Yo-Yo Attack - Vulnerability in auto-scaling mechanism

Mor Sides
Interdisciplinary Center
Herzliya, Israel
mor.sides@idc.ac.il

Anat Bremler-Barr
Interdisciplinary Center
Herzliya, Israel
bremler@idc.ac.il

Elisha Rosensweig
Cloudband, Alcatel-Lucent
elisha.rosensweig@alcatel-lucent.com

CCS Concepts

•**Security and privacy** → **Denial-of-service attacks;** *Distributed systems security;* •**Computer systems organization** → **Cloud computing;** *Reliability;* •**Networks** → *Middle boxes / network appliances;*

1. INTRODUCTION

In the last few years, more and more public and private networks rely on cloud and virtualization to provide the service while meeting their SLA commitments. One attractive property of the cloud is its support for rapid elasticity - the ability to scale the number of machines up and down according to the load on the machine, which can be configured to occur automatically, according to customer-set thresholds.

This auto-scaling mechanism provides an ability to cope with many of the basic Distributed Denial of Service (DDoS) attacks (as describe in [4]), but opens the door to a new type of attack, the Economic Denial of Sustainability attacks (EDoS) [2]. In DDoS, an attacker overwhelms the victim with bogus traffic, blocking the service from legitimate users. With a cloud-based operation, the auto-scaling mechanism ensures that a victim can cope with an attack by providing the victim with more resources to handle the attack. This solution, however, comes with an economic penalty termed EDoS, since the victim needs to pay for the extra not beneficial resources that process the bogus traffic.

In many DoS attacks, the danger of the attack impact is mitigated by the expected cost to the attacker: the more effort required on the side of the attacker, who has to invest in generating large amounts of traffic, the less likely it is to occur. In this work we present the 'Yo-Yo attack', an efficient attack on the auto-scaling mechanism, which results in an Economic Denial of Sustainability attack (EDoS) that is difficult to detect. The attack cycles between two phases repeatedly: In the *on-attack* phase, the attacker sends a short burst of traffic that causes the auto-scaling mechanism to perform a scale up. In the *off-attack* phase, the attacker stops sending the excess traffic. This second phase takes place when the attacker identifies that the scale up has occurred. Once the attacker determines that the scale down occurred, the process is repeated. Clearly, the strength of the attack is partially determined by the ability of the attacker to determine when to switch between the two phases.

SIGCOMM '15 August 17-21, 2015, London, United Kingdom
© 2015 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-3542-3/15/08.
DOI: http://dx.doi.org/10.1145/2785956.2790017

We demonstrate the Yo-Yo attack on Amazon cloud service, and show how the attacker can detect the state of the auto-scaling mechanism. The incentive of the attacker is to reduce the cost of the attack, i.e., minimize the attack traffic, and produce maximum damage, i.e., maximize the number of extra VMs used to process the attack. Reducing the attack cost has two impacts: the attacker uses his resources efficiently, and the attack is harder to detect. In our experiments over AWS, the attacker remained in the *off-attack* phase approximately 77% of the time, while the victim is constantly in flux.

The Yo-Yo attack can also be considered a Reduction of Quality (RoQ) attack [3]. RoQ attacks aim to keep an adaptive mechanism oscillating between over-load and under-load conditions. In other areas, it has been shown that load-balancing mechanisms are vulnerable to such attacks. To the best of our knowledge, our work is the first to demonstrate such a vulnerability in the auto-scaling mechanism.

2. AUTO-SCALING MECHANISM

An auto-scaling mechanism is standardly managed by the orchestrator component (also known as service controller). Each cloud solution comes with its own auto-scaling engine: Heat in Openstack, autoscaler in Google Cloud, and auto-scaling in Amazon Elastic Compute Cloud (Amazon EC2) [1]. Basically, in each of these systems the underlying algorithms lets the cloud customer, referred to in our work as the *user*, to define a scaling criteria, and the corresponding thresholds for overload and underload. In Amazon auto-scaling, which is where we conducted our experiment, the possible metrics to use for this criteria are CPU utilization, in/out network traffic in bytes and disk read/write in bytes or operations. In our experiment we choose the CPU utilization criteria, and correspondingly implement a service where each connection requires high CPU consumption.

Each user needs to configure rules for performing scale up and down, as well as the minimum and maximum number of machines allowed. Each scale rule is defined by a threshold and a scale-interval, s.t. if the threshold was exceeded for the duration of the scale-interval, the scaling operation is performed. For example in our experiment, we defined that scale down is performed if the CPU utilization is below 10% for 1 minute. We begin with a single machine, the minimum

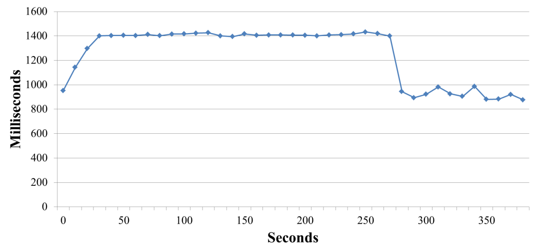

Figure 1: Scale up approximation

amount allowed, and cap it at two machines at maximum. Another important configurable parameter is the cooldown period, i.e., the number of seconds after a scaling activity completes before another can begin (in our experiment this was set to two minutes). We note that according to our experiments the initialization of a new machine takes around 2-2.5 minutes, which includes the time duration between the scale up event starts and a new machine is allocated, until the service is up on the new machine.

The attacker motivation is to maximize the attack potency, which as defined in [3] is equal to k/m, where k is the fraction of time an extra machine is required by user, and m is the fraction of time an attacker must be transmitting traffic. If the internal parameters are known to the attacker, which is not the usual case, the attacker can easily optimize its attack cost. The attack should be in the *on-attack* phase for a duration equal to the scale-up-interval, and then switch to *off-attack* for the duration equal to the initialization of a new machine plus the maximum time between scale-down-interval and the cooldown period. While these parameters are usually unknown to attacker, we find that the attacker can determine a good approximation for them. The key idea is that scale is done usually in order to improve the response time, thus the response time reveals some information on the state of the auto-scaling mechanism. Figure 1 shows the response time (in milliseconds) as a function of the elapsed time (in seconds), where the attack began at $t = 10$ seconds. It is clear that after 280 seconds a new machine is up and running, since the response time is reduced by half.

3. YO-YO ATTACK

We named our attack the *Yo-Yo attack* since the attacker oscillates from the *on-attack* phase to the *off-attack* phase. We demonstrate the Yo-Yo attack on Amazon, using simple attack where the goal of the attacker is to add extra machine and thus performing a EDoS attack. In our work here we simulate the best case attack from the attacker perspective, where we assume the attacker is aware of when scaling occurs relying on diagnosis Fig.1.

Figure 2 shows the number of machines as a function of attack time, where we compares a brute force DDoS attack that sends a constant amount of traffic to a Yo-Yo attack where the factor between *off-attack* to *on-attack* is almost 3.5, i.e., reducing the attack cost in about 77%. In our experiment the damage is the time the user has two machines running, and the cost is the fraction of time during which the *on-attack* phase is active. In a full DDoS attack, the attack is always on and thus the potency is 1, while in our simulated Yo-Yo attack the potency is 0.71/0.23 which is 3.08. Thus the attacker is 3 time more effective.

Figure 3 demonstrates that in Yo-Yo attack, there is additional damage in the form of high response time to a regular client. The regular client suffer a high response time while the system is under attack and the scale up is in process. While in constant DDoS attack, it happens only once at the begining, under the Yo-Yo attack it happens every *on-attack* phase. We note that in stateful services the damage can be larger. If the client's state move to another VM in every scale action it will increase the scaling duration. Hence a scheduling mechanism should minimize the number of times a client changes machine. Designing such scheduling mechanism is part of our ongoing research.

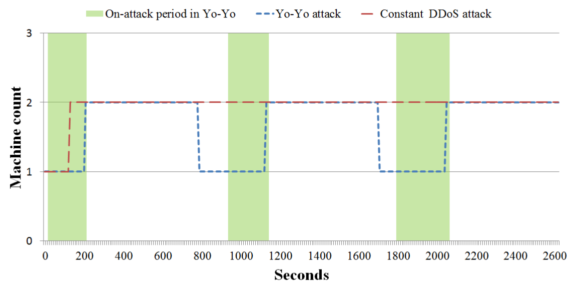

Figure 2: Machine count comparison

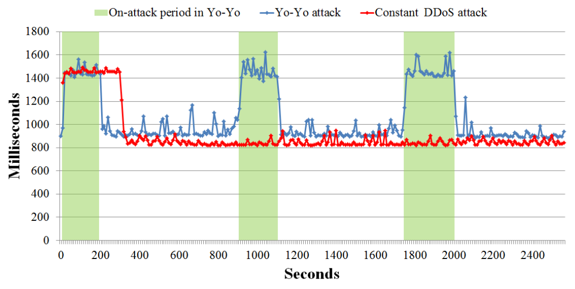

Figure 3: Response time comparison

4. CONCLUSION

In this work we shed some light on the potential of exploiting the auto-scaling mechanism to perform an efficient attack, that impacts the cost of a service and the response time of standard users. While part of the promise of Network Function Virtualization (NFV) is that network services and middleboxes will enjoy the auto-scaling property, our work shows that special care should be taken to prevent attackers from using this mechanisms to reduce the value of NFV.

5. REFERENCES

[1] Amazon web services, auto scaling. http://aws.amazon.com/autoscaling/.

[2] Z. A. Baig and F. Binbeshr. Controlled virtual resource access to mitigate economic denial of sustainability (edos) attacks against cloud infrastructures. In *Cloud Computing and Big Data (CloudCom-Asia)*, 2013.

[3] M. Guirguis, A. Bestavros, I. Matta, and Y. Zhang. Reduction of quality (RoQ) attacks on dynamic load balancers: Vulnerability assessment and design tradeoffs. In *INFOCOM*, 2007.

[4] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield. Split/merge: System support for elastic execution in virtual middleboxes. In *NSDI*, 2013.

This research was supported by European Research Council (ERC) Starting Grant no. 259085.