DDoS Attack on Cloud Auto-scaling Mechanisms

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

Eli Brosh
Nexar Inc
elibrosh@getnexar.com

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

***Abstract*—Auto-scaling mechanisms are an important line of defense against Distributed Denial of Service (DDoS) in the cloud. Using auto-scaling, machines can be added and removed in an on-line manner to respond to fluctuating load. It is commonly believed that the auto-scaling mechanism casts DDoS attacks into Economic Denial of Sustainability (EDoS) attacks. Rather than suffering from performance degradation up to a total denial of service, the victim suffers only from the economic damage incurred by paying for the extra resources required to process the bogus traffic of the attack.**

Contrary to this belief, we present and analyze the Yo-Yo attack, a new attack against the auto-scaling mechanism, that can cause significant performance degradation in addition to economic damage. In the Yo-Yo attack, the attacker sends periodic bursts of overload, thus causing the auto-scaling mechanism to oscillate between scale-up and scale-down phases. The Yo-Yo attack is harder to detect and requires less resources from the attacker compared to traditional DDoS. We demonstrate the attack on Amazon EC2 [4], and analyze protection measures the victim can take by reconfiguring the auto-scaling mechanism.

***Index Terms*—Cloud attack; Auto-scaling; Denial-of-service attack; Economic-Denial-of-Sustainability attack; Distributed systems security;**

I. Introduction

In the last few years, more and more public and private networks have come to rely on cloud environments and virtualization to provide 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 virtual machines up and down according to the load. This scaling can often be configured to occur automatically, according to customer-set thresholds.

The main purpose of the auto-scaling mechanism is to cope with changes in the traffic load. While it is mainly used to cope with predictable changes that arise from known characteristics of the service (i.e., peak hours), it is also recommended as a remedy to cope with unpredictable loads that may arise from flash crowds or even malicious Distributed Denial of Service (DDoS) attacks. Amazon lists the auto-scaling mechanism as one of the best practices for dealing with Distributed Denial of Service (DDoS) [7]. In DDoS, an attacker overwhelms the victim with bogus traffic, blocking the service from legitimate users. There are a large variety of DDoS attack types and correspondingly a large variety of defense lines. Auto-scaling mechanisms in particular serve as a defense line against application-level attacks, when the served content is dynamic. [1]

With a cloud-based operation, the auto-scaling mechanism ensures that a victim can cope with an attack by providing additional resources for handling the extra traffic. This solution, however, comes with an economic penalty, termed Economic Denial of Sustainability attacks (EDoS). The victim needs to pay the cloud infrastructure provider for the extra resources required to process the bogus traffic, resources which provide no real benefit to the victim. However, auto-scaling is regarded as a good enough solution, since it ensures that the service will continue to run with good performance. There is a clear incentive for the cloud provider to market auto-scaling since it benefits from extra income. Moreover, alternative remedies, such as DDoS scrubbers middleboxes or DDoS scrubber cloud services [1], [30], which also cost the victim money, might fail to mitigate the attacks. [2]

In this paper, we show that contrary to the common belief, a shrewd attacker can cause substantial performance damage, up to repeated episodes of total denial of service, on top of the economic damage. We present and analyze the Yo-Yo attack, where the attacker sends periodic bursts of overload, thus causing the auto-scaling mechanism to oscillate between scale-up and scale-down. During the repetitive scale-up process, which takes several minutes and cannot be prevented [20], the cloud service suffers from substantial performance penalty. When the scale-up process is finished, the attacker stops sending traffic, and waits for the scale-down process to start. When the scale-down ends, the attacker begins the attack again, and so on. Overall the cloud service will suffer a substantial performance penalty for almost half the duration of the attack. Moreover, when the scale-up process ends, there are extra machines but no extra traffic. Thus the victim pays for the machines in vain. Notice that these short bursts are harder to detect and also reduce the cost of the attack to the attacker compared to a traditional continuous DDoS attack.

We demonstrate the Yo-Yo attack on the Amazon cloud service under different configurations and analyze the damages. Auto-scaling mechanisms employ one of two common policy

[1]TCP DDoS attacks, such as the spoofed SYN-attack [35], are mitigated at the TCP level. In application level DDoS attacks, CDN solutions can help deal with with faked static requests, while the defense line for dynamic content is the auto-scaling mechanism. In Amazon the best practices for DDoS resiliency include the following defense lines: running the CloudFront CDN solution [5] with Route 53 [9] to mitigate DNS DDoS attacks, and the auto-scaling mechanism to cope with dynamic content. The Elastic Load balancer [8] and CloudFront also mitigate TCP level attacks, such as the SYN-attack.

[2]In the case of dynamic content, most DDoS scrubbers try to find a signature of attacks, which might not exist.

types: The first is the discrete scale policy, which adds one or a few machines at a time, checks whether the problem has been resolved, and if not continues to add machines iteratively. The second is the adaptive scale policy, which tries to estimate the number of machines required to cope with the traffic load and adds them at once. We model the Yo-Yo attack under the two polices. In the discrete scale policy, we show that if the burst in the load is up to k more than the original load, the victim will pay for approximately $\frac{k}{2}$ more machines, and an average extra load on machine will be logarithmic in k. In the adaptive model, on the other hand, we show that the economic damage and the extra load are linear with k. In a representative use case, this is translated to a requirement of extra $\frac{k}{2}$ machines and the average extra load will be $\frac{k}{2}$. Thus, while under the adaptive auto-scaling policy the system is guaranteed to adapt to the extra load in shorter time, this policy is shown to be more vulnerable than the discrete policy.

The Yo-Yo attack requires the attacker to infer the state of the auto-scaling mechanism of the victim (i.e., whether the system is in the middle of scale-up or scale-down). We show that it is feasible for the attacker to detect the state of the auto-scaling mechanism by sending probe packets that measure the response time. To to best of our knowledge, we are the first to analyze such attacks. We discuss possible defense strategies using the auto-scaling mechanisms. Our work shows a possible explanation for the recently reported behavior of attacks which come in repeated waves.

The remainder of this paper is organized as follows. Section II describes cloud scaling characteristics and existing attacks in the cloud area. Section III presents the Yo-Yo attack in detail. Section IV introduces the related work. In Section V we model the attack, analyze it mathematically and compare it to the DDoS attack. In Section VI we evaluate the Yo-Yo attack and assess the impact of our attack on a real cloud service infrastructure. In Section VII we discuss possible defense strategies. In Section VIII we present our conclusions.

II. Cloud Auto-Scaling

Auto-scaling is a cloud computing service feature that automatically adds or removes compute resources depending upon actual usage. Each cloud solution comes with its own auto-scaling engine: Heat in Openstack [27], autoscaler in Google Cloud [12], Azure Autoscale in Microsoft Azure [11] and auto-scaling in Amazon Elastic Compute Cloud (Amazon EC2) [4]. In each of these systems the underlying algorithm lets the cloud customer, referred to in our work as the *user*, to define a scaling criterion and the corresponding thresholds for overload and underload. In Amazon auto-scaling, which we used in our experiments, the possible metrics for this criterion are CPU utilization, in/out network traffic in bytes, and disk read/write in bytes or operations.

Each user needs to configure rules for performing scale-up and scale-down of the group of machines, as well as the minimum and maximum number of machines allowed. Each scale rule is defined by a threshold, scale interval and action, s.t. if the threshold was exceeded for a duration of the scale interval, the action is performed. We denote by I_{up} and I_{down} the scale interval for scale-up and scale-down actions.

Another important configurable parameter is the scale policy type, which determines how the scaling action is performed: discretely or adaptively. In the *discrete policy* the number of machines increases or decreases by a fixed, predefined number. If the overload was not resolved the process is continued iteratively. In the *adaptive policy* the number of machines increases or decreases differentially and is adaptive to the system load.

Google cloud scaling is always adaptive [3]. The user sets the target criterion value, e.g., target CPU utilization of 75%, and the autoscaler makes scaling decisions proportionally and maintains that level without the user having to set rules. It is assumed that Google uses a machine learning algorithm for the adaptive scaling.

In Amazon EC2 the user should specify a configuration for the auto-scaling algorithm. [4] In the *adpative policy* the user defines several thresholds for the relevant scaling criterion, and the corresponding number of machines to upload for each threshold. For example, for the CPU utilization criterion, the number of machines to upload if the CPU utilization during the scale interval is above 50% will be different from the number of machines to upload if it is above 80%. Thus, the *adaptive policy* is more adaptive to the condition of the service; however, it is more complex to configure. In this paper we chose to demonstrate the Yo-Yo attack on Amazon EC2 since it allows us more control over the auto-scaling algorithm. However, we note that we observed similar results in the Google environment.

After a scaling decision is made, it takes time until the machine is ready to function with the appropriate service software. This time is called the *Warming time* and, we denote it by W_{up}. Mao et al. [20] show that this time is between 1 and 13 minutes, depending on the infrastructure provider, OS server, service initialization time and other factors. In the auto-scaling solution for predicted rush-hour, machines can be loaded early [28] in order to avoid the latency of warming time. While this solution is useful in the case of predictable load, it is obviously not applicable to our case of DDoS attacks.

Scaling down, denoted by W_{down}, is the time it takes to backup the service and release all the resources. It might be shorter than W_{up}, if the service does not have a long backup operation.

Table I summarizes the notation we use throughout the paper.

The auto-scaling policy might influence the autoscaling parameters. We denote by superscript letter 'd', the parameters for the discrete policy, and by superscript letter 'a' the parameters for the adaptive policy.

[3]Google autoscaler is a black-box to the user and there is no public information about the algorithm.

[4]The discrete policy carried out using 'Simple scaling' or single step of 'Step scaling'. The Adaptive policy carried out using multiple steps of 'Step scaling'.

Parameter	Definition	Configured by
r	Average requests rate per unit time of legitimate clients	Given by system usage
m	Number of machines	System administrator
$I_{up} \backslash I_{down}$	Threshold interval for scale-up and scale-down	
$W_{up} \backslash W_{down}$	Warming time of scale-up and scale-down	Given by system infrastructure
k	The power of the attack	Attacker
n	Number of attack cycles	
T	Cycle duration	
$t_{on} \backslash t_{off}$	Time of *on-attack* phase and *off-attack* phase. $T = t_{on} + t_{off}$	

TABLE I: Notation used in the model description

For example, I^d_{up} might be a different value than I^a_{up}. Note that W_{up} and W_{down} are not influenced by the scaling policy because they depend on the properties of the machine and the application running on it.

Auto-scaling is a defense line against DDoS attacks, which flood a target with fake requests of dynamic content. Nowadays, a cloud environment usually has extensive resources, and dynamic resource allocation , that can handle effectively such attacks.

In order to understand how auto-scaling helps to mitigate DDoS, we use here a very simplified and basic model (see Section V) using the parameters of a medium-size e-commerce web site [29] (See Table II for a summary of the parameter values). We assume the site has, in steady state, an average rate of 10,000 dynamic requests, with 10 machines. We define *power of the attack* as the extra load per machine during the attack, which in our case is equal to 2. Thus the site is attacked with an additional 20,000 requests (triple the average request rate).

Parameter	Value
r	10000 Requests per minute
m	10 machines
$I^d_{up}, I^d_{down}, I^a_{up}, I^a_{down}$	1 minutes
W_{up}, W_{down}	2 minutes
k	2 - power of the attack

TABLE II: Parameter values of use case example

As Figure 1 shows, adaptive scaling action occurred after an interval of I_{up}, and 20 machines were added and ran for the duration of the attack (see Figure 1, top). The system overload was experienced only for the duration of I_{up} plus W_{up}, until the machines were up and function (see Figure 1,bottom). Figure 2 shows the impact of the same attack on an environment with a discrete policy. Here, the system overload is felt for a longer duration.

Since the cloud services are provided as pay-per-use [17], the services under attack cost more, thus leading to an Economic Denial of Sustainability attack (EDoS)[16]. In our example the user would be requested to pay for almost 20 extra machines for the entire duration of the attack.

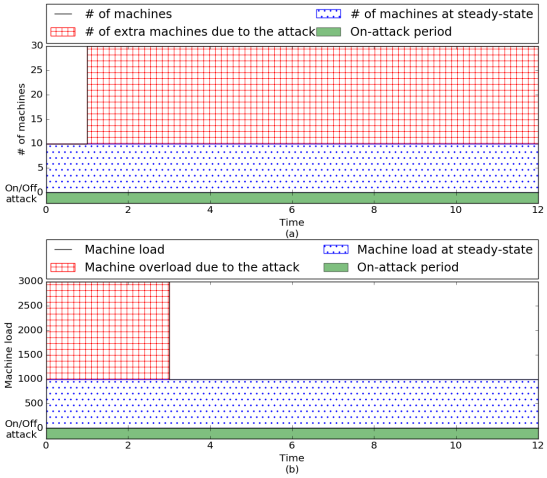

Fig. 1: DDoS attack on system with adaptive scale policy

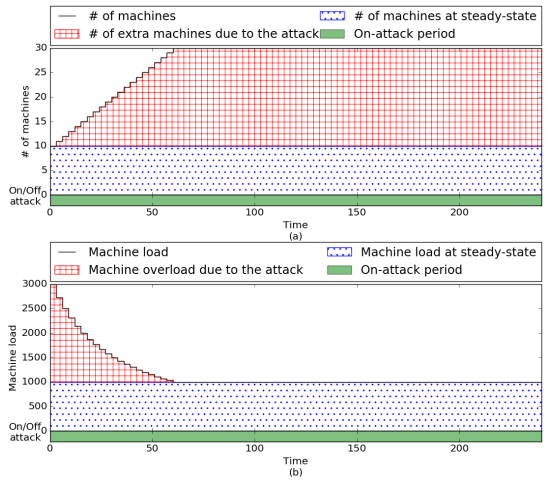

Fig. 2: DDoS attack on system with discrete scale policy

III. Yo-Yo Attack

In the *Yo-Yo attack* the attacker oscillates between the *on-attack* phase and the *off-attack* phase. In the *on-attack* phase, the attacker sends a 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

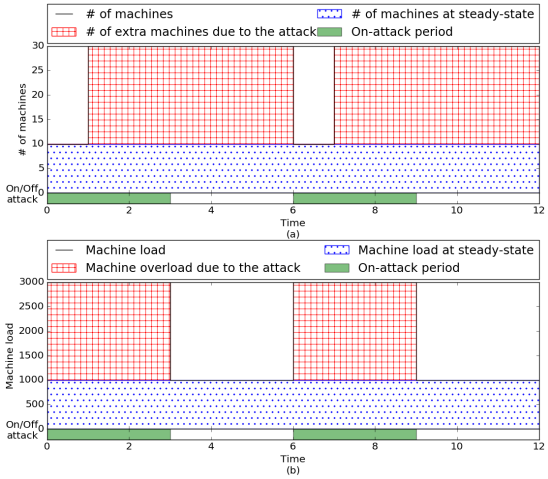

Fig. 3: Yo-Yo attack on system with adaptive scaling policy

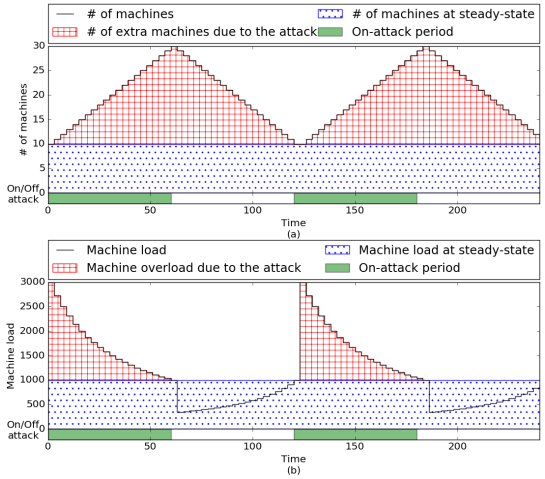

Fig. 4: Yo-Yo attack on system with discrete scaling policy

has occurred (all machines are up and the service is fully functional) and continues until the attacker determines that scale down has occurred. Then, the process is repeated and the attacker will be on-attack again.

Figure 3 demonstrates the Yo-Yo attack on an environment with adaptive scaling policy and Figure 4 on an environment with discrete policy on the same use case of a medium-size commercial site (see table II). The attack causes both economic and performance damage. In section V we present a damage analysis model ; Figures 3 and 4 are derived from that model.

In the adaptive scaling policy, during the *on-attack*, after the scale-up interval (I^a_{up}) of 1 minute, all the extra machines are up and the site needs to pay for them, but note that the machines are still not fully functional, since they require an extra two minutes warning time (W_{up}). Only then will there be no extra load on the machines, and hence no performance degradation. The attacker will stop sending traffic at this point and the off-attack stage will begin, during which the victim will experience economic damage, until scale-down occurs. Then, the entire process will be repeated.

In the discrete scaling policy, the number of machines increases linearly during the scale-up and decreases linearly during scale-down. In addition, at the beginning of each on-attack phase, the site suffers from substantial performance degradation. Afterwards, the load on the machines drops logarithmically, because the load is distributed over a linearly increasing number of machines.

Clearly, the damage from the attack is partially determined by the ability of the attacker to determine when to switch between the two phases. In subsection VI-A we show how an attacker can estimate the state of autoscaling by sending probe messages to the site and measuring the response time. The attacker can also utilize the fact that some cloud providers, such as Amazon, have default values for these parameters. For now, we simulate the best-case attack from the attacker's perspective, where we assume the attacker is aware of the scaling parameters and whether the system is scaling up or down.

The attack, as described here, requires some synchronizing between the on-attack and off-attack phases in the case of multiple origins of attack, such as multiple botnets. However, it is clear that low precision synchronization is enough for this attack. Nowadays computers commonly run some basic clock synchronization protocol such as NIST[25] service or NTP[26].

A. *Other Variants of the Yo-Yo Attack*

In the Yo-Yo attack, the attacker aims to optimize the damage of both economic and performance. We note that there can be other variants of the Yo-Yo attack that optimize only one damage criteria, economic or performance. For example, attacker might choose to inflict only performance damage, by sending during the on-attack phase, bursts with duration smaller than but close to the scale-up interval. Thus scale-up process will not occur, and the only damage will be to performance damage. Similarly, might choose to inflict only economic damage by sending small bursts at frequency high enough to prevent scale down from occuring.

Further discussions of these variants can be found in a technical report [36]. For lack of space we note here only that these variants are hard to launch, since doing so requires more detailed knowledge of the auto-scaling system.

IV. Related Work

Security, and DDoS prevention in particular, are crucial to every computing environment, especially in the cloud [39]. Several recent works and whitepapers [7], [23] recommend auto-scaling as a possible solution to mitigate DDoS attacks especially when dealing with dynamic content. The papers [38], [32] describe how a traditional DDoS attack can be transformed into an Economic Denial of Sustainability attack (EDoS) in the cloud environment.

Several work have tried to mitigate EDoS attacks [33], [10], [18], [22]. Their proposed solutions are focused on classification of the clients and the ability to determine whether the user is legitimate or whether it is a malicious bot. All of these works ignore the effect of the autoscaling mechanism and attacks of the Yo-Yo type.

The Yo-Yo attack can also be seen as a type of Reduction of Quality (RoQ) [15] attack. RoQ attacks aim to keep an adaptive mechanism oscillating between overload and underload conditions. In other areas, it has been shown that many mechanisms, such as load-balancing [14], or even TCP retransmission [19], are vulnerable to such attacks. The Yo-Yo attack is the first to show that the auto-scaling mechanism is vulnerable as well.

There are several papers [3], [2], [34], [37] that discuss how to perform efficient auto-scaling, but they ignore the security vulnerabilities. A brief announcement of this work was presented at [31].

V. Analysis of YoYo Attack

In this section we analyze the Yo-Yo attack using a simplified model. From the model we gain insight into the attack and the effect of different autoscaling policies. Reader interested only in the outcomes of this model should skip to subsection V-D.

Consider a common setting in Cloud environment that includes identical service machines behind a load balancer. Requests arrive with an average rate of r requests per unit time, and the load balancer distributes them to m machines in the steady state.

The scaling policy configuration contains threshold intervals, denoted I_{up} and I_{down}, for the scale-up and scale-down processes, respectively, and warming times, denoted W_{up} and W_{down}, respectively. Recall that, we denote by superscript letter 'd', parameters for the discrete policy, and by superscript letter 'a' the parameters for the adaptive policy.

A. Yo-Yo attack formalization

We model the Yo-Yo attack as n cycles where each cycle duration is T and is comprised of an *on-attack* period, denoted as t_{on}, and an *off-attack* period, denoted as t_{off}. Thus T is equal to t_{on} plus t_{off}. Let k be the power of the attack. We assume that in the *on-attack* period, the attacker adds fake requests k times more than the rate in the steady state (i.e., a total rate of $(k+1)\cdot r$), while in the *off-attack* period t_{off}, the attacker does not send any traffic. See Table I for notation summary.

In order to simplify the analysis, we assume that in order to cope with additional $k\cdot r$ traffic, the auto-scaling will scale up to all additional $k\cdot m$ machines. We also assume the attacker has full knowledge of the autoscaling parameters. Thus the results give an upper bound on the damage from Yo-Yo attacks.

Under the adaptive scaling policy, after the attack has been perpetrated for $I^a_{up}+W_{up}$ time, the scaled up mechanism triggers the addition of $k\cdot m$ machines. Thus, in the adaptive policy the attack would choose: $t^a_{on} = I^a_{up}+W_{up}$; and $T^a = I^a_{up}+W_{up}+I^a_{down}+W_{down}$.

Under the discrete scaling policy, every $I^d_{up}+W_{up}$, only one machine is added. Thus the attacker would choose : $t^d_{on} = k\cdot m\cdot(I^d_{up}+W_{up})$; and the total duration of a cycle, including scaling up and scaling down $k\cdot m$ machines is: $T^d = k\cdot m\cdot(I^d_{up}+W_{up}+I^d_{down}+W_{down})$.

B. Damage analysis

We define $D^{attack}_p(k)$, the performance damage caused by an $attack$, DDoS or Yo-Yo, as a function of k, the power of the attack, and assess it as the average extra load on a machine in the system during the total attack time. We note that this is a simplified assumption, since due to network protocols (such as TCP and HTTP), the actual impact on client performance is more complicated to analyze. Similarly, we define $D^{attack}_e(k)$, the economic damage caused by the attack, and assess it as the average extra machines running in the system for the duration of attack.

We define relative damage as the ratio between the damage following the attack and the corresponding value measured at steady state. We denote $RD^{attack}_p(k)$, $RD^{attack}_e(k)$ as the performance relative damage and economic damage correspondingly.

From the definition it is clear that in DDoS without auto-scaling, $RD^{DDoS}_p(k)=k$ and $RD^{DDoS}_e(k)=0$, i.e. the extra load on each machine is k times the regular load and there is no extra machines. In DDoS with auto-scaling, $RD^{DDoS}_e(k)=k$ but $RD^{DDoS}_p(k)=0$. The load is as in the steady state since the solution uses extra k times machine than in the steady state.

1) Yo-Yo attack with adaptive scaling policy: Here all the machines are scaled at once (see Figure 3). Thus the economic damage is:

$$D^{YoYo^a}_e = \quad (m\cdot(k+1)-m)\cdot\frac{(T^a-I^a_{up})\cdot n}{T^a\cdot n} = k\cdot m\cdot\frac{T^a-I^a_{up}}{T^a}$$

In the general case, with respect to the economic state in the steady-state, the relative economic damage is:

$$RD^{YoYo^a}_e(k) = \quad \frac{k\cdot m\cdot\frac{T^a-I^a_{up}}{T^a}}{m} = k\cdot\frac{T^a-I^a_{up}}{T^a}$$

The performance damage is estimated as the overload on a machine as a result of the attack. In the steady state the load on a machine is $\frac{r}{m}$ while during the attack the load is $\frac{r\cdot(k+1)}{m}$. The performance damage is experienced until the machines are running and functioning, i.e., until I^a_{up} plus W_{up}.

$$D^{YoYo^a}_p = \quad (\frac{r\cdot(k+1)}{m}-\frac{r}{m})\cdot\frac{(I^a_{up}+W_{up})\cdot n}{T^a\cdot n} = \frac{k\cdot r}{m}\cdot\frac{I^a_{up}+W_{up}}{T^a}$$

In the general case, with respect to the load in steady-state, the relative performance damage is :

$$RD^{YoYo^a}_p(k) = \quad \frac{\frac{k\cdot r}{m}\cdot\frac{I^a_{up}+W_{up}}{T^a}}{\frac{r}{m}} = k\cdot\frac{I^a_{up}+W_{up}}{T^a}$$

In order to simplify the equation of relative damage, here we made the simplified assumption that the scale up and scale down parameters are equal, i.e., $I^d_{up}=I^d_{down}$ and $W_{up}=W_{down}$. Thus, we obtain $RD^{YoYo^a}_p(k)=\frac{k}{2}$.

2) Yo-Yo attack with discrete scaling policy: Note that the number of machines increases and decreases linearly (see Figure 4). Thus:

$$D_e^{YoYo^d} = \sum_{i=m}^{m\cdot(k+1)} (i-m)\cdot\frac{(I_{up}^d + W_{up} + I_{down}^d + W_{down})\cdot n}{T^d\cdot n} = \frac{k\cdot m\cdot(k\cdot m+1)}{2}\cdot\frac{I_{up}^d + W_{up} + I_{down}^d + W_{down}}{T^d}$$

By assigning the value of T^d and normalizing it by the number of machines in the steady state, we obtain:

$$RD_e^{YoYo^d}(k) = \frac{\frac{k\cdot m\cdot(k\cdot m+1)}{2}\cdot\frac{1}{k\cdot m}}{m} = \frac{k\cdot m+1}{2\cdot m} \approx \frac{k}{2}$$

During the attack the general load $r\cdot(k+1)$ is divided on a linearly increasing number of machines. This results in the following a harmonic series: [5]

$$D_p^{YoYo^d} = \sum_{i=m}^{m\cdot(k+1)}\left(\frac{r\cdot(k+1)}{i} - \frac{r}{m}\right)\cdot\frac{(I_{up}^d+W_{up})\cdot n}{T^d\cdot n} \approx r\cdot\left((k+1)\cdot ln(k+1) - k - \frac{1}{m}\right)\cdot\frac{(I_{up}^d+W_{up})}{T^d}$$

In order to simplify the equation of relative damage, we again made the simplified assumption that the scale up and scale down parameters are equal. We then assigning the value of T^d and normalize it by the load on a machine in the steady state, obtaining:

$$RD_p^{YoYo^d}(k) \approx \frac{r\cdot((k+1)\cdot ln(k+1)-k-\frac{1}{m})}{\frac{r}{m}}\cdot\frac{1}{2\cdot k\cdot m} \approx \frac{(k+1)\cdot ln(k+1)-k}{2\cdot k} \approx \frac{ln(k+1)-1}{2}$$

C. Cost and Potency Analysis

Potency is a metric that measures the effectiveness of DDoS attack and the RoQ attack in particular. It was first defined in [13], as the ratio between the damage caused by an attack and the cost of mounting such an attack. An attacker would be interested in maximizing the potency, i.e., the damage per unit cost.

Specifically, here we define $P_e^{attack}(k)$, the economic potency of an *attack*, DDoS or Yo-Yo, to be the ratio between the relative economic damage $RD_e^{attack}(k)$ caused by attack with power k and $C^{attack}(k)$, the cost of mounting such an attack. I.e: $P_e^{attack}(k) = \frac{RD_e^{attack}(k)}{C^{attack}(k)}$. Similarly, we define $P_p^{attack}(k) = \frac{RD_p^{attack}(k)}{C^{attack}(k)}$.

We define the cost of an attack as the average power of the attack. Thus, cost of DDoS attack is k and the cost of Yo-Yo attack is $C^{YoYo}(k) = k\cdot\frac{t_{on}}{T}$. We simplified and assume again that values of the scale-up parameters equal to those of the scale- down parameters. Thus, in our analysis $C^{YoYo}(k) = \frac{k}{2}$.

Table III summarizes and compares the relative damage and the potency of the DDoS and Yo-Yo attack with different auto-scaling policies.

[5]Note that during t_{off} the load on the machine is better than in the steady state. This fact is insignificant to our analysis.

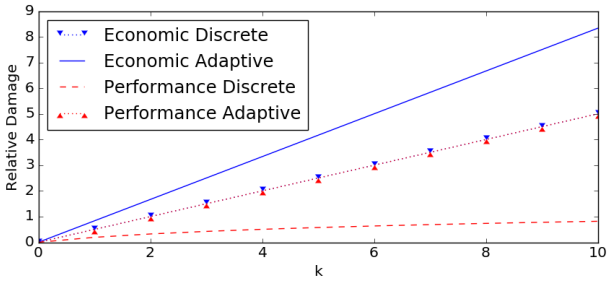

Fig. 5: Relative damage of YoYo attack with different scaling policy as function of the attack's power.

D. Discussion

In the discrete scale policy, the victim will pay for approximately $\frac{k}{2}$ more machines, and an average extra load on each machine will be logarithmic in k. In the adaptive model, on the other hand, the economic damage and the extra load are linear with k.

Figure 5 compares the relative damage as a function of k for the two auto-scaling policies [6]. Although the adaptive policy responds more quickly to changes, we can see that the relative economic and performance damages are larger for the adaptive policy than for the discrete policy. This is because the greater adaptivity to the load under the former policy is countered by the substantial W_{up} time during which the user has to pay for *all* the machines but receives no performance boost.

In Table III, the performance and economic potency values (which is the relative damage divided by cost) show that the Yo-Yo attack is able to do more damage per unit cost than the DDoS attack. Nowadays, the cost of an attack often translates to the real cost of renting an army of botnets. The fact that the Yo-Yo attack is less costly to perpetrate opens the door to attackers who wish to cause harm to larger services.

In order to get some sense of the actual damage inflicted by the Yo-Yo attack, we analyze the results of the representative use case of a medium-size commercial site (see Table II for the parameters of the site). Table IV summarizes our findings. While the Yo-Yo attack can be carried out at half the cost of DDoS attack, the damage with the adaptive policy is 160% more machines as opposed to of 200% more machines in DDoS with autoscaling. Moreover, in DDoS with autoscaling, in the longrun there is no extra load on machine, while in Yo-Yo attack the load on each machine is 100% more comparing to the steady state.

VI. Experimental Evaluation

In this section, we show a proof of concept of the Yo-Yo attack on Amazon's Cloud service and evaluate it.

Our environment in Amazon consists of a simple HTTP server, front-end side stateless without back-end.[7] In our experiment we choose the CPU utilization criterion, and correspondingly implement the http server where each connection

[6]Again we assume that the scale up and scale down parameters are equal

[7]We wanted to evaluate the damage at front-end side without being influenced by damage at back-end side.

	DDoS	DDoS with autoscaling	Yo-Yo Adaptive policy	Yo-Yo Discrete Policy
Attack cost C(k)	k	k	$\frac{k}{2}$	$\frac{k}{2}$
Relative Economic Damage $RD_e(k)$	0	k	$k \cdot \frac{T - I^a_{up}}{T^a}$	$\approx \frac{k}{2}$
Relative Performance Damage $RD_p(k)$	k	0	$\frac{k}{2}$	$\approx \frac{ln(k+1)-1}{2}$
Economic Potency $P_e(k)$	0	1	$\frac{T^a - I^a_{up}}{I^a_{up} + W_{up}}$	1
Performance Potency $P_p(k)$	1	0	1	$\approx \frac{ln(k+1)-1}{k}$

TABLE III: Summary of damage and potency in various environments according to the analysis

Attack Scale policy	DDoS with autoscaling	Yo-Yo Adaptive	Yo-Yo Discrete
$[\frac{t_{on}}{T}, \frac{t_{off}}{T}]$	$[1, 0]$	$[\frac{1}{2}, \frac{1}{2}]$	$[\frac{1}{2}, \frac{1}{2}]$
Cost with power of attack = 2	2	1	1
Relative Economic Damage	$\approx 200\%$	$\approx 166\%$	$\approx 100\%$
Relative Performance Damage	$\approx 0\%$	$\approx 100\%$	$\approx 30\%$
Economic Potency	$\approx 100\%$	$\approx 166\%$	$\approx 100\%$
Performance Potency	$\approx 0\%$	$\approx 100\%$	$\approx 30\%$

TABLE IV: Use-case analysis

Fig. 6: Yo-Yo attack on system with adaptive scale policy

requires high CPU consumption. We use Amazon CloudWatch [6] to monitor our instances and manage the auto-scale group for scale triggers on high or low CPU utilization. In our experiments, we begin with a single machine. In the steady state our http server handles 10 dynamic http requests per second, and in the on-attack phase, our attacker implements an attack with power of attack 4, i.e., adds an additional 40 requests per second.

Under the discrete policy type, we configure scale-up to be performed if CPU utilization is over 50% for 1 minute, and scale-down to be performed if CPU utilization is below 10% for 1 minute.

Under the adaptive policy type we configure scaling with steps, which scales up to 4 machines, when the CPU utilization is for 1 minute in a scale between 50% to 80% and scales down to 1 machines, when CPU utilization is for 1 minute in a scale between 10% to 30%.

Figures 7 and 6 show the number of machines and the response time under adaptive and discrete policies, respectively. The figures clearly show economic damage in both cases, along with an increase in the number of machines, as predicted by the model. Likewise, the response time graphs in both figures are similar in shape to the load per machines graphs produced by our model. Clearly the load on each machine and the response time are correlated. We represent in one plot the http error percent [8] and in the other the average response time. Note that our system still suffers from errors after the attack has ended. We speculate that there are two reasons for this behavior: first, it takes the Amazon load balancer time to recover [14], and second, there is an overload of Http retries after the attacker has stopped sending traffic. We want to model these phenomena in the future. We did a similar test with Https, and similar to the findings of [24], the performance degradation increased even further.

Note that even with our small program we measured a warming time of around 3 minutes.

A. *Detecting Scale Policy*

In this subsection we show how an attacker can approximate the autoscaling state and configuration in order to maximize the damage from the attack.

In order to detect when the scale-up has ended, the attacker can send probe requests and check their response time. Figure

[8] Http Errors are usually thrown when the server is under heavy load and is able to receive requests but not able to process them.

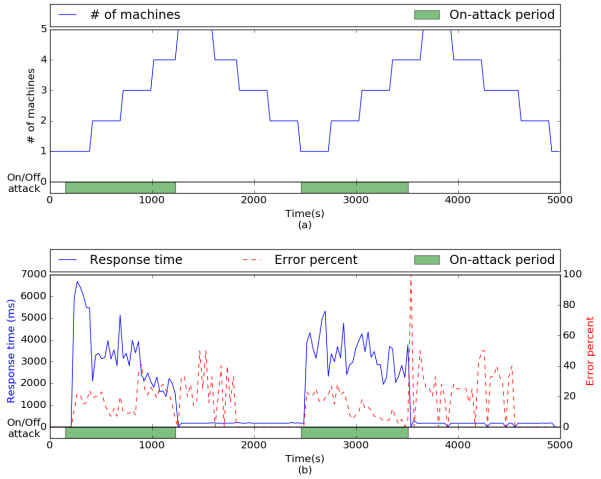

Fig. 7: Yo-Yo attack on system with discrete scale policy

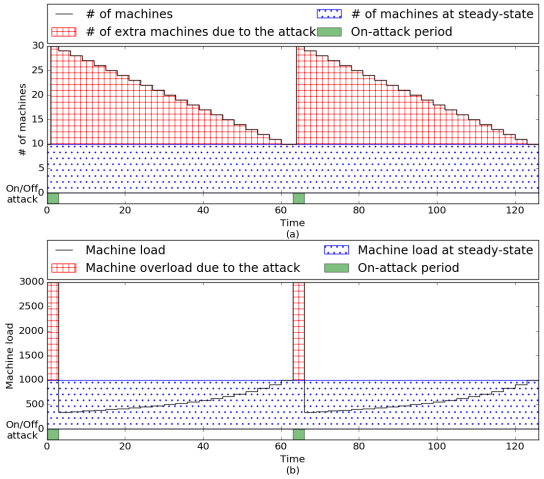

Fig. 8: Yo-Yo attack on system with adaptive scale up policy and discrete scale down policy

7 shows that the probe requests will suffer from a slow response time and high packet lose if the scale-up process has not ended. Our rule of thumb was that the scale-up process has not yet ended if the response time is over $1000ms$. Note that this technique applies both to the discrete and the adaptive policy.

Identifying when the scale-down process has ended is much more complicated. The basic technique is to send a small burst of traffic (small enough not to trigger the scaling mechanism), and to send in parallel probe packets to measure the response time. The impact of the extra load will be immediately noticed in the response time (in our experiments it was noticeable in less than 5 seconds).

In a pure adaptive policy, all the extra machines can be up or down; it is never the case that only some remains up or down. If the response time is smaller than $1000ms$, then all the extra machines are still up; otherwise, the scale-down process has ended. But under the discrete policy, scale down is a continuous process in which the extra machines are gradually shut down. This makes things more difficult for the attacker, who first must learn the response time when there is a burst in the load but no extra machines. The attacker can check this before lunching the attack. This is part of our future work.

VII. Defense Strategies For The Yo-Yo Attack

One obvious remedy is to mitigate the "DDoS attack parts" in the Yo-Yo attack, i.e., identify the fake requests, using a DDoS scrubber that filters out the attack. Handling Yo-Yo attacks also requires that adjustments be made to scrubber solutions, since they would need to be able to analyze and identify attacks that come in waves, even short ones. Moreover, for some dynamic DDoS attacks it is very difficult to identify and filter the malicious traffic.

Here, we focus on auto-scaling configuration changes that can be used to further decrease the effect of the attack.

- **Scaling configuration - Scale up early, Scale down slowly:** Performance damage can be reduced by early scale-up. A closer look at the model of the attack shows that warming time plays a major role in the damage, especially in the discrete model. During the scale-up warming time, the user pays for the extra machines but they do not help cope with the load. Thus one effort should be to minimize the warming time. Intensive efforts have been made to reduce that warming time [21] but this is a challenging task, due to the time required (several minutes) for the application to setup.
  An additional option is to keep some unused capacity available for quick response. Under this approach the user pays for the unused machines all the time.
  Another option is not to rush to scale down the machines immediately after the attack stops. In other words, the idea is to configure a longer I_{down}. This is also part of the recommendation of Amazon [7], due to the evidence that many attacks come in waves. We note that in the Google cloud, scaling operations are adaptive and the autoscaler has a fixed scale-down delay of 10 minutes built-in. From a discussion with several cloud users, we learned that many of them have configured I_{down} to be much longer than I_{up}, in order to handle DDoS attacks that come in waves. We note that while it was known that there are DDoS attacks that comes in waves, it was not known that auto-scaling could be used by an attacker to further increase the damage.
  The phrase "scale up early, scale down slowly" appeared in Netflix's recommendation [28] for auto-scaling in Amazon's EC2 environment but not in the context of attacks. This solution does reduce the performance damage, but increases the economic damage. Figure 8 shows a simulation of the Yo-Yo attack on such an environment.
- **Restrictions - Limiting the resources:**
  In order to avoid unexpected expenses, it is possible to set the maximum number of machines allowed to scale and limit the system resources. However, resource limitation also restricts the autoscaler operating range and may cause denial of service, so attention is required when

employing this strategy.

To cocnlude, there is a trade-off between paying for high service cost (while scaling down slowly) or suffering low performance (while limiting resources). System administrator can configure their system in accordance with what they are willing to compromise .

VIII. Conclusions And Future Work

In this work we shed light on the potential of exploiting the auto-scaling mechanism to perform an efficient attack, Yo-Yo attack, that impacts the cost and the performance. We discuss various auto-scaling parameters and their influence on the damage inflicted by the Yo-Yo attack.

An open question is how to fully cope with Yo-Yo and similar attacks that leverage the trade-off between DDoS and EDoS in cloud-based services. We proposed some preliminary thoughts on this complex issue.

The trend of virtualization and cloud services also impact the area of network services, which are part of the Network Function Virtualization (NFV) revolution. Part of the promise of NFV is that network services, among them middleboxes, will enjoy the auto-scaling property. Our work shows that the auto-scaling feature is vulnerable to attack, and that special care should be taken to prevent attacks in the NFV scenario where Yo-Yo attack will harm crucial network services.

Acknowledgments

This research was supported by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013)/ERC Grant agreement no 259085.

References

[1] Akamai, "How to protect against DDoS attacks - stop denial of service," https://www.akamai.com/us/en/resources/protect-against-ddos-attacks.jsp.

[2] A. Ali-Eldin, M. Kihl, J. Tordsson, and E. Elmroth, "Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control," in *Proceedings of the 3rd Workshop on Scientific Cloud Computing Date, 2012*, pp. 31–40.

[3] A. Ali-Eldin, J. Tordsson, and E. Elmroth, "An adaptive hybrid elasticity controller for cloud infrastructures," in *Network Operations and Management Symposium (NOMS), 2012*, pp. 204–212.

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

[5] "Amazon CloudFronth," https://aws.amazon.com/cloudfront.

[6] "Amazon CloudWatch," https://aws.amazon.com/cloudwatch.

[7] "AWS best practices for DDoS resiliency, 2015," https://d0.awsstatic.com/whitepapers/DDoS_White_Paper_June2015.pdf.

[8] "Amazon Elastic Load Balancing," https://aws.amazon.com/elasticloadbalancing.

[9] "Amazon Route 53," https://aws.amazon.com/route53.

[10] 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 International Conference on*, pp. 346–353.

[11] A. D. George, "How to autoscale an application," https://azure.microsoft.com/en-us/documentation/articles/cloud-services-how-to-scale, Microsoft Azure.

[12] Google, "Autoscaling groups of instances," https://cloud.google.com/compute/docs/autoscaler.

[13] M. Guirguis, A. Bestavros, and I. Matta, "Exploiting the transients of adaptation for RoQ attacks on internet resources," in *Network Protocols, 2004. ICNP 2004. Proceedings of the 12th IEEE International Conference on*. IEEE, pp. 184–195.

[14] 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*, pp. 857–865.

[15] ——, "Reduction of quality (RoQ) attacks on internet end-systems," in *INFOCOM 2005*, vol. 2, pp. 1362–1372.

[16] C. Hoff, "Cloud computing security: From DDoS (distributed denial of service) to EDoS (economic denial of sustainability)," http://www.rationalsurvivability.com/blog/?p=66, 2008.

[17] R. Jellinek, Y. Zhai, T. Ristenpart, and M. Swift, "A day late and a dollar short: The case for research on cloud billing systems," in *HotCloud 2014*.

[18] P. S. M. K. M. Kumar, R. Korra, P. Sujatha, and M. Kumar, "Mitigation of economic distributed denial of sustainability (EDDoS) in cloud computing," in *Proc. of the IntlConf. on Advances in Engineering and Technology, 2011*.

[19] A. Kuzmanovic and E. W. Knightly, "Low-rate tcp-targeted denial of service attacks: the shrew vs. the mice and elephants," in *Proceedings of SIGCOMM 2003*. ACM, pp. 75–86.

[20] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *Cloud Computing (CLOUD) 2012*, pp. 423–430.

[21] J. Martins, M. Ahmed, C. Raiciu, and F. Huici, "Enabling fast, dynamic network processing with clickos," in *Proceedings of HotSDN 2013*, pp. 67–72.

[22] I. M. Mary, P. Kavitha, M. Priyadharshini, and V. S. Ramana, "Secure cloud computing environment against DDoS and EDoS attacks." Citeseer, 2014.

[23] R. Miao, M. Yu, and N. Jain, "NIMBUS: cloud-scale attack detection and mitigation," in *ACM SIGCOMM Computer Communication Review, 2014*, vol. 44, no. 4, pp. 121–122.

[24] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste, "The cost of the s in https," in *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies, 2014*. ACM.

[25] "NIST Internet Time Service," http://www.nist.gov/pml/div688/grp40/its.cfm.

[26] "NTP - Network Time Protocol," https://en.wikipedia.org/wiki/Network_Time_Protocol.

[27] "Heat: Openstack orchestration," https://wiki.openstack.org/wiki/Heat.

[28] G. Orzell and J. Becker, "The Netflix tech blog: Auto scaling in the Amazon cloud," http://techblog.netflix.com/2012/01/auto-scaling-in-amazon-cloud.html, 2012.

[29] "Private discussion with site administrators," 2016.

[30] Radware, "DDoS attack protection and mitigation," http://www.radware.com/Products/DefensePro.

[31] M. Sides, A. Bremler-Barr, and E. Rosensweig, "Yo-Yo Attack: vulnerability in auto-scaling mechanism," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pp. 103–104.

[32] G. Somani, M. S. Gaur, and D. Sanghi, "DDoS/EDoS attack in cloud: affecting everyone out there!" in *Proceedings of the 8th International Conference on Security of Information and Networks, 2015*. ACM, pp. 169–176.

[33] M. H. Sqalli, F. Al-Haidari, and K. Salah, "EDoS-shield-a two-steps mitigation technique against edos attacks in cloud computing," in *Utility and Cloud Computing (UCC), 2011*, pp. 49–56.

[34] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova, "Resource allocation algorithms for virtualized service hosting platforms," *Journal of Parallel and Distributed Computing 2010*, vol. 70, no. 9, pp. 962–974.

[35] "TCP SYN Flooding and IP Spoofing Attacks," http://www.cert.org/historical/advisories/CA-1996-21.cfm?

[36] "Technical report," https://www.dropbox.com/s/m19lbn41e5dpywp/DDoSandCloudAuto-Scaling.rar?dl=0.

[37] L. M. Vaquero, L. Rodero-Merino, and R. Buyya, "Dynamically scaling applications in the cloud," *ACM SIGCOMM Computer Communication Review, 2011*, vol. 41, no. 1, pp. 45–52.

[38] S. VivinSandar and S. Shenai, "Economic denial of sustainability (EDoS) in cloud services using http and xml based DDoS attacks," *International Journal of Computer Application 2012*, vol. 41, no. 20.

[39] S. Yu, Y. Tian, S. Guo, and D. O. Wu, "Can we beat DDoS attacks in clouds?" *IEEE Transactions on Parallel and Distributed Systems, 2014*, vol. 25, no. 9, pp. 2245–2254.