2015 48th Hawaii International Conference on System Sciences

DDoS Attacks in Service Clouds

Sarra Alqahtani
University of Tulsa
sarra-alqahtani@utulsa.edu

Rose Gamble
University of Tulsa
gamble@utulsa.edu

Abstract

The scalability and dynamic configuration of service clouds can be susceptible to Distributed Denial of Service (DDoS) attacks. The attack on web services causes a performance decrease in the cloud applications or can shut them down. Additionally, due to the high distribution of the service cloud components, finding the original attacking service becomes a far more complex task. This paper advocates a DDoS attack detection approach for service clouds and develops efficient algorithms to resolve the originating service for the attack. The detection approach is composed of four levels such that each level detects symptoms of DDoS attacks from its local data. The detection results of all levels are corroborated to confirm the victim and attacking services. We evaluate our proposed solution by using a random dataset. The results indicate that it is a promising solution to mitigate the DDoS attack in the service cloud.

1. Introduction

The service cloud is a cloud computing model that provisions Service Oriented Architectures (SOAs) of web services from different vendors. Service vendors contract with a cloud provider as cloud tenants to host their services and provision them in a web service composition (i.e. application) to respond to a client's request [1, 2]. The request is placed through the service cloud interface to initiate the service provisioning process. This process provisions services, potentially owned by distinct tenants, into a composite application (i.e. *service chain*) based on their functional and non-functional attributes (e.g. QoS parameters) [3, 4]. The service chain uses end-to-end round trip messaging to exchange data between services. In a service chain $S = <s_0, s_1, \ldots, s_n>$ where the client making the request is s_0. Each service s_i $(1 \leq i \leq n)$ takes a request from the prior service, completes its own processing, adds its output to the original request, and then invokes the next service s_{i+1} in the chain. The response is returned by s_n through the sequence back to s_0. This deployment takes advantage of the flexible publication and access features from cloud computing, allowing the service vendors to enhance the utilization rate of their services and reduce the cost spent on infrastructure deployment [5, 6]. However, messages between services can be used to expose vulnerabilities in service clouds.

For trusted conversations [7], the service cloud employs an identity management service using a Security Token Service (STS). The STS is a token-based authority element that forms trusted conversations by encrypting and authorizing transmitted SOAP messages inside the cloud to protect them from being tainted or disclosed to an unauthorized party.

The service cloud not only inherits features and issues of SOA but, in some cases, exaggerates them. One such issue is the Distributed Denial of Service (DDoS) attack. A DoS attack is a malicious attempt by the attacker to make web services unavailable to intended clients. A DDoS attack has multiple attackers target multiple services or tenants that can affect deployed service chains. DDoS attacks have been a major threat to SOA security. In applications built on different web services owned by different tenants in distributed environment, such as service clouds, the detection and even the mitigation of DDoS attacks is harder than for single services. First, applications in service clouds have multiple interfaces that can be used by attackers as entry points for their attacks, such as the URI of the service WSDL, certain unsecure tenants, an application with clients sending useless requests, and the cloud by targeting more than one of the previous targets simultaneously. Another reason is the diversity of tenants in the service clouds. Although each tenant may have mechanisms to detect anomalous events at its services, such as by watching their ingress traffic, their services are involved in distributed service chains that interact with services from different tenants. It may not be able to trace a malicious event within the application to detect the cause. The cloud provider, however, can trace and discover the causes and consequences of the detected attack among several applications and tenants given the right facilities.

A successful attack prevents the victim from providing desired QoS to its legitimate clients. The most commonly researched type of service denial is the

1530-1605/15 $31.00 © 2015 IEEE
DOI 10.1109/HICSS.2015.627

flooding-based attacks intended to overwhelm a limited, critical set of targets [8]. Such flooding attacks attempt to perform continued resource exhaustion at a well-provisioned target.

The major economic driver behind running a web service on a cloud is charging the customers according to their actual usage. Hence, cloud computing allows customers to scale up their services to serve a large number of requests. This feature of "pay-as-you-use" has transformed the problem of DDoS attacks in the cloud into a financial one. In the event of a DDoS attack, the impact of flooding requests detected at a particular service may have passed several intermediate services before reaching this service. Thus, the owners of intermediate services that have received, processed, and passed illegitimate requests from the original attacking service will be charged for processing those undesired requests. This problem of finding the service originating the attack (i.e. the attack entry point) is an accountability issue [9].

This paper offers a solution to detect DDoS attacks early and identify the attack originating service to isolate it and protect other services in the service cloud. The solution is distributed across service, tenant, application, and cloud levels. At the service level, attack detection and mitigation includes authenticating and syntactically validating received requests before getting processed at the web service. Moreover, the service can detect DDoS attacks by watching the number of messages received from other services. The tenant level corroborates the detection information across its services to find if it is facing a DDoS attack. At the application level, the performance degradation of the service chain can be used to detect DDoS attacks. The cloud level determines the real services that have been used by attackers by corroborating and verifying the information received from other levels.

The contributions of this paper are as follows: (i) A service-cloud DDoS attack detection approach based on a distributed architecture that uses message flow as a metric for detection and (ii) a distance algorithm that measures the deviation between the flow rates of requests at different time intervals to monitor symptoms of DDoS attacks. By using flow distance variation rather than fingerprints of specific attacks; monitoring is independent of attack features.

2. Background

2.1. DoS and DDoS Attacks on Web Services

Web services are vulnerable to several types of DoS attacks, especially as they relate to XML usage. A service that neglects the request authentication process can allow attackers to present numerous requests. The authentication of every request could itself be exploited by attackers. Table 1 summarizes some DoS attacks on web services, that specifically exploit XML vulnerabilities and includes appropriate mitigation techniques and efforts spent to overcome the attacks in web service applications.

Table 1. Sample DoS attacks

Attack Type	Exploitation Method	Mitigation
Oversized Payload	Querying a service using an overly expanded request message [10, 11].	XML size checking before parsing [12]
Coercive Parsing (Deep-Nested XML, XDoS)	Sending a SOAP message with an unlimited amount of opening tags in the SOAP Body [11]	Strong input validation given a specified schema [12]
Flooding Attack(XML ping of death)	Sending numerous requests to the service to make it unavailable for legitimate requests.	Throttling mechanisms for resource allocation [12]

Web services in service clouds must deploy security instruments to authenticate requests before processing them and mitigate the attack types shown in Table 1. Otherwise, attackers may directly flood a service using its WSDL or altering the message content. Attackers can spoof or hijack services and use them to flood other services with authenticated but useless requests. These attacking services can only flood services that are part of the same applications. Thus, victim services work as agents for the attacker by receiving, processing, and passing the malicious requests to other services in the affected applications.

There are four types of DDoS attack participants in the service-cloud:

(1) **Attacker**: the true originator of the attack. It can spoof or hijack services in the service chain so that its requests are syntactically correct but semantically invalid
(2) **Attacking service**: the web service generating attacking requests.
(3) **Victim service**: the service receiving, processing, and passing attacker requests.
(4) **Suspicious service**: the service used initially as the attacking service.

Some services may be falsely identified as suspicious services for attacking other services. A good detection solution should be able to find the actual attacking services that have been hijacked.

For a service cloud DDoS attack, the malicious traffic comes from a group of services (e.g. zombies) operated by people who have stealthily gained access to them. Such traffic is distributed across different

entry points of the cloud making it harder to locate and stop [13]. One proposed approach is to use a form of XML schema validation middleware to filter out malicious packets with the logic being that most web services DoS attacks stem from the vulnerability in the XML structure [14], [15]. Metrics to measure the impact of DDoS attacks on QoS include packet loss, traffic throughput, request/response delay, transaction duration, and allocation of resources [8].

To address potential performance issues of XML message content-based filtering approaches, Padmanabhuni, et al. [16] use a Patricie Trie data structure as a flexible means of storing, indexing, and retrieving XML information in a large file. The approach's use of XML validation and filtering firewalls to counter various types of XML-based DoS attacks is supported in industry [17].

Another approach to mitigate DoS attacks is by incorporating neural networks, clustering techniques, association and sequential rules, and fuzzy logic to analyze and identify malicious SOAP messages to reject them [18, 19]. Offloading the verification of message authenticity to other components allows web services to process only those requests that have been successfully authenticated and validated [20].

In [11], an intrusion tolerant approach employs three phases – monitoring, intrusion and diagnosis – to detect a type of DoS called Deeply-Nested XML attack. In the monitoring phase, the malicious behavior is detected by observing the attack symptoms at different levels. The diagnosis phase evaluates the results to determine if the detected attack was successful or a false positive. The intrusion phase measures the intrusion effects in order to react against it in the recovery phase.

To find the malicious entry points of the flooding attacks, the key is to track and identify the origin of nonsense traffic that tries to saturate other services in the cloud [21]. Jensen and Schwenk [9] suggest methods to identify the entry point of flooding attacks in SOAs. One method centralizes attack monitoring by receiving notifications from all services enabling global tracking of a detected attack. This method can increase performance overhead since each request message in the service composition is sent to the central unit to check its status as a genuine or malicious message. The method also requires all services operated by different companies to accept a single and centralized unit which is not applicable to SOAs and service clouds. Another method is to enforce local logging for each service, which requires searching all service log files to identify the attack entry point [22]. Both approaches advocate carrying service invocation information within the request message, which can cause send/receive performance issues and the potential for a service in the chain to be compromised and disclose its log.

2.3. DoS and DDoS Attacks in Clouds

The Notorious Nine Cloud Computing Top Threats in 2013 [23] show that DDoS attacks are still an prevalent in service clouds. Research in flooding attack detection and mitigation for the cloud has been performed at the network-level. Researchers propose a Multivariate Correlation Analysis (MCA) to investigate geometrical correlations between features in a single network traffic record [24]. Significant changes in the correlations indicate intrusions.

A Confidence-Based Filtering (CBF) method [25] can be deployed under both non-attack and attack periods for detection. Legitimate packets are collected at non-attack periods to extract attribute pairs to generate a nominal profile. With the nominal profile, the CBF method calculates the score of a particular packet at an attack period and determines whether to discard it or not.

A multistage anomaly detection scheme for symptoms of DDoS attacks for secured cloud computing resources has been proposed in [26]. This scheme is composed of three stages. The monitoring stage performs misuse detection by comparing attack patterns to known DDoS attacks. The lightweight anomaly detection stage classifies volume data into large volume data and small volume data using Bayesian methods. The focused anomaly detection stage detects novel and modified DDoS attacks by unsupervised learning algorithm.

Economic Denial of Sustainability (EDoS) attacks target the ability of the service provider to dynamically accommodate increasing numbers of requests from end-users. The attack makes it economically unviable for the service provider to sustain further demand for resources from legitimate end users. The difference between DDoS and EDoS attacks is that EDoS attacks exploit the auto-scale feature of the cloud infrastructure while DDoS attacks exhaust available resources to make a service unavailable [27, 28]. The EDoSShield technique [27] was proposed to mitigate EDoS attacks. The main idea is to verify whether incoming requests are from a legitimate user or generated by bots. The strategy forwards the first request of each requester to a verifier node that performs a verification process and updates white and black lists. Subsequent requests from bots are blocked by a virtual firewall while the subsequent requests coming from legitimate clients are forwarded to the target cloud service.

Hinkhor and Nakao [28] describe a self-verifying proof of work, sPoW, as a cloud-based EDoS attack mitigation mechanism by introducing an asymmetric

step before committing the server's resources. The server requires a proof of work from the client, before committing its resources to the client. The consumed time by clients to solve difficult puzzles reduces the number of illegitimate requests which helps to mitigate EDoS attacks in clouds.

3. Case Study

Our approach distributes DDoS detection and mitigation processes across the service cloud architecture at the Software-as-a-Service (SaaS) and Platform-as-a-Service (PaaS) levels. The DDoS attack detection algorithms are distributed across the service level, tenant level, session or application level, and cloud level as depicted in Figure 1. This distribution allows for load balancing between the local detection performed at each service, detection by the tenant over its hosted services, during execution of service chains to satisfy client requests, and by the cloud to verify the extent of the attack.

Although the distributed detection architecture depicted in Figure 1 may cause some performance degradation because of the required data exchange, the centralized approach suffers from the same issue since each service also sends data to a central unit for analysis. In addition, our method enables each service to send a summary of its received messages in a specified time period to the tenant level in the cloud. The tenant level analyzes the data for DDoS attack evidence. We believe this approach reduces overhead across all service cloud application while providing earlier alerts than centralized solutions, since each level examines local evidence. Moreover, risk mitigation can be triggered at any level even before the information is sent to the central unit.

Service level detection is conducted within the service using local information that is not shared with the application level. A local service handler examines ingress messages from other services in the cloud to detect any dramatic increase in the distance between ingress messages at two different time intervals. An increase beyond a specific threshold indicates the potential for a DDoS attack on the service. The threshold can be manually defined or by using learning techniques. Tenant detection algorithms aggregate and corroborate results from its local services to determine if more than one of its services has been attacked, indicating the tenant itself may be the attack target.

An application detection method is deployed and triggered at the application level for the service chain. It is assumed that provisioned services communicate with a STS to provide a trusted conversation to complete the application request. Thus, the STS log file retains messages exchanged with the client and services as it manages the creation, delegation, validation, and cancellation of the security token. The records in the STS log have the following structure for secure conversation SC_i:

$$\text{STS\_log}(SC_i) = (\text{SessionID, Start\_time, End\_time, Status, <request\_response\_pair>})$$

where SessionID is unique to a session of a single application or service chain. Start_time and End_time provide the time interval when the session was active. Status holds the success or failure status of the session. <request_response_pair> holds the list of requests and their responses which are extracted from the communication exchange between the STS and services. The STS log is used by the application detection process to identify symptoms of DDoS attacks, such as the message flow rate in the service chain and the performance degradation of the application's services.

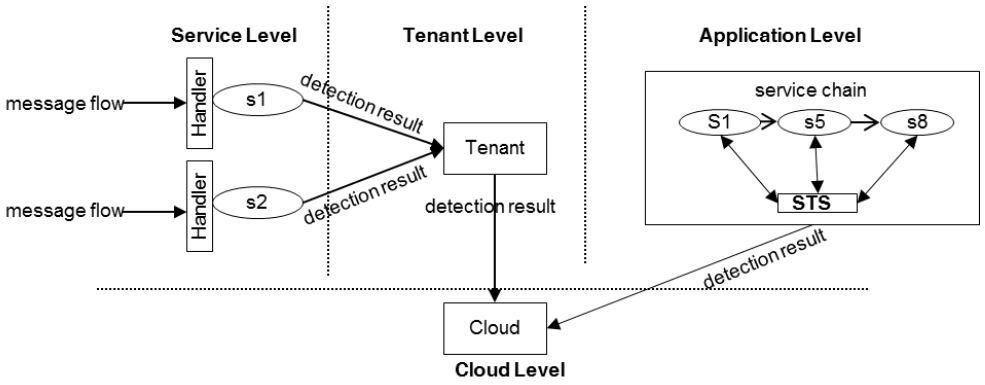

Figure 1. Architecture for DDoS Attack Detection in the Service Cloud

The cloud level is the only level where some centralized forensic investigation information is gathered and examined for DDoS attacks. The cloud level processes corroborate the evidence from other levels and verify the attack occurrence.

The case study we use for the remainder of the paper is a service cloud hosting nine distinct services, four tenants owning and offering these services, and four applications deployed by using a combination of service instances to satisfy client requests. Figure 2 shows the entities used in the case study to evaluate the proposed DDoS attack detection approach.

Tenant	Services
Tenant 1	Z,K
Tenant 2	X,F
Tenant 3	Y,G,E
Tenant 4	A,C

Application	Service Chain
App1	X→Y→Z
App2	A→F→Z→G
App3	Z→E→C
App4	K→A→Z

Figure 2. Case Study

4. DDoS Attack Detection Approach

In this section, we present a DDoS attack detection approach for flooding attacks in a service cloud and identify their entry points into the cloud and responsible services. The approach mitigates and detects DDoS attacks, and attempts to reduce resource consumption and delay by dividing the effort over four functional levels: service, tenant, application, and cloud. Flooding attacks leave traces in the system that provide preliminary evidence of a DDoS attack [29, 30]. For instance, the victim service often experiences a sudden burst in incoming flow over a relatively short period of time.

The service level assures that web services are security instrumented to face DDoS attacks from illegitimate services outside the cloud's boundary. For illegitimate messages from legal services, the service level identifies malicious messages based on the ingress message flow from each other service. The tenant level corroborates detection results from its services to decide if the tenant faces a DDoS attack. At the application level, performance metrics for services in the service chain are used to detect attacks targeting the application. If the detected anomaly appears, the cloud level corroborates the diagnosis from the involved levels to pinpoint the services originating the attack.

The attacking service identification is triggered only when a local anomaly is detected by employing our detection algorithms at the service, tenant, and application levels. This method can reduce the number of services that need to be checked by the multiple detection levels, and thus, allow scaling to more complex service cloud environments.

4.1. Service Level Detection

Since the web service is the fundamental entity in the service cloud that interfaces with the client, ingress flow to web services can be monitored for evidence of anomalous behaviors that manifest DDoS attacks. If the ingress flow rate increases significantly in a short period of time, then the alarm for a potential malicious event is raised. With this evidence, the next step is to identify the list of requesters contributing to the increased flow rate and label them as suspicious services. We measure the difference between the received flow rates from each suspicious service at different times using an abstract information distance metric. If the information distances among the flow rates are greater than a given threshold, a malicious event is confirmed.

The anomaly detection method performed locally at the service level measures message passing using a hash map, which is commonly used to summarize a data stream. We use a hash map H, deployed in a local service handler called *MsgCounter,* to count the number of messages sent by each requester. For each time interval, w, *MsgCounter* generates a time-dependent H, which counts all messages arriving during w. H is defined as follows:

- H: $Interval \times Key \rightarrow Value$, where *Interval* is the time interval of interest, *Key* represents a service ID, and *Value* has the total number of messages received from that service.
- $H(w, x)$, returns the number of messages received from service x at the time interval w.

At the end of each designated time interval, *MsgCounter* at service s examines H for evidence of malicious events. By aggregating the ingress flows for each service in H at s during the specified time interval, fundamental observations about the flow can be made about service requests to s.

Using H at time interval *w-1* and H at time interval w, we detect anomalous flows by calculating the information distance between messages sent at time intervals w and time *w-1* to a service s by using the total variation distance [31]. A distance greater than a preset threshold indicates the service faces a potential DDoS attack at time interval w. Let

$$\text{Distance(s, w-1, w)} = \sum_{x \in X} H(w,x) - H(w-1,x) > \beta \qquad \text{Eq.1}$$

where x is a service that communicates with service s during time interval w or *w-1*. If x is not present in H at both time intervals, then its value is 0. The choice of the flow rate threshold β (and other thresholds

used in the paper) has direct influence on the performance of the detection process. A tight threshold generates a high false positive rate since it decreases the acceptable range of results from the equation, while a looser threshold may cause low detection accuracy because the range in this case would be too expansive to detect abnormal results. Thus, the threshold should be carefully chosen and constantly monitored by the service owner and the cloud provider. Methods exist for such choices, but are outside the scope of the paper.

The next step is to determine those requesters that cause the biggest distance between hash maps given time intervals *w* and *w-1*. We compute the distance between each cell (serviceID) in *H* during *w* and its counterpart in *H* during *w-1* by using the deviation rate, DR, as defined in Eq. 2 below.

$$\text{DR(s, x, w, w-1)} = \frac{H[w,x] - H[w-1,x]}{H[w-1,x]} > \theta \qquad \text{Eq. 2}$$

We consider a requester *x* to be a suspicious service if DR is greater than a preset threshold θ given the values in *H* for service *s* at time intervals *w* and *w-1*. The service *s* is considered to be facing a distributed anomaly if the results from Eq. 2 include more than one requester. However, a suspicious service could be causing a DoS attack that might be a DDoS attack when viewed at the tenant level across its other services hosted in the service cloud. Thus, hash maps that have been built at the service level are forwarded to the tenant level for further investigation.

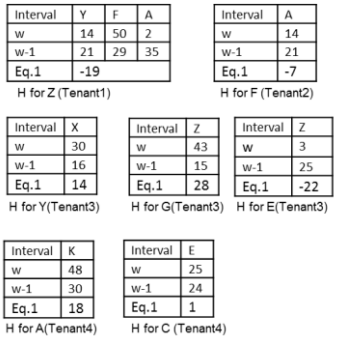

Figure 3. Services with Normal Events

Using the case study from Section 3, the number of requests sent by each service in each application is randomly generated between 1 and 70 requests. We generate *H* for each service in the service chain and compute Eqs. 1 and 2 with β = 50 and θ = 0.8. Figure 3 shows a snapshot of the services at two time intervals. It shows the hash map for each service (except services that receive requests from clients). Eq.1 is computed for each service by using its H hash map. Since Eq. 1 for all services does not exceed β =50, there is no need to compute Eq. 2. The results of Eq.1 at the service level show no DDoS attacks at any service in the cloud.

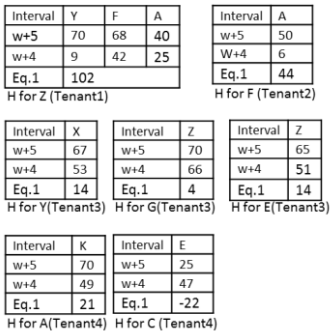

Figure 4: Services with malicious events

Another snapshot at time intervals *w+4* and *w+5* shows how algorithms detect evidence of DDoS attacks. Figure 4 shows the hash maps of the services and their Eq. 1. Service *Z* is the only service that potentially faces a DDoS attack because Eq. 1 produces a value greater than 50 (β). To find the potential attacking services for service *Z*, we compute Eq. 2 for all services communicating with service *Z* during time intervals *w+4* and *w+5*. Based on the results in Table 2, service *Y* is a suspicious service since the deviation rate between its message flow in time intervals *w+5* and *w+4* is greater than the preset threshold θ = 0.8. This information is sent to the tenant for further investigation.

Table 2. Values of Eq. 2 for Service Z

Service	DR (Z, x, w+5, w+4)
x=Y	6.78
x=F	0.61
x=A	0.6

4.2. Tenant Level Detection

Since the tenant itself can be a target for DDoS attackers through its services, the tenant detector combines hash maps received from its local services to detect a potential attack. The tenant detector has its own local hash map called *bigH* which can handle all distinctive keys in the received hash maps.

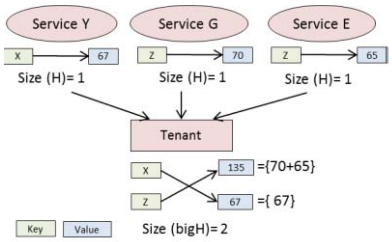

Figure 5. *bigH* for Tenant 3 (w+5)

The *bigH* for Tenant 3 in the case study during the time interval *w+5* is shown in Figure 5, which

presents the process of merging all *H*'s received from the tenant's local services (i.e., services *Y*, *G*, and *E*) into *bigH*.

By using *bigH* at time intervals *w+5* and *w+4*, the tenant detector can detect anomalous flows by reusing the previously defined Eq. 1 and Eq. 2. The tenant detector then produces a list of suspicious service IDs that triggers an important symptom of DDoS attacks as shown in Algorithm 1.

ALGORITHM 1: Tenant_detection

Input:
1. The hash map structures bigH for tenant T obtained in w and w-1 time intervals.
2. The distance threshold β for Eq.1.
3. The deviation rate threshold θ for Eq.2.
4. The set of services communicate with the tenant's services X.

Output:
5. A list of suspicious services attacking the tenant along with the detection interval time w.

Begin
6. If(Distance(T, w-1, w) > β) //Eq.1
7. For each x ∈ X
8. If (DR (T, x, w, w-1) > θ) //Eq.2
9. L=^ x
10. send L, w to the cloud level

End

Figure 6 shows the tenants of the case study during the time intervals *w* and *w-1* along with the hash map *bigH* for each tenant. At the service level, no service has detected a DDoS attack during the same time interval. Tenants still compute Eq.1 to determine if they have potential DDoS attacks in their domain. Since Eq. 1 in Figure 6 for all tenants does not exceed β =50, there is no need to compute Eq. 2. Hence, the results of Eq.1 at the service and tenant levels show no DDoS attacks.

Interval	Y	F	A
w	14	50	2
w-1	21	29	35
Eq.1	-19		

bigH for Tenant1

Interval	A
w	14
w-1	21
Eq.1	-7

bigH for Tenant2

Interval	X	Z
w	30	46
w-1	16	40
Eq.1	20	

bigH for Tenant3

Interval	K	E
w	48	25
w-1	30	24
Eq.1	19	

bigH for Tenant4

Figure 6. *bigH* and Eq. 1 for tenants (*w, w-1)*

As shown in the service level, service *Z* faces a potential DDoS attack. Figure 7 shows *bigH* of all tenants during the same intervals *w+4* and *w+5*. Eq. 1 for Tenant 1 detected the same attack since it has a value exceeding β = 50. This triggers computing Eq. 2 for Tenant 1. However, *bigH* for Tenant 1 is the same as *H* for service *Z* so the results of Eq.2 are also the same (depicted in Table 2). This confirms that service *Y* is the attacking service at the tenant level. The tenant level passes this information to the cloud level to find the original attacking service. Since messages are exchanged between different services in a service chain, service *Y* may be a victim of other attacking services which indirectly flood the tenant by sending their messages to intermediaries.

4.3. Application Level Detection

We measure the effects of DDoS attacks on the service chain performance to find the originating service(s) that started the detected attack in the application. Our approach correlates DDoS attacks, flow rate, and the performance degradation of the web services to detect the first service starting and spreading the DDoS attack in the application. The application detector monitors and identifies the extent of the damages of the attacked services by measuring their degradation based on known performance metrics. Each application is provisioned by the cloud to have a local detector service, which sends results to the cloud level.

Interval	Y	F	A
w+5	70	68	40
w+4	9	42	25
Eq.1	102		

bigH for Tenant1

Interval	A
w+5	50
w+4	6
Eq.1	44

bigH for Tenant2

Interval	X	Z
w+5	67	135
w+4	53	117
Eq.1	32	

bigH for Tenant3

Interval	K	E
w+5	70	25
w+4	49	47
Eq.1	-1	

bigH for Tenant4

Figure 7. *bigH* and Eq. 1 for tenants (*w+4, w+5*)

This correlation is inspired by the model proposed in [8] to evaluate DoS attacks in web applications by using its QoS. Instead of examining the services' static QoS properties that may reside in their WSDL, we compute a performance snapshot from the application log file that is associated with its STS. Let *Perf*(*s*,*w*) be the overall performance of a service *s* at time interval *w* as shown in Eq. 3,

$$\text{Perf(s,w)} = \text{Availability(s,w)} + \text{Successability(s,w)} + \text{Throughput (s,w)} - \text{Response(s,w)} \quad \text{Eq. 3}$$

where Availability, Successability, Throughput, and Response are calculated according to [32].

The deviation rate of the service s performance at w and w-1 is calculated in Eq. 4. If the deviation rate goes lower than a preset threshold α then the service faces a DDoS attack within the application:

$$\text{DR\_Perf(s,w,w-1)} = \frac{Perf(s,w) - Perf(s,w-1)}{Perf(s,w-1)} < \alpha \quad \text{Eq. 4}$$

Another aspect of the service that needs to be measured to evaluate DDoS attacks is its flow of

ingress and egress messages. This parameter helps to find whether the service is an attacking service if it is sending request and response messages more than the received requests. In the service chain, an intermediate service processes requests and forwards them to the next service in the chain. For example, if application A has a service chain <createCustomer, setProfile, setPreferences> then *n* requests to createCustomers should lead to $k \times n$ requests to setProfiles, where *k* is a constant number of requests used by createCustomer per invocation of setProfile. An unexpected change in the number of the requests at any service is a strong indicator that hijacking or spoofing occurred. Thus, each service in the application should be limited to a flow rate of at most some predefined threshold λ. The flow rate for service *s* as computed for time interval *w* using the STS log file (i.e., <request_response_pair>) is:

$$FR(s,w) = K*(outboundMessages) - (inboundRequests + inboundRequests*serviceDegree) > \lambda \quad \text{Eq.5}$$

For $FR(s,w)$, the degree of service *s* refers to the number of children that *s* has, which affects its outbound requests. Thus, the number of outbound requests should be equivalent to the number of inbound requests multiplied by the service degree in *inboundRequests*serviceDegree* unless service s is the attacking service. The number of response messages on the requests from the preceding service should be equivalent to the number of received requests which is considered in the second operand of the minus operator (inboundRequests).Thus, the rest of the outbound messages as a combination of requests and responses represent the irregular flow that needs to be evaluated against a predetermined threshold to find the attacking services in the application. The algorithm is expanded as follows:

ALGORITHM 2: App_detection

Fun:
Input:
1. Threshold of performance degradation α. //Eq.3
2. Threshold of flow rate λ. //Eq.4

Output:
3. Original attacking service or an annotated sequence with flow rate (FR) information.

Begin:
4. If app was not active during the time interval w
5. Send null to the cloud level
else
6. Create a sequence Seq to represent the service composition of the active session.
7. For each node s ∈ Seq
8. annotate s with FR
9. If DR_Perf(s,w,w-1) < α & FR(s,w) > λ
10. return s
11. return Seq

End

Though each application has several instances running simultaneously to satisfy client requests, we use a snapshot of the instances of the deployed applications for illustration. To apply Eq. 3 then Eq. 4, we compute the flow rate and the performance for each service in the deployed service chain. We preset a value of 50 for the threshold λ and -0.25 for the threshold α within the case study.

Figure 8 shows App 1, App 2, App 3, and App 4 sequences during time interval *w+5* annotated with the flow rate of each service. App 1 has the service *Y* with a flow rate 60, exceeding the preset threshold λ = 50. The high flow rate is considered evidence of the occurrence of a DDoS attack in App 1 with service *Y* as the potential attacking service. However, this indicator still needs to be confirmed by a degradation in service *Y*'s performance during the attack time (*w+5* and *w+4*).

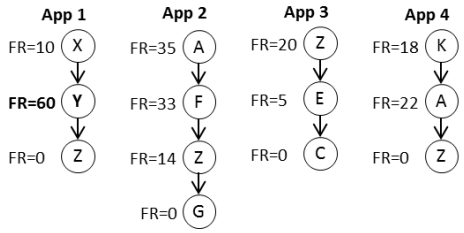

Figure 8. Annotated trees during the attack

Table 3 shows the performance information for services of App 1 during time intervals *w+5* and *w+4*. Information about the performance deviation rate (less than -0.25) and flow rate at service *Y* in App 1 illustrates clear symptoms of a DDoS attack targeting the application with the attacking service Y. This information would be sent to the cloud level.

Table 3. Performance deviation for services in App 1

Service	Perf(s,w+5) (Eq.3)	Perf(s,w+4) (Eq.3)	DR_Perf (s,w+5,w+4) (Eq.4)
s=X	20	22	-0.09
s=Y	13	18	-0.28
s=Z	15	18	-0.17

4.4. Cloud Level Detection

Detection results from the tenant and application levels are sent to the cloud detection level service for more verifiable information about the detected attacks. This result includes a list of IDs of suspicious services attacking the tenant along with the detection time interval (*L*, *w*) from Algorithm 1 and the annotated sequence of services of the application or its attacking service from Algorithm 2. The cloud detector corroborates received information to match

the detection results and extract the exact attacking services that originated the attack. To investigate each suspicious service, the cloud detector extracts the suspicious service IDs and the time interval *w* from the tenant results and compares them with the application results.

For each detection time interval w, a set of the attacking services can be generated after combining the annotated sequences received from the previous level. Although the affected applications have already determined the local attacking service using Algorithm 2, the cloud level can discover attacking services that distributed their flooding loads across multiple, different applications as shown in Algorithm 3. This algorithm starts by filtering applications to only focus on applications that have one or more services from the suspicious list of services detected by the tenant level.

ALGORITHM 3: Cloud_detection

Input:
1. The list of suspicious services' IDs received from the tenant level L.
2. Time interval w.
3. Threshold of flow rate λ.

Output: list of attacking services.
Begin:
4. Initialize list A, list attackingServices, list Seqs, Seq, and result to null
5. Initialize totalFR=0
6. A=^ AppsFilter(L,w)
7. For i=0 to A.length
8. result= A[i].App_detection(λ,α)
9. If result=null
10. skip
11. elseif result.length=1 // single attacking service (line 10 in Algorithm 2)
12. attackingServices=^result
13. else // annotated sequence (line 11 in Algorithm 2)
14. Seqs=^result
15. For j=0 to Seqs.length
16. Seq= Seqs[j]
17. For x=0 to Seq.length //travers all services in Seq
18. Find Seq[x] in all Seqs and sum their FR in totalFR:
19. For each Seq[x] ∈ Seqs
20. totalFR += Seq[x].FR
21. If totalFR > λ
22. attackingServices=^Seq[x]
23. return attackingServices

End

Some important information can be deduced from the list of attacking services generated by Algorithm 3, such as the tenant that was the target for the attack, by finding the owner tenant of each attacking service.

For the case study at time intervals *w+4* and *w+5*, Table 4 shows the results of algorithm 3 that is applied in the cloud level. Service *Z* has been attacked directly by service *Y* (as detected at the application level) and indirectly by service *A*. Service *A* distributed the attack load over two different applications, App 2 and App 4, with two locally acceptable flow rate 35 and 22, respectively. However, the cloud detector, by summing the flow rates of different instances of the same services across all applications, detects this attack.

Table 4. Results of Algorithm3

Service	Total FR across all applications
X	10
Y	***60***
Z	34
A	***57***
F	33
G	0
E	5
C	0
K	18

5. Conclusion

In this paper, we propose a distributed detection of flooding-based DDoS attacks and identify the attacking services within a service cloud. Using a case study, we illustrate the detection algorithms with $O(n^2)$ time in the worst case. Moreover, we observe a similarity between DDoS attacks and numerous illegitimate messages between services. This case is called Flash Crowd (FC), which is a sudden surge in incoming requests to a target, thereby rendering it overloaded and congesting the communication channel leading to it. We intend to expand our system to distinguish between DDoS attacks and FC events.

6. References

[1] She, W., Yen, I.,Thuraisingham, B., and Bertino, E., "Policy-Driven Service Composition with Information Flow Control", IEEE Int'l Conf. on Web Services, 2010, pp. 50-57.

[2] She, W., Yen, I.,Thuraisingham, B., and Bertino, E., "Rule-Based Run-Time Information Flow Control in Service Cloud", IEEE Int'l Conf. on Web Services, 2011, pp. 524-531.

[3] Paik, I., Chen, W., and Huhns, M.N., "A Scalable Architecture for Automatic Service Composition", IEEE Transactions on Services Computing, 7(1), 2014, pp. 82-95.

[4] Wu, C., And Khoury, I., "Tree-Based Search Algorithm for Web Service Composition in Saas", 9th Int'l Conf. on Information Technology - New Generations, 2012, pp. 132-138.

[5] Hu, R., Dou, W., Liu, X.F., and Liu, J., "WSrank: A Method for Web Service Ranking in Cloud Environment", 9th IEEE Int'l Conf. on Dependable, Autonomic and Secure Computing, 2011, pp. 585-592.

[6] Marshall, P., Keahey, K., and Freeman, T., "Improving Utilization of Infrastructure Clouds", 11th IEEE/ACM Int'l Symposium on Cluster, Cloud and Grid Computing, 2011, pp. 205-214.

[7] Rahaman, M.A., And Schaad, A., "Soap-Based Secure Conversation and Collaboration", IEEE 13th conf. on Web Services, 2007, pp. 471-480.

[8] Mirkovic, J., Fahmy, S., Reiher, P., and Thomas, R.K., "How to Test DoS Defenses", Conf. on Cybersecurity Applications & Technology, 2009, pp. 103-117.

[9] Jensen, M., and Schwenk, J., "The Accountability Problem of Flooding Attacks in Service-Oriented Architectures", Int'l Conf. on Availability, Reliability and Security, 2009, pp. 25-32.

[10] Suriadi, S., Clark, A., and Schmidt, D., "Validating DoS Vulnerabilities in Web Services", 4th Int'l Conf. on Network and System Security, 2010, pp. 175-182.

[11] Ficco, M., and Rak, M., "Intrusion Tolerant Approach for Denial of Service Attacks to Web Services", 1st Int'l Conf. on Data Compression, Communications and Processing, 2011, pp. 285-292.

[12] Alqahtani, S., and Gamble, R., "Embedding a Distributed Auditing Mechanism in the Service Cloud", 10th IEEE World Congress on Services, 2014.

[13] Zaffar, F., and Kedem,G., "A Service Architecture for Countering Distributed Denial of Service Attacks", 21st Int'l Conf. on Advanced Information Networking and Applications, 2007, pp. 36-42.

[14] Gruschka, N., and Luttenberger, N. , "Protecting Web Services from DoS Attacks by Soap Message Validation", The IFIP TC-11 21st Int'l Conf. on Information Security, 2006, pp. 171-182.

[15] Loh, Y., Yau, W., Wong, C., and Ho, W., "Design and Implementation of an Xml Firewall", Int'l Conf. on Computational Intelligence and Security, 2006, pp. 1147-1150.

[16] Padmanabhuni, S., Singh, V., Kumar, K M. S., and Chatterjee, A., "Preventing Service Oriented Denial of Service (Presodos): A Proposed Approach", IEEE Int'l Conf. on Web Services, 2006, pp. 577-584.

[17] Suriadi, S., Stebila, D., Clark, A., and Liu, H., "Defending Web Services against Denial of Service Attacks Using Client Puzzles", IEEE Int'l Conf. on Web Services, 2011, pp. 25-32.

[18] Pinz, C.I., De Paz, J.F. , Bajo, J., and Corchado,J.M., "A Multiagent Solution to Adaptively Classify Soap Message and Protect against DoS Attack", 13th Conf. on the Current Topics in Artificial Intelligence, 2010, pp. 181-190.

[19] Yee, C., Shin, W., and Rao, G., "An Adaptive Intrusion Detection and Prevention (Id/Ip) Framework for Web Services", Int'l Conf. on Convergence Information Technology, 2007, pp. 528-534.

[20] Ye, X., "Countering DDoS and XDoS Attacks against Web Services", IEEE/IFIP Int'l Conf. on Embedded and Ubiquitous Computing, 2008, pp. 346-352.

[21] Jensen, M., Schwenk, J., Gruschka, N., and Iacono, L., "On Technical Security Issues in Cloud Computing", IEEE Int'l Conf. on Cloud Computing, 2009, pp. 109-116.

[22] Xie, R., Gamble, R., and Ahmed, N. , "Diagnosing Vulnerability Patterns in Cloud Audit Logs": High Performance Cloud Auditing and Applications, Springer New York, 2014, pp. 119-146.

[23] https://downloads.cloudsecurityalliance.org/initiatives/top_threats/The_Notorious_Nine_Cloud_Computing_Top_Threats_in_2013.pdf, accessed 01/21/2014, 2014.

[24] Tan, Z., Jamdagni, A., He, X., Nanda, P., and Liu, R., "Triangle-Area-Based Multivariate Correlation Analysis for Effective DoS Attack Detection", 11th Int'l Conf. on Trust, Security and Privacy in Computing and Communications, 2012, pp. 33-40.

[25] Chen, Q., Lin, W., Dou, W., and Yu, S., "Cbf: A Packet Filtering Method for DDoS Attack Defense in Cloud Environment", IEEE 9th Int'l Conf. on Dependable, Autonomic and Secure Computing, 2011, pp. 427-434.

[26] Cha, B., and Kim, J., "Study of Multistage Anomaly Detection for Secured Cloud Computing Resources in Future Internet", IEEE 9th Int'l Conf. on Dependable, Autonomic and Secure Computing, 2011, pp. 1046-1050.

[27] Sqalli, M.H., Al-Haidari, F., and Salah, K., "EDoS-Shield - a Two-Steps Mitigation Technique against EDoS Attacks in Cloud Computing", 4th IEEE Int'l Conf. on Utility and Cloud Computing, 2011, pp. 49-56.

[28] Hinkhor, S., and Nakao, A., "Spow: On-Demand Cloud-Based Eddos Mitigation Mechanism", 5th Workshop on Hot Topics in System Dependability, 2009.

[29] Bhatia, S., Mohay, G., Tickle, A., and Ahmad, E., "Parametric Differences between a Real-World Distributed Denial-of-Service Attack and a Flash Event", Int'l Conf. on Availability, Reliability and Security (ARES), 2011, pp. 210-217.

[30] Le, Q., Zhanikeev, M., and Tanaka, Y., "Methods of Distinguishing Flash Crowds from Spoofed DoS Attacks", 3rd EuroNGI Conf. on Next Generation Internet Networks, 2007, pp. 167-173.

[31] Denuit, M., And Bellegem,S.V., "On the Stop-Loss and Total Variation Distances between Random Sums", Statistics and Probability Letters, 53(2), 2001, pp. 153-165.

[32] Issa, H., Assi, C., And Debbabi, M., "QoS-Aware Middleware for Web Services Composition - a Qualitative Approach", 11th IEEE Symposium on Computers and Communications (ISCC'06), 2006, pp. 359-364.