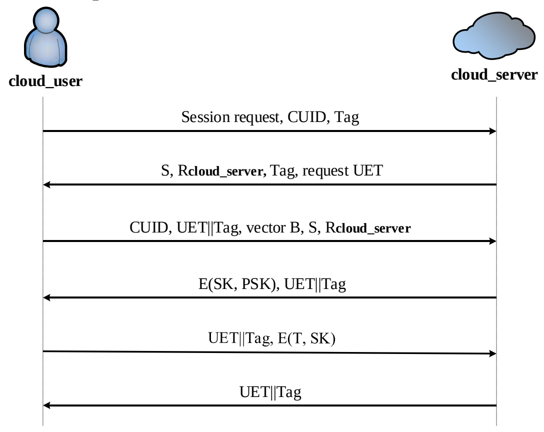

cloud_user
cloud_server
Session request, CUID, Tag
S, R**cloud_server**, Tag, request UET
CUID, UET||Tag, vector B, S, R**cloud_server**
E(SK, PSK), UET||Tag
UET||Tag, E(T, SK)
UET||Tag

Figure 8. Adaptive-based identification and authentication protocol

As shown in Fig. 8, the steps of the adaptive-based identification and authentication protocol are as follows:

1) Cloud_user sends an initial session request with the CUID and tag to cloud_server.

Cloud_server prevents requests from the same CUID once the consecutive failures reach the maximum allowed (three) in a short timeframe. Cloud_server identifies the CUID by easily looking it up in the lookup table on the database and then comparing the received tag to the stored tag value in the table. If they are not identical, the request is considered an illegitimate request; otherwise, the CUID is identified by cloud_server. Note that cloud_server is not required to decrypt the tag until the tag value validated in the lookup table.

2) Cloud_server responds to the request by sending a puzzle challenge value S, sending a nonce (Rcloud_server), and requesting the UET from cloud_user. In addition, cloud_server generates a new tag value and updates the lookup table in the database. Note that the new tag generation process is repeated every time cloud_server checks for the tag value so that the refreshment property has achieved in this protocol.

Cloud_server then sends the newly generated tag to cloud_user. To prove cloud_user's sincere commitment, cloud_server requests the expected puzzle solution (vector B) along with the UET and tag from cloud_user.

3) Cloud_user computes the puzzle solution (vector B) and sends it along with the S value, received nonce (Rcloud_server), CUID, and UET in conjunction with tag to cloud_server.

4) Cloud_server validates the received tag and CUID by checking the lookup table. Cloud_server then verifies that vector B is identical to the result obtained by secure hashing (CUID, Rcloud_server, MK). Note that the MK size should not be short in order to avoid any possibility of using a brute-force attack to guess the MK. Once vector B is verified, cloud_server achieved the goal of identifying the legitimate cloud_user. On the other hand, if vector B is not verified, the request is rejected and assumed as a forgery. Cloud_server then decrypts the UET and checks the CUID registered in the UET. Furthermore, cloud_server issues a new tag, as mentioned in the second step of this protocol. After cloud_server authenticates cloud_user, both participants agree to the session key (SK) for future communications; therefore, cloud_server creates the SK and encrypts it using the stored PSK. It then adds it and the current T to the UET, and it sends this modified UET in conjunction with the new tag and the encrypted SK to cloud_user.

5) To confirm receipt, cloud_user encrypts the current T using the SK and returns it and the received UET along with the received tag to cloud_server.

6) Cloud_server validates the received tag and UET. Cloud_server then decrypts the UET to get the registered SK, which then is used to decrypt T. Note that in order to prevent a DoS attack on storage space, the UET is not stored in cloud_server at all. In addition, in this case, T is used instead of the nonce because cloud_server can verify cloud_user's operation time. Furthermore, cloud_server can handle SK's refreshment property for future communications by simply adding the new SK information to the UET and re-applying the last three steps of this protocol.

Thus, cloud_server in the identification and authentication phases was able to prevent external DoS attacks. Additionally, it was able to identify and authenticate the legitimate cloud_user. Therefore, cloud_user in the next phase can request for available services in the cloud service provider.

C. Service management and allocation phase

One of the most common causes of the internal DoS attack is the misuse of cloud resources allocation. The service management and allocation phase in ECSA enables cloud_server to allocate the cloud_user to the requested available service(s) in a way that help prevent risks of internal cloud-based DoS attacks. To achieve these goals, this work proposes a service allocation protocol. In this protocol, cloud_server can organize multiple requests for a specific service so that deadlocks cannot occur, thus protecting the services from internal cloud-based DoS attacks.

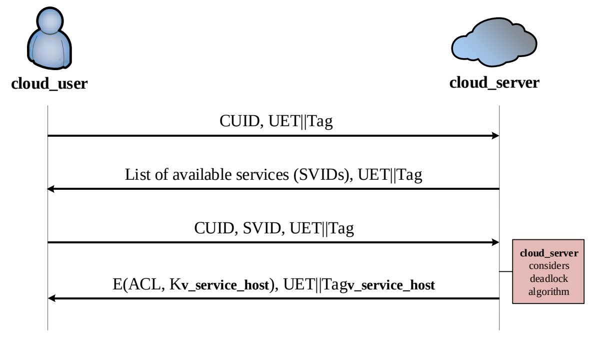

cloud_user
cloud_server
CUID, UET||Tag
List of available services (SVIDs), UET||Tag
CUID, SVID, UET||Tag
E(ACL, Kv_service_host), UET||Tagv_service_host
cloud_server considers deadlock algorithm

Figure 9. Service allocation protocol

In the first line of the service allocation protocol, cloud_user sends the CUID and UET along with the tag to cloud_server as shown in Fig. 9, asking for available services (SVIDs). Cloud_server validates the received CUID by first comparing the received tag with the tag value registered in the lookup table. If the tag value is valid, cloud_server then decrypts the UET and validates the CUID; cloud_server also adds the "service requested" status to the UET. Cloud_server then checks for SVIDs available in the system and sends a list of available SVIDs to cloud_user along with the UET and a newly generated tag as shown in the second line of the protocol. In the third line of the service allocation protocol, cloud_user sends a request for a selected SVID from the received list along with the CUID and UET as well as the tag. Cloud_server denies any request that includes an invalid tag by comparing the received tag with one in the lookup table. If the CUID validated, the requested service may lead to a deadlock problem because many clients have multiple accesses to the requested service or other services through the requested service. Therefore, in order to avoid a deadlock problem and to prevent an internal DoS attack due to a huge number of forged requests for a specific service, cloud_server controls the service allocation process by applying a deadlock avoidance strategy. If the current request can lead to a deadlock problem by, for example, by flooding services with false requests, the request will denied. Hence, cloud_server prevents internal cloud-based DoS attack and achieves the phase's goal.

ISSN: 2454-4248
485 – 496

One approach for deadlock avoidance is to implement the banker's algorithm [15], so that the "requested services" replace the "current processes". Moreover, the "required resources" are those "directly and indirectly accessible services" that cloud_user needs, as shown in service request algorithm in Fig. 10(a) and safety algorithm in Fig. 10(b).

```
(a) algorithm service_request
    input:
    integer n: number of services
    integer m: number of users
    vector available[m]: available[j] = k means there are k instances of services type $S_j$ available
    matrix max[n,m]: max[i,j] = k means user $U_i$ may request at most k instances of services type $S_j$
    matrix allocation[n,m]: allocation[i,j]= k means user $U_i$ is currently allocated k instances of $S_j$
    matrix need[n,m]: need[i,j]= k means user $U_i$ may need k more instances of $S_j$ to complete its task
    vector request[]: vector for user $U_i$, $request_i[j]$ = k means user $U_i$ wants k instances of service type $S_j$
    output:
    Determine the availability of the requested service

    1.  need[i,j] = max[i,j] – allocation[i,j]
    2.  if $request_i$ [j] ≤ $need_i$ then
            go to step 3
        else
            error: user has exceeded maximum claim
    3.  if $request_i$ ≤ available, then
            go to step 4
        else
            user must wait, services are not available
    4.  available = available - $request_i$
        $allocation_i = allocation_i + request_i$
        $need_i = need_i – request_i$
        if safe then
            the services are allocated to user
        if unsafe then
            user must wait, and the old service allocation state is restored

(b) algorithm safety
    input:
    vector work[m]
    vector finish[n]
    output:
    Determine the safety of the system

    1.  work = available
    2.  for i = 0 ... n
            finish[i] = false
    3.  for i = 0 ... n
            if finish[i] = false AND $need_i$ ≤ work then
                go to step 4
            else
                go to step 5
    4.  work = work + $allocation_i$
        finish[i] = true
        go to step 3
    5.  for i = 0 ... n
            if finish[i] = true then
                system is safe
            else
                system is unsafe
```

Figure 10. (a) Service request algorithm for the user
(b) Safety algorithm

If the service can be allocated to cloud_user based on the result of the deadlock avoidance process, cloud_server issues an access control list (ACL) that is encrypted using the secret key of the service's host and adds this additional information to the UET. The ACL contains information such as the CUID, the requested SVID with permissions, and all other SVIDs accessible through the requested service with permissions, as shown in Fig. 11.

CUID
SVIDs SVIDs that are directly accessible with permission
SVIDs Related SVIDs that are indirectly accessible with permission

Figure 11. Access control list (ACL) structure

Cloud_server also issues a new tag to access the virtual service host (v_service_host). However, this tag structure is different than the one used before because it will be used for identifying cloud_user to v_service_host. The new tag includes a timestamp (T) along with the CUID that are both encrypted by v_service_host's secret key. Then, cloud_server sends the encrypted ACL, which is a modified UET in conjunction with new generated Tagv_service_host , to cloud_user as shown in the fourth line of the protocol. Hence, cloud_user received all required information to access the requested service after a lightweight authentication process by the service's host.

D. *Host session and authentication phase*

The main goal of the host session and authentication phase is to authenticate the cloud_user by the host of the requested service in a lightweight process. To achieve this goal, this work proposes a service-host lightweight authentication protocol. In this protocol, cloud_user communicates internally with the required virtual service host. Consequently, cloud_user must be authenticated by v_service_host before accessing the requested service; however, this authentication is a lightweight authentication process to ensure that the requester is a legitimate authenticated cloud_user. As shown in Fig. 12, the service-host lightweight authentication protocol works as follows:

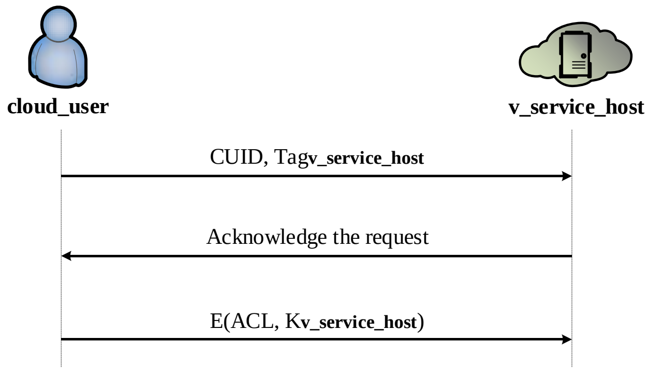


Figure 12. Service-host lightweight authentication protocol

1) Cloud_user sends a request that includes the CUID and Tagv_service_host to v_service_host.

2) V_service_host maintains its own lookup table based on the received CUID and Tagv_service_host. V_service_host then validates cloud_user by decrypting the received tag and then comparing the CUID in Tagv_service_host with the received CUID. This authentication is a lightweight process because cloud_user is already authenticated by cloud_server. Once the CUID is identified, v_service_host acknowledges the request.

3) Cloud_user then sends the encrypted ACL to v_service_host to decrypt it. Next, v_service_host allows cloud_user to access the requested service based on the registered information in the ACL.

Hence, cloud_user is authenticated by the host of the requested service, and the phase's goal has been achieved

IV. ECSA PROTOCOL SUITE VALIDATION

To validate the design of proposed ECSA protocol suite, two experiments have been conducted. The former evaluates the risk of DoS attacks using a data storage technique during the authentication process in the cloud-computing environment. The latter validates the effectiveness of ECSA protocols' processes on the cloud computing system against DoS attacks. Both experiments are implemented using the GreenCloud simulator tool developed by Kliazovich et al. [16] as an extension of the network simulator Ns2 [18].

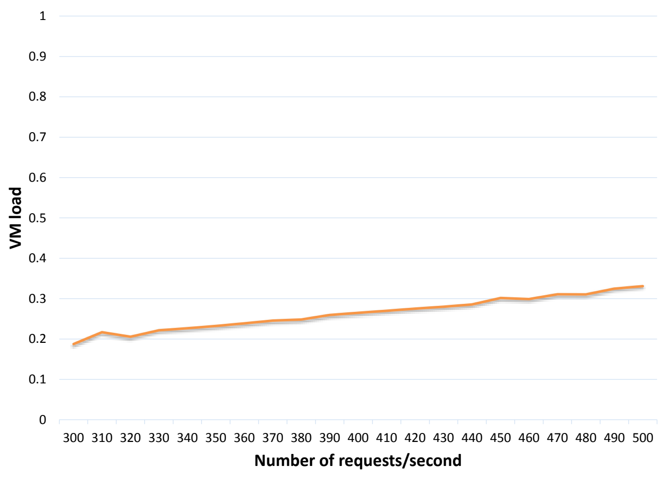
A. *Evaluate a cloud-based authentication protocol with a storage process*

We have mentioned in section 1 above that insecure implementation of OAuth 2.0 could be vulnerable to DoS attack, in particular when storing data during the authentication process. In Google's implementation of OAuth 2.0 for web server applications, the server should store the refresh tokens and keep accessing these stored tokens until they are revoked by a user request. Consequently, the server could be flooded by a huge number of stored refresh tokens' requests until the server becomes unable to process legitimate users' requests even if issuing the refresh token is limited to one token per user. Therefore, this experiment evaluates the effect of DoS attacks due to exhausted storage resources on a cloud computing system.

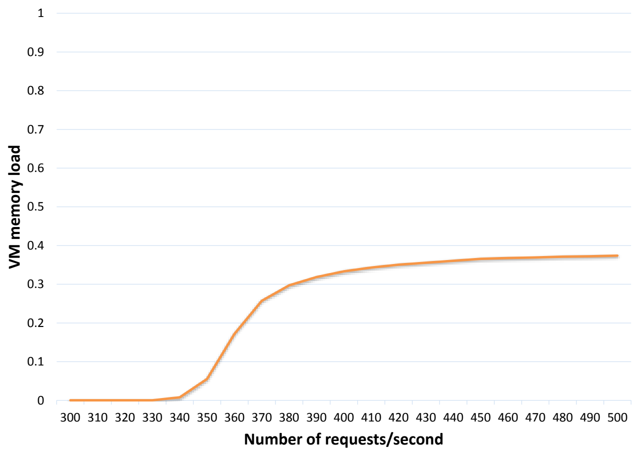
We started our experiment by determining the actual size of the refresh token used among different vendors. Based on Google's documentation for using OAuth 2.0 for web server applications [5], and based on Twitter's documentation [19], the approximate size of the refresh token is 42 bytes. In Yahoo's documentation for OAuth 2.0 [20], the approximate refresh token size is 52 bytes. Therefore, this experiment considers that the average expected refresh token size is 48 bytes.

This experiment involved a main server that run and manage one VM. The expected scenario is that the VM is sharing the resources of the main server with other VMs. Therefore, the expected specifications of the VM is a CPU processor with 4 cores, 4 GB of memory, and 100 GB hard disk drive. This experiment considers the web server has 1 million stored refresh tokens out of 1.5 million users who use social login monthly that is close to real example in industry [21]. Then, the experiment evaluates the performance of the virtual web server according to its load, its memory load, and its storage device load with stored tokens. We assume that reaching approximately 70% load of VM storage device during processing users' requests exhausts the storage system.

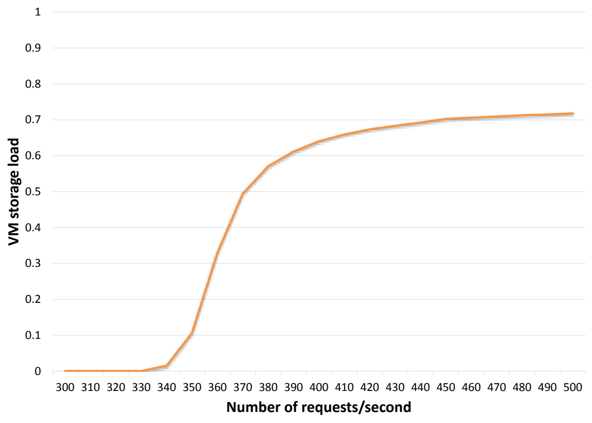
The charts in Fig. 13 show the experiment results of VM load, VM memory load, and VM storage load. We noted that virtual machine storage consumptions start dramatically increasing when processing approximately 300 requests/second up to 500 requests/second as shown in Fig.13(c). The web servers, in general, can handle up to 870 to 1230 requests/second on heavy workload based on the web server specifications [22]. Note that this storage size (48 MB) is expandable because it depends on two factors: the number of stored refresh tokens and the size of the refresh token.



(a)



(b)



(c)

Figure 13. (a) VM load with multiple requests/second
(b) VM memory load with multiple requests/second
(c) VM storage load with multiple requests/second

This experiment shows that depending on the storage device during the authentication process can lead to vulnerability to DoS attacks. Therefore, ECSA protocol suites avoid this serious flaw and eliminate the need for storing any information during the authentication phase.

To validate the effectiveness of the ECSA protocol suite against the risks of DoS during the authentication phase, the following sub-section shows the effect of the most expensive computation processes of ECSA protocols on the cloud server.

B. *Validate ECSA protocol suite*

The cloud_server in the proposed ECSA protocol suite involved in several processes. This experiment evaluates the effect of the most expensive computation processes of the ECSA protocol, including generating random numbers, encrypting plain text by a well-known key (consider AES256 encryption), decrypting cipher text by a well-known key (consider AES256 decryption), and the hashing process (consider SHA2-512) on the cloud server. This experiment simulates these processes using the GreenCloud simulation tool by configuring the cloud server specification as the same as the specification of a virtual web server on the first experiment. This similarity is to compare the performance of our proposed ECSA protocol with the currently used cloud-computing authentication protocol. In this experiment, we evaluated the effect of the most computationally expensive processes of the ECSA protocol on VM load, VM memory load, and VM storage load when executing 300 to 500 requests/second.

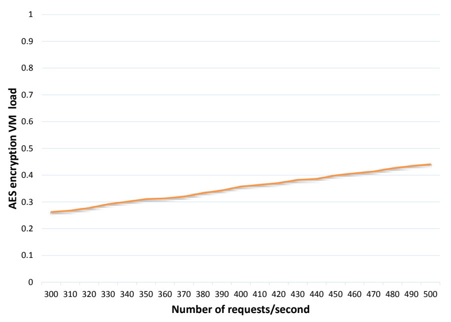
To obtain the properties of ECSA protocol's processes and to use them in the simulation tool, we implemented a real virtual machine with four cores, 4 GB of memory, and 100 GB hard disk drive. We configured Linux OS onto this VM to benchmark these processes and to obtain their properties. This is because any process in GreenCloud simulator has its properties. These properties include million instructions per second (MIPS), input size, output size, and storage size that are used during process execution. To measure process's MIPS, Linux OS has a CPU speed metric called bogoMIPS. Therefore, after the process is executed in the Linux OS, we calculated the MIPS of the process by multiplying the used CPU bogoMIPS by the process's execution time [23]. Then, we obtained the information of the process's file size before and after execution. Finally, we obtained the process's storage size that was used during the process execution such as the key size of the AES-256 encryption process.

After executing the four processes in Linux OS, we obtained the process' properties, as shown in Table 2. These collected properties were then used in the GreenCloud simulation tool.

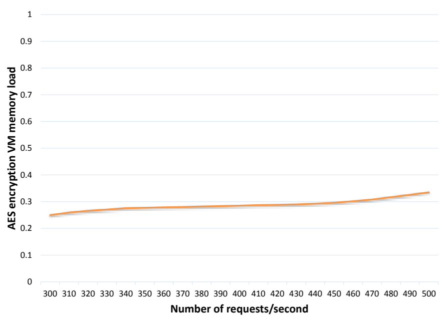
TABLE II. .PROCESS' PROPERTIES THAT OBTAINED FROM LINUX OS BENCHMARK EXPERIMENT

Process type	AES-256 encryption	AES-256 decryption	SHA-512 hashing	Generating random numbers
MIPS	3989	3826	3663	3582
Input size	10000 bytes	10016 bytes	10000 bytes	10 bytes
Output size	10016 bytes	10000 bytes	149 bytes	5121 bytes
Storage size	48 bytes	48 bytes	10 bytes	10 bytes

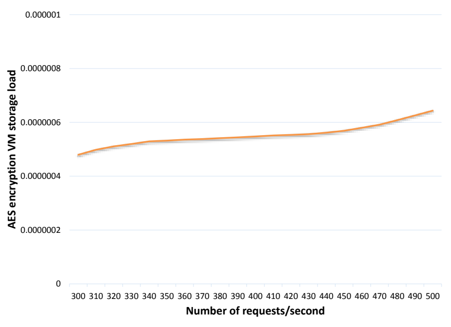
The charts in Fig. 14 show the experiment results after simulating the four different processes: AES-256 encryption, AES-256 decryption, SHA2-512 hashing, and generating random numbers.



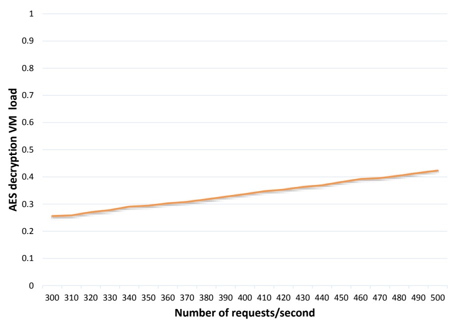
(a) AES encryption VM load



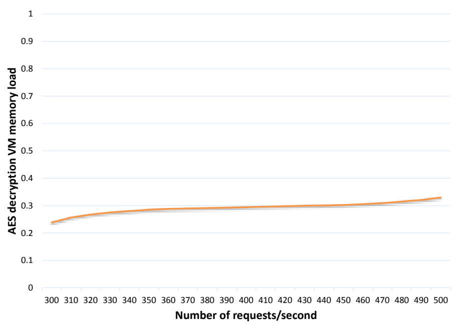
(b) AES encryption VM memory load



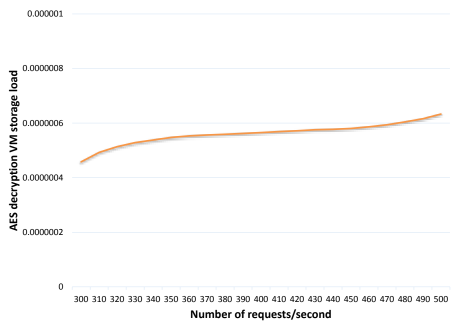
(c) AES encryption VM storage load



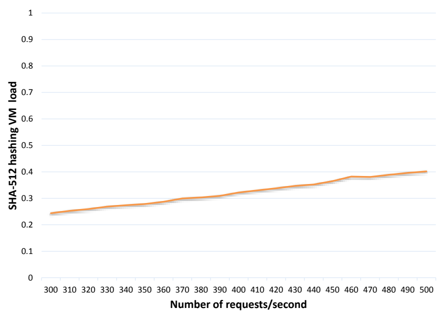
(d) AES decryption VM load



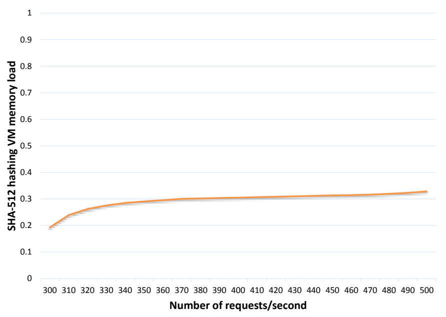
(e) AES decryption VM memory load



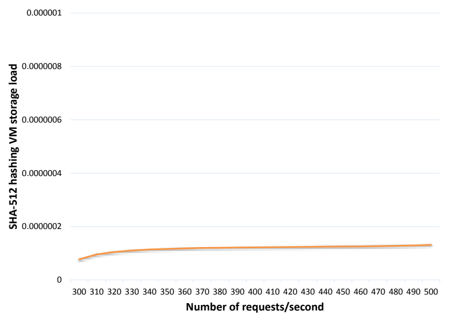
(f) AES decryption VM storage load



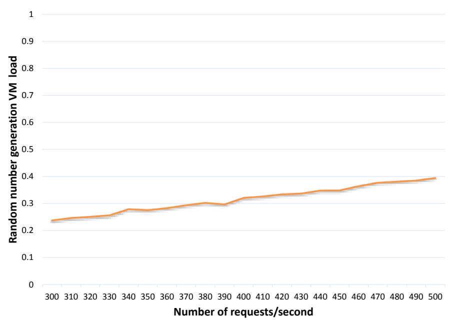
(g) SHA512 hashing VM load



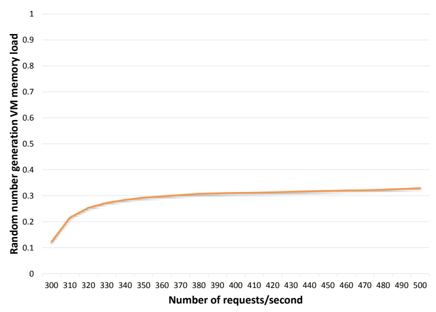
(h)SHA512 hashing VM memory load



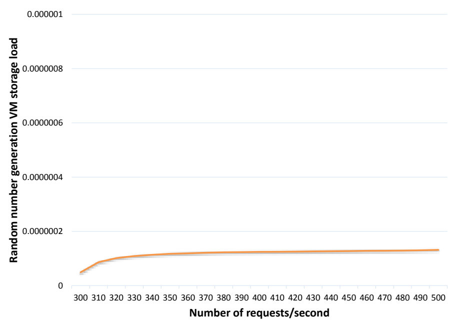
(i) SHA512 hashing VM storage load



(j) Generating random number VM load

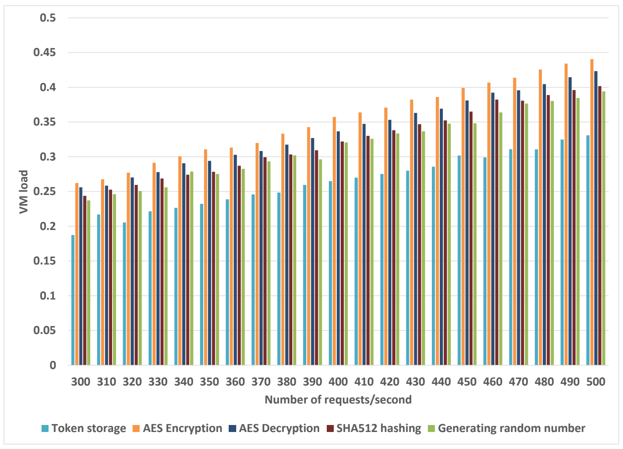


(k) Generating random number VM memory load

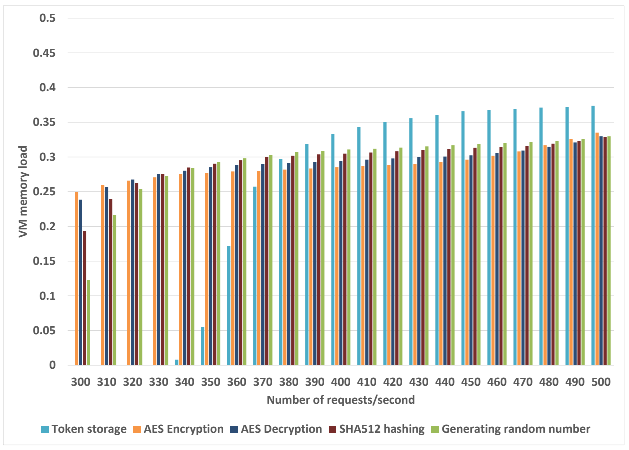


(l) Generating random number VM storage load

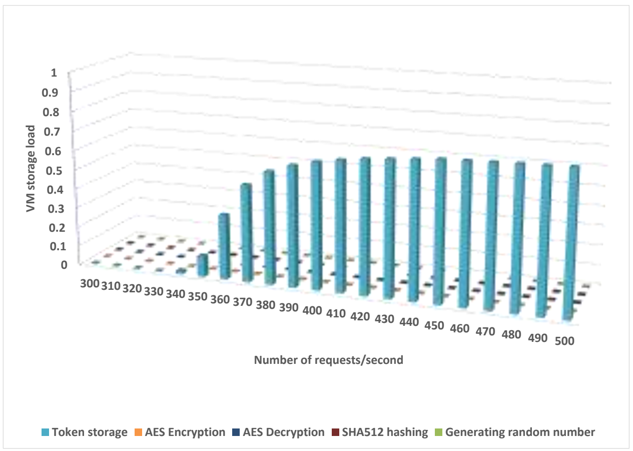
Figure 14. VM load, VM memory load, and VM storage load of ECSA protocol processes with multiple requests/second



(a)



(b)



(c)

Figure 13. (a) Comparison of VM load
(b) Comparison of VM memory load
(c) Comparison of VM storage load

After that, we compared the effect of the computationally expensive processes of our proposed ECSA protocol on the VM to the effect of the implementation of OAuth2.0 authentication protocol for web server's processes that was simulated in our first experiment. Fig.15(a) shows that AES encryption and AES decryption are the most processes that affect VM load. However, all processes including AES encryption and AES decryption are executed without noticeable change between these processes' loads on the VM. In addition, all processes are exacted with less than 45% of VM load. Similarly, Fig. 15(b) shows that all processes are executed in less than 40% load of VM memory without noticeable change between the processes' loads on the VM memory. On the other hand, there is a significant change in the VM storage load between the proposed ECSA protocol's processes and the simulated implementation of OAuth2.0 as shown in Fig. 15(c). This significant change is because of the 1,000,000 tokens that stored in the web server, and then they were used in the authentication process. Therefore, the web server can be vulnerable to DoS attack due to storing data during the authentication processes; this technique is avoided in the proposed ECSA protocol design

V. Discussion

An enhanced cloud-based secure authentication protocol suite was proposed in this work in order to securely authenticate and identify a cloud user and to prevent external and internal risks of DoS attacks. This protocol suite described in four phases to prevent the risks of DoS attacks in multiple cloud-computing layers. In this technique, we combine a UET application, a client puzzle problem, and a deadlock avoidance algorithm to prevent security threats that allow attackers to perform cloud-based DoS attacks as stated in section 1.

In this work, we integrated the proposed ECSA protocol suite with several techniques. We developed the protocol suite to identify and authenticate the legitimate users' requests. Moreover, the protocol suite is developed to prevent external DoS attackers based on the theory of computational complexity. Therefore, a cloud-computing server can adjust the difficulty level of the client puzzle based on the sensitivity of the requested service(s) and thus protect the cloud server from external DoS attackers. Additionally, we implemented deadlock avoidance algorithm to allocate the requested service(s) to the authenticated cloud user with awareness to threats of internal DoS attacks. Furthermore, the ECSA protocol suite is developed so that the host of the requested service is able to authenticate the cloud user.

The proposed ECSA protocol suite is practically applicable on different cloud-computing layers. This is because the protocol is developed based on basic software and hardware components of both participants. Our validation experiment demonstrates the applicability of implementing ECSA protocol on existing hardware and software. Additionally, the experiment shows the reliability of the proposed ECSA protocol suite because of two reasons. First, the processes of the protocol suite do not overload cloud's resources. Second, the protocol suite controls the access to the cloud service(s) by implementing the deadlock problem with no cache process. Therefore, the cloud server does not allocate the cloud user to the requested service(s) if the number of maximum allowed accesses on the same time to these service(s) is achieved.

Furthermore, the maximum number of allowed accesses to the cloud's services is adaptive based on the properties of these services. As a result, the ECSA protocol suite is scalable as it can be expandable based on the size of the cloud-computing system without effect on the cloud computing resources.

This work considers awareness of DoS attacks during authentication processes through different cloud-computing layers. However, this work does not consider an existing implementation of other firewall's software/hardware or intrusion detection systems (IDS); this will be considered in future work.

VI. Conclusion

The usage of cloud-computing systems is increasing nowadays. In these systems, authenticating cloud users within different cloud layers (i.e., SaaS, PaaS and IaaS) is considered a security challenge. Because the communication processes between cloud users and a cloud server are considerable, attackers may take advantage of these processes to perform DoS attacks. Using an existing DoS-resistance authentication protocol in traditional network environments is inapplicable to prevent these types of attacks because the structures of cloud-computing systems and traditional networks are extremely different. This work proposes an enhanced cloud-based secure authentication (ECSA) protocol suite to authenticate the cloud user securely through different cloud layers and to prevent DoS attacks. This protocol relies on an adaptive puzzle challenge technique and a deadlock avoidance technique to prevent external and internal cloud-based DoS attacks. At the same time, this protocol does not require third party devices for cloud users. The risk of DoS attacks in currently used cloud-based authentication protocols was assessed in this work. Finally, the ECSA protocols' performance was analyzed using the GreenCloud simulation tool. This analysis shows that our proposed ECSA protocol suite is a lightweight authentication protocol suite in terms of reliability and scalability. In addition, the analysis of ECSA protocol suite shows that proposed protocol suite is aware of DoS attacks due to storage flaw and it can be implemented in multiple cloud-computing layers.

Acknowledgment

This work was partially supported by King Abdulaziz University through the Cultural Bureau of Saudi Arabia in Canada. This support is greatly appreciated.

References

[1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," 2011.

[2] C. Modi, D. Patel, B. Borisaniya, A. Patel, and M. Rajarajan, "A survey on security issues and solutions at different layers of Cloud computing," The Journal of Supercomputing, vol. 63, no. 2, pp. 561–592, Oct. 2013.

[3] M. Darwish, A. Ouda, and L. F. Capretz, "Cloud-based DDoS Attacks and Defenses," in International Conference on Information Society (i-Society), 2013, pp. 67–71.

[4] D. Hardt, "The OAuth 2.0 Authorization Framework," Rfc 6749, no. 6749, p. 85, 2012.

[5] Google Developers, "Using OAuth 2.0 for Web Server Applications - Google Accounts Authentication and Authorization," 2014. [Online]. Available: https://developers.google.com/accounts/docs/OAuth2WebServer?hl=de. [Accessed: 05-Apr-2015].

[6] R. Moskowits and P. Nikander, "Host Identity Protocol (HIP) Architecture," Rfc 4423, no. 4423, p. 24, 2006.

[7] A. J. Choudhury, P. Kumar, M. Sain, H. Lim, and H. Jae-Lee, "A Strong User Authentication Framework for Cloud Computing," in IEEE Asia-Pacific Services Computing Conference, 2011, pp. 110–115.

[8] J. Kim and J. Moon, "Approach of Secure Authentication System for Hybrid Cloud Service," in Advances in Computer Science and its Applications, vol. 279, H. Y. Jeong, M. S. Obaidat, N. Y. Yen, and J. J. Park, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 543–550.

[9] C. Rigney, A. Rubens, W. Simpson, and S. Willens, "Remote Authentication Dial In User Service (RADIUS)," Rfc 2865, no. 2865, p. 76, 1997.

[10] C. Meadows, "A Cost-Based Framework for Analysis of Denial of Service in Networks," Journal of Computer Security, vol. 9, no. 1–2, pp. 143–164, Jan. 2001.

[11] M. Darwish, A. Ouda, and L. F. Capretz, "A cloud-based secure authentication (CSA) protocol suite for defense against Denial of Service (DoS) attacks," Journal of Information Security and Applications, vol. 20, pp. 90–98, Jan. 2015.

[12] A. Juels and J. Brainard, "Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks," in Network and Distributed System Security Symposium (NDSS), 1999, pp. 151–165.

[13] A. Salomaa, Public-Key Cryptography, 2nd ed. Berlin: Springer, 1996.

[14] U. Manber, Introduction to Algorithms: A Creative Approach, 1st ed. Addison-Wesley, 1989.

[15] E. W. Dijkstra, Cooperating sequential processes. Springer, 2002.

[16] D. Kliazovich, P. Bouvry, and S. U. Khan, "GreenCloud: A packet-level simulator of energy-aware cloud computing data centers," Journal of Supercomputing, vol. 62, no. 3, pp. 1263–1283, 2012.

[17] E. Southern, A. Ouda, and A. Shami, "Wireless security: securing mobile UMTS communications from interoperation of GSM," Security and Communication Networks, vol. 6, no. 4, pp. 498–508, Apr. 2013.

[18] K. Fall and K. Varadhan, "The network simulator (ns-2)," 2007. [Online]. Available: http://www.isi.edu/nsnam/ns. [Accessed: 05-Apr-2015].

[19] Twitter, "Tokens from dev.twitter.com," Twitter, Inc., 2015. [Online]. Available: https://dev.twitter.com/oauth/overview/application-owner-access-tokens. [Accessed: 21-Apr-2015].

[20] Yahoo, "Yahoo OAuth 2.0 Guide," Yahoo Developer Network, 2015. [Online]. Available:

https://developer.yahoo.com/oauth2/guide/index.html. [Accessed: 21-Apr-2015].

[21] LoginRadius, "EasyToBook: Simple Registration and Boosted User Engagement," 2015. [Online]. Available: http://www.loginradius.com/easytobook-simplify-registration-and-boost-user-engagement. [Accessed: 12-Jul-2015].

[22] Parallels, "Delivering Extraordinary Density for Cloud Service Providers," 2010.

[23] E. Pacini, M. Ribero, C. Mateos, A. Mirasso, and C. G. Garino, "Simulation on cloud computing infrastructures of parametric studies of nonlinear solids problems," in Advances in New Technologies, Interactive Interfaces and Communicability, vol. 7547, F. Cipolla-Ficarra, K. Veltman, D. Verber, M. Cipolla-Ficarra, and F. Kammüller, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 58–70.