Mitigating Yo-Yo attacks on cloud auto-scaling

1st Meraj Mostamand Kashi
Department of Computer Science)
OsloMet – Oslo Metropolitan University
Oslo, Norway
meraj.m.kashi@gmail.com

2nd Anis Yazidi
Department of Computer Science)
OsloMet – Oslo Metropolitan University
Oslo, Norway
anisy@oslomet.no

3rd Hårek Haugerud
Department of Computer Science)
OsloMet – Oslo Metropolitan University
Oslo, Norway
haugerud@oslomet.no

***Abstract*—In recent years, global businesses have witnessed a significant cloud adoption that provides considerable value compared to traditional data centers, achieving greater scalability, cost efficiency, and improved performance. Cloud auto-scaling is a cloud service feature that copes with variations in the workload by spinning up or down instances on the fly. Attackers may exploit auto-scaling mechanisms to transform the traditional DDoS attacks into Economic Denial of Sustainability attacks (EDoS). In this perspective, a new type of attack, called Yo-Yo attack, has been recently reported in the literature where the attackers send a burst of traffic periodically to oscillate the auto-scaling system between scale-out and scale-in status inducing economic loss to the tenant. These new types of attacks cause are harder to detect compared with traditional DDoS, and they require fewer resources from the attacker. In this paper, we present a simple solution that is capable of detecting a Yo-Yo attack and mitigating it. In this quest, a legacy approach named Trust-based Adversarial Scanner Delaying (TASD) [1] is implemented and tested in a cloud production environment. The TASD method assigns a trust value number as a Quality of Service (QoS) value to each user. The original TASD system used an Additive Decrease method to update the trust value. Inspired by the TCP rate control mechanisms, we introduce two variants of TASD, ADAI (Additive Decrease/Additive Increase) and MDAI (Multiplicative Decrease/Additive Increase). The devised TASD approaches are deployed on Amazon Web Services (AWS). The experiment evaluations show that our proposed TASD variants can efficiently detect and mitigate Yo-Yo attacks in an actual cloud application.**

***Index Terms*—Yo-Yo attack, Auto-scaling, Cloud Security**

I. Introduction

For a decade, DoS and DDoS attacks have become an immense threat to the Internet infrastructure. Attacks have become commonplace, with many global victims from commercial websites, educational institutions, public chat servers, and government organizations [2] [3]. A detailed analysis from Cloudflare shows that DDoS attacks have increased dramatically over the past few years. The report reveals that the number of attacks grew by 50% in Q3 2020 compared to Q3 2019 [4]. Another report from Kaspersky researchers observed a massive increase in attacks in Q3 and Q4 2021. The report shows 400% growth in Q4 2021 in comparison with Q4 2020 [5]. According to the Cloud Security Alliance report [6], the number of DDoS attacks on cloud services has also risen over the years. Increasing the number of attacks causes emerging new detection and mitigation solutions as well [7]. In November 2020, Alibaba Cloud Security Team detected the most substantial resource exhaustion DDoS attack on their cloud platform, with a peak of 5.369 million QPS (Queries Per Second) [8]. Microsoft mitigated 359,713 DDoS attacks against their Azure Cloud infrastructures during the second half of 2021 [9].

On the other hand, attackers do not surrender. New kinds of DDoS attacks have emerged to exploit cloud anti-DDoS solutions. Burst attacks, also known as hit-and-run DDoS, is a new kind of DDoS attack where the attacker launches periodic bursts of traffic overload at random intervals on online targets [10] [11]. A general rise of Burst attacks has reached such an extent that in a comprehensive survey in 2017, half of the participants cited an increase in burst attacks [12]. This points to the increasing sophistication of hackers in their ability to better leverage large botnets and develop mechanisms that can evade detection. Botnet malware technologies accelerate burst attacks due to scheduling and synchronizing features [13].

Enterprises using cloud services primarily benefit from cloud features. Enhanced cloud scalability and elasticity through auto-scaling allow customers to scale their applications dynamically [14]. Incoming traffic is distributed evenly across multiple endpoints, so individual back-end services cannot be overwhelmed until the traffic volume approaches the entire network's capacity.

Hostile actors adjust their tactics to correspond to the realities posed by the cloud. Yo-Yo attack is a new technique against the cloud auto-scaling feature [15]. In this method, attackers send a burst of request traffic to increase the significant load on the cloud server. This large amount of requests causes the triggering of an auto-scaling mechanism. Consequently, the overload of requests will again result in the server scaling up. Therefore, during this confirmation period, the victim's system had already deployed resources far exceeding the required amount. Burst traffic will be stopped after scale-out and waiting for the auto-scaling mechanism to scale-in the server. The attacker continues the latter attack procedure and forces the cloud services to scale-out and scale-in continuously [1]. It causes adding extra load to scale-out the services to respond to the fake requests. In effect, the attacker forces the victim to pay for large amounts of resources that are not necessary to handle the legitimate workload. Yo-Yo attack can affect any platform using auto-scaling mechanisms such

This work was partially supported by the research project CyberHunt (Grant No. 303585 - funded by the Research Council of Norway)

as container-based-environment [16] and Kubernetes platform [17].

II. Approach

A. Overview of TASD system

Figure 1 depicts an overview of the TASD system architecture to detect and mitigate the Yo-Yo attack. The load balancer acts as the ”traffic cop” sitting in front of the auto-scaling and routing client requests across all servers capable of fulfilling those requests. It helps in a manner that maximizes speed and capacity utilization and ensures that no one server is overworked, which could degrade performance. When a new server is added to the auto-scaling group, the load balancer automatically starts to send requests to it, and in the case of scaling-in, the load balancer redirects traffic to the remaining online servers. The TASD architecture has been designed as a distributed system and runs as a ”side-car.” It means that the TASD service runs on each instance where the web application is running. Therefore, when the system scales out and a new instance is added to the auto-scaling group, TASD distributed service can work as a defense mechanism independently.

TASD system has two main functions: detection and mitigation. The attack detection module firstly determines the type of request for each user. If it is identified as suspicious, this request will be forwarded to the second auto-scaling group. The instances inside the second auto-scaling group run a web service, which replies to the requests with a two-second delay. If a request identifies as malicious, the request will be dropped directly. Otherwise, it will be marked as normal and forwarded to the main auto-scaling module. The next section discusses the TASD modules in more detail.

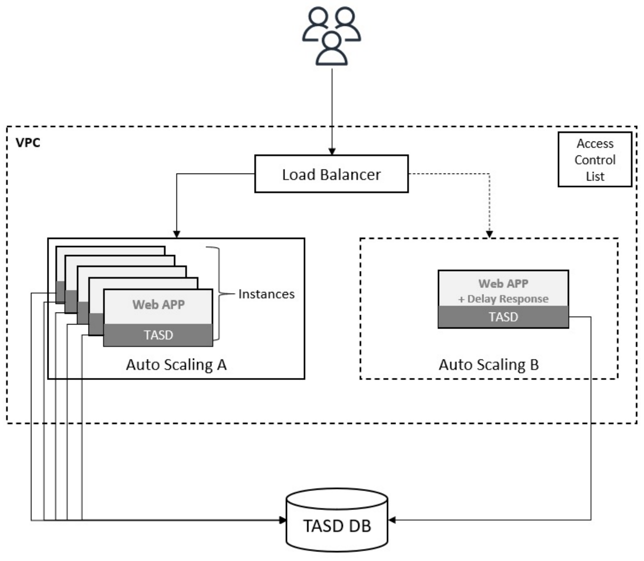

Fig. 1. The project architecture diagram using TASD system as a ”side-car” inside each instances of auto-scaling group.

B. TASD modules

According to the characteristics of the Yo-Yo attack, the attacker frequently triggers the auto-scaling mechanism. The attacker continuously sends burst traffic to increase the load of the system. It causes scaling out the instances. Then attacker stops sending traffic and waits for scale-in. Subsequently, compared with benign traffic, the requests from Yo-Yo attackers frequently occur during the server overload. This feature shows that there is a large difference in load volumes between scale-out and scale-in phases. TASD system can detect the attack by comparing the request numbers and cloud auto-scaling status.The detection module adds a Quality of Service (QoS) number, which is known as a trust value for each user. Therefore, the trust value is selected based on the below indexes:

- Status of the auto-scaling.
- The difference of load volumes between scale-out and scale-in phases.

The detection module is responsible for capturing received packages and maintaining a list of trust values for all users. Firstly it checks whether the user exists in the trust value database. If not, it adds a new entry for the user IP address and initial trust value (T_{init}). The module checks the auto-scaling status and counts the request number of each user during the scale-out process. In addition, in each scale-out process, the detection module records k users that have the most requests load volume in the trust value list. The user's IP address and the number of requests are stored in an array called S. Then after scaling-in, the request number of k user in the array S compares with the current request number; if the request is loaded down by M, the attack detection module decreases the user's trust value by one. Algorithm 1 shows the Pseudo code for assigning trust value by TASD. Then, based on the trust value, the module marks different requests as normal, malicious, or suspicious. As the module increases the trust value of a suspicious user by a constant amount, it is an Additive Decrease (AD) method.

The mitigation module is responsible for forwarding or dropping the packets. For each received request, The TASD mitigation module checks the trust value number of each user. As shown in the algorithm 2, if the trust value is lower than $T(suspicious)$, then the TASD adds a new rule to the load balancer to forward all the requests from the specific source IP address toward the second auto-scaling group. Moreover, a user can be denied access by blocking its IP address in the Network Access Control List if the trust value is lower than $T(malicious)$.

Considering a situation where a benign user has a suspicious behavior in a period, then the trust value assigned to the user will be decreased wrongly, and it may cause degradation in the quality of service. This is a bottleneck in the mitigation mechanism in the suggested algorithm 2 and may cause false-positive mitigation. To optimize the algorithm, TASD either calculates the duration of request forwarding $t_{forward}$ or denies t_{block} rules. So, it keeps rules for a specific period

Algorithm 1 Pseudo code for the AD trust value detection method

```
1: for each request $r$ do
2:     if the user $u$ in request $r$ is not in trust value db then
3:         Add user $u$ with default trust value $T_{init}$ to trust value db;
4:     end if
5:     if Auto-scaling in Scale-out then
6:         $N_i \leftarrow$ number of requests for each users;
7:         $\{S\} \leftarrow k$ users with top $t$ request numbers;
8:     end if
9:     if Auto-scaling in Scale-in then
10:        for each user $u_i$ in $\{S\}$ do
11:            $N`_i \leftarrow$ number of requests for $u_i$;
12:            if $N`$ - $N > M$ then
13:                Update trust value of user $u_i$: $T_i$ = $T_i$ -1;
14:            end if
15:        end for
16:    end if
17: end for
```

Algorithm 2 Pseudo code for the AD trust value mitigation method

```
1: for each request $r$ do
2:     if the trust value of request $T_i < T_{suspicious}$ then
3:         Add user $u$ to ALB forwarding rule;
4:     end if
5:     if the trust value of request $T_i < T_{malicious}$ then
6:         Add deny rule for user $u$ in VPC ACL;
7:     end if
8: end for
```

of $t_{release}$ and then increases the trust value by one. So, if the user shows suspicious/malicious behavior, the trust value will be decreased, and if it shows normal behaviour, the trust value can be increased again. The method is called ADAI (Additive Decrease / Additive Increase). Algorithm 3 describes the Pseudo code of the ADAI algorithm.

Algorithm 3 Pseudo code for the ADAI trust value mitigation method

```
1: for each request $r$ do
2:     if the trust value of request $T_i < T_{suspicious}$ then
3:         if $t_{forward}$ of $u_i > t_{release}$ then
4:             Update trust value of user $u_i$: $T_i$ = $T_i$ +1;
5:         else
6:             Add user $s$ to ALB forwarding rule;
7:         end if
8:         if the trust value of request $T_i$ ¡ $T_{malicious}$ then
9:             if $t_{block}$ of $u_i > t_{release}$ then
10:                Update trust value of user $u_i$: $T_i$ = $T_i$ +1;
11:            else
12:                Add deny rule for $u_i$ in VPC ACL;
13:            end if
14:
```

The algorithm 3 can help to decrease the rate of false positives. We also tried to increase the detection rate. Inspired by the TCP congestion control method called additive-increase/multiplicative-decrease (AIMD) algorithm [18] [19], we introduced a method with a Multiplicative Decrease of trust value to provide a higher detection rate and also Additive Increase of trust value to mitigate false-positive errors. Algorithm 4 describes the Pseudo code of the MDAI algorithm.

Algorithm 4 Pseudo code for the MDAI trust value detection method

```
1: for each request $r$ do
2:     if the user $u$ in request $r$ is not in trust value db then
3:         Add user $u$ with default trust value $T_{init}$ to trust value db;
4:     end if
5:     if Auto-scaling in Scale-out then
6:         $N_i \leftarrow$ number of requests for each users;
7:         $\{S\} \leftarrow k$ users with top $t$ request numbers;
8:     end if
9:     if Auto-scaling in Scale-in then
10:        for each user $u_i$ in $\{S\}$ do
11:            $N`_i \leftarrow$ number of requests for $u_i$;
12:            if $N`$ - $N > M$ then
13:                Update trust value of user $u_i$: $T_i$ = $T_i \times \alpha$ ;
14:            end if
15:        end for
16:    end if
```

Algorithm 5 shows the Pseudo code to mitigate attack with the MDAI method.

Algorithm 5 Pseudo code for the MDAI trust value mitigation method

```
1: for each request $r$ do
2:     if the trust value of request $T_i < T_{suspicious}$ then
3:         if $t_{forward}$ of $u_i > t_{release}$ then
4:             Update trust value of user $u_i$: $T_i$ = $T_i + \beta$ ;
5:         else
6:             Add user $s$ to ALB forwarding rule;
7:         end if
8:         if the trust value of request $T_i$ ¡ $T_{malicious}$ then
9:             if $t_{block}$ of $u_i > t_{release}$ then
10:                Update trust value of user $u_i$: $T_i$ = $T_i$ + $\beta$;
               else
12:                Add deny rule for $u_i$ in VPC ACL;
13:            end if
14:
```

The idea behind the experiments shown in this paper is to test how the proposed algorithms are practical, as opposed to testing them theoretically. This means that the tests will be run like a real business would run a Yo-Yo attack detection system in a customer-facing cloud platform.

C. Test Environment on AWS Resources

The AWS cloud platform is used to test TASD methods. A simple web application has been deployed on AWS Elastic Compute Cloud (EC2), managed by the AWS Auto Scaling group. The TAWS Application Elastic Load Balancer is used to distribute the incoming traffic across multiple EC2 instances. It monitors the health of its registered targets and routes traffic only to the healthy targets.

An AWS launch template without any detection system is created to evaluate and compare the DoS and Yo-Yo attacks. A custom AMI has been created to implement the non-TASD project. Figure 2 illustrates the high-level service design of the AMI without any Yo-Yo attack detection system. It contains a flask web application responding to the requests behind an Nginx web server. To Forward web server requests to the Flask web application, the Gunicorn WSGI middleware has been configured.

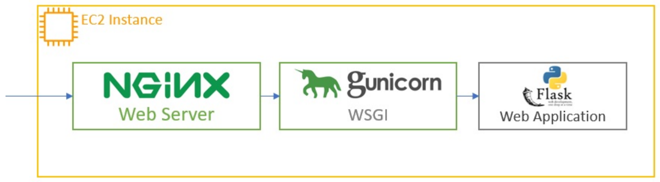

Fig. 2. High-level design services running on EC2 instance; Nginx web server, Gunicorn WSGI and sample Flask web application.

The AWS Auto scaling has been used to activate the auto-scaling feature. It has been configured with the desired capacity of one instance with a minimum instance of one and a maximum of five. After scaling-out or scaling-in instances, auto-scaling waits for a cooldown period to end before any further scaling activities initiated by simple scaling policies can start. The default cooldown was set to a minimum value equal to 30 seconds to have a sharp scaling.

AWS provides different scaling strategies [20] [21]. A simple scaling policy for Amazon EC2 auto-scaling has been configured in the project's experiments. The AWS CloudWatch is used to capture all metrics and trigger the AWS Auto scaling to scale out or scale in the instances.

Elastic Load Balancer publishes data points to Amazon CloudWatch. CloudWatch retrieves statistics about those data points as an ordered set of time-series data, known as metrics. The metric to trigger the auto-scaler is set to ELB RequestCount [22].

The scale-out policy is triggered if the sum of the Request Count exceeds over 1K and the scale-in policy is triggered by the sum of the Request Count goes less than 100. The threshold values have been selected based on the Flask application and Gunicorn WSGI workers.

III. Experiments

The mitigation solution should be used to recognize the Yo-Yo attack from benign traffic. Before implementation of the actual solution, the behavior of a classical DoS attack and a Yo-Yo attack on a cloud service with an auto-scaling feature will be tested. The first experiment simulates a DoS attack on a web service running on EC2 instances inside an auto-scaling group. Then, another test will be applied to show the effect of the Yo-Yo attack on the same service and resources. Finally, a series of Yo-Yo attack scenarios will be simulated to the web service shielded by the AD, ADAI, and MDAI TASD solutions. In all experiments, the Locust tool has been used to send burst traffic toward the cloud service. It is configured with the peak number of concurrent ten users and a spawn rate of two users per second. It means that when Locust starts sending burst traffic, it will spawn two new users every second until it fulfills the total number of 10 users to simulate.

a) Experiment 1: The first experiment is implemented to show the effect of the DoS attack on the auto-scaling mechanism.

b) Experiment 2: The second experiment focuses on the simulation of a Yo-Yo attack on a cloud service with an auto-scaling feature. According to [15], the attacker sends prob requests and compares response time to detect the scale-out status. The same technique is used to detect scale-in.

c) Experiment 3: The experiment focuses on whether the AD TASD system is able to differentiate malicious Yo-Yo traffic from benign traffic. The system is made to evaluate traffic requests' numbers and add a trusted number to each client based on the algorithm 1 and 2. Figure 3 shows the experiment architecture diagram. In the algorithm 1, T_{init} is set to 10, and M is configured to 100 to update the trust value. Also in the algorithm 2 $T_{suspicious}$ and $T_{malicious}$ thresholds set to 7 and 5 respectively.

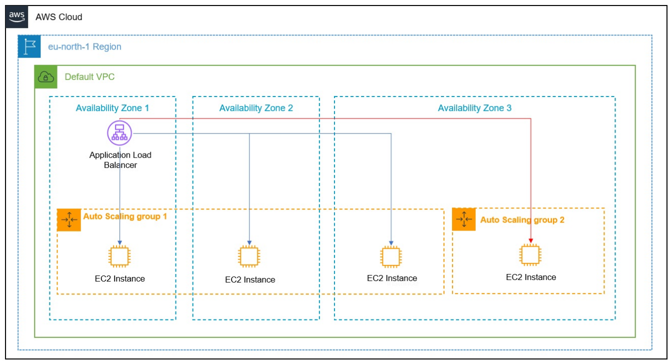

Fig. 3. The project architecture diagram of the deployment TASD system on AWS cloud.

d) Experiment 4: This experiment is implemented to test the ADAI method. The original AD TASD system could add a trust value to each client based on the number of requests. According to section II-B, the original TASD system introduced by [1] may have a false positive error. To optimize the performance of assigning the trust value, the algorithm 2 is modified to algorithm 3.

In the algorithm 1, T_{init} is set to 10, and M is configured to 100 to update the trust value. Also in the algorithm

3 $T_{suspicious}$ and $T_{malicious}$ thresholds set to 7 and 5 respectively. The waiting time to increase the trust value, $t_{release}$, is set to 15 minutes.

e) Experiment 5: This experiment is implemented to test the MDAI method. In the algorithm 4, T_{init} is set to 100, and M is configured to 100 to update the trust value. Alpha which is the multiplicative argument is set to 0.5. Also in the algorithm 5 $T_{suspicious}$ and $T_{malicious}$ thresholds set to 50 and 20 respectively. Also, the incremental argument (Beta) is set to 5 and it increases by $t_{release}$ every 5 minutes.

IV. Results

Different experiments were conducted to assess the efficiency of our algorithms under different scenarios. In this section, we report some representative experiments.

a) Experiment 1: DoS attack and auto-scaling: The first experiment was a DoS attack attempt on a web service running on an EC2 instance inside the AWS Auto scaling group. Figure 4 shows the total requests per second sent with Locust as a DoS attack traffic. The graph has been plated based on the data generated by Locust.

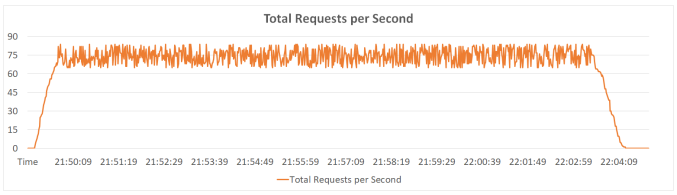

Fig. 4. Total requests per second sent from Locust as a burst traffic to simulate DoS attack.

Figure 5 shows the both in-service and desired instances. The graph obviously shows the time takes from deploying an EC2 instance until the service is getting up.

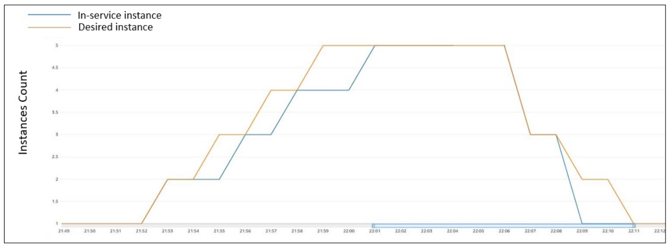

Fig. 5. Auto-scaling group instance count including the in-service and desired capacity.

b) Experiment 2: Yo-Yo attack and auto-scaling: The second experiment was a Yo-Yo attack on a web service running on an EC2 instance inside AWS auto-scaling group. Figure 4 shows the total requests per second from Locust as Yo-Yo attack traffic. The graph has been plated based on the data generated by Locust.

Figure 7 shows the sum of the requests received by each instance in a target group assigned to the load balancer. The red line shows the threshold set as a trigger for the auto-scaling policy.

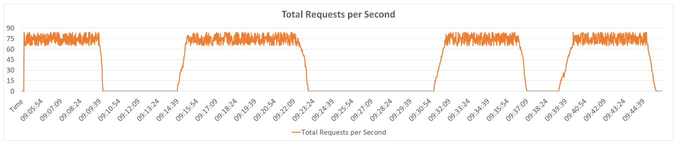

Fig. 6. Total requests per second sent from Locust as a burst traffic to simulate Yo-Yo attack.

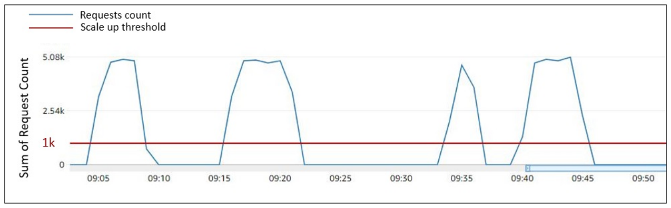

Fig. 7. The sum number of requests received by each instance in a target group assigned to the load balancer.

Figure 8 presents the number of in-service and desired EC2 instances in the auto-scaling group during the experiment.

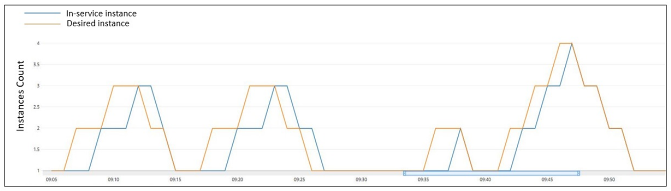

Fig. 8. Auto-scaling group instance count including the in-service and desired capacity.

c) Experiment 3: Yo-Yo attack and AD algorithm : Figure 9 illustrates the number of requests sent via Locust as an attacker toward the web application. The figure shows the AWS ALB traffic shifting and VPC ACL blocking IP address moments. Moreover, figure 10 shows the median response time from the web application captured by the attacker. The response time from instances running in auto-scaling group 1 is around 200 milliseconds. After decreasing the trust value to below 7, the traffic has shifted toward the second auto-scaling group. As the web application was configured to respond with a two-second delay, the response time doubled to 400 milliseconds. Finally, the trust value reached lower than 5, and the IP address was blocked in the AWS VPC ACL.

Figure 11 presents the trust numbers assigned to the attacker during the experiment. It shows that the trust value of the attacker decreases as the number of updates performed by TASD increases. The trust value of the attacker dropped sharply, and it can help the TASD system quickly mitigate the attack.

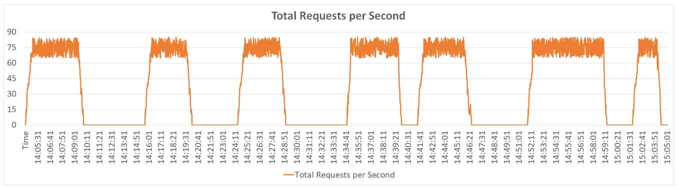

Fig. 9. Total requests per second sent from Locust as a burst traffic to simulate Yo-Yo attack to simulate Yo-Yo attack.

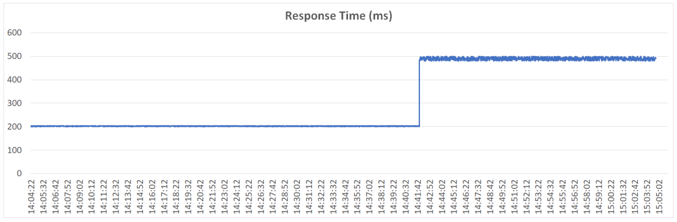

Fig. 10. Web service response time from attacker side view.

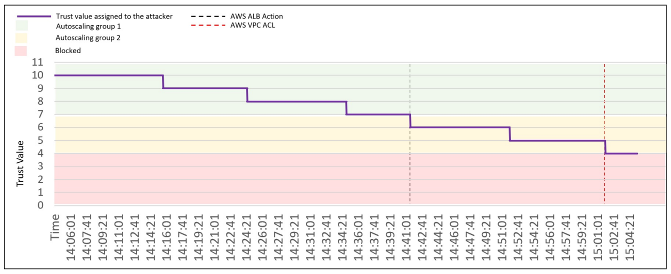

Fig. 11. Trust value assigned to the attacker IP address during the experiment of AD algorithm.

Figure 12 shows the sum of the requests received by each instance in a target group assigned to the load balancer. The red line shows the threshold set as a scale-out trigger for the auto-scaling policy.

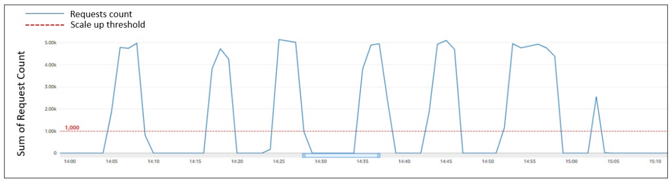

Fig. 12. The sum of requests received by each instance in a target group assigned to the load balancer.

TASD system decides to send traffic toward the auto-scaling group based on the trust value assigned to the clients. Figure 13 shows the number of in-service instance in auto-scaling group 1. The web application running inside this auto-scaling group is the main service for the clients. The figure shows that after shifting traffic toward the second auto-scaling group, the scaling is in a stable status. Figure 14 illustrates the number of instances inside the second auto-scaling group. Although the Yo-Yo attack causes scaling oscillation in the second auto-scaling, the main service running in the first auto-scaling is kept normal and stable.

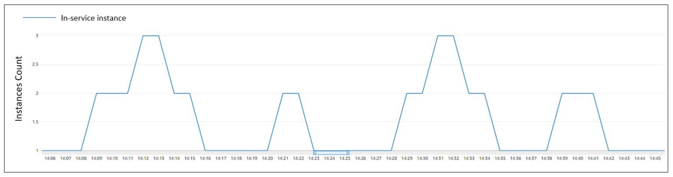

Fig. 13. Number of in-service instances in auto-scaling group 1.

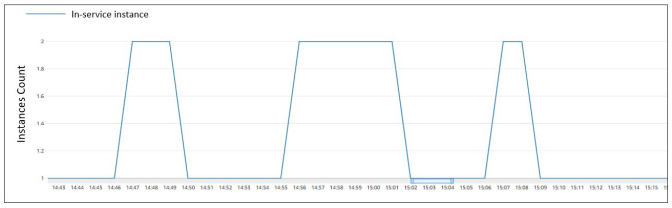

Fig. 14. Number of in-service instances in auto-scaling group 2.

d) Experiment 4: Yo-Yo attack and ADAI algorithm: Figure 15 illustrates the number of requests sent via Locust as an attacker toward the web application. The figure shows the AWS ALB traffic shifting and VPC ACL blocking IP address moments. ADAI calculates the trust value of each user and decides to send traffic to the correct auto-scaling group or block the request. The blocking interval was set to 15 minutes. It means that the TASD increases the user trust value after 15 minutes of normal behaviour. 16 shows the trust value of the attacker during the experiment.

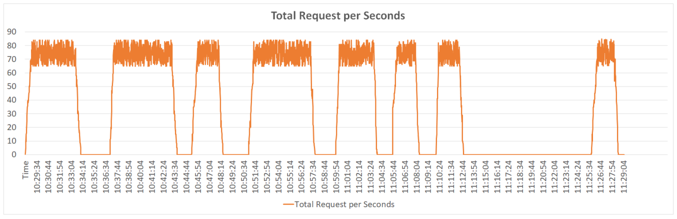

Fig. 15. Total requests per second sent from Locust as a burst traffic to simulate Yo-Yo attack.

Figure 17 illustrates response time of the attacker requests. It shows the response time from the instances inside the first auto-scaling group is approximately 200 milliseconds, and then the traffic shifted toward the second auto-scaling. As a result, the response time increased to 400 milliseconds. Then, the traffic is blocked by decreasing the trust value.

e) Experiment 5: Yo-Yo attack and MDAI algorithm: Figure 18 illustrates the number of requests sent via Locust

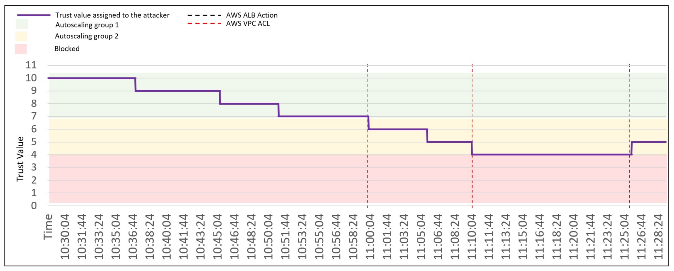

Fig. 16. Trust value assigned to the attacker IP address during the experiment of ADAI algorithm.

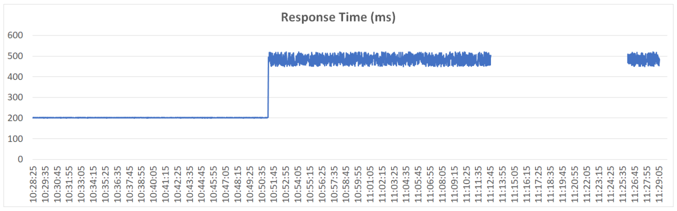

Fig. 17. Web service response time from attacker side view. The gap in the graph shows the blocking interval.

as an attacker toward the web application inside the MDAI detection and mitigation system. MDAI calculates the trust value of each user and decides to send traffic to the correct auto-scaling group or block the request. The blocking interval was set to 5 minutes.

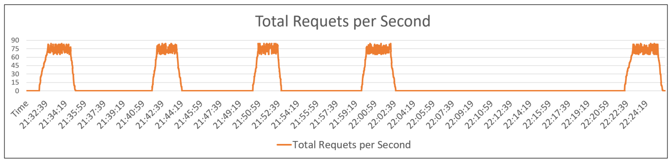

Fig. 18. Total requests per second sent from Locust as a burst traffic to simulate Yo-Yo attack.

Figure 19 shows the amount of traffic received by load balancer. According to the 4 and 5 trust value decreased sharply and it increased additive. Figure 20 illustrates the changes of trust value of the attacker during the experiment.

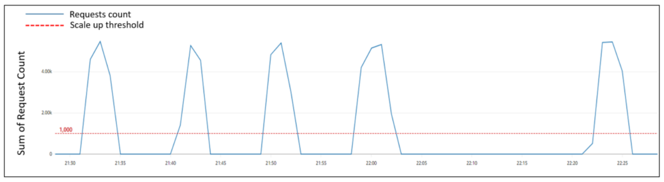

Fig. 19. The sum of requests received by each instance in a target group assigned to the load balancer.

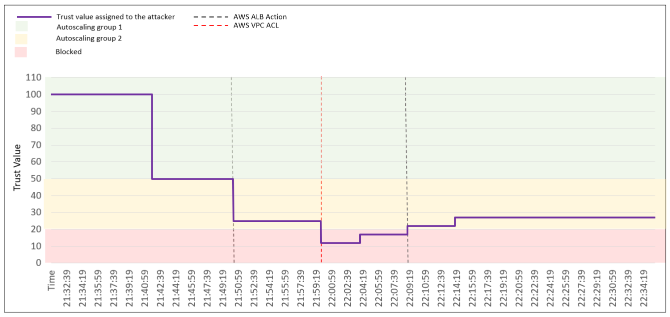

Fig. 20. Trust value assigned to the attacker IP address during the experiment of MDAI algorithm.

V. Discussion

a) DoS/DDoS attack and cloud auto-scaling: Cloud auto-scaling mechanism helps cloud services to meet the demands a workload makes for resources to maintain availability and performance as utilization increases. Sending bursts of traffic via DoS or DDoS attacks causes an increased load on the cloud services. During the experiment, the AWS Auto Scaling policy was configured to trigger based on the Incoming Request Count metric. Also, the maximum instance number was set to five. In this case, as shown in figure 5, the instances increased to maximum capacity during the test. It means the service degraded afterward. Therefore, auto-scaling is not a solution to mitigate DoS/DDoS attacks.

b) Yo-Yo attack and cloud auto-scaling: The term Yo-Yo attack is due to oscillation from the on-attack to the off-attack phase. The experiment proved that the Yo-Yo attack could cause oscillation on instances inside the auto-scaling group. Figure 8 shows the instance oscillation during the experiment. The graph illustrates the time interval between the in-service instance and desired capacity. The interval was nominated as warming time. The warming time of scale-out is the time that it takes to get ready to function, and the warming time of scale-in is the time that an instance allocates to close all services and release resources. 8 shows that the warming time plays a major role in the damage, especially with the simple auto-scaling policy. Whenever the scaling metric threshold was selected close to the maximum capacity of the web application, the service degraded during scale-out because of warming time.

c) Yo-Yo attack and TASD methods: The TASD mitigation module can decrease the number of Yo-Yo attacks as well as the number of scaling. Figure 8 shows the instance scaling status for a non-TASD solution. It shows that during the Yo-Yo attack, instances cycle continuously. The same experiment on a system with the TASD solution shows that the scaling status is getting stable after detecting the attack and blocking the attacker. Figure 13 illustrates that after forwarding the traffic to the second auto-scaling group, as a defense mechanism, the number of instances stays at the stable status. According to the algorithm 2, forwarding to the second auto-scaling helps clients to continue their main workloads on the main servers. Moreover, the TASD can block the malicious server at the VPC level. Blocking at the VPC level helps to prevent increasing load on the load balancer. Therefore, TASD can efficiently mitigate the request load on the web application running on the cloud. Algorithm 3 added an optimization to the initial TASD solution. To decrease the false-positive error of detecting a normal user as a suspicious or malicious user, TASD can update the trust value dynamically. The mitigation module can increase the trust value of a user after an interval. The method is called ADAI (Additive Decrease/Additive Increase). Also, to increase the mitigation speed, another method called MDAI (Multiplicative Decrease/Additive Increase).

d) Yo-Yo attack and Cost: One of the main discussions regarding the Yo-Yo attack and auto-scaling is the economic penalty. The Yo-Yo attack victim needs to pay for the extra resources required to process the bogus traffic and resources, which provide no real benefit to the victim. The cloud users also may affect the financial loss indirectly because of performance degradation and then cause the quality of services decrement.

VI. Conclusion

In this paper, we presented a simple solution to detect a Yo-Yo attack and mitigate it in the cloud auto-scaling mechanism. Our study investigates to what extent the Yo-Yo attack differs from traditional DoS/DDoS attacks. An approach named Trust-based Adversarial Scanner Delaying (TASD) and introduced in [1], is implemented and tested in a real cloud production environment. The original TASD system used an Additive Decrease method to update the trust value. Inspired by the TCP rate control mechanisms, we introduce two optimization methods, ADAI (Additive Decrease/Additive Increase) and MDAI (Multiplicative Decrease/Additive Increase), to optimize the TASD detection and mitigation system. The TASD system is deployed on Amazon Web Services (AWS). The experiment evaluations show that our proposed TASD variants can efficiently detect and mitigate Yo-Yo attacks in a real cloud application.

References

[1] X. Xu, J. Li, H. Yu, L. Luo, X. Wei, and G. Sun, "Towards yo-yo attack mitigation in cloud auto-scaling mechanism," *Digital communications and networks*, vol. 6, no. 3, pp. 369–376, 2020.

[2] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage, "Inferring internet denial-of-service activity," *ACM Trans. Comput. Syst.*, vol. 24, no. 2, p. 115–139, may 2006. [Online]. Available: https://doi.org/10.1145/1132026.1132027

[3] Gupta, B. B, Badve, Omkar P. , "Taxonomy of DoS and DDoS attacks and desirable defense mechanism in a Cloud computing environment," *Neural Computing and Applications*, vol. 28, p. 3655–3682, Dec 2017. [Online]. Available: https://doi.org/10.1007/s00521-016-2317-5

[4] Sam Cook, "DDoS attack statistics and facts for 2018-2022," https://www.comparitech.com/blog/information-security/ddos-statistics-facts/, February 2022, accessed: October 15, 2022.

[5] Kaspersky, "DDoS attacks hit a record high in Q4 2021," https://www.kaspersky.com/about/press-releases/2022_ddos-attacks-hit-a-record-high-in-q4-2021/, February 2022, accessed: October 15, 2022.

[6] Ko R, Lee SSG, "Cloud Computing Vulnerability Incidents: A Statistical Overview," https://downloads.Cloudsecurityalliance.org/initiatives/cvwg/CSA_Whitepaper_Cloud_Computing_Vulnerability_Incidents.zip/, August 2013, accessed: 2014.

[7] M. Darwish, A. Ouda, and L. F. Capretz, "Cloud-based ddos attacks and defenses," in *International Conference on Information Society (i-Society 2013)*, 2013, pp. 67–71.

[8] Alibaba Clouder , "DDoS Attack Statistics and Trend Report by Alibaba Cloud," https://www.alibabacloud.com/blog/ddos-attack-statistics-and-trend-report-by-alibaba-cloud_597607, April 2021, accessed: October 15, 2022.

[9] Alethea Toh, "Azure DDoS Protection—2021 Q3 and Q4 DDoS attack trends," https://azure.microsoft.com/en-us/blog/azure-ddos-protection-2021-q3-and-q4-ddos-attack-trends//, January 2022, accessed: October 15, 2022.

[10] A. B. Usman and J. Gutierrez, "Toward trust based protocols in a pervasive and mobile computing environment: A survey," *Ad Hoc Networks*, vol. 81, pp. 143–159, 2018.

[11] K. Singh, P. Singh, and K. Kumar, "Application layer http-get flood ddos attacks: Research landscape and challenges," *Computers & security*, vol. 65, pp. 344–372, 2017.

[12] Amir Dahan, "Are You Protected Against Burst Attacks?" https://blog.radware.com/security/2018/02/burst-attack-protection/, Feb 2018, accessed: October 15, 2022.

[13] R. B. David and A. B. Barr, "Kubernetes autoscaling: Yoyo attack vulnerability and mitigation," *arXiv preprint arXiv:2105.00542*, 2021.

[14] Tania Lorido-Botran, Jose Miguel-Alonso, Jose A. Lozano, "A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments," pp. 559–592, Oct 2014.

[15] A. Bremler-Barr, E. Brosh, and M. Sides, "Ddos attack on cloud auto-scaling mechanisms," pp. 1–9, May 2017.

[16] V. Danielsen, "Detecting yo-yo dos attack in a container-based environment," Master's Thesis, Oslo Metropolitan University, 2021. [Online]. Available: https://hdl.handle.net/11250/2774518

[17] R. B. David and A. B. Barr, "Kubernetes autoscaling: Yoyo attack vulnerability and mitigation," 2021.

[18] "Additive increase/multiplicative decrease," https://en.wikipedia.org/wiki/Additive_increase/multiplicative_decrease, accessed: October 15, 2022.

[19] "TCP congestion control," https://en.wikipedia.org/wiki/TCP_congestion_control, accessed: October 15, 2022.

[20] "Manual scaling for Amazon EC2 Auto Scaling," https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-manual-scaling.html, accessed: October 15, 2022.

[21] "Dynamic scaling for Amazon EC2 Auto Scaling," https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scale-based-on-demand.html, accessed: October 15, 2022.

[22] "CloudWatch metrics for your Application Load Balancer," https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-cloudwatch-metrics.html, accessed: October 15, 2022.