Trust Management Framework for attenuation of Application Layer DDoS Attack in Cloud Computing

Dipen Contractor and Dhiren Patel

Department of Computer Engineering, NIT Surat
India 395007
contractor.dipen@yahoo.co.in
dhiren29p@gmail.com

Abstract. There is a new breed of denial-of-service attacks intended to misuse resources and drive up the cost of cloud computing. Although the impact is less widespread than a traditional Network layer DDoS. Crashing a server is not always easy in the cloud because additional resources can be made available as needed to support sharp spikes in demand. However those resources are not free and an attack could make it economically prohibitive to keep the attacked cloud or its services running.

In this paper, we propose a Trust Management Framework as a partial solution to this problem. It is a lightweight mitigation mechanism that uses trust to differentiate legitimate users from attackers. The trust is evaluated on the basis of clients' visiting history, and used to schedule the service to their requests to access cloud. It uses a new feature called a license (composed of three parameters; client ID, IP address of the client, and computed Trust), for user identification (even beyond NATs) and store the trust information at clients. The license is cryptographically secured against forgery or replay attacks.

Keywords: DDoS attack, Cloud Computing, Trust Management,

1 Introduction

DoS/DDoS attacks are not new and are not directly related to the use of cloud computing. The issue with these attacks and cloud computing is an increase in an organization's risk at the network level due to some increased use of resources external to your organization's network. For example, there continue to be rumors of DDoS attacks on AWS, making the services unavailable for hours to AWS users [14].

However, when using IaaS [9], the risk of a DDoS attack is not only external but there is also the risk of an internal DDoS attack. That internal (non-routable) network is a shared resource, used by customers for access to their non-public instances (e.g., Amazon Machine Images or AMIs[15]) as well as by the provider for management of its network and resources (such as physical servers). If I become a rogue customer, there would be nothing to prevent me from using my customer access to this internal network to find and attack other customers, or the IaaS provider's infrastructure. Provider would probably not have any detective controls in place to even notify it of such an attack.

Application layer DDoS attack [7] is a DDoS attack that sends out requests following the communication protocol and thus these requests are indistinguishable from legitimate requests in the network layer. Most application layer protocols, for example, HTTP1.0/1.1, FTP and SOAP [10], are built on TCP and they communicate with users using sessions which consist of one or many requests (and hence the requester does not use spoofed IP addresses). An application layer DDoS attack may be of one or a combination of the following types [7, 8]: (1) session flooding attack sends session connection requests at a rate higher than legitimate users; (2) request flooding attack sends sessions that contain more requests than normal sessions and (3) asymmetric attack sends sessions with more high-workload requests.

In this paper, we focus on how to mitigate the session flooding attack in cloud. In this paper, we propose a lightweight mechanism, named Trust Management Framework that uses trust management to mitigate session flooding DDoS attack. For every established connection it records four aspects of trust to the user: short-term trust, long-term trust, negative trust and misusing trust which are used to compute an overall trust that helps in determining whether to accept a client’s next connection request. These values are stored as part of a license at clients and when a client revisits the cloud; he attaches his license to the session connection request. Based on the license computes the client’s overall trust, updates his license, and decides whether to accept his request. The license is designed such that the framework can easily identify the client and verify his associated trusts, but license forgery or replay is computationally infeasible. We can also extend Trust Management Framework to collaborative trust management in Hybrid Cloud [2].

The organization of this paper is as follows. In Section 2, we describe the legitimate user model and attacker model. In Section 3 we propose our design considerations. Then defense mechanism in Section 4 and in Section 5, we concluded

2 Basic User Models

Before proposing the mitigation mechanism, behaviors of both normal and abnormal users should be investigated and described carefully. In this section, we build the legitimate user model. Firstly, we would like to make two assumptions.

Assumption 1 Under session flooding attacks, the bottleneck is the maximal number of simultaneous session connections, called as *MaxConnector*. It depends not only on the bandwidth of the server, but also on other resources of the server, e.g. CPU, memory, maximal database connections.

Assumption 2 Without attacks, the total number of session connections of the server should be much smaller than *MaxConnector*, e.g., smaller than 20% of *MaxConnector*, as a cloud controller [3] would set the threshold much higher to tolerate the potential burst of requests.

2.1 Legitimate User Model

In contrast to attackers, legitimate users are people who request services for their benefit from the content of the services. Therefore, the interarrival time of requests from a legitimate user would form a certain density distribution *density(t)* [5]. With this insight, we build the user model in the following way:

1. Use traces of Internet accesses to build an initial model $density_0(t)$, where t is a inter-arrival time and *density(t)* is the probability a legitimate user will revisit the service after t seconds. Many traces has been done by researchers, e.g. F. Douglis et al. [5] traced web users to investigate caching technique in World Wide Web, and M. Arlitt et al. [1] presents a workload characterization study for Internet Web servers.

2. Rebuild user model $density_{i+1}(t)$ with the newly collected inter-arrival times of all legitimate users after Framework runs d days under model $density_i(t)$, where d is randomly chosen from [*dmin* , *dmax*]. It means that $density_{i+1}(t)$ is tightly derived from $density_i(t)$ and hence is difficult to be fooled by attackers.

As a practical legitimate user model, it should satisfy the following properties: firstly, it should converge fast to the users' accesses interval distribution; secondly, it should be dynamic as the distribution may change from time to time; and most importantly, it should be lightweight to be easily implemented and monitored in the defense mechanism.

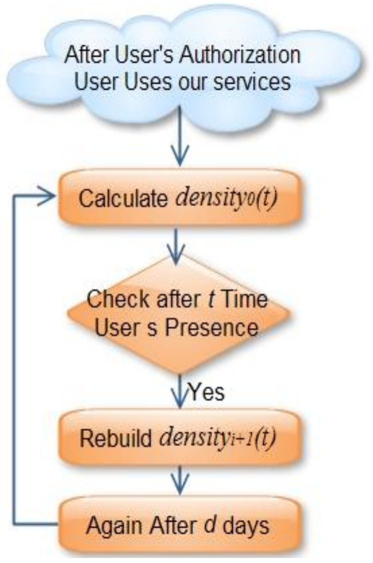
After User's Authorization User Uses our services
Calculate $density_0(t)$
Check after t Time User s Presence
Yes
Rebuild $density_{i+1}(t)$
Again After d days

Fig 1.Basic Flow chart

2.2 Attacker Model

The goal of an attacker is to keep the number of simultaneous session connections to cloud's resources as large as possible to stop new connection requests from legitimate users being accepted. So, an attacker may consider using the following strategies.

He controls a lot of zombie machines or can misuse P2P network as an attack platform

1. Send session connection requests at a fixed rate, without considering the response or the service ability of victim.

2. Send session connection requests at a random rate, without considering the response or the service ability of victim.

3. Send session connection requests at a random rate and consider the response or the service ability of victim by adjusting request rate according to the proportion of accepted session connection requests by the cloud provider [3]. Note that this behavior is different from legitimate behavior, since the random range and random model are different.

4. First send session connection requests at a rate similar to legitimate users to gain trust from server, then start attacking with one of the above attacking strategies.

3 Design Considerations

We have considered the following properties in designing our mitigation mechanism:
(1) it should be deployed at the server for incentive and performance reasons [6].
(2) It should be lightweight, to reduce the processing delay and to avoid being a new target of attacks.
(3) It should be easy to deploy and independent to the details of servers. The defense mechanism need not know what services the server runs or what configuration it uses.
(4) It should be adaptive to the server's resource consumption and differentiate between concurrent requests.

Here we define several components of it before defining trust.

Definition 1: Short-term trust T_s, estimating the recent behavior of a client. It is used to identify those clients who send session connection requests at a high rate when the server is under session flooding attacks.

Definition 2: Long-term trust T_l , estimating the long-term behavior of a client. It is used to distinguish clients with normal visiting history and those with abnormal visiting history.

Definition 3: Negative trust T_n, cumulating the distrust to a client. Distrust means each time the client's overall trust falls below the initial value T_0. It is used to penalize a client if he is less trustworthy than a new client.

Definition 4: Misusing trust T_m, cumulating the suspicious behavior of a client who misuses its cumulated reputation.

Definition 5: Trust *T*, representing the overall trustworthiness of a client, which takes into account all of his short-term trust, long-term trust, negative trust and misusing trust.

Definition6: Blacklist, a list of clients whose trust value is below some minimum level.

Definition7: Whitelist, a list of clients whose trust value is above some threshold value.

When a client's trust *T* drops below defined minimum, that client moves into the blacklist with an expiration time. That client is then banned from accessing the services until his blacklist record expires. When session connection request reaches trust management framework (as shown in fig-2), it checks whether the client is blacklisted; if not, it computes the new trust *T* and use trust-based scheduling to schedule the connection request.

When trusted client starts behaving as an attacker, the number of sessions requested by that client differs. Such client can be moved from whitelist to blacklist.

4 Trust Management Framework Architecture

This architecture is not a monolithic solution that can be easily deployed to gain capabilities immediately. Our proposed architecture as depicted in Fig-2 is a collection of technology components, processes, and standard practices for cloud computing. Standard enterprise access architecture encompasses several layers of technology, services, and processes. Broadly categorized as follows:

1. User management Activities for the effective governance and management of identity life cycles [11]
2. Authentication management Activities for the effective governance and management of the process for determining that an entity is who or what it claims to be.
3. Authorization management Activities for the effective governance and management of the process for determining entitlement rights that decide what resources an entity is permitted to access in accordance with the organization's policies
4. Access management Enforcement of policies for access control in response to a request from an entity (user, services) wanting to access and IT resource within the organization
5. Data management and provisioning of identity and data for authorization to IT resources via automated or manual processes
6. Monitoring, auditing, and reporting compliance by users regarding access to resources within the organization based on the defined policies. Authenticate user records are stored parentally in cloud for future user through Legitimate User Model.

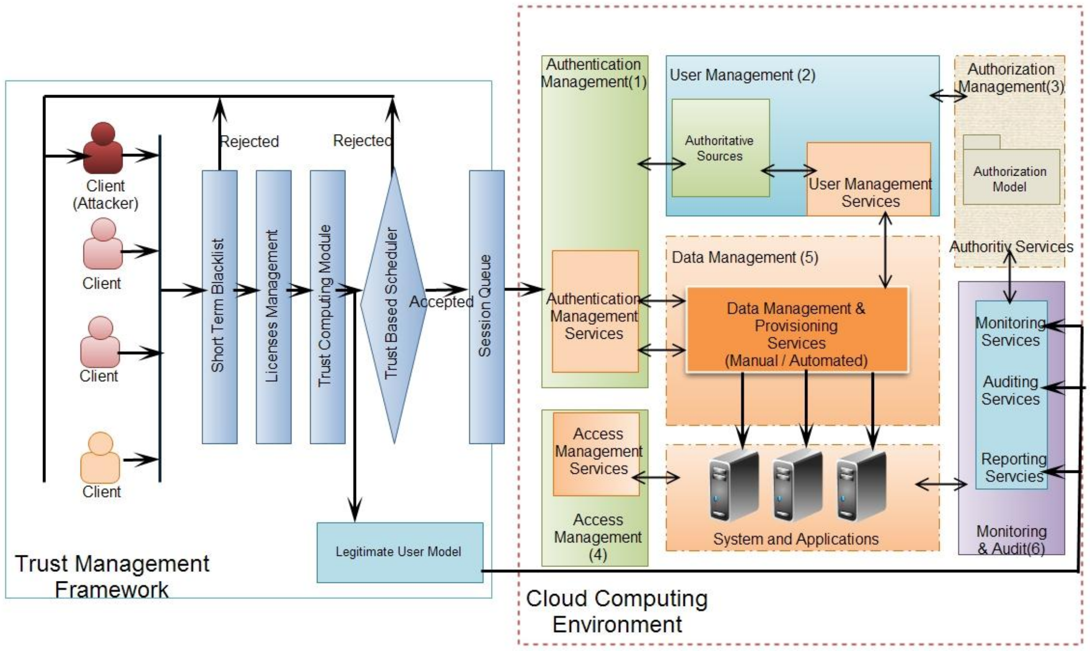

Fig.2. Fundamental proposed model and with its components

4.1 License Management

The identification information and trust states can be stored at clients and verified by the server. We call the information stored at clients as *license*. It contains the following: 64-bit identifier *ID*, *IP* address of client IP, the overall trust *T* to the client, negative trust T_n, missus trust T_m, last access time *LT*, average access interval *AT*, the total number of accesses *AN*, and a keyed hash *H* of the concatenation of all the above, with a 128-bit server password *SP* as the key. *SP* is private to the server. We identify a client by his public IP and the server assigned identifier. If IP address alone is used, clients behind NATs cannot be distinguished, because they share the same public IP address.

A license serves two functions for user identification and trust computation. The identification information, such as *ID* and *IP*, must be stored at the client license. The state variables for trust computation can be stored at the client or at the server. Each has its advantages and drawbacks. Keeping licenses at a server largely prevents attackers from tempering them, but it is a single point of data failure.

4.2 Adaptive Trust Computing

The computation of trust T' employs T, T_n, T_m, LT, AT and AN in license, current time *now*, and *usedRate* (i.e., the percentage of connected sessions over *MaxConnector*) of the server. Based on Assumption 2 in Section 3 *usedRate* is much lower than 1. As we explained, a server should give priority to protect the connectivity of good users during session flooding attacks, instead of identifying all the attack requests. Since a higher trust value means a request is more likely to be accepted, it is desired to satisfy: $T_{legitimate\ user} > T_{new\ client} > T_{attacker}$

We give the formula of short-term trust as follows:

$$T_s' = \left(\frac{density(now - LT)}{e^{alpha \times usedRate}}\right) \tag{1}$$

Where *alpha* is a weight factor deciding the influence of *usedRate*. It is a positive real number with default value 1 and can be modified by servers as needed. When *alpha* ≈ 0, the short-term trust mainly relies on the interval of the latest two accesses of the client.

Similarly long-term behavior of a client. The formula of long-term trust is:

$$T_l' = \left(\frac{lg(AN) \times density(AT)}{e^{T_n}}\right) \tag{2}$$

Using the short-term trust and long-term trust computed above and the misusing trust provided in license, we can then compute trust T' as follows:

$$T' = \min\left(2 \times \frac{\beta \times T_s' + (1-\beta) \times T_l'}{e^{T_m}}, 1\right) \tag{3}$$

Where $\beta \in [0, 1]$ with default value 0.5, it decides the weight of short-term trust and long term trust in the overall trust computation. For a client accessing the server for the First time, its initial value of the overall trust is 0.1, and its initial value of negative trust and misusing trust are both 0, i.e. $T_0 = 0{:}1$, $T_{n0} = T_{m0} = 0$.

4.3 Trust-based Scheduler

When a session connection request is made, this framework firstly validates the license of that client. If passed, it will compute the client's new overall trust, negative trust and misusing trust and then update this information into the license. Afterwards, the scheduler in Framework decides whether to redirect it to the server based on the trust values. It schedules session connection requests once every time slot. If the total number of the on-going sessions and the sessions waiting to be connected is not larger than the *MaxConnector* of the server, the scheduler will redirect all requests to the server. Otherwise, suppose there are N session connection requests waiting to be connected and the percentage of requests should be dropped is μ.

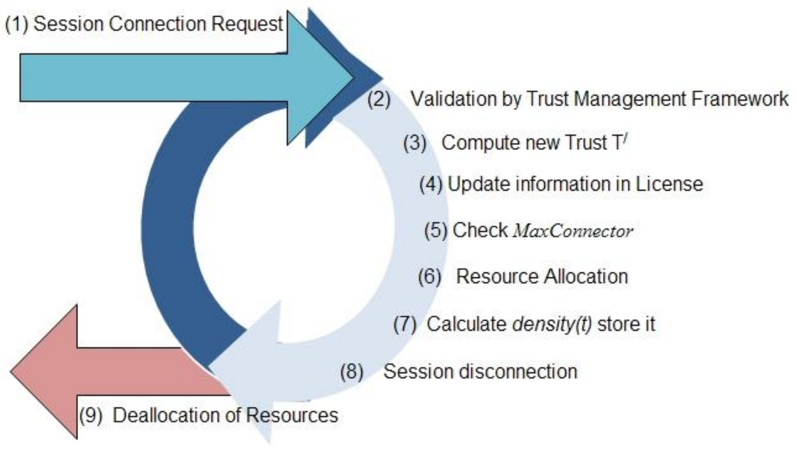
(1) Session Connection Request
(2) Validation by Trust Management Framework
(3) Compute new Trust T/
(4) Update information in License
(5) Check MaxConnector
(6) Resource Allocation
(7) Calculate density(t) store it
(8) Session disconnection
(9) Deallocation of Resources

Fig.3. Flow of Operations

We propose the following scheduling policies to drop suspicious requests:

Foot-*n*: sort all requests in current time slot by the clients' trusts in the decreasing order. For clients that have the same overall trust, sort them by their misusing trusts in the increasing order. We then drop the last $n = \theta \times N$ requests.

5 Conclusion & Future Work

Defending against application DDoS attacks is a pressing problem of the Internet. Motivated by the fact that it is more important for the cloud service provider to accommodate good users when there is a scarcity of resources. Our proposed mechanism Trust Management Framework will mitigate session flooding attack using trust evaluated from user's history. We will try to compare this to with other defense mechanism. Trust Management Framework is lightweight, independent to the service details, adaptive to the Cloud's resource consumption and extendable to allow collaboration among different clouds.

References

1. M.F.Arlitt, C.L.Williamson.: Web Server Workload Characterization: The Search for Invariants In: Proceedings of the ACM SIGMETRICS '96 Conference, pp. 126-137, Pennsylvania (1996)
2. Peter Mell., Timothy Grance.: The NIST Definition of Cloud Computing (Draft). NIST Special Publication 800-145, pp. 6-10 (2011)
3. Nurmi, D., Wolski, R., Grzegorczyk.C., Obertelli. G., Soman. S., Youseff. L.: The Eucalyptus Open-Source Cloud-Computing System. Cluster computing and grid 9th IEEE ACM Proceedings, pp. 124-131, IEEE Press (2009)
4. F. Cornelli, E. Damiani, S. Vimercati, S. Paraboschi, P. Samarati.: Choosing reputable servents in a p2p network. In: Proceedings of the 11th international conference, pp. 65-69 (2002)
5. F. Douglis, A. Feldmannz., B. Krishnamurthy.: Rate of change and other metrics: a live study of the World Wide Web. In: Proceedings of USENIX Symposium on Internetworking Technologies and Systems, pp. 1-13 (1997)
6. M. Natu, J. Mirkovic.: Fine-Grained Capabilities for Flooding DDoS Defense Using Client Reputations. In: Proceedings of LSAD'07, pp. 105-112, Japan (2007)
7. S. Ranjan, R. Swaminathan, M. Uysal, E. Knightly.: DDoS-Resilient Scheduling to Counter Application Layer Attacks under Imperfect Detection. In: Proceedings of INFOCOM'06, pp.1-13 (2006)
8. J. Yu, Z. Li, H. Chen, X. Chen.: A Detection and Offense Mechanism to Defend Against Application Layer DDoS Attacks. In: Proceedings of ICNS'07, pp. 54-56 (2007)
9. Khajeh-Hosseini. A., Greenwood D., Sommerville I.: Cloud Migration: A Case Study of Migrating an Enterprise IT System to IaaS. In: Proceedings IEEE Cloud computing 3rd conference, pp.55-65, IEEE Press (2010).
10. Francisco Curbera., Matthew Duftler., Rania Khalaf, William Nagy, Nirmal Mukhi, Sanjiva Weerawarana.: Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI. IEEE Internet Computing, vol. 6, no. 2, pp. 86-93, IEEE Press (2002)
11. Anu Gopalakrishnan.: Cloud Computing Identity Management. SETLabs Briefings, Vol7 No7, pp.45-55 (2009)
12. Yi Xie, Shun-Zheng Yu.: A Novel Model for Detecting Application Layer DDoS Attacks Computer and Computational Sciences. IMSCCS '06 First International Multi-Symposiums (2006)
13. Kevin J Houle., George M Weaver., Neil Long., Rob Thomas.: Trends in Denial of Service Attack Technology CERT, Issue Oct. (2008)
14. Article: Rumor Amazon Hit With Denial-of-Service Attack, Again. posted June 6, 2008, http://www.appscout.com/2008/06/rumor_amazon_hit_with_denial of.php
15. Amazon machine images, http://en.wikipedia.org/wiki/Amazon_Machine_Image